1 Alphabete, Wörter und Sprachen

Alphabet Σ:	Eine endliche, nichtleere Menge bestehend aus **Buchstaben**
Wort:	Ein Wort über Σ ist eine endliche Folge von Buchstaben. (Σ^* ist die Menge aller Wörter)
Sprache L:	Eine Menge von Wörtern, also eine Teilmenge von Σ^*

Für ein **Wort** in Σ^* gilt:

Konkatenation:	$\text{Kon}(x, y) = x \cdot y = xy$
Reversal:	$a^R = (a_1 a_2 ... a_n)^R = a_n a_{n-1} ... a_1$
Exponent:	$(abc)^3 = abcabcabc$
Vorkommen:	$\|x = abcabcabc\|_a = 3$ (Anzahl a in x)
kanonische Ordnung:	$u < v \Leftrightarrow (\|u\| < \|v\|) \vee (\|u\| = \|v\| \wedge u = x \cdot s_i \cdot u' \wedge v = x \cdot s_j \cdot v')$ (mit $i < j$)

Für eine **Sprache** gilt:

Konkatenation:	$L_1 \cdot L_2 = L_1 L_2 = \{vw \mid v \in L_1 \wedge w \in L_2\}$
Exponent:	$L^{i+1} = L^i \cdot L$
Kleene'schen Stern:	$L^* = \bigcup_{i \in \mathbb{N}} L^i$
Plus:	$L^+ = LL^*$

1.1 Algorithmische Probleme

1.1.1 Entscheidungsproblem

Für jedes $x \in \Sigma^*$ ist 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:

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

A **erkennt** somit die Sprache. L ist **rekursiv**, falls so ein Algorithmus A existiert.

1.1.2 Funktion (Transformation)

Ein **Algorithmus** A berechnet eine Funktion $f : \Sigma^* \mapsto \Gamma^*$, falls

$$A(x) = f(x) \quad \text{für alle } x \in \Sigma^*$$

Entscheidungsprobleme sind spezielle Fälle von Funktionsberechnungen.

1.1.3 Relationsproblem

Sei $R \subseteq \Sigma^* \times \Gamma^*$ eine Relation, dann löst ein **Algorithmus** A das Relationsproblem R, falls für alle $(x, y) \in R$ gilt:

$$(x, A(x)) \in R$$

Bemerkung: Es ist nur nötig, ein einziges $y = A(x)$ von potenziel unendlich vielen y zu finden.

1.2 Kolmogorov Komplexität

Die Kosten für die binäre Kodierung von w_n der Länge n sind

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

Vorgehen:

1. Schreibe ein Programm, welches das gewünschte Resultat liefert
2. Finde den Term, der von n abhängig ist (log wird mit der Anzahl n im Programm multipliziert)
3. Dieser Term hat die Komplexität $\lceil \log_2(n+1) \rceil + c$
4. Jetzt muss nur noch n in Abhängigkeit von $|w_n|$ ausgedrückt werden

Ein Wort $x \in (\Sigma_{\text{bool}})^*$ heisst **zufällig**, falls gilt:

$$K(x) \geq |x|$$

Analog dazu heisst eine Zahl n **zufällig**, falls gilt:

$$K(n) = K(\text{Bin}(n)) \geq \lceil \log_2(n+1) \rceil - 1$$

2 Endliche Automaten

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

Q**:**	Eine endliche Menge von Zuständen
Σ**:**	Das Eingabealphabet
$q_0 \in Q$**:**	Der Anfangszustand
$F \subseteq Q$**:**	Die Menge der akzeptierenden Zustände (Zustände, die das momentan gelesene Wort als valid erklären)
δ**:**	Die Übergangsfunktion ($\delta : (Q \times \Sigma) \mapsto Q$)

Weiter definiert sind folgende:

Konfiguration:	Ein Element aus $(Q \times \Sigma^*)$. Befindet sich M in der Konfiguration (q, w) , bedeutet das, dass M im Zustand q ist und noch das Suffix w lesen soll.
Startkonfiguration:	Die Berechnung von M auf x muss in der Konfiguration $(q_0, x) \in (\{q_0\} \times \Sigma^*)$ anfangen
Endkonfiguration:	Jedes Element in $(Q \times \{\lambda\})$
Schritt:	Eine Relation auf Konfigurationen definiert als $(q, w) \vdash_M (p, x) \Leftrightarrow w = ax, a \in \Sigma$ und $\delta(q, a) = p$. Sie entspricht einer Anwendung der Übergangsfunktion auf die aktuelle Konfiguration.
Berechnung C**:**	Eine endliche Folge $C = C_0, C_1, C_2, ..., C_n$ von Konfigurationen, so dass $C_i \vdash_M C_{i+1}$. Weiter ist C eine **Berechnung von** M **auf eine Eingabe** $x \in \Sigma^*$, falls C_0 eine Startkonfiguration und C_n eine Endkonfiguration ist. Ist $C_n \in (F \times \{\lambda\})$, so ist C eine **akzeptierende Berechnung** (im Gegensatz zu **verwerfenden Berechnungen**).
Akzeptierte Sprache $L(M)$**:**	$L(M) = \{w \in \Sigma^* \mid$ die Berechnung auf w endet in einer Endkonfiguration (q, λ) mit $q \in F\}$
Klasse der regulären Sprachen $\mathcal{L}_{EA}$**:**	Die Klasse der Sprachen, die von endlichen Automaten akzeptiert werden. Jede Sprache in $\mathcal{L}_{EA}$ wird **regulär** genannt.

