Theoretische Informatik

> ❗ This summary contains all relevant material for the mid- and endterm exam 2021. The content for the session exam or future endterms might be different. E.g. include 10.4.

> ❗ This and many more summaries can be found on https://n.ethz.ch/~dcamenisch. Feel free to leave a comment in the document if you spot any mistakes!

▼ 0. Table of Contents

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

Definition: Ein **Wort** über ein Alphabet Σ ist eine endliche Folge von Buchstaben aus Σ. 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.

Definition: a^R bezeichnet die Umkehrung des Wortes a.

Definition: v ist genau dann ein echtes Teilwort on w, wenn $v \neq \lambda \ \wedge \ v \neq w$.

2.3 Algorithmische Probleme

Definition: Das **Entscheidungsproblem** für ein gegebenes Alphabet und eine gegebene Sprache, ist zu entscheiden, ob $x \in L, \forall x \in \Sigma^*$. Wenn ein Algorithmus A das Entscheidungsproblem löst, sagen wir A erkennt L und L ist rekursiv.

Definition: Ein **Aufzählungsalgorithmus** für eine Sprache L, gibt für eine Eingabe $n \in \mathbb{N} - \{0\}$ die Wortfolge $x_1, x_2, ..., x_n$ aus. Für eine rekursive Sprache L existiert immer ein Aufzählungsalgorithmus.

2.4 Kolmogorov-Komplexität

> ❗ Die **Kolmogorov-Komplexität** $K(x)$ eines Wortes x ist das Minimum der binären Länge der Pascal-Porgramme, welche das Wort generieren.

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

Lemma 2.5: Für jede Zahl $n \in \mathbb{N} - \{0\}$ existiert ein Wort $x \in \Sigma^n_{bool}$, so dass $K(x) \geq |x| = n$, d.h es existiert ein Wort der Länge n, welches nicht komprimierbar ist.

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

Satz 2.2 Für eine boolsche Sprache gilt, wenn die Sprache rekursiv ist und x_n das n-te Wort in kanonischer Ordnung ist, so ist:

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

3. Endliche Automaten

Ein **endlicher Automat** ist ein Quintupel $M = (Q, \Sigma, \delta, q_0, F)$, wobei
Q eine endliche Menge von *Zuständen* ist,
Σ ein Alphabet, genannt *Eingabealphabet*, ist,
$q_0 \in Q$ der *Anfangszustand* ist,
$F \subseteq Q$ die *Menge der akzeptierten Zustände* ist und
δ eine Funktion von $Q \times \Sigma$ nach Q ist, die *Übergangsfunktion* genannt wird.

Eine **Konfiguration** ist ein Element $Q \times \Sigma^*$, wobei wir einen Zustand und ein noch zu lesendes Suffix haben.

Ein **Schritt** $\vdash_M$ ist ein Schritt von einer zur nächsten Konfiguration (es gibt dann noch die transitive Hülle gekennzeichnet mit einem *).

Eine **Berechnung** ist eine endliche Folge $C = C_0, C_1, ..., C_n$ von Konfigurationen, so dass $C_i \vdash_M C_{i+1}$ für alle $0 \leq i \leq n-1$.

Die von M aktzeptierte Sprache ist $L(M)$. $\mathcal{L}_{EA}$ ist die Klasse der von EA aktzeptierten Sprachen. Diese wird auch als die Klasse der **regulären Sprachen** bezeichnet.

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.

Folgende Lemmas / Sätze sind essentiel für **Beweise der Nichtexistenz**:

Lemma 3.3: Sei A ein EA und $x, y \in \Sigma^*, x \neq y$, so dass x, y in der selben Klasse sind. Dann existiert für jedes $z \in \Sigma^*$ ein $r \in Q$, so dass xz und $yz \in Kl[r]$.

