TheoInf - Complete

Type	Book
Date	Oct 14, 2020
Additional Files	Empty
Class	TheoInf
PPT	Empty

Add a property

Add a comment…

2. Alphabete, Wörter, Sprachen und die Darstellung von Problemen

2.2 Alphabete, Wörter und Sprachen

Definition: Eine endliche nichtleere Menge $\sum$ heisst **Alphabet**. Die Elemente eines Alphabets werden **Buchstaben** (Zeichen, Symbole) genannt.

Einige Beispiele:

- $\sum_{\text{bool}} = \{0, 1\}$
- $\sum_{\text{lat}} = \{a, b, c, ..., z\}$
- $\sum_{\text{Tastatur}} = \{A, B, ..., Z, <, >, ..., !\}$

Definition: Ein **Wort** über ein Alphabet $\sum$ ist eine endliche (eventuell leere) Folge von Buchstaben aus $\sum$. Das leere Wort λ ist die leere Buchstabenfolge. Σ^* ist die Menge aller Wörter über Σ, Σ^+ ist die Menge aller Wörter über Σ^* ohne das leere Wort, also $\Sigma^+ = \Sigma^* - \{\lambda\}$.

Definition: Die **Verkettung** für ein Alphabet $\sum$ ist eine Abbildung $\text{Kon}: \sum^* \times \sum^* \to \sum^*$, so dass $\text{Kon}(x, y) = x \cdot y = xy$ ist.

Definition: Eine Sprache L über ein Alphabet $\sum$ ist eine Teilemenge von $\sum^*$. Das Komplement L^C der Sprache L bezüglich $\sum$ ist die Sprache $\sum^* - L$.

Lemma 2.1: Seien $L_1,\ L_2$ und L_3 Sprachen über einem Alphabet $\sum$. Dann gilt $L_1L_2 \cup L_1L_3 = L_1(L_2 \cup L_3)$.

Lemma 2.2: Seien $L_1,\ L_2$ und L_3 Sprachen über ein Alphabet $\sum$. Dann gilt $L_1(L_2 \cap L_3) \subseteq L_1L_2 \cap L_1L_3$.

Definition: Seien $\sum_1$ und $\sum_2$ zwei beliebige Alphabete. Ein **Homomorphismus** von $\sum_1^*$ nach $\sum_2^*$ ist eine Funktion $h : \sum_1^* \to \sum_2^*$ mit den folgenden Eigenschaften:

- $h(\lambda) = \lambda$
- $h(uv) = h(u) \cdot h(v)$ für alle $u, v \in \sum_1^*$

2.3 Algorithmische Probleme

Definition: Das **Entscheidungsproblem** (Σ, L) für ein gegebenes Alphabet Σ und eine gegebene Sprache $L \subseteq \sum^*$ ist, für jedes $x \in \Sigma^*$ zu entscheiden, ob $x \in L$ oder $x \notin L$. Ein Algorithmus A **löst** das Entscheidungsproblem (Σ, L), falls für alle $x \in \Sigma^*$ gilt:

$$A(x) = \begin{cases} 1, & \text{falls } x \in L \\ 0, & \text{falls } x \notin L \end{cases}$$

Wir sagen auch, dass A die Sprache L erkennt.

Wenn für eine Sprache L ein Algorithmus existiert, der L erkennt, werden wir sagen, dass L **rekursiv** ist.

Definition: Sei Σ ein Alphabet, und sei $x \in \Sigma^*$. Wir sagen, dass ein Algorithmus A das Wort x **generiert**, falls A für die Eingabe λ die Ausgabe x liefert.

Definition: Sei Σ ein Alphabet, und sei $L \subseteq \Sigma^*$. A ist ein **Aufzählungsalgorithmus für** L, falls A für jede Eingabe $n \in \mathbb{N} - \{0\}$ die Wortfolge $x_1,\ x_2, ...,\ x_n$ ausgibt, wobei $x_1,\ x_2, ...,\ x_n$die kanonisch n ersten Wörter in L sind.

2.4 Kolmogorov-Komplexität

Definition: Für jedes Wort $x \in (\Sigma_{\text{bool}})^*$ ist die **Kolmogorov-Komplexität** $K(x)$ des Wortes x das Minimum der binären Längen der Pascal-Programme, die x generieren.

Lemma 2.4: Es existiert eine Konstante d, so dass für jedes $x \in (\Sigma_{\text{bool}})^*$ gilt, dass $K(x) \leq |x| + d$.

Definition: Die Kolmogorov-Komplexität einer natürlichen Zahl n ist $K(n) = K(Bin(n))$.

Lemma 2.5: Für jede Zahl $n \in \mathbb{N} - \{0\}$ existiert ein Wort $w_n \in (\Sigma_{\text{bool}})^n$, so dass $K(w_n) \geq |w_n| = n$, d.h., es existiert für jede Zahl n ein nichtkomprimierbares Wort der Länge n.

Definition: Ein Wort $x \in (\Sigma_{\text{bool}})^*$ heisst **zufällig**, falls $K(x) \geq |x|$. Eine Zahl n heisst **zufällig**, falls $K(n) = K(\text{Bin}(n)) \geq \lceil \log_2(n+1) \rceil - 1$.

Satz 2.2: Sei L eine Sprache über Σ_{bool}. Sei, für jedes $n \in \mathbb{N} - \{0\}$, z_n das n-te Wort in L bezüglich der kanonischen Ordnung. Wenn ein Programm A_L existiert, das das Entscheidungsproblem $(\Sigma_{\text{bool}},\ L)$ löst, dann gilt für alle $n \in \mathbb{N} - \{0\}$, dass

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

wobei c eine von n unabhängige Konstante ist.

> **Satz 2.3 (Primzahlsatz):** Es gilt $\lim_{n \to \infty} \frac{\text{Prim}(n)}{n / \ln(n)} = 1$.

Satz 2.4: Für unendlich viele $k \in \mathbb{N}$ gilt:

$$\text{Prim}(k) \geq \frac{k}{2^{17} \log_2(k) \cdot (\log_2(\log_2(k)))^2}$$

3. Endliche Automaten

3.2 Die Darstellung endlicher Automaten

Wir führen die folgende visuelle Definition zu endlichen Automaten ein: Ein endlicher Automat lässt sich als Graphen beschreiben. Dabei hat der Graph $G(A)$ genau so viele Knoten wie das Programm Zeilen hat. Zwischen zwei Knoten i und j setzten wir eine Kante mit Markierung b, falls wir im Programm von Zeile i zu j gelangen, in dem wir in Zeile i die Eingabe b erhalten.

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

1. Q eine endliche Menge von *Zuständen* ist,
2. Σ ein Alphabet, genannt *Eingabealphabet*, ist,
3. $q_0 \in Q$ der *Anfangszustand* ist,
4. $F \subseteq Q$ die *Menge der akzeptierten Zustände* ist und
5. δ eine Funktion von $Q \times \Sigma$ nach Q ist, die *Übergangsfunktion* genannt wird.

Eine *Konfiguration* von M ist ein Element aus $Q \times \Sigma^*$. Die Konfiguration $(q_0, x) \in \{q_0\} \times \Sigma^*$ heisst die *Startkonfiguration von M auf x*. Jede Konfiguration aus $Q \times \{\lambda\}$ nennt man die *Endkonfiguration*. Ein *Schritt* von M ist eine Relation auf Konfigurationen $\vdash_M \subseteq (Q \times \Sigma^*) \times (Q \times \Sigma^*)$, definiert durch

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

Eine *Berechnung* C von M ist eine endliche Folge $C = C_0, C_1, ..., C_n$ von Konfigurationen, so dass $C_i \vdash_M C_{i+1}$ für alle $0 \leq i \leq n-1$. C ist die Berechnung 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 $C_n \in F \times \{\lambda\}$, so sagen wir, dass C eine *akzeptierte Berechnung* von M auf x ist, und dass M *das Wort akzeptiert*. Falls $C_n \in (Q - F) \times \{\lambda\}$, sagen wir, dass C eine *verwerfende Berechnung* von M auf x ist, und dass M *das Wort verwirft*. Die von M akzeptierte Sprache $L(M)$ ist definiert als

$$L(M) = \{w \in \Sigma^* \mid die\ Berechnung\ von\ M\ auf \\ w\ endet\ in\ einer\ Endkonfiguration\ (q, \lambda)\ mit\ q \in F\}.$$

$\mathcal{L}_{EA} = \{L(M) \mid M\ ist\ ein\ EA\}$ ist die Klasse der Sprachen, die von endlichen Automaten akzeptiert werden. $\mathcal{L}_{EA}$ bezeichnet man auch als Klasse der regulären Sprachen, und jede Sprache L aus $\mathcal{L}_{EA}$ wird *regulär* genannt.

Definition: Sei $M = (Q, \Sigma, \delta, q_0, F)$ ein endlicher Automat. Wir definieren $\vdash_M^*$ als die **reflexive und translative Hülle der Schrittrelation** $\vdash_M$; daher ist

$$(q, w) \vdash_M^* (p, u) \Leftrightarrow (q = p \text{ und } w = u) \text{ oder } \exists k \in \mathbb{N} - \\ \{0\},\ \text{so dass}$$

1. $w = a_1 a_2 ... a_k u,\ a_i \in \Sigma$ für $i = 1, 2, ..., k$, und
2. $\exists r_1, r_2, ..., r_{k-1} \in Q$, so dass $(q, w) \vdash_M (r_1, a_2 ..., a_k u) \vdash_M (r_2, a_3 ... a_k u) \vdash_M ... (r_{k-1}, a_k u) \vdash_M (p, u)$.