Zusätzliche sind folgende Operationen definiert:

$\vdash_M^*$**:**	Die *reflexive* und *transitive* Hülle der Schrittrelation $\vdash_M$. $(q, w) \vdash_M^* (p, u)$ ist, dass es eine Berechnung von M gibt, die ausgehend von der Konfiguration (q, w) zu der Konfiguration (p, u) führt.

$\hat{\delta} : (Q \times \Sigma^*) \mapsto Q$**:** (i) $\hat{\delta}(q, \lambda) = q$ für alle $q \in Q$
(ii) $\hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a)$
Die Aussage $\hat{\delta}(q, w) = p$ bedeutet, dass wenn M im Zustand q das Wort w zu lesen beginnt, dann endet M im Zustand p.

Zwei endliche Automaten A und B heissen **äquivalent**, falls gilt:

$$L(A) = L(B)$$

2.1 Nichtexistenz

Um zu zeigen, dass eine Sprache nicht regulär ist ($L \notin \mathcal{L}_{EA}$), genügt es, zu beweisen, dass es keinen Automaten gibt, der die Sprache akzeptiert.

2.1.1 Lemma 3.3

Sei $A = (Q, \Sigma, \delta_A, q_0, F)$ ein EA. Seien weiter $x, y \in \Sigma^*$ mit $x \neq y$, wobei x und y nicht unbedingt in L sein muss, so dass

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

Das heisst, dass nach dem Lesen der unterschiedlichen Wörter x und y der EA im gleichen Zustand endet. Daraus folgt, dass der EA in Zukunft zwischen den Wörtern nicht unterscheiden kann. Anders ausgedrückt:

$$\hat{\delta}_A(q_0, x) = \hat{\delta}_A(q_0, y) = p \quad (x, y \in \mathrm{Kl}[p])$$

Für jedes Suffix $z \in \Sigma^*$ gilt dann, dass beide Wörter xz und yz entweder akzeptiert oder beide verworfen werden, ansonsten ist die Sprache nicht regulär:

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

Vorgehen: Um zu beweisen, dass eine Sprache nicht regulär ist, zeigt man, dass zwei Wörter zum selben Zustand führen müssen.
Wir wählen m als die Anzahl zustände im EA. Zusätzlich wählen wir $m + 1$ Wörter, die alle vom Automaten akzeptiert werden. Da es $m + 1$ Wörter gibt, aber nur m Zustände, müssen zwei unterschiedliche Wörter zum selben Zustand y_0 führen. Es folgt, dass beide Wörter mit einem z konkateniert zu der Sprache L gehören müssen. Wir wählen also einfach ein z so dass nur eines der beiden Wörter sich in L befindet.

2.1.2 Pumping Lemma für reguläre Sprachen

Sei L regulär, dann existiert eine Konstante $n_0 \in \mathbb{N}$, so dass sich jedes Wort $w \in \Sigma^*$, welches nicht zwingend in L sein muss, mit $|w| \geq n_0$ in drei Teile x, y, z zerlegen lässt, so dass $w = yxz$, wobei

(i) $|yx| \leq n_0$
(ii) $|x| \geq 1$
(iii) entweder $\{yx^k z \mid k \in \mathbb{N}\} \subseteq L$ oder $\{yx^k z \mid k \in \mathbb{N}\} \cap L = \emptyset$

Vorgehen: Wähle ein **konkretes** Wort mit Länge $|w| \geq n_0 \in \mathbb{N}$ aus, so dass keine seiner Zerlegungen die Eigenschaften (i), (ii) und (iii) erfüllt.