Pumping-Lemma: Sei L regulär. Dann existiert ein $n_0 \in \mathbb{N}$, so dass sich jedes Wort $w \in \Sigma^*, |w| \geq n_0$ in drei Teile zerlegen lässt, also $w = yxz$, wobei
(i) $|yx| \leq n_0$
(ii) $|x| \geq 1$
(iii) $\{yx^k z \mid k \in \mathbb{N}\} \subseteq L \quad \vee \quad \{yx^k z \mid k \in \mathbb{N}\} \cap L = \emptyset$

Satz 3.1: Sei $L \subseteq \Sigma^*_{bool}$ eine reguläre Sprache und sei L_x eine Prefixsprache. Dann existiert eine Konstante c, so dass für alle $x, y \in \Sigma^*_{bool} : K(y) \leq \lceil \log_2(n+1) \rceil + c$ gilt, fall y das n-te Wort in der Sprache L_x ist.

Oft wollen wir zeigen, dass kein EA A mit weniger als k Zuständen existiert, so dass $L(A) = L$. Wir gehen dabei wiefolgt vor:

1. Bestimme k Wörter aus L
2. Nach Lemma 3.3 muss nun gelten, dass für zwei Wörter w_i, w_j, welche im gleichen Zustand landen, auch $w_i z, w_j z$ im gleichen Zustand landen müssen.

3. Mithilfe einer Tabelle geben wir jetzt für jedes Paar von Wörtern ein Suffix z_{ij} an, so dass nicht beide in den selben Zustand führen. Woraus folgt, dass A mindestens k Zustände haben muss.

Beweise mit der Kolmogorov-Komplexität, wobei die Wörter nur aus einem Buchstaben bestehen (hier 0), können wie folgt geführt werden:

1. Sei w_n das n-te Wort in kanonischer Ordnung
2. L_{w_n} hat λ als erstes Wort und $v_n = 0^{|w_{n+1} - w_n|}$ als zweites Wort
3. Daher gilt $K(v_n) \leq \lceil \log_2(2+1) \rceil + c = 2 + c$
4. Es gibt unendlich viele v_n, da $|v_n| = |w_{n+1}| - |w_n| \to^{n \to \infty} \infty$
5. Aber nur endlich viele Programme mit Länge höchstens $2 + c$
6. Wir haben einen Widerspruch zur Annahme, dass L regulär ist

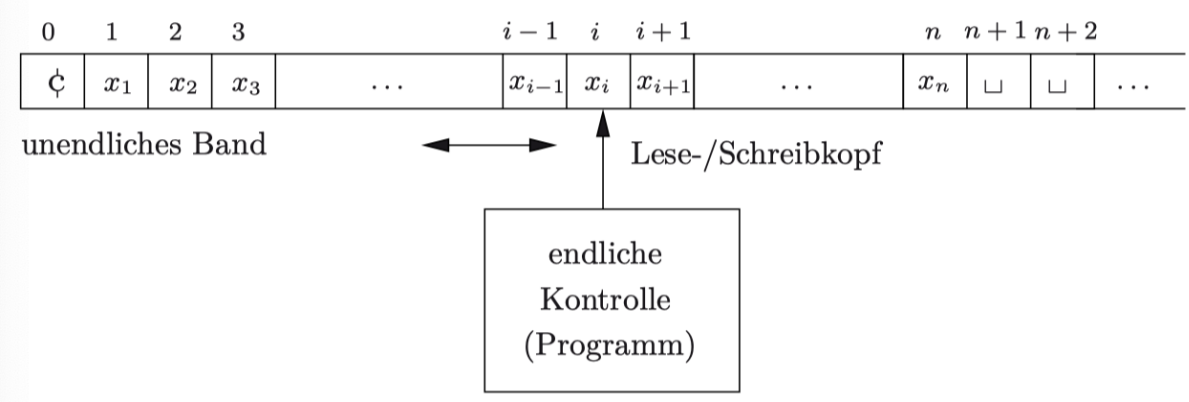
3.5 Nichtdeterminismus

Weiter existieren nichtdeterministische EA (NEA). Die Definition erfolgt analog zum EA, jedoch ist δ nun eine Funktion von $Q \times \Sigma$ nach $\mathcal{P}(Q)$. Dabei ist wichtig, dass diese nicht mächtiger sind als EA.