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

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

Lemma: $L(M) = \{w \in \{0, 1\}^* \mid |w|_0 + |w|_1 \equiv 0 \bmod 2\}$.

3.3 Simulationen

Lemma 3.2: Sei Σ 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 $\odot \in \{\cup, \cap, -\}$ existiert eine EA M, so dass $L(M) = L(M_1) \odot L(M_2)$. Wir konstruieren $M = (Q, \Sigma, \delta, q_0, F_\odot)$ wie folgt:

- $Q = Q_1 \times Q_2$
- $q_0 = (q_{01}, q_{02})$
- für alle $q \in Q_1, p \in Q_2$ und $a \in \Sigma, \delta((q, p), a) = (\delta_1(q, a), \delta_2(p, a))$

- falls $\odot = \cup$, dann ist $F = F_1 \times Q_2 \cup Q_1 \times F_2$

  falls $\odot = \cap$, dann ist $F = F_1 \times F_2$

  falls $\odot = -$, dann ist $F = F_1 \times (Q_2 - F_2)$

3.4 Beweise der Nichtexistenz

Um zu zeigen, dass eine Sprache L nicht regulär ist ($L \notin \mathcal{L}_{EA}$), genügt es zu beweisen, dass es keinen EA gibt, der die Sprache akzeptiert. Im Allgemeinen zu zeigen, dass von einer gewissen Klasse von Programmen (Algorithmen) eine konkrete Aufgabe nicht lösbar ist (kein Programm aus der Klasse löst die Aufgabe), gehört zu den schwersten Problemstellungen in der Informatik. Beweise solcher Aussagen nennen wir die **Beweise der Nichtexistenz.**

Lemma 3.3: Sei $A = (Q, \Sigma, \delta_A, q_0, F)$ ein EA. Seien $x, y \in \Sigma^*, x \neq y$, so dass

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

für ein $p \in Q$ (also $\hat{\delta}_A(q_0, x) = \hat{\delta}_A(q_0, y) = p$ $(x, y \in \mathrm{Kl}[p])$). Dann existiert für jedes $z \in \Sigma^*$ ein $r \in Q$, so dass xz und $yz \in \mathrm{Kl}[r]$, also gilt insbesondere

$$xz \in L(A) \Leftrightarrow yz \in L(A).$$

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, das heisst $w = yxz$, wobei

1. $|yx| \leq n_0$,
2. $|x| \geq 1$ und
3. entweder $\{yx^kz \mid k \in \mathbb{N}\} \subseteq L$ oder $\{yx^kz \mid k \in \mathbb{N}\} \cap L \neq \emptyset$.

Satz 3.1: Sei $L \subseteq (\Sigma_{bool})^*$ eine reguläre Sprache. Sei $L_x = \{y \in (\Sigma_{bool})^* \mid xy \in L\}$ für jedes $x \in (\Sigma_{bool})^*$. Dann existiert eine Konstante const, so dass für alle $x, y \in (\Sigma_{bool})^*$

$$K(y) \leq \lceil \log_2(n+1) \rceil + \mathrm{cost},$$

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

3.5 Nichtdeterminismus

Die Wahl einer von mehreren Möglichkeiten nennen wir **nichtdeterministische Entscheidung.** Für ein Entscheidungsproblem $(\Sigma,\ L)$ bedeutet dies, dass ein nichtdeterministisches Programm A eine Sprache L akzeptiert, falls für jedes $x \in L$ mindestens eine akzeptierte Berechnung von A auf x existiert und für jedes $y \in \Sigma^* - L$ alle Berechnungen nicht-akzeptiert sind.

Definition: Ein **nichtdeterministischer endlicher Automat (NEA)** ist ein Quintupel $M = (Q,\ \Sigma,\ \delta,\ q_0,\ F)$. Dabei ist:

- Q eine endliche Menge, *Zustandsmenge* genannt,
- Σ ein Alphabet, *Eingabealphabet* genannt,
- $q_0 \in Q$ der *Anfangszustand*,
- $F \subseteq Q$ die Menge der *akzeptierten Zustände* und
- δ eine Funktion von $Q \times \Sigma$ nach $\mathcal{P}(Q)$, *Übergangsfunktion* genannt.

Eine Konfiguration von M ist ein Element aus $Q \times \Sigma^*$. Die Konfiguration $(q_0,\ x)$ ist die Startkonfiguration für das Wort x. Ein Schritt von M ist eine Relation $\vdash_M \subseteq (Q \times \Sigma^*) \times (Q \times \Sigma^*)$, definiert durch

$$(q,\ w) \vdash_M (p,\ x) \Leftrightarrow w = ax \text{ für ein } a \in \Sigma \text{ und } p \in \delta(q,\ a).$$

Eine Berechnung von M ist eine endliche Folge $D_1,\ D_2, ...,\ D_k$ von Konfigurationen, wobei $D_i \vdash D_{i+1}$ für $i = 1, ...,\ k-1$. Eine Berechnung von M auf x ist eine Berechnung $C_0,\ C_1, ...,\ C_m$ von M, wobei $C_0 = (q_0,\ x)$ und entweder $C_m \in Q \times \{\lambda\}$ oder $C_m = (q,\ ay)$ für ein $a \in \Sigma$, ein $y \in \Sigma^*$ und ein $q \in Q$, so dass $\delta(q,\ a) = \emptyset$. Die Berechnung $C_0,\ C_1, ...,\ C_m$ ist eine akzeptierte Berechnung von M auf x, wenn $C_m = (p,\ \lambda)$ für ein $p \in F$. Falls eine akzeptierte Berechnung von M auf x existiert, sagen wir auch, dass M das Wort akzeptiert. Die Relation $\vdash_M^*$ ist die reflexive und transitive Hülle von $\vdash_M$, genau wie bei einem EA. Die Sprache

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

ist die von M **akzeptierte Sprache.** Zu der Übergangsfunktion δ definieren wir die Funktion $\hat{\delta}$ von $Q \times \Sigma^*$ in $\mathcal{P}(Q)$ wie folgt:

1. $\hat{\delta}(q,\ \lambda) = \{q\}$ für jedes $q \in Q$,

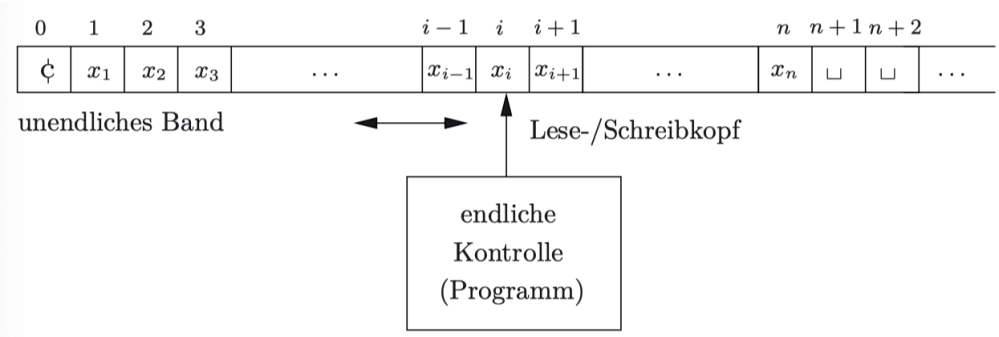
2. $\hat{\delta}(q,\ wa) = \{p \in Q \mid \text{es existiert ein } r \in \hat{\delta}(q,\ w),\ \text{so dass } p \in \delta(r,\ a)\} = \cup_{r \in \hat{\delta}(q,\ w)} \delta(r,\ a)$ für alle $q \in Q,\ a \in \Sigma,\ w \in \Sigma^*$.

Satz 3.2: Zu jedem $NEA\ M$ existiert ein $EA\ A$, so dass $L(M) = L(A)$.

Lemma 3.6: Für alle $k \in \mathbb{N} - \{0\}$ muss jeder EA, der L_k akzeptiert, mindestens 2^k Zustände haben.

4. Turingmaschinen

4.3 Das Modell der Turingmaschine



Eine **Turingmaschine** kann als eine Verallgemeinerung eines EA gesehen werden. Informell besteht sie aus:

- einer endlichen Kontrolle, die das Programm enthält,
- einem unendlichen Band, das als Eingabeband, aber auch als Speicher (Arbeitsband) zur Verfügung steht, und
- einem Lese-/Schreibkopf, der sich in beiden Richtungen auf dem Band bewegen kann.

Die Ähnlichkeit zu einem EA besteht in der Kontrolle über einer endlichen Zustandsmenge und dem Band, das am Anfang das Eingabewort enthält. Der Hauptunterschied zwischen Turingmaschine und endlichen Automaten besteht darin, dass eine Turingmaschine das Band auch als Speicher benutzen kann.