2.1.3 Pumping Lemma für kontextfreie Sprachen

Sei L kontextfrei. Dann existiert eine nur von L abhängige Kontante n_L, so dass für alle Wörter $z \in L$ mit $|z| \geq n_L$ eine Zerlegung $z = uvwxy$ von z existiert, so dass:

(i) $|vx| \geq 1$
(ii) $|vwx| \leq n_L$
(iii) $\{uv^iwx^iy \mid i \in \mathbb{N}\} \subseteq L$

2.1.4 Methode der Kolmogorov Komplexität

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

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

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

2.2 Reguläre Ausdrücke

Die Menge $RA(\Sigma)$ aller regulären Ausdrücke über dem Alphabet Σ ist rekursiv definiert durch:

$$\begin{aligned}
\lambda \in RA(\Sigma),\ \emptyset \notin RA(\Sigma) \qquad & L(\lambda) = \{\lambda\},\ L(\emptyset) = \{\emptyset\} \\
a \in RA(\Sigma) \text{ für alle } a \in \Sigma \qquad & L(a) = \{a\} \\
\alpha, \beta \in RA(\Sigma) \Rightarrow (\alpha, \beta) \in RA(\Sigma) \qquad & L(\alpha, \beta) = L(\alpha) \cdot L(\beta) \\
\alpha, \beta \in RA(\Sigma) \Rightarrow (\alpha + \beta) \in RA(\Sigma) \qquad & L(\alpha + \beta) = L(\alpha) \cup L(\beta) \\
\alpha \in RA(\Sigma) \Rightarrow \alpha^* \in RA(\Sigma) \qquad & L(\alpha^*) = (L(\alpha))^*
\end{aligned}$$

Sei $\alpha_{ij}^{(k)}$ der reguläre Ausdruck, der alle Transitionen vom Zustand i zu Zustand j über die Zwischenzustände $k \in \{0, ..., k\}$ beschreibt. Das heisst, es wird kein Zwischenzustand (Start- und Endzustand ausgenommen) q berührt, mit $q > k$.

$$\alpha_{ij}^{(k)} = \alpha_{ij}^{(k-1)} + \alpha_{ik}^{(k-1)} \cdot (\alpha_{kk}^{(k-1)})^* \cdot \alpha_{kj}^{(k-1)}$$

2.3 Nichtdeterminismus

Ein **nichtdeterministischer endlicher Automat** (NEA) ist wie ein endlicher Automat ein Tupel $M = (Q, \Sigma, \delta, q_0, F)$. Der einzige Unterschied ist, dass ein NEA zu einem Zustand q und gelesenen Zeichen a zu mehreren oder gar keinem Zustand übergehen kann.
Dementsprechend ist x in $L(M)$, falls es mindestens eine akzeptierte Berechnung auf x gibt. Zusätzlich kann eine nicht akzeptierte Berechnung auch frühzeitig enden, falls von dem momentanen Zustand q keine weiteren Zustände für das gelesene Zeichen gibt ($\delta(q, a) = \emptyset$).

$$L(M) = \{w \in \Sigma^* \mid \hat{\delta}(q, a) \cap F \neq \emptyset\}$$

Ausserdem gilt $\mathcal{L}_{NEA} = \mathcal{L}_{EA}$: Zu jedem NEA M existiert ein EA A, so dass

$$L(M) = L(A)$$

2.3.1 λ-NEA

λ-NEAs sind genau gleich definiert wie NEAs, also auch ein Tupel $M = (Q, \Sigma, \delta, q_0, F)$. Die übergangsfunktion δ ist jedoch wie folgt definiert:

$$\delta : (Q \times (\Sigma \cup \{\lambda\})) \mapsto Q$$

Es kann also zusätzlich auch dass leere Wort λ eingelesen werden.

Durch die Elimination der λ-Kanten lässt sich auch zeigen, dass $\mathcal{L}_{\lambda-NEA} = \mathcal{L}_{NEA}$.

3 Grammatiken

Eine Grammatik $G = (\Sigma_N, \Sigma_T, P, S)$ ist ein Tupel mit $\Sigma = \Sigma_N \cup \Sigma_T$:

Σ_N**:**	Die Nichtterminalsymbole spielen die Rolle von Variablen und dürfen in der Sprache nicht vorkommen.
Σ_T**:**	Die Terminalsymbole, die bisherige Nutzung zur Gestaltung von Wörtern aus der betrachteten Sprache.
$S \in \Sigma_N$**:**	Das Startsymbol
P**:**	Die Menge der Ableitungsregeln ist eine Teilmenge von $\Sigma^*\Sigma_N\Sigma^* \times \Sigma^*$
$(\alpha, \beta) \in P$**:**	Die Regel, die besagt, dass α kann in G durch β ersetzt werden (auch $\alpha \to_G \beta$)

Sei weiter $\gamma, \delta \in \Sigma^*$, dann sagen wir dass δ aus γ in einem Ableitungsschritt in G ableitbar ist, genau dann wenn w_1 und $w_2 \in \Sigma^*$ und eine Regel $(\alpha, \beta) \in P$ existieren, so dass

$$\gamma \Rightarrow_G \delta :\Leftrightarrow \gamma = w_1\alpha w_2 \text{ und } w_1\beta w_2$$

Weiter bedeutet, $S \Rightarrow_G^* w$, dass w von G erzeugt wird. Die von G erzeugte Sprache ist:

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

Typ-0:	Die Klasse der allgemeinen, uneingeschränkten Grammatiken
kontextsensitiv/Typ-1:	$\lvert\alpha\rvert \leq \lvert\beta\rvert$ für alle Regeln $(\alpha, \beta) \in P$. Es folgt, dass kein Teilwort α durch ein kürzeres Teilwort β ersetzen kann.
kontextfrei/Typ-2:	$\alpha \in \Sigma_N$ und $\beta \in (\Sigma_N \cup \Sigma_T)^*$ für alle Regeln $(\alpha, \beta) \in P$. Es folgt, dass alle Regeln die Form $X \to \beta$ für ein Nichtterminal X haben.
regulär/Typ-3:	$\alpha \in \Sigma_N$ und $\beta \in \Sigma_T^* \cdot \Sigma_N \cup \Sigma_T^*$ für alle Regeln $(\alpha, \beta) \in P$. Das bedeutet, dass die Regeln die Form $X \to u$ oder $X \to uY$ für ein $u \in \Sigma_T^*, X, Y \in \Sigma_N$ haben (enthält höchstens ein Nichtterminal welches nur als letztes Symbol vorkommen darf).
normiert:	falls alle Regeln nur eine der folgenden drei Regeln haben: (i) $S \to \lambda$, wobei S das Startsymbol ist (ii) $A \to a$ für $A \in \Sigma_N$ und $a \in \Sigma_T$ (iii) $B \to bC$ für $B, C \in \Sigma_N$ und $b \in \Sigma_T$

Es gilt:

$$\mathcal{L}_3 = \mathcal{L}_{EA}$$

3.1 Beweis durch Induktion

Zu zeigen: $L(G_1) = \{...\}$

1. Zeige, dass: $w \in L(G_1)$, eine Ableitung fur w in G_1 zu geben.
2. G1 nur Wörter vom w abgeleitet werden können. Zeigen wir durch induktion. Zuerst mit ein ableitung es gilt. Dann nehmen wir ein α bei dem wir annehmen dass es geht, wir ableiten noch einmal und zeigen dass es immer noch zu A gehort.

4 Turingmaschinen

Eine Turingmaschine ist eine Verallgemeinerung eines EA und besthet informell aus:

Kontrolle:	Enthält das Programm
Unendliches Band:	Eingabeband und Speicher
Lese-/Schreibkopf:	Kann sich in beide Richtungen auf dem Band bewegen

Bemerkung:

- Das Band ist nur nach rechts unendlich, links ist es mit dem Randsymbol ¢ begrenzt, welches von der Maschine nicht überschritten oder überschrieben werden darf. Dies erlaubt es, das Band **durchzunummerieren**.
- Die Felder, die nicht beschriftet sind, enthalten das Leerfeldsymbol ␣

Die **Ähnlichkeit** zu den EAs besteht in der Kontrolle über eine endliche Zustandsmenge und dem Band, das am Anfang das Eingabewort enthält.
Der **Hauptunterschied** liegt darin, dass eine Turingmaschine mit dem Band über einen unendlich grossen Speicher verfügt.

Formell ist eine **Turingmaschine** (TM) ein Tupel $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, wobei

Q**:**	Eine endliche Menge von Zuständen
Σ**:**	Das Eingabealphabet
Γ**:**	Das Arbeitsalphabet (alle Symbole, die in Feldern des Bandes auftreten dürfen), wobei $\Sigma \subseteq \Gamma$ und ¢, ␣ $\in \Gamma$ gelten, wie auch $\Gamma \cap Q = \emptyset$
δ**:**	Die Übergangsfunktion $(\delta : (Q - \{q_{\text{accept}}, q_{\text{reject}}\} \times \Gamma) \mapsto Q \times \Gamma \times \{L, R, N\})$. M kann also eine Aktion $(q, X, Z) \in Q \times \Gamma \times \{L, R, N\}$ aus einem aktuellen Zustand q beim Lesen eines Symbols $Y \in \Gamma$ durchführen, falls $\delta(p, Y) = (q, X, Z)$. Dies bedeutet der übergang von p nach q, das Ersetzen von Y durch X und die Bewegung des Kopfes gemäss Z (**L**inks, **R**echts, **N**ichts).
$q_0 \in Q$**:**	Der Anfangszustand
$q_{\text{accept}} \in Q$**:**	Der akzeptierende Zustand
$q_{\text{reject}} \in Q - \{q_{\text{accept}}\}$**:**	Der verwerfende Zustand