Satz 3.2: Zu jedem NEA M existiert ein EA A, so dass $L(M) = L(A)$. Einen EA zu einem NEA können wir mithilfe der Potenzmenge der Zustände konstruieren.

4. Turingmaschinen

4.3 Das Modell der Turingmaschine



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

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

> Wir definieren die Begriffe Schritt, Berechnung und aktzeptierte Sprache analog zu den EA.

Eine Sprache heisst **rekursiv aufzählbar**, falls eine TM existiert, so dass die Sprache von der TM akzeptiert wird.

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

Eine Sprache heisst **rekursiv**, falls $L = L(M)$ für eine TM, für die für alle $x \in \Sigma^*_{bool}$ gilt:
(i) $q_0 cx \vdash^*_M yq_{accept}z$ falls $x \in L$
(ii) $q_0 cx \vdash^*_M yq_{reject}v$ falls $x \notin L$
Gelten diese Bedingungen immer, so sagen wir, dass M auf jeder Eingabe **hält**.

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

4.4 Mehrband-TM und Church'sche These

Eine Mehrband-TM (MTM), hat zusätzlich zu einem endlichen, nur lesbaren Programm(band), auch noch k viele Arbeitsbänder, auf welchen sowohl geschrieben als auch gelesen werden kann.

Satz: Die Maschinenmodelle von Turingmaschinen und Mehrband-Turingmaschinen sind äquivalent.

> **Chuch'sche These:** Die TM sind die Formalisierung des Begriffes Algorithmus, d.h. die Klasse der rekursiven Sprachen stimmt mit der Klasse der algorithmisch erkennbaren Sprachen überein.

4.5 Nichtdeterministische Turingmaschinen

Analog zu den endlichen Automaten, können wir auch den **Nichtdeterminismus** für TMs definieren.

4.6 Kodierung von Turingmaschinen

Die **Kodierung** einer TM $Kod(M) = h(Code(M))$ besteht aus einer Anneinanderreihung von Codierungen der einzelnen Symbole, Transitionen und Zuständen. Zuerst werden dabei die Anzahl der Zustände und Symbole angegeben.

Wir bezeichnen die Menge der Kodierungen aller TM als:

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

5. Berechenbarkeit

5.2 Die Methode der Diagonalisierung

❗ Eine Menge A heisst **abzählbar**, falls A endlich ist oder $|A| = |\mathbb{N}|$. Zwei Mengen sind gleich mächtig, falls eine Bijektion zwischen den beiden Mengen existiert.

Definition: Eine Menge A ist kleiner oder gleich **Mächtig** wie eine Menge B ($|A| \leq |B|$), wenn eine Injektion von A nach B existiert. $|A| = |B|$ gilt wenn eine Bijektion existiert.

Folgendes sind **abzählbare Mengen**:

$$\Sigma^*, \text{KodTM}, \mathbb{N} \times \mathbb{N}$$

Folgendes sind **nichtabzählbare Mengen**:

$$[0,1], \mathcal{P}(\Sigma^*_{bool})$$

Da die Menge der Turingmaschinen abzählbar ist, aber die Menge $\mathcal{P}(\Sigma^*_{bool})$ unabzählbar ist, existieren unendliche viele nicht rekursiv aufzählbare Sprache $L \subseteq \Sigma^*_{bool}$.