Definition: Eine **Turingmaschine** (TM) ist ein 7-Tupel $M = (Q,\ \Sigma,\ \Gamma,\ \delta,\ q_0,\ q_{\text{accept}},\ q_{\text{reject}})$. Dabei ist:

- Q eine endliche Menge, die *Zustandsmenge von* M genannt wird,
- Σ das *Eingabealphabet*, wobei ¢ und das Blanksymbol ␣ nicht in Σ sind,
- Γ ein Alphabet, *Arbeitsalphabet* genannt, wobei $\Sigma \subseteq \Gamma,\ ¢,\ ␣ \in \Gamma,\ \Gamma \cap Q = \emptyset$,

- $\delta : (Q - \{q_{\text{accept}},\ q_{\text{reject}}\}) \times \Gamma \to Q \times \Gamma \times \{L,\ R,\ N\}$ eine Abbildung, *Übergangsfunktion von* M genannt, mit der Eigenschaft $\delta(q,\ ¢) \in Q \times \{¢\} \times \{R,\ N\}$ für alle $q \in Q$,
- $q_0 \in Q$ der *Anfangszustand*,
- $q_{\text{accept}} \in Q$ der *akzeptierte Zustand*,
- $q_{\text{reject}} \in Q - \{q_{\text{accept}}\}$ der *verwerfende Zustand*.

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

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

Eine *Startkonfiguration* für ein Eingabewort x ist $q_0¢x$. Ein *Schritt von* M ist eine Relation $\vdash_M$ auf der Menge der Konfigurationen ($\vdash_M \subseteq \text{Konf}(M) \times \text{Konf}(M)$) definiert durch

1. $x_1x_2...x_{i-1}qx_ix_{i+1}...x_n \vdash_M x_1x_2...x_{i-1}pyx_{i+1}...x_n$, falls $\delta(q,\ x_i) = (p,\ y,\ N)$ → Abb. 4.3(a)
2. $x_1x_2...x_{i-1}qx_ix_{i+1}...x_n \vdash_M x_1x_2...x_{i-2}px_{i-1}yx_{i+1}...x_n$, falls $\delta(q,\ x_i) = (p,\ y,\ L)$ → Abb. 4.3(b)
3. $x_1x_2...x_{i-1}qx_ix_{i+1}...x_n \vdash_M x_1x_2...x_{i-1}ypx_{i+1}...x_n$, falls $\delta(q,\ x_i) = (p,\ y,\ R)$ für $i < n$ → Abb. 4.3(c)
4. $x_1x_2...x_{n-1}qx_n \vdash_M x_1x_2...x_{n-1}yp␣$, falls $\delta(q,\ x_n) = (p,\ y,\ R)$ → Abb. 4.3(d)

Eine *Berechnung von* M ist eine potentiell unendliche Folge von Konfigurationen $C_0,\ C_1, ...$, so dass $C_i \vdash_M C_{i+1}$ für alle $i = 0,\ 1,\ 2, ...$ Wenn $C_0 \vdash_M C_1 \vdash \cdots \vdash_M C_i$ für ein $i \in \mathbb{N}$, dann gilt $C_0 \vdash_M^* C_i$. Die **Berechnung von M auf einer Eingabe von** x ist eine Berechnung, die mit der Startkonfiguration $C_0 = q_0¢x$ beginnt und entweder unendlich ist oder in einer Konfiguration w_1qw_2 endet, wobei $q \in \{q_{\text{accept}},\ q_{\text{reject}}\}$. Eine Berechnung von M auf x heisst:

- *akzeptiert*, falls sie in $w_1q_{\text{accept}}w_2$ endet,
- *verwerfend*, falls sie in $w_1q_{\text{reject}}w_2$ endet,
- *nicht-akzeptiert*, falls sie verwerfend oder unendlich ist.

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

$$L(M) = \{w \in \Sigma^* \mid q_0¢w \vdash_M^* yq_{\text{accept}}z,\ \text{für irgendwelche } y, z \in \Gamma^*\}.$$

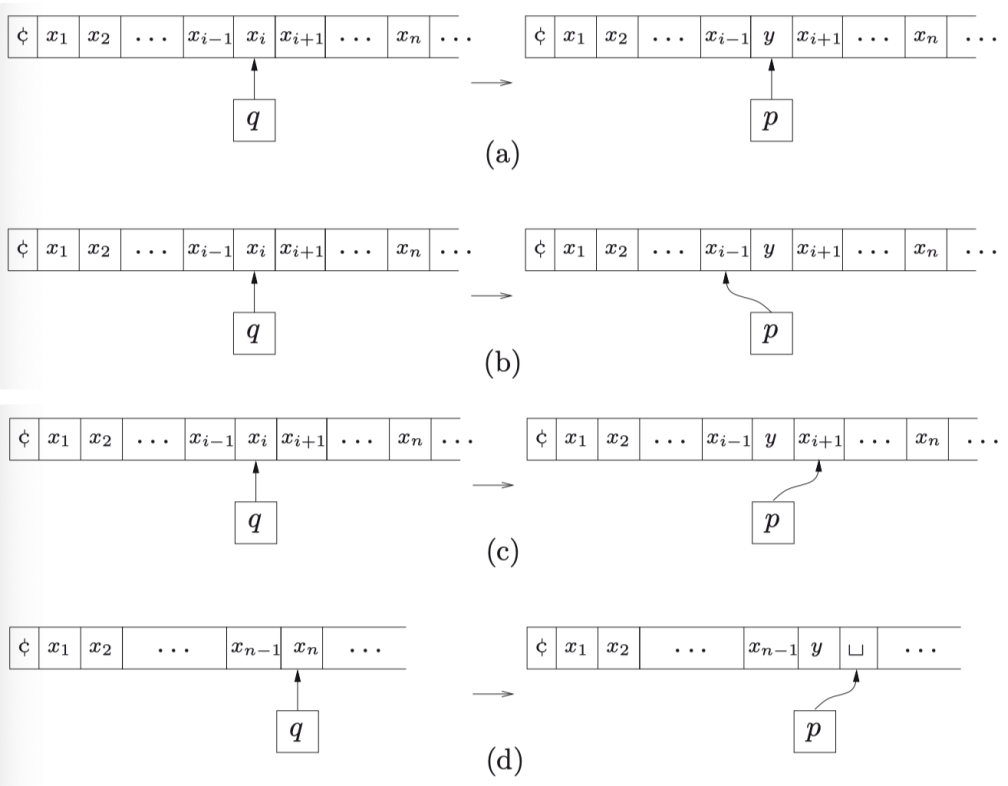
Wir sagen, dass M eine Funktion $F : \Sigma^* \to \Gamma^*$ *berechnet*, falls für alle $x \in \Sigma^* : q_0 ¢ x \vdash_M^* q_{\text{accept}} ¢ F(x)$. Eine Sprache $L \subseteq \Sigma^*$ heisst **rekursiv aufzählbar,** falls eine $TM\ M$ existiert, so dass $L = L(M)$.

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

ist die *Klasse aller rekursiv aufzählbaren Sprachen*. Eine Sprache $L \subseteq \Sigma^*$ heisst **rekursiv,** falls $L = L(M)$ für eine $TM\ M$, für die für alle $x \in \Sigma^*$ gilt:

1. $q_0 ¢ x \vdash_M^* y q_{\text{accept}} z,\ y, z \in \Gamma^*,$ falls $x \in L$ und
2. $q_0 ¢ x \vdash_M^* u q_{\text{reject}} v,\ u, v \in \Gamma^*,$ falls $x \notin L$.

Wenn (1) und (2) gelten, sagen wir, dass M auf *jeder Eingabe hält* oder dass M *immer hält*. Eine Funktion $F : \Sigma_1^* \to \Sigma_2^*$ für zwei Alphabete $\Sigma_1,\ \Sigma_2$ heisst *total berechenbar*, falls eine $TM\ M$ existiert, die F berechnet.



4.4 Mehrband-Turingmaschinen und Church'sche These

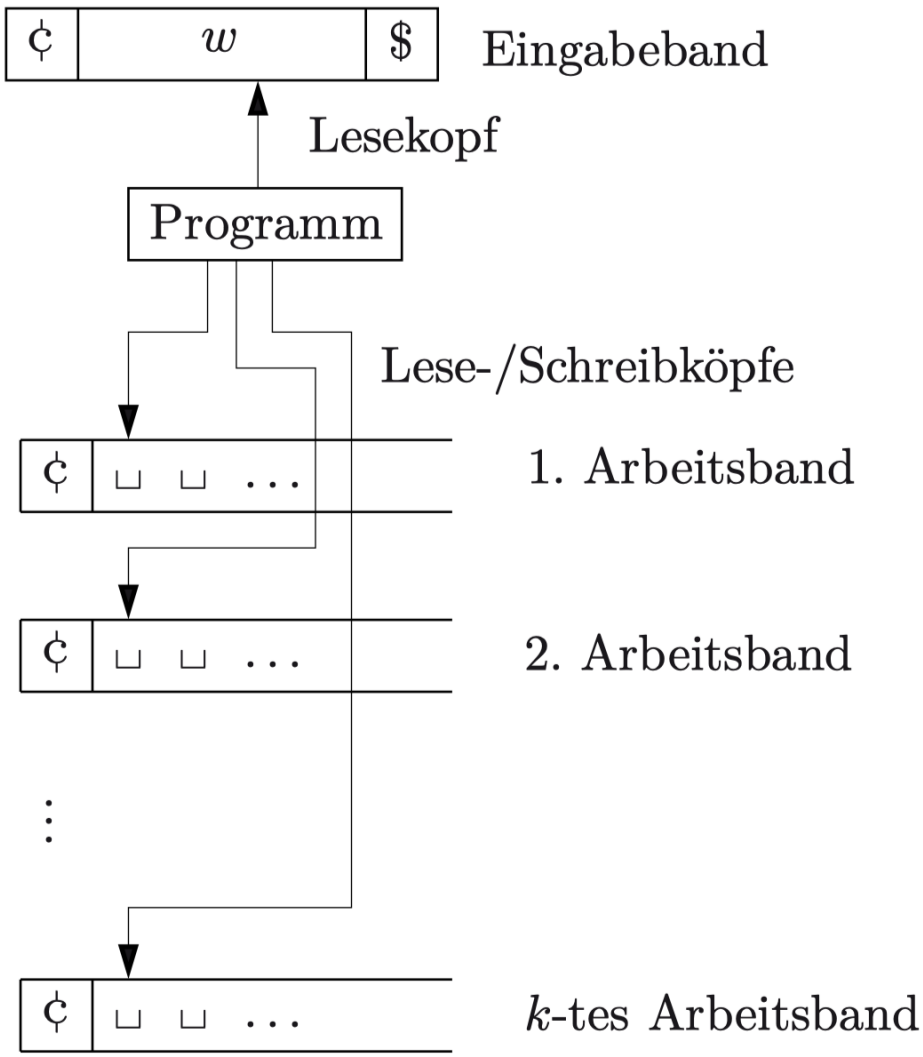
Das folgende Modell der **Mehrband-Turingmaschine** ist das grundlegende Modell der *Komplexitätstheorie*. Für jede positive ganze Zahl k hat eine k -Band-Turingmaschine folgende Komponenten:

- eine endliche Kontrolle (Programm)

- ein endliches Band mit einem Lesekopf
- k Arbeitsbänder, jedes mit eigenem Lese-/Schreibkopf

Am Anfang jeder Berechnung auf einem Wort w ist die k-Band-Turingmaschine in folgender Situation:

- Das Eingabeband enthält $¢w\$$, wobei $¢$ und $\$$ die linke bzw. rechte Seite der Eingabe markieren
- Der Lesekopf des Eingabebandes zeigt auf $¢$
- Der Inhalt aller Arbeitsbänder ist $¢ \sqcup \sqcup \sqcup \ldots$ und deren Lese-/Schreibköpfe zeigen auf $¢$
- Die endliche Kontrolle ist im Anfangszustand q_0



Für jedes $k \in \mathbb{N} - \{0\}$ nennen wir eine k-Band-Turingmaschine auch **Mehrband-Turingmaschine (*MTM*).**

Lemma: Zu jeder $TM\ A$ existiert eine zu A äquivalente 1-Band-Turingmaschine $TM\ B$.

Lemma: Zu jeder Mehrband-$TM\ A$ existiert eine zu A äquivalente $TM\ B$.

Definition: Zwei Maschinenmodelle (*Maschinenklassen*) $\mathcal{A}$ und $\mathcal{B}$ für Entscheidungsprobleme sind **äquivalent,** falls:

1. für jede Maschine $A \in \mathcal{A}$ eine zu A äquivalente Maschine $B \in \mathcal{B}$ existiert, und
2. für jede Maschine $C \in \mathcal{B}$ eine zu C äquivalente Maschine $D \in \mathcal{A}$ existiert.

Satz: 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.*

4.5 Nichtdeterministische Turingmaschinen

Definition: Eine **nichtdeterministische Turingmaschine (*NTM*)** ist ein 7-Tupel $M = (Q,\ \Sigma,\ \Gamma,\ \delta,\ q_0,\ q_{accept},\ q_{reject})$, wobei

- $Q,\ \Sigma,\ \Gamma,\ \delta,\ q_0,\ q_{accept},\ q_{reject}$ die gleiche Bedeutung haben wie bei einer TM haben, und
- $\delta : (Q - \{q_{accept},\ q_{reject}\}) \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L,\ R,\ N\})$ die *Übergangsfunktion* von M ist und die folgende Eigenschaft hat:

$$\delta(p,\ ¢) \subseteq \{(q,\ ¢,\ X) \mid q \in Q,\ X \in \{R,\ N\}\}$$

  für alle $p \in Q$.

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

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

Die Konfiguration $q_0¢w$ ist die *Anfangskonfiguration* für das Wort $w \in \Sigma^*$. Eine Konfiguration heisst *akzeptiert*, falls sie den Zustand q_{accept} enthält. Eine Konfiguration ist *verwerfend*, falls sie den Zustand q_{reject} enthält.

Ein ***Schritt*** von M ist eine Relation $\vdash_M$, die auf der Menge der Konfigurationen ($\vdash_M \subseteq \text{Konf}(M) \times \text{Konf}(M)$) wie folgt definiert ist. Für alle $p, q \in Q$ und alle $x_1,\ x_2, ...,\ x_n,\ y \in \Gamma$ gilt

- $x_1x_2...x_{i-1}qx_ix_{i+1}...x_n \vdash_M x_1x_2...x_{i-1}pyx_{i+1}...x_n$ falls $(p,\ y\ N) \in \delta(q,\ x_i)$
- $x_1x_2...x_{i-2}x_{i-1}qx_ix_{i+1}...x_n \vdash_M x_1x_2...x_{i-2}px_{i-1}yx_{i+1}...x_n$ falls $(p,\ y,\ L) \in \delta(q,\ x_i)$
- $x_1x_2...x_{i-1}qx_ix_{i+1}...x_n \vdash_M x_1x_2...x_{i-1}ypx_{i+1}...x_n$ falls $(p,\ y,\ R) \in \delta(q,\ x_i)$ für $i < n$
- $x_1x_2...x_{n-1}qx_n \vdash_M x_1x_2...x_{n-1}yp␣$ falls $(p,\ y,\ R) \in \delta(q,\ x_n)$

Die Relation $\vdash_M^*$ ist die reflexive und transitive Hülle von $\vdash_M$.

Eine ***Berechnung von*** M ist eine Folge von Konfigurationen $C_0,\ C_1, ...$, so dass $C_i \vdash_M C_{i+1}$ für $i = 0,\ 1,\ 2, ...$ Eine *Berechnung von M auf einer Eingabe x* ist eine Berechnung, die mit der Anfangskonfiguration $q_0¢x$ beginnt und entweder unendlich ist oder in einer Konfiguration w_1qw_2 endet, wobei $q \in \{q_{accept},\ q_{reject}\}$. Eine Berechnung von M auf x heisst *akzeptierend*, falls sie in einer akzeptierenden Konfiguration endet und *verwerfend*, falls sie in einer verwerfenden Konfiguration endet.

Die ***von der NTM M akzeptierten Sprache*** ist

$$L(M) = \{w \in \Sigma^* \mid q_0¢w \vdash_M^* \\ yq_{accept}z \text{ für irgendwelche } y, z \in \Gamma^*\}.$$

Definition: Sei $M = (Q,\ \Sigma,\ \Gamma,\ \delta,\ q_0,\ q_{accept},\ q_{reject})$ eine NTM und sei x ein Wort über dem Eingabealphabet Σ von M. Ein **Berechnungsbaum $T_{M,\,x}$ von M auf x** ist ein potentiell unendlicher gerichteter Baum mit einer Wurzel, der wie folgt definiert wird

1. Jeder Knoten von $T_{M,\,x}$ ist mit einer Konfiguration beschriftet.

2. Die Wurzel ist der einzige Knoten von $T_{M,x}$ mit dem Eingangsgrad 0 und ist mit der Startkonfiguration $q_0 ¢ x$ beschriftet.
3. Jeder Knoten des Baumes, der mit einer Konfiguration C beschriftet ist, hat genau so viele Kinder wie C Nachfolgekonfigurationen hat, und diese Kinder sind mit diesen Nachfolgekonfigurationen von C markiert.

Satz: Sei M eine NTM. Dann existiert eine TM A, so dass

1. $L(M) = L(A)$ und
2. falls M keine unendlichen Berechnungen auf Wörtern aus $(L(M))^C$ hat, dann hält A immer.

4.6 Kodierung von Turingmaschinen

Sei M eine TM wobei $Q = \{q_0,\ q_1, ...,\ q_m,\ q_{accept},\ q_{reject}\}$ und $\Gamma = \{A_1,\ A_2, ...,\ A_r\}$. Wir definieren zuerst die ***Kodierung*** der einzelnen Symbole wie folgt:

- $\text{Code}(q_i) = 10^{i+1}1$ für $i = 0,\ 1, ...,\ m$
- $\text{Code}(q_{accept}) = 10^{m+2}1$
- $\text{Code}(q_{reject}) = 10^{m+3}1$
- $\text{Code}(A_j) = 110^{j}11$ für $j = 0,\ 1, ...,\ r$
- $\text{Code}(N) = 1110111$
- $\text{Code}(R) = 1110^{2}111$
- $\text{Code}(L) = 1110^{3}111$

Um eine Kodierung über Σ_{bool} zu erhalten, benutzen wir folgenden Homomorphismus

$$h : \{0,\ 1,\ \#\}^*$$

Definition: Für jede Turingmaschine M wird

$$\text{Kod}(M) = h(\text{Code}(M))$$

wir die ***Kodierung der*** TM M genannt.

$$\text{KodTM} = \{\text{Kod}(M) \mid M \text{ ist eine } TM\}$$

bezeichnet die Menge der Kodierungen aller Turingmaschinen.

Definition: Sei $x \in (\Sigma_{bool})^*$. Für jedes $i \in \mathbb{N} \setminus \{0\}$ sagen wir, dass x die ***Kodierung der i-ten TM*** ist, falls

1. $x = \text{Kod}(M)$ für eine TM M und
2. die Menge $\{y \in (\Sigma_{bool})^* \mid y \text{ ist vor } x \text{ in der kanonischen Ordnung}\}$ enthält genau $i-1$ Wörter, die Kodierungen von Turingmaschinen sind.