Die **akzeptierte Sprache** der Turingmaschine M ist

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

Weiter **berechnet** M eine Funktion $F : \Sigma^* \mapsto \Gamma^*$, falls

$$q_0 ¢ x \vdash_M^* q_{\text{accept}} ¢ F(x)$$

4.1 Kodierung von Turingmaschinen

Sei M eine TM mit $Q = \{q_0, q_1, ..., q_m, q_{\text{accept}}, q_{\text{reject}}\}$ und $\Gamma = \{A_1, A_2, ..., A_r\}$

$$\begin{aligned}
\text{Code}(q_i) &= 10^{i+1}1 \quad \text{für } i = 0, 1, ..., m \\
\text{Code}(q_{\text{accept}}) &= 10^{m+2}1 \\
\text{Code}(q_{\text{reject}}) &= 10^{m+3}1 \\
\text{Code}(A_j) &= 110^j11 \quad \text{für } j = 1, ..., r \\
\text{Code}(N) &= 1110111 \\
\text{Code}(R) &= 1110^2111 \\
\text{Code}(L) &= 1110^3111 \\
\text{Code}(\delta(p, A_l) = (q, A_m, \alpha)) &= \#\text{Code}(p)\text{Code}(A_l)\text{Code}(q)\text{Code}(A_m)\text{Code}(\alpha)\#
\end{aligned}$$

4.2 Mehrband-Turingmaschinen

Eine **k-Band-Turingmaschine** (MTM) hat im Gegensatz zu einer normalen TM ein endliches Band, welches nur die Eingabe enthält und k unendlich lange Arbeitsbänder, jedes mit eigenem Lese-/ Schreibkopf.
Zwei **Maschinenmodelle** $\mathcal{A}$ und $\mathcal{B}$ sind äquivalent, falls gilt:

- zu jeder Maschine $A \in \mathcal{A}$ existiert eine zu A äquivalente Maschine $B \in \mathcal{B}$
- zu jeder Maschine $C \in \mathcal{B}$ existiert eine zu C äquivalente Maschine $D \in \mathcal{A}$

Zu jeder MTM A existiert eine zu A äquivalente 1-Band-TM B. Dies heisst jedoch nicht, dass A und B äquivalent sind. Es kann zum Beispiel sein, dass A auf einem Wort $w \notin L(A) = L(B)$ unendlich lange läuft, während B dieses verwirft.
Die Maschinenmodelle von MTM und TM sind daher äquivalent.

5 Nichtdeterministische Turingmaschinen

Eine **nichtdeterministische Turingmaschine** M ist ein Tupel mit der gleichen Definition wie eine normale TM, ausser der übergangsfunktion. Diese wird mehr als Relation betrachtet:

$$\delta : (Q - \{q_{\text{accept}}, q_{\text{reject}}\} \times \Gamma) \mapsto \mathcal{P}(Q \times \Gamma \times \{L, R, N\})$$

6 Sprachenhierarchie

Die Sprachen sind der Reihe nach aufgelistet (d.h. die unteren Sprachen enthalten die oberen)

Klasse	**Notation**	**Erklärung**	**Beispiel**
regulär	$\mathcal{L}_{EA}$	Wird von einem EA akzeptiert	$x010y$
kontextfrei			0^n1^n, ww^R
kontextsensitiv			
rekursiv (entscheidbar)	$\mathcal{L}_R$	Wird von einer TM akzeptiert Die TM hält bei jedem eingegebenen Wort	
rekursiv aufzählbar (semientscheidbar)	$\mathcal{L}_{RE}$	Wird von einer TM akzeptiert Die TM hält bei jedem akzeptierten Wort	$(L_{\text{diag}})^C$
unentscheidbar		Wird von keiner TM und EA akzeptiert	L_{diag}

7 Berechenbarkeit

7.1 Methode der Reduktion

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

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

$L_1 \leq_R L_2$ bedeutet so viel wie "L_2 ist bezüglich algorithmischer Lösbarkeit mindestens so schwer wie L_1".
Für jede Sprache L gilt:

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

7.1.1 EE-Reduktion

Bei der **Eingabe-zu-Eingabe-Reduktion** ist die Idee, eine TM (einen Algorithmus) M zu finden, die für jede Eingabe x für das Entscheidungsproblem (Σ_1, L_1) eine Eingabe y für das Entscheidungsproblem (Σ_2, L_2) konstruiert, so dass die Lösung des Problems (Σ_2, L_2) für y der

Lösung des Problems (Σ_1, L_1) für x entspricht.

Falls es so eine TM M gibt, die die Abbildung $f_M : \Sigma_1^* \mapsto \Sigma_2^*$ berechnet, so dass

$$L_1 \leq_{EE} L_2 :\Leftrightarrow x \in L_1 \Leftrightarrow f_M(x) \in L_2$$

Dann sagen wir, dass L_1 auf L_2 **EE-reduzierbar** bzw. M die Sprache L_1 auf L_2 reduziert.

Bemerkung: Wichtig ist dabei, dass f berechenbar ist, d.h. für alle x muss die Berechnung terminieren, egal ob $x \in L_1$ oder $x \notin L_1$.