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

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

Es gilt $L_{diag} \notin \mathcal{L}_{RE}$. Dies können wir wiefolgt zeigen:

> Wir definieren eine Matrix mit $d_{ij} = 1 \Leftrightarrow w_j \in L(M_i)$. Nun definieren wir $L_{diag} = \{w \in \Sigma^*_{bool} \mid w = w_i \text{ für ein } i \in \mathbb{N} - \{0\} \text{ und } d_{ij} = 0\}$. Durch das Argument der Diagonalisierung können wir nun zeigen, dass L_{diag} von keiner TM erkannt wird.

5.3 Die Methode der Reduktion

💡 Mithilfe der **Reduktion** zeigen wir, dass ein Problem A unlösbar ist (in einer gewissen Komplexität), indem wir ein Problem B finden, von dem bereits bekannt ist, dass es unlösbar ist, und zeigen, dass die Lösbarkeit von A die Lösbarkeit von B implizieren würde.

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

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

Intuitiv bedeutet dies, dass L_2 mindestens 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.

> Es gibt einige Tricks für die Reduktion, so ist f(x) oft eine TM A, die den Input x so modifiziert, dass wir eine TM erhalten, die...
>
> ... ihren Input ignoriert und M auf w simuliert
>
> ... gleicht ist, aber alle Transitionen zu q_{accept} und q_{reject} geändert wurden
>
> ... ∞ Loops hinzufügt
>
> ... bei bestimmten Inputs sofort aktzeptiert und nur bei den Restlichen simuliert

R-Reduktionen verlaufen nach folgendem Schema:

1. Wir nehemen an, es existiert ein Algorithmus B für L_2, wir konstruieren nun eine Algorithmus A für L_1, der B verwendet.
2. Zeige, dass für alle Inputs das korrekte Resultat ausgegeben wird.

EE-Reduktionen haben folgendes Schema:

1. Angabe der $TM\ M$ für die Abbildung f_M
2. Beweis dass $x \in L_1 \Leftrightarrow f_M(x) \in L_2$

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: 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 $L_U \in \mathcal{L}_{RE}$. *Weiter gilt, dass* $L_U \notin \mathcal{L}_R$.

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

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

Es gilt, dass $L_H \notin \mathcal{L}_R$, aber $L_H \in \mathcal{L}_{RE}$.

Definition: Wir betrachten die leere Sprache:

$$L_{empty} := \{\mathrm{Kod}(M) \,|\, L(M) = \emptyset\}.$$

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 $L_{EQ} := \{\mathrm{Kod}(M)\#\mathrm{Kod}(\overline{M}) \,|\, L(M) = L(\overline{M})\}$ ist unentscheidbar, das heisst $L_{EQ} \notin \mathcal{L}_R$.

5.4 Der Satz von Rice

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

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

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

Wir betrachten zudem kurz folgende Sprache als ein spezifisches Halteproblem:

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

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

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

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 $\mathrm{Kod}_L \notin \mathcal{L}_R$.

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* $x \in (\Sigma_{bool})^*$.

Wichtige Sprachen und ihre Klassifizierung

$L_{diag} := \{w \in \{0,1\}^* \mid w = w_i \text{ für ein } i \in N^+ \text{ und } M_i \text{ akzeptiert } w_i \text{ nicht}\}$

$L_U := \{Kod(M)\#w \mid w \in \{0,1\}^* \text{ und } M \text{ akzeptiert } w\}$

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

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

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

$(L_{empty})^C := \{w \in \{0,1\}^* \mid w \neq \mathrm{Kod}(M) \text{ oder } w = \mathrm{Kod}(M) \text{ und } L(M) \neq \emptyset\}$

$L_{EQ} := \{\mathrm{Kod}(M)\#\mathrm{Kod}(\overline{M}) \mid L(M) = L(\overline{M})\}$