Falls $x = \text{Kod}(M)$ die Kodierung der i-ten TM ist, dann ist M die i-te Turingmaschine M_i. Die Zahl i ist die ***Ordnung der*** TM M_i.

---

5. Berechenbarkeit

5.2 Die Methode der Diagonalisierung

Definition: 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). Wir sagen, dass

$$|A| < |B|,$$

falls $|A| \leq |B|$ und keine injektive Abbildung von B nach A existiert.

Definition: Eine Menge A heisst ***abzählbar,*** falls A endlich ist oder $|A| = |\mathbb{N}|$.

Lemma: Sei Σ ein beliebiges Alphabet. Dann ist Σ^* *abzählbar*.

Satz: Die Menge KodTM der Turingmaschinenkodierungen ist abzählbar.

Lemma: $(\mathbb{N} - \{0\}) \times (\mathbb{N} - \{0\})$ ist abzählbar.

Die oben genannte Menge kann durch die Darstellung einer Matrix nummeriert werden (Diagonale für Diagonale). Dann ist die Nummerierung geben durch:

$$f((a,b)) = \binom{a+b-1}{2} + b.$$

Satz: $\mathbb{Q}^+$ ist abzählbar. $[0,1]$ ist nicht abzählbar. $\mathcal{P}((\Sigma_{bool})^*)$ ist nicht abzählbar.

Definition: Wir definieren die ***Diagonalsprache*** L_{diag}, die keiner der Sprachen $L(M_i)$ enstpricht, wie folgt:

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

Es gilt $L_{diag} \notin \mathcal{L}_{RE}$.

5.3 Die Methode der Reduktion

Definition: 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

$$L_2 \in \mathcal{L}_R \Rightarrow L_1 \in \mathcal{L}_R.$$

Inuitiv bedeuted dies, dass L_2 bezüglich der algorithmischen Lösbarkeit mindestens genau so schwer wie L_1 ist.

Definition: 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

$$x \in L_1 \iff f_M(x) \in L_2$$

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: Seien L_1 und L_2 zwei Sprachen. Falls $L_1 \leq_{EE} L_2$, dann auch $L_1 \leq_R L_2$.

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

$$L \leq_R L^C \text{ und } L^C \leq_R L.$$

Desweiteren gilt:

- $(L_{diag})^C \notin \mathcal{L}_R$
- $(L_{diag})^C \in \mathcal{L}_{RE}$
- $(L_{diag})^C \in \mathcal{L}_{RE} - \mathcal{L}_R$ und daher $\mathcal{L}_R \subsetneq \mathcal{L}_{RE}$

Definition: Die ***universelle Sprache*** ist die Sprache

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

Satz: Es gibt eine $TM\ U$, ***universelle*** TM genannt, so dass

$$L(U) = L_U.$$

Daher gilt auch $L_U \in \mathcal{L}_{RE}$. Desweiteren gilt, dass $L_U \notin \mathcal{L}_R$.

Definition: Das ***Halteproblem*** ist das Entscheidungsproblem $(\{0,\ 1,\ \#\},\ L_H)$, wobei

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

Es gilt, dass $L_H \notin \mathcal{L}_R$.

Definition: Wir betrachten nun folgende Sprache:

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

Für diese gilt:

- $(L_{empty})^C \in \mathcal{L}_{RE}$
- $(L_{empty})^C \notin \mathcal{L}_R$

- $L_{empty} \notin \mathcal{L}_R$

Korollar: Die Sprache $L_{EQ} := \{\text{Kod}(M)\#\text{Kod}(\overline{M}) \mid L(M) = L(\overline{M})\}$ ist nicht entscheidbar, das heisst $L_{EQ} \notin \mathcal{L}_R$.

5.4 Der Satz von Rice

Definition: Eine Sprache $L \subseteq \text{KodTM}$ heisst ***semantisch nichttriviales Entscheidungsproblem über Turingmaschinen***, falls folgende Bedingungen erfüllt sind:

1. Es gibt eine TM M_1, so dass $\text{Kod}(M_1) \in L$ (daher $L \neq \emptyset$)
2. Es gibt eine Tm M_2, so dass $\text{Kod}(M_2) \notin L$ (daher enthält L nicht die Kodierungen aller Turingmaschinen)
3. Für zwei Turingmaschinen A und B impliziert $L(A) = L(B)$

$$\text{Kod}(A) \in L \iff \text{Kod}(B) \in L.$$

Wir betrachten zudem kurz folgende Sprache als ein spezifisches Halteproblem:

$$L_{H,\lambda} := \{\text{Kod}(M) \mid M \text{ hält auf } \lambda\}$$

Es gilt, dass $L_{H,\lambda} \notin \mathcal{L}_R$.

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

Der Satz von Rice hat folgende Konsequenz. Sei L eine beliebige rekursive Sprache und sei

$$\text{Kod}_L := \{\text{Kod}(M) \mid M \text{ ist eine } TM \text{ und } L(M) = L\}$$

die Sprache der Kodierungen aller Turingmaschinen, die die Sprache L akzeptieren. Es gilt, dass Kod_L ein semantisch nichttriviales Entscheidungsproblem über Turingmaschinen ist, und daher gilt nach dem Satz von Rice, dass $\text{Kod}_L \notin \mathcal{L}_R$.

5.6 Die Methode der Kolmogorv-Komplexität

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

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

6. Komplexitätstheorie

6.2 Komplexitätsmasse

Wir definieren zwei grundlegende Komplexitätsmassen: die Zeitkomplexität und die Speicherkomplexität. Die Zeitkomplexität einer Berechnung entspricht der Anzahl elementarer Operationen, die in dieser Berechnung ausgeführt werden. Damit steht sie in linearer Beziehung zu der Energie, die die Ausführung der Berechnung auf einem Rechner kosten würde. Die Speicherkomplexität ist die Grösse des benutzen Speichers, ausgedrückt in der Anzahl der gespeicherten Rechnerwörter.

Definition: Sei M eine Mehrband-Turingmaschine oder TM, die immer hält. Sei Σ das Eingabealphabet von M. Sei $x \in \Sigma^*$ und sei $D = C_1, C_2, ..., C_k$ die Berechnung von M auf x. Dann ist die ***Zeitkomplexität*** $\mathrm{Time}_M(x)$ ***der Berechnung von*** M ***auf*** x definiert durch

$$\mathbf{Time}_M(x) = k - 1,$$

also durch die Anzahl der Berechnungsschritte in D.

Die ***Zeitkomplexität von*** M ist die Funktion $\mathrm{Time}_M : \mathbb{N} \to \mathbb{N}$, definiert durch

$$\mathbf{Time}_M(n) = \max\{\mathrm{Time}_M(x) \mid x \in \Sigma^n\}.$$

Definition: Sei $k \in \mathbb{N} - \{0\}$. Sei M eine k-Band-Turingmaschine, die immer hält. Sei

$$C = (q,\, x,\, i,\, \alpha_1,\, i_1,\, \alpha_2,\, i_2, ...,\, \alpha_k,\, i_k) \text{ mit } 0 \leq i \leq |x| + 1 \text{ und } 0 \leq i_j \leq |\alpha_j| \text{ für } j = 1, ..., k$$

eine Konfiguration von M. Die ***Speicherkomplexität von*** C ist

$$\mathbf{Space}_M(C) = \max\{|\alpha_i| \mid i = 1, ..., l\}.$$

Die ***Speicherkomplexität von*** M ist die Funktion $\mathrm{Space}_M : \mathbb{N} \to \mathbb{N}$, definiert durch

$$\mathbf{Space}_M(n) = \max\{\mathrm{Space}_M(C_i) \mid i = 1, ..., l\}.$$

Lemma: Sei k eine positive ganze Zahl. Für jede k-Band-TM A, die immer hält, existiert eine äquivalente 1-Band-TM B, so dass

$$\mathrm{Space}_B(n) \leq \mathrm{Space}_A(n).$$

Lemma: Sei k eine positive ganze Zahl. Für jede k-Band-TM A existiert eine k-Band-TM B, so dass $L(A) = L(B)$ und

$$\mathrm{Space}_B(n) \leq \frac{\mathrm{Space}_A(n)}{2} + 2.$$

Definition: Für jede Funktion $f : \mathbb{N} \to \mathbb{R}^+$ definieren wir

$$O(f(n)) := \{r : \mathbb{N} \to \mathbb{R}^+ \mid \exists n_0 \in \mathbb{N},\, \exists c \in \mathbb{N},\, \text{so dass für alle } n \geq n_0 : r(n) \leq c \cdot f(n)\}.$$

Für jede Funktion $r \in O(f(n))$ sagen wir, dass r ***asymptotisch nicht schneller wächst als*** f.

Für jede Funktion $g : \mathbb{N} \to \mathbb{R}^+$ definiren wir

$$\Omega(g(n)) := \{s : \mathbb{N} \to \mathbb{R}^+ \mid \exists n_0 \in \mathbb{N},\, \exists d \in \mathbb{N},\, \text{so dass für alle } n \geq n_0 : s(n) \geq \frac{1}{d} \cdot g(n)\}.$$

Für jede Funktion $s \in \Omega(g(n))$ sagen wir, dass ***s asymptotisch mindestens so schnell wächst wie*** g.