7.1.2 Diagonalsprache

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

7.1.3 Universelle Sprache

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

Es gibt eine TM U, die **universelle TM**, so dass

$$L(U) = L_U$$

Es folgt, dass $L_U \in \mathcal{L}_{\text{RE}}$, jedoch nicht $L_U \in \mathcal{L}_{\text{R}}$. Dies leuchtet ein, da die simulierte Maschine M für das eingegebene Wort w nicht zwingend terminiert.

7.1.4 Halteproblem

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

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

Da es kein Algorithmus gibt, der Testen kann, ob ein gegebenes Programm immer terminiert, gilt:

$$L_H \notin \mathcal{L}_R$$

7.1.5 Leere Sprache

Die **leere Sprache** ist die Sprache, die alle Kodierungen aller Turingmaschinen enthält, die die leere Menge (kein Wort) akzeptieren.

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

Es gilt

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

7.1.6 Satz von Rice

Jedes semantisch nichttriviale Entscheidungsproblem über Turingmaschinen ist unentscheidbar.

7.1.7 Post'sche Korrespondenzproblem

Zielsetzung ist es, mit Hilfe der Reduktion die Untentscheidbarkeit aus der Welt der Turingmaschinen in die Welt der Spiele (Domino) zu übertragen. Das Dominospiel wid auch **Post'sches Korrespondenzproblem (PKP)** gennant.

Eine **Instanz des Post'schen Korrespondezproblems** ist ein Paar (A, B), wobei $A = w_1, ..., w_k$ und $B = x_1, ..., x_k$. Für jedes $i = 1, ..., k$ wird (w_i, x_i) ein **Dominiostein** genannt. Es existiert eine **Lösung**, falls $i_1, i_2, ..., i_m$ so gewählt werden können, so dass:

$$w_{i_1} w_{i_2} ... w_{i_m} = x_{i_1} x_{i_2} ... x_{i_m}$$

Zusätzlich dazu gibt es noch die modifizierte Variante, **MPKP**. Es ist ein Spezialform des **PKP**, bei der das Dominospiel immer mit dem ersten Dominosteintyp anfangen muss. Eine Lösung dazu ist die folgende:

$$u_1 u_{j_1} u_{j_2} ... u_{j_m} = v_1 v_{j_1} v_{j_2} ... v_{j_m}$$

Bemerkung: Falls PKP entscheidbar ist, ist auch MPKP entscheidbar. Da MPKP untentscheidbar ist, ist somit auch PKP **unentscheidbar**.

7.2 Methode der Kolmogorovkomplexität

Wir benutzen diese Methode, um zu beweisen, dass kein Algorithmus existiert, der die Kolmogorovkomplexität $K(x)$ von x aller Wörter $x \in \Sigma^*_{\text{bool}}$ berechnen kann. Dies ist nämlich **algorithmisch unlösbar**.

8 Komplexitätstheorie

8.1 Komplexitätsmasse

8.1.1 Zeitkomplexität

Sei M eine MTM oder TM, die immer hält und sei $D = C_1, C_2, ..., C_k$ die Berechnung von M auf $x \in \Sigma^*$. Dann ist die **Zeitkomplexität** $\text{Time}_M(x)$ die Berechnung von M auf x definiert durch die Anzahl Berechnungsschritte in D:

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

Die Zeitkomplxität $\text{Time}_M : \mathbb{N} \mapsto \mathbb{N}$ ist als Funktion definiert durch:

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

$\text{Time}_M(n)$ ist die Zeitkomplexität der längsten Berechnung auf Eingaben der Länge n. Man nennt dies auch **die Komplexität im schlechtesten Fall**.

8.1.2 Speicherplatzkomplexität

Sei M eine k-Band-Turingmaschine, die immer hält und sei weiter $C = (q, x, i, \alpha_1, i_1, \alpha_2, i_2, ..., \alpha_k, i_k)$ eine Konfiguration von M. Dann ist die **Speicherplatzkomplexität von** C gleich:

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

Sei $C_1, C_2, ..., C_l$ die Berechnung von M auf x, dann ist die **Speicherplatzkomplexität von** M **auf** x geich

$$\text{Space}_M(x) = \max\{\text{Space}_M(C_i) \mid i = 1, ..., l\}$$

Die **Speicherplatzkomplexität von** M ist die Funktion $\text{Space}_M : \mathbb{N} \mapsto \mathbb{N}$, definiert durch:

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

Die Speicherplatzkomplexität einer Konfiguration ist also die **maximale beschriftete Länge** eines Arbeitsbandes.

8.1.3 Asymptotische Komplexität

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

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

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

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

Mit Bedeutung:

$r \in 0(f(n))$**:** r wächst asymptotisch nicht schneller als f
$s \in \Omega(f(n))$**:** r wächst asymptotisch mindestens so schnell wie f
$q \in \Theta(f(n))$**:** r wächst asymptotisch gleich schnell wie f

Sei L eine Sprache und f, g zwei Funktionen, dann ist:

- $O(g(n))$ eine **obere Schranke für die Zeitkomplexität von** L, falls eine MTM A existiert, so dass $L(A) = L$ und $\text{Time}_A(n) \in O(g(n))$
- $\Omega(f(n))$ eine **untere Schranke für die Zeitkomplexität von** L, falls eine MTM B existiert, so dass $L(B) = L$ und $\text{Time}_B(n) \in \Omega(f(n))$
- Eine MTM C heisst **optimal für** L, falls $\text{Time}_C(n) \in \emptyset(f(n))$ gilt und $\Theta(f(n))$ eine untere Schranke für die Zeitkomplexität von L ist.

8.1.4 Komplexitätsklassen

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

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

(i) $\text{Space}_M(n) \leq s(n)$ für alle $n \in \mathbb{N}$
(ii) für jede Eingabe 0^n generiert generiert M das Wort $0^{s(n)}$ und hält in q_{accept}

Weiter ist eine Funktion $t : \mathbb{N} \mapsto \mathbb{N}$ **zeitkonstruierbar**, falls eine MTM M existiert, so dass:

(i) $\text{Time}_M(n) \in O(t(n))$
(ii) für jede Eingabe 0^n generiert generiert M das Wort $0^{t(n)}$ und hält in q_{accept}

Bemerkung: Die meisten gewöhnlichen monotonen Funktionen sind platzkonstruierbar bzw. zeitkonstruierbar.

8.2 Nichtdeterministische Komplexitätsklassen

8.2.1 Zeitkomplexität

Sei M eine NTM mit $x \in L(M)$. Die **Zeitkomplexität von** M **auf** x gleich:

$$\text{Time}_M(x) = \text{die Länge einer kürzesten akzeptierenden Berechnung}$$

Die **Zeitkomplexität von** M ist die Funktion $\text{Time}_M : \mathbb{N} \mapsto \mathbb{N}$:

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

8.2.2 Speicherplatzkapazität

Sei $C = C_1C_2...C_m$ eine akzeptierende Berechnung von M auf x. Dann ist

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

Die **Speicherplatzkapazität von M auf x** ist dann definiert als

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

Die **Speicherplatzkomplexität von M** ist die Funktion $\text{Space}_M : \mathbb{N} \mapsto \mathbb{N}$:

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

8.2.3 Komplexitätsklassen

$$\begin{aligned}
\text{NTIME}(f) &= \{L(B) \mid \text{B ist eine nichtdeterministische MTM mit } \text{Time}_B(n) \in O(f(n))\} \\
\text{NSPACE}(g) &= \{L(A) \mid \text{A ist eine nichtdeterministische MTM mit } \text{Space}_A(n) \in O(g(n))\} \\
\text{NLOG} &= \text{NSPACE}(\log_2 n) \\
\text{NP} &= \bigcup_{c \in \mathbb{N}} \text{NTIME}(n^c) \\
\text{NPSPACE} &= \bigcup_{c \in \mathbb{N}} \text{NSPACE}(n^c)
\end{aligned}$$

Für jede Funktion $t : \mathbb{N} \mapsto \mathbb{R}^+$ und $s : \mathbb{N} \mapsto \mathbb{N}$ mit $s(n) \geq \log_2(n)$ gelten folgende Vergleiche:

Deterministisch - Deterministisch

$$\begin{aligned}
\text{DLOG} \subseteq \text{P} \subseteq \text{PSPACE} \subseteq \text{EXPTIME} \\
\text{TIME}(s(n)) \subseteq \text{SPACE}(s(n)) \\
\text{SPACE}(s(n)) \subseteq \bigcup_{c \in \mathbb{N}} \text{TIME}(c^{s(n)})
\end{aligned}$$

Nicht Deterministisch - Nicht Deterministisch

$$\begin{aligned}
\text{NTIME}(t) \subseteq \text{NSPACE}(t) \\
\text{NSPACE}(s) \subseteq \bigcup_{c \in \mathbb{N}} \text{NTIME}(c^{s(n)})
\end{aligned}$$

Deterministisch - Nicht Deterministisch

$$\begin{aligned}
\text{TIME}(t) \subseteq \text{NTIME}(t) \\
\text{SPACE}(t) \subseteq \text{NSPACE}(t) \\
\text{NTIME}(s(n)) \subseteq \text{SPACE}(s(n)) \subseteq \bigcup_{c \in \mathbb{N}} \text{TIME}(c^{s(n)}) \\
\text{NP} \subseteq \text{PSPACE} \\
\text{NLOG} \subseteq \text{P} \\
\text{NPSPACE} \subseteq \text{EXPTIME} \\
\text{PSPACE} = \text{NPSPACE}
\end{aligned}$$