Für jede Funktion $h : \mathbb{N} \to \mathbb{R}^+$ definieren wir

$$\Theta(h(n)) := \{q : \mathbb{N} \to \mathbb{R}^+ \mid \exists c,\, d,\, n_0 \in \mathbb{N},\, \text{so dass für alle } n \geq n_0 : \frac{1}{d} \cdot h(n) \leq q(n) \leq c \cdot h(n)\} = O(h(n)) \cap \Omega(h(n)).$$

Falls $g \in \Theta(h(n))$ sagen wir, dass g und h ***asymptotisch gleich schnell wachsen.***

Seien f und g zwei Funktionen von $\mathbb{N}$ nach $\mathbb{R}^+$. Falls

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0,$$

dann sagen wir, dass g ***asymptotisch schneller wächst als*** f und wir schreiben $f(n) = o(g(n))$.

Satz: Es existiert ein Entscheidungsproblem $(\Sigma_{bool},\, L)$, so dass für jede $MTM\ A$, die $(\Sigma_{bool},\, L)$ entscheidet, eine $MTM\ B$ existiert, die auch $(\Sigma_{bool},\, L)$ entscheidet, und für die gilt

$$\text{Time}_B(n) \leq \log_2(\text{Time}_A(n))$$

für unendlich viele $n \in \mathbb{N}$.

Definition: Sei L eine Sprache. Seien f und g zwei Funktionen von $\mathbb{N}$ nach $\mathbb{R}^+$. Wir sagen, dass $O(g(n))$ eine ***obere Schranke für die Zeitkomplexität von*** L ist, falls eine $MTM\ A$ existiert, so dass A die Sprache L entscheidet und $\text{Time}_A(n) \in O(g(n))$.

Wir sagen, dass $\Omega(f(n))$ eine ***untere Schranke für die Zeitkomplexität von*** L ist, falls für jede $MTM\ B$, die L entscheidet, $\text{Time}_B(n) \in \Omega(f(n))$.

Eine $MTM\ C$ heisst ***optimal für*** L, falls $\text{Time}_C(n) \in O(f(n))$ gilt und $\Omega(f(n))$ eine untere Schranke für die Zeitkomplexität von L ist.

6.3 Komplexitätsklassen und die Klasse P

Definition: Für alle Funtkionen $f,\ g$ von $\mathbb{N}$ nach $\mathbb{R}^+$ definieren wir:

- $\mathbf{TIME}(f) = \{L(B) \mid B \text{ ist eine } MTM \text{ mit } \mathrm{Time}_B(n) \in O(f(n))\}$
- $\mathbf{SPACE}(g) = \{L(A) \mid A \text{ ist eine } MTM \text{ mit } \mathrm{Space}_A(n) \in O(g(n))\}$
- $\mathbf{DLOG} = \mathrm{SPACE}(\log_2(n))$
- $\mathbf{P} = \bigcup_{c\in\mathbb{N}} \mathrm{TIME}(n^c)$
- $\mathbf{PSPACE} = \bigcup_{c\in\mathbb{N}} \mathrm{SPACE}(n^c)$
- $\mathbf{EXPTIME} = \bigcup_{d\in\mathbb{N}} \mathrm{TIME}(2^{n^d})$

Lemma: Für jede Funktion $t : \mathbb{N} \to \mathbb{R}^+$ gilt

$$\mathrm{TIME}(t(n)) \subseteq \mathrm{SPACE}(t(n)).$$

Korollar: $\mathrm{P} \subseteq \mathrm{PSAPCE}.$

Definition: Eine Funktion $s : \mathbb{N} \to \mathbb{N}$ heisst ***platzkonstruierbar,*** falls eine 1 -Band-TM M existiert, so dass

1. $\mathrm{Space}_M(n) \leq s(n)$ für alle $n \in \mathbb{N}$ und
2. für jede Eingabe 0^n, $n \in \mathbb{N}$, generiert M das Wort $0^{s(n)}$ auf ihrem Arbeitsband und hält in q_{accept}.

Eine Funktion $t : \mathbb{N} \to \mathbb{N}$ heisst ***zeitkonstruierbar,*** falls eine MTM A existiert, so dass

1. $\mathrm{Time}_A(n) \in O(t(n))$ und
2. für jede Eingabe $0^n,\ n \in \mathbb{N}$, generiert A das Wort $0^{t(n)}$ auf dem ersten Arbeitsband und hält in q_{accept}.

Lemma: Sei $s : \mathbb{N} \to \mathbb{N}$ eine platzkonstruierbare Funktion. Sei M eine MTM mit $\mathrm{Space}_M(x) \leq s(|x|)$ für alle $x \in L(M)$. Dann existiert eine MTM A mit $L(A) = L(M)$ und

$$\mathrm{Space}_A(n) \leq s(n),$$

d.h., es gilt $\mathrm{Space}_A(y) \leq s(|y|)$ für alle y über dem Eingabealphabet von M.

Lemma: Sei $t : \mathbb{N} \to \mathbb{N}$ eine zeitkonstruierbare Funktion. Sei M eine MTM mit $\text{Time}_M(x) \leq t(|x|)$ für alle $x \in L(M)$. Dann existiert eine MTM A mit $L(A) = L(M)$ und

$$\text{Time}_A(n) \in O(t(n)).$$

Die obigen Lemmata zeigen, dass es für die Definition der Komplexitätsklassen $\text{SPACE}(s)$ und $\text{TIME}(t)$ für eine platzkonstruierbare Funktion s und eine zeitkonstruierbare Funktion t unwesentlich ist, ob mann Space_M und Time_M einer TM M als

- $\text{Space}_M(n) = \max\{\text{Space}_M(x) \mid x \in \Sigma^n\}$ und
- $\text{Time}_M(n) = \max\{\text{Time}_M(x) \mid x \in \Sigma^n\}$

oder als

- $\text{Space}_M(n) = \max(\{\text{Space}_M(x) \mid x \in L(M) \text{ und } |x| = n\} \cup \{0\})$ und
- $\text{Time}_M(n) = \max(\{\text{Time}_M(x) \mid x \in L(M) \text{ und } |x| = n\} \cup \{0\})$

definiert.

Satz: Für jede Funktion s mit $s(n) \geq \log_2(n)$ gilt

$$\text{SPACE}(s(n)) \subseteq \bigcup_{c \in \mathbb{N}} \text{TIME}(c^{s(n)}).$$

Korollar: $\text{DLOG} \subseteq P$ und $\text{PSPACE} \subseteq \text{EXPTIME}$.

Dadurch können wir die folgende ***fundamentale Hierarchie deterministischer Komplexitätsklassen*** geben:

$$\text{DLOG} \subseteq P \subseteq \text{PSPACE} \subseteq \text{EXPTIME}$$

Satz: Seien s_1 und s_2 zwei Funktionen von $\mathbb{N}$ nach $\mathbb{N}$ mitt folgenden Eigenschaften:

1. $s_2(n) \geq \log_2(n)$
2. s_2 ist platzkonstruierbar
3. $s_1(n) = o(s_2(n))$

Dann gilt $\text{SPACE}(s_1) \subsetneq \text{SPACE}(s_2)$.

Satz: Seien t_1 und t_2 zwei Funktionen von $\mathbb{N}$ nach $\mathbb{N}$ mit folgenden Eigenschaften:

1. t_2 ist zeitkonstruierbar
2. $t_1(n) \cdot \log_2(t_1(n)) = o(t_2(n))$

Dann gilt $\mathrm{TIME}(t_1) \neq \mathrm{TIME}(t_2)$.

6.4 Nichtdeterministische Komplexitätsmasse

Definition: Sei M eine NTM oder eine nichtdeterministische MTM. Sei $x \in L(M) \subseteq \Sigma^*$. Die ***Zeitkomplexität von M auf x,*** $\mathrm{Time}_M(x)$, ist die Länge einer kürzesten akzeptierten Berechnung von M auf x. Die ***Zeitkomplexität von*** M ist die Funktion $\mathrm{Time}_M : \mathbb{N} \to \mathbb{N}$, definiert durch

$$\mathbf{Time}_M(n) = \max(\{\mathrm{Time}_M(x) \mid x \in L(M) \text{ und } |x| = n\} \cup \{0\}).$$

Sei $C = C_1,\ C_2, ...,\ C_m$ eine akzeptierende Berechnung von M auf x. Sei $\mathrm{Space}_M(C_i)$ die Speicherkomplexität der Konfiguration C_i. Wir definieren

$$\mathbf{Space}_M(C) = \max\{\mathrm{Space}_M(C_i) \mid i = 1,\ 2, ...,\ m\}.$$

Die ***Speicherplatzkomplexität von M auf x*** ist

$$\mathbf{Space}_M(x) = \min\{\mathrm{Space}_M(C) \mid C \text{ ist eine akzeptierende Berechnung von } M \text{ a}$$

Die ***Speicherkomplexität von*** M ist die Funktion $\mathrm{Space}_M : \mathbb{N} \to \mathbb{N}$ definiert durch

$$\mathbf{Space}_M(n) = \max(\{\mathrm{Space}_M(x) \mid x \in L(M) \text{ und } |x| = n\} \cup \{0\}).$$

Definition: Für alle Funktionen $f, g : \mathbb{N} \to \mathbb{R}^+$ definieren wir

- $\mathrm{NTIME}(f) = \{L(M) \mid M \text{ ist eine nichtdeterministische } MTM \text{ mit } \mathrm{Time}_M(n) \in O(f(n))\}$

- $\mathrm{NSPACE}(g) = \{L(M) \mid M \text{ ist eine nichtdeterministische } MTM \text{ mit } \mathrm{Space}_M(n) \in O(g(n))\}$
- $\mathrm{NLOG} = \mathrm{NSPACE}(\log_2(n))$
- $\mathrm{NP} = \bigcup_{c \in \mathbb{N}} \mathrm{NTIME}(n^c)$
- $\mathrm{NPSPACE} = \bigcup_{c \in \mathbb{N}} \mathrm{NSPACE}(n^c)$.

Lemma: Für alle Funktionen t und s mit $s(n) \geq \log_2(n)$ gilt

1. $\mathrm{NTIME}(t) \subseteq \mathrm{NSPACE}(t)$
2. $\mathrm{NSPACE}(s) \subseteq \bigcup_{c \in \mathbb{N}} \mathrm{NTIME}(c^{s(n)})$.

Satz: Für jede Funktion $t : \mathbb{N} \to \mathbb{R}^+$ und jede platzkonstruierbare Funktion $s : \mathbb{N} \to \mathbb{N}$ mit $s(n) \geq \log_2(n)$ gilt

1. $\mathrm{TIME}(t) \subseteq \mathrm{NTIME}(t)$
2. $\mathrm{SPACE}(t) \subseteq \mathrm{NSPACE}(t)$
3. $\mathrm{NTIME}(s(n)) \subseteq \mathrm{SPACE}(s(n)) \subseteq \bigcup_{c \in \mathbb{N}} \mathrm{TIME}(c^{s(n)})$

Korollar: $\mathrm{NP} \subseteq \mathrm{PSPACE}$

Satz: Für jede platzkonstruierbare Funktion s, $s(n) \geq \log_2(n)$, gilt

$$\mathrm{NSPACE}(s(n)) \subseteq \bigcup_{c \in \mathbb{N}} \mathrm{TIME}(c^{s(n)}).$$

Satz von Savitch: Sei s mit $s(n) \geq \log_2(n)$ eine platzkonstruierbare Funktion. Dann gilt

$$\mathrm{NSAPCE}(s(n)) \subseteq \mathrm{SPACE}(s(n)^2).$$

Die Zusammenfassung der vorgestellten Resultate führt zu der sogenannten ***fundamentalen Komplexitätsklassenhierarchie der sequentiellen Berechnungen:***

> $\mathrm{DLOG} \subseteq \mathrm{NLOG} \subseteq \mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} \subseteq \mathrm{EXPTIME}$

6.5 Die Klasse NP und die Beweisverifikation

In der fundamentalen Komplexitätsklassenhierarchie konzentriert sich das Interesse auf die Relation zwischen P und NP. Das Problem, ob $\mathrm{P} = \mathrm{NP}$ oder $\mathrm{P} \subsetneq \mathrm{NP}$ gilt, ist das wohl bekannteste offene Problem der Informatik, und heutzutage zählt es auch zu den wichtigsten offenen Problemen der Mathematik.

Definition: Wir definieren die Sprache SAT wie folgt:

$$\mathrm{SAT} := \{x \in (\Sigma_{logic})^* \mid x \text{ kodiert eine erfüllbare Formel in KNF}\}.$$

Definition: Sei $L \subseteq \Sigma^*$ eine Sprache und sei $p : \mathbb{N} \to \mathbb{N}$ eine Funktion. Wir sagen, dass eine MTM (ein Algorithmus) A ein ***p-Verifizierer für*** L ist, $V(A) = L$, falls A mit folgenden Eigenschaften auf allen Eingaben aus $\Sigma^* \times (\Sigma_{bool})^*$ arbeitet:

1. $\mathrm{Time}_A(w,\ x) \leq p(|w|)$ für jede Eingabe $(w,\ x) \in \Sigma^* \times (\Sigma_{bool})^*$.
2. Für jedes $w \in L$ existiert ein $x \in (\Sigma_{bool})^*$, so dass $|x| \leq p(|w|)$ und $(w,\ x) \in L(A)$ (d.h. A akzeptiert $(w,\ x)$). Das Wort x nennt man einen ***Beweis*** oder einen ***Zeugen*** der Behauptung $w \in L$.
3. Für jedes $y \notin L$ gilt $(y,\ z) \notin L(A)$ für alle $z \in (\Sigma_{bool})^*$.

Falls $p(n) \in O(n^k)$ für ein $k \in \mathbb{N}$, so sagen wir, dass A ein ***Polynomialzeit-Verifizierer*** ist. Wir definieren die ***Klasse der in Polynomialzeit verifizierbaren Sprachen*** als

$$\mathbf{VP} := \{V(A) \mid A \text{ ist ein Polynomialzeit-Verifizierer}\}.$$

Satz: $\mathrm{VP} = \mathrm{NP}$.

6.6 NP-Vollständigkeit

Definition: Seien $L_1 \subseteq \Sigma_1^*$ und $L_2 \subseteq \Sigma_2^*$ zwei Sprachen. Wir sagen, dass L_1 ***polynomiell auf*** L_2 ***reduzierbar ist,*** $L_1 \leq_p L_2$, falls eine polynomielle TM (ein polynomieller Algorithmus) A existiert, die für jedes Wort $x \in \Sigma_1^*$ ein Wort $A(x) \in \Sigma_2^*$ berechnet, so dass

$$x \in L_1 \iff A(x) \in L_2.$$

A wird eine ***polynomielle Reduktion*** von L_1 auf L_2 genannt.

Definition: Eine Sprache L ist $\mathbf{NP}$**-schwer,** falls für alle Sprachen $L' \in \mathrm{NP}$ gilt $L' \leq_p L$. Eine Sprache ist $\mathbf{NP}$**-vollständig,** falls

1. $L \in \mathrm{NP}$ und
2. L ist NP-schwer.

Lemma: Falls $L \in \mathrm{P}$ und L ist NP-schwer, dann gilt $\mathrm{P} = \mathrm{NP}$.

Satz von Cook: SAT ist NP-vollständig.

Lemma: Seien L_1 und L_2 zwei Sprachen. Falls $L_1 \leq_p L_2$ und L_1 ist NP -schwer, dann ist auch L_2 NP-schwer.

Für den Rest dieses Kapitels definieren wir folgende Sprachen:

- $\mathrm{SAT} := \{\phi \mid \phi \text{ ist eine erfüllbare Formel in KNF}\}$
- $\mathrm{CLIQUE} :=$
  $\{(G,\, k) \mid G \text{ ist ein ungerichteter Graph, der eine } k\text{-Clique enthält}\}$
- $\mathrm{VC} :=$
  $\{(G,\, k) \mid G \text{ ist ein ungerichteter Graph mit einer Knotenüberdeckung der Mächtigkeit}$

Lemma: Wir stellen die folgenden Ungleichungen:

- $\mathrm{SAT} \leq_p \mathrm{CLIQUE}$
- $\mathrm{CLIQUE} \leq_p \mathrm{VC}$
- $\mathrm{SAT} \leq_p \mathrm{3SAT}$

Definition: $\mathbf{NPO}$ ist die Klasse der Optimierungsprobleme, wobei

$$U = (\Sigma_I,\, \Sigma_O,\, L,\, \mathcal{M},\, \mathrm{cost},\, \mathrm{goal}) \in \mathbf{NPO},$$

falls folgende Bedingungen erfüllt sind:

1. $L \in \mathrm{P}$,

2. es existiert ein Polynom p_U, so dass
    1. für jedes $x \in L$ und jedes $y \in \mathcal{M}(x)$, $|y| \leq p_U(|x|)$,
    2. es existiert ein polynomieller Algorithmus A, der für jedes $y \in \Sigma_O^*$ und jedes $x \in L$ mit $|y| \leq p_U(|x|)$ entscheidet, ob $y \in \mathcal{M}(x)$ oder nicht,
3. die Funktion cost kann man in polynomieller Zeit berechnen.

Definition: $\mathbf{PO}$ ist die Klass der Optimierungsprobleme $U = (\Sigma_I, \Sigma_O, L, \mathcal{M}, \mathrm{cost}, \mathrm{goal})$, so dass

1. $U \in \mathrm{NPO}$ und
2. es existiert ein polynomieller Algorithmus A, so dass $A(x)$ für jedes $x \in L$ eine optimale Lösung für x ist.

Definition: Sei $U = (\Sigma_I, \Sigma_O, L, \mathcal{M}, \mathrm{cost}, \mathrm{goal})$ ein Optimierungsproblem aus NPO. Die ***Schwellenwert-Sprache für*** U ist

$$\mathbf{Lang}_U := \{(x, a) \in L \times (\Sigma_{bool})^* \mid Opt_U(x) \leq \mathrm{Nummer}(a)\},$$

falls goal = Minimum, und

$$\mathbf{Lang}_U := \{(x, a) \in L \times (\Sigma_{bool})^* \mid Opt_U(x) \geq \mathrm{Nummer}(a)\},$$

falls goal = Maximum. Wir sagen, dass U **$\mathbf{NP}$-schwer** ist, falls Lang_u NP-schwer ist.

Lemma: Wir stellen folgende Sätze und Lemmas:

- Falls ein Optimierungsproblem $U \in \mathrm{PO}$, dann $\mathrm{Lang}_U \in \mathrm{P}$.
- Sei $U \in \mathrm{NPO}$. Falls U NP-schwer ist und $\mathrm{P} \neq \mathrm{NP}$, dann $U \notin \mathrm{PO}$.
- $\mathrm{MAX\text{-}SAT}$ ist NP-schwer.
- $\mathrm{MAX\text{-}CL}$ ist NP-schwer.