8.3 Beweisverifikation

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

(i) $\text{Time}_A(w, x) \leq p(|w|)$ für jede Eingabe $(w, x) \in \Sigma^* \times (\Sigma_{\text{bool}})^*$
(ii) Für jedes $w \in L$ existiert ein $x \in (\Sigma_{\text{bool}})^*$, so dass $|x| \leq p(|w|)$ und $(w, x) \in L(A)$. Das Wort x ist ein **Beweis** für die Behauptung $w \in L$
(iii) Für jedes $y \notin L$ gilt $(y, z) \notin L(A)$ für alle $z \in (\Sigma_{\text{bool}})^*$

Ist $p(n) \in O(n^k)$ für ein $k \in \mathbb{N}$, nennen wir A einen **Polynomialzeitverifizierer**. Somit wird VP als die folgende Sprache definiert:

$$VP = \{V(A) \mid \text{A ist ein Polynomialzeitverifizierer}\}$$

Bemerkung: $V(A)$ und $L(A)$ sind verschiedene Sprachen;

$$V(A) = \{w \in \Sigma^* \mid \text{es existiert ein } x \in (\Sigma_{\text{bool}})^* \text{ mit } |x| \leq p(|w|)\}$$

Ein **p Verifizierer** A für eine Sprache L ist also ein deterministischer Algorithmus, der für die Eingabe (w, x) verifiziert, ob x ein Beweis für die Behauptung $w \in L$ ist.
Jede polynomielle NTM kann in eine äquivalente NTM umgewandelt werden, die alle nichtdeterministischen Entscheidungen am Anfang macht und dann nur die Richtigkeit des geratenen verifiziert.

$$VP = NP$$

Daraus folgt, dass die Klasse NP die Klasse aller Sprachen L ist, die für jedes $x \in L$ einen in $|x|$ polynomiell langen Beweis von $x \in L$ haben, welchen man deterministisch in polynomieller Zeit bezüglich $|x|$ verifizieren kann.

8.4 NP-Vollständigkeit

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

$$x \in L_1 \Leftrightarrow A(x) \in L_2$$

Dies bedeutet also, dass L_2 mindestens so schwer wie L_1 ist.

Eine Sprache L ist **NP-schwer**, falls für alle Sprachen $L' \in NP$ gilt $L' \leq_p L$. Eine Sprache L ist **NP-vollständig**, falls gilt

(i) $L \in NP$
(ii) L ist NP-schwer

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

8.4.1 Vergleiche verschiedener Sprachen

Betrachten wir folgende Sprachen:

$$\text{SAT} = \{x \in (\Sigma_{\text{logic}})^* \mid x \text{ kodiert eine erfüllbare Formel in KNF}\}$$
$$k - \text{SAT} = \{x \in (\Sigma_{\text{logic}})^* \mid x \text{ kodiert eine erfüllbare Formel in k-KNF}\}$$
$$\text{CLIQUE} = \{(G, k) \mid G \text{ ist ein ungerichteter Graph, der eine } k\text{-Clique enthält}\}$$
$$\text{VC} = \{(G, k) \mid G \text{ ist ein ungerichteter Graph mit der Knotenüberdeckung der Mächtigkeit höchstens } k\}$$

Eine **k-KNF** ist eine KNF, bei der jede Klausel aus höchstens 3 Literalen besteht. Eine **Clique** vom Graphen $G = (V, E)$ ist ein Subgraph G', der vollständig ist. Eine **Knotenüberdeckung** vom Graphen $G = (V, E)$ ist wenn jede Kante von G mindestens einen Knoten aus U (einer Teilmenge von V) enthält.
Dann gelten:

$$\begin{aligned} &\text{SAT ist NP-vollständig (Satz von Cook)} \\ &\text{SAT} \leq_p \text{CLIQUE} \\ \text{CLIQUE} &\leq_p \text{VC} \\ &\text{SAT} \leq_p \text{3SAT} \end{aligned}$$

8.4.2 Klasse der Optimierungsprobleme

NPO ist die **Klasse der Optimierungsprobleme**, wobei

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

falls folgende Bedingungen erfüllt sind:

(i) $L \in P$
(ii) Es existiert ein Probelm p_U, so dass
  (a) Für jedes $x \in L$ und jedes $y \in \mathcal{M}(x)$ gilt $|y| \leq p_U(|x|)$
  (b) 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
(iii) Die Funktion *cost* kann man in polynomieller Zeit berechnen

Weiter ist **PO** die **Klasse der Optimierungsprobleme** $U = (\Sigma_I, \Sigma_O, L, \mathcal{M}, \text{cost}, \text{goal})$, so dass

(i) $U \in NPO$
(ii) Es existiert ein polynomieller Algorithmus A, so dass $A(x)$ für jedes $x \in L$ eine optimale Lösung ist