10. Grammatiken und Chomsky-Hierarchie

10.2 Das Konzept der Grammatiken

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

1. Σ_N ist ein Alphabet, genannt *Nichtterminalalphabet* oder die Menge der Nichtterminale von G. Die Symbole aus Σ_N nennt man ***Nichtterminale.***
2. Σ_T ist ein Alphabet, genannt *Terminalalphabet* oder die Menger der Terminalsymbole von G. Die Symbole aus Σ_T nennt man ***Terminalsymbole.*** Es gilt

$$\Sigma_N \cap \Sigma_T = \emptyset.$$

3. $S \in \Sigma_N$ heisst ***Startsymbol*** oder Startnichtterminal.
4. P ist eine endliche Teilmenge von $\Sigma^*\Sigma_N\Sigma^* \times \Sigma^*$ für $\Sigma = \Sigma_N \cup \Sigma_T$, genannt die ***Menge der Ableitungsregeln von*** G. Die Elemente von P heissen *Regeln* oder auch Produktionen. Statt $(\alpha,\ \beta) \in P$ schreiben wir auch

$$\alpha \to_G \beta,$$

   in Worten: α kann in G durch β ersetzt werden.

Seien $\gamma, \delta \in (\Sigma_N \cup \Sigma_T)^*$. Wir sagen, dass δ aus γ in einem Ableitungsschritt in G ableitbar ist (oder dass γ in G zu δ übergeht),

$$\gamma \Rightarrow_G \delta,$$

genau dann, wenn ω_1 und ω_2 aus $(\Sigma_N \cup \Sigma_T)^*$ und eine Regel $(\alpha,\ \beta) \in P$ existieren, so dass

$$\gamma = \omega_1\alpha\omega_2 \text{ und } \delta = \omega_1\beta\omega_2.$$

Wir sagen, dass δ aus γ in G ableitbar ist,

$$\gamma \Rightarrow_G^* \delta,$$

genau dann, wenn

1. entweder $\gamma = \delta$,
2. oder ein $n \in \mathbb{N} - \{0\}$ und $n+1$ Wörter $\omega_0, \omega_1, ..., \omega_n \in (\Sigma_N \cup \Sigma_T)^*$ existieren, so dass

$$\gamma = \omega_0,\ \delta = \omega_n \text{ und } \omega_i \Rightarrow_G \omega_{i+1} \text{ für } i = 0,\ 1,\ 2, ...,\ n - 1.$$

Eine Folge von Ableitungsschritten

$$\omega_0 \Rightarrow_G \omega_1 \Rightarrow_G \omega_2 \Rightarrow_G ... \Rightarrow_G \omega_n$$

heisst eine ***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

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

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

1. G heisst *Typ-0-Grammatik*.
2. G heisst ***kontextsensitiv*** oder *Typ-1-Grammatik*, falls für alle Paare $(\alpha,\ \beta) \in P$ gilt:

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

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

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

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

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

Für $i = 0,\ 1,\ 2,\ 3$ ist eine Sprache L vom Typ i genau dann, wenn sie von einer Grammatik G vom Typ i erzeugt wird. Die *Familie aller Sprachen vom Typ i* wird mit $\mathcal{L}_i$ bezeichnet.

10.3 Reguläre Grammatiken und endliche Automaten

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

Wir sagen, dass eine Sprachklasse $\mathcal{L}$ *abgeschlossen bezüglich einer binären Operation* $\bigcirc$ über Sprachen ist, falls für alle $L_1, L_2 \in \mathcal{L}$ gilt:

$$L_1 \bigcirc L_2 \in \mathcal{L}.$$

Wir sagen, dass eine Sprachklasse $\mathcal{L}$ *abgeschlossen bezüglich einer unären Operation* $\triangle$ ist, falls für jede Sprache $L \in \mathcal{L}$ gilt:

$$\triangle(L) \in \mathcal{L}.$$

Lemma: $\mathcal{L}_3$ ist abgeschlossen bezüglich Vereinigung und Konkatenation.

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

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

1. $S \to \lambda$, wobei S das Startsymbol ist,
2. $A \to a$ für $A \in \Sigma_N$ und $a \in \Sigma_T$, und
3. $B \to bC$ für $B, C \in \Sigma_N$ und $b \in \Sigma_T$.

Lemma: Für jede reguläre Grammatik G existiert eine äquivalente reguläre Grammatik G', so dass G' keine Regel der Form $X \to Y$ für zwei Nichtterminale X und Y enthält.

Lemma: Für eine reguläre Grammatik G existiert eine äquivalente Grammatik G', die keine Regel $A \to \lambda$ für ein Nichtterminal A, welches nicht gleich dem Startsymbol ist, enthält.

Satz: Zu jeder regulären Grammatik existiert eine äquivalente normierte reguläre Grammatik.

Satz: $\mathcal{L}_3 = \mathcal{L}_{EA}$.

10.4 Kontextfreie Grammatiken und Kellerautomaten

Definition: Sei $G = (\Sigma_N,\ \Sigma_T,\ P,\ S)$ eine kontextfreie Grammatik. Ein ***Syntaxbaum (Ableitungsbaum)*** T für G ist ein markierter, geordneter Baum mit folgenden Eigenschaften:

1. T hat eine Wurzel, die mit S markiert ist.
2. Jeder Knoten von T ist mit einem Symbol aus $\Sigma_N \cup \Sigma_T \cup \{\lambda\}$ markiert.
3. Innere Knotetn sind mit Symbolen aus Σ_N markiert.
4. Alle Blätter sind mit Symbolen aus $\Sigma_T \cup \{\lambda\}$ markiert. Hierbei kann ein Blatt nur dann mit λ markiert sein, wenn es das einzige Kind seines Vorgängerknotens ist.
5. Wenn ein innerer knoten, der mit $A \in \Sigma_N$ markiert ist, genau k Kinder besitzt, die von links nach rechts mit $\alpha_1,\ \alpha_2,\ \alpha_3, \ldots,\ \alpha_k$ markiert sind, dann existiert eine Regel $A \to \alpha_1\alpha_2 \ldots \alpha_k$, mit $\alpha_i \in \Sigma_N \cup \Sigma_T \cup \{\lambda\}$ in P.

Die Markierungen der Blätter von T von links nach rechts gelesen ergeben ein Wort $x = x_T \in L(G)$. Wir sagen auch, dass T ein Syntaxbaum zur Generierung von x in G ist.

Definition: Sei $G = (\Sigma_N,\ \Sigma_T,\ P,\ S)$ eine kontextfreie Grammatik mit $\lambda \notin L(G)$. Wir sagen, dass G in ***Chromsky-Normalform*** ist, falls alle Regeln von der Form

- $A \to BC$ für $A, B, C \in \Sigma_N$, oder
- $A \to a$ für $A \in \Sigma_N$ und $a \in \Sigma_T$

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

- $A \to a\alpha$ für $A \in \Sigma_N,\ a \in \Sigma_T$ und $\alpha \in \Sigma_N^*$

sind.

Satz: Für jede kontextfreie Grammatik G mit $\lambda \notin L(G)$ existieren zu G äquivalente Grammatiken in Chromsky-Normalform und Greibach-Normalform.

Pumping-Lemma für kontektfreie Sprachen: Sei L kontextfrei. Dann existiert eine nur von L abhängige Konstante n_L, so dass für alle Wörter $z \in L$ mit $|z| \geq n_L$ eine Zerlegung

$$z = uvwxy$$

von z existiert, so dass

1. $|vx| \geq 1$,
2. $|vwx| \leq n_L$ und
3. $\{uv^iwx^iy \mid i \in \mathbb{N}\} \subseteq L$.

Lemma von Ogden: Für jede kontextfreie Sprache L gibt es eine nur von L abhängige Konstante n_L, so dass für alle Wörter $z \in L$ mit $|z| \geq n_L$ und alle Markierungen von mindestens n_L Buchstaben in z eine Zerlegung

$$z = uvwxy$$

von z existiert, so dass

1. vx mindestens einen markierten Buchstaben enthält,
2. vwx höchstens n_L markierte Buchstaben enthält und
3. $\{uv^iwx^iy \mid i \in \mathbb{N}\} \subseteq L$.

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

- Q ist eine endliche Menge, die *Zustandsmenge* genannt wird,
- Σ ist das *Eingabealphabet,*
- Γ ist das *Kelleralphabet,*
- $q_0 \in Q$ ist der *Anfangszustand,*

- $Z_0 \in \Gamma$ ist das *Initialisierungssymbol* des Kellers,
- δ ist eine Abbildung von $Q \times (\Sigma \cup \{\lambda\}) \times \Gamma$ in endliche Teilmengen von $Q \times \Gamma^*$.

Eine ***Konfiguration*** von M ist ein Tripel

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

wobei

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

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

1. $(q,\, aw,\, \alpha X) \vdash_M (p,\, w,\, \alpha\beta)$, falls $(p,\, \beta) \in \delta(q,\, a,\, X)$ für $p, q \in Q,\, a \in \Sigma,\, X \in \Gamma$ und $\alpha, \beta \in \Gamma^*$, und
2. $(q,\, w,\, aX) \vdash_M (p,\, w,\, \alpha\beta)$, falls $(p,\, \beta) \in \delta(q,\, \lambda,\, X)$.

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

$$(q_0,\, x,\, Z_0)$$