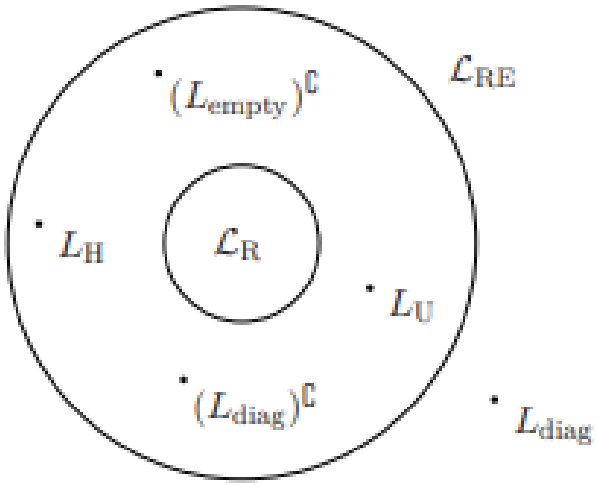
$L_{EQ} \notin \mathcal{L}_R$

$\mathcal{L}_{RE} := \{L(M) \mid \mathrm{M \text{ ist eine TM}}\} = \text{rekursiv aufzählbare Sprachen}$

Zu zeigen: Konstruiere eine NTM oder TM welche die Sprache erkennt

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

Zu zeigen: Mittels Reduktion von einer Sprache die $\in \mathcal{L}_R$ oder $\notin \mathcal{L}_R$



> Sprachen in $\mathcal{L}_R$: —— (können durch Angabe einer TM bewiesen werden)
>
> Sprachen in $\mathcal{L}_{RE}$: L_U, L_H, L^C_{empty}, L^C_{diag}, $L_{H,\lambda}$, Kod_L
>
> Sprachen die weder in $\mathcal{L}_R$ noch in $\mathcal{L}_{RE}$ sind: L_{diag}, L_{empty}, L^C_U, L^C_H

6. Komplexitätstheorie

6.2 Komplexitätsmasse

Wir unterscheiden zwei grundlegende Komplexitätsmassen: die Zeitkomplexität und die Speicherkomplexität. Die Zeitkomplexität einer Berechnung entspricht der Anzahl elementarer Operationen. 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

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

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

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

Die ***Speicherkomplexität von*** M auf x ist

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

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

$$\mathbf{Space}_M(n) = \max\{\mathrm{Space}_M(x) \,|\, x \in \Sigma^n\}.$$

Wir bemerken, dass die Komplexität unabhängig vom Alphabet ist. Darauf folgt:

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

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

Da diese Angaben relativ grob sind, verwenden wir die asymptotische Masse. Diese sind bereits aus AnD bekannt, bis auf eine.

Definition: Seien f, g zwei Funktionen $\mathbb{N} \to \mathbb{R}^+$. 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 (Σ_{bool}, L), so dass für jede MTM A, die (Σ_{bool}, L) entscheidet, eine MTM B existiert, die auch (Σ_{bool}, L) entscheidet, und für die gilt

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

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

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 Mexistiert, 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}.

Die meisten gewöhnlichen monotonen Funktionen mit $f(n) \geq \log_2(n+1)$ sind platzkonstruierbar, und analog dazu sind die meisten Funktionen $f(n) \geq n$ zeitkonstruierbar.

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 $\mathrm{Time}_M(x) \leq t(|x|)$ für alle $x \in L(M)$. Dann existiert eine $MTM\ A$ mit $L(A) = L(M)$ und

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

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

- $\mathrm{Space}_M(n) = \max\{\mathrm{Space}_M(x) \,|\, x \in \Sigma^n\}$ und
- $\mathrm{Time}_M(n) = \max\{\mathrm{Time}_M(x) \,|\, x \in \Sigma^n\}$

oder als

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

definiert.

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

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

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

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

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

Satz: Seien s_1 und s_2 zwei Funktionen von $\mathbb{N}$ nach $\mathbb{N}$ mit 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 $\mathrm{SPACE}(s_1) \subsetneq \mathrm{SPACE}(s_2)$.

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

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

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

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

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

$$\mathbf{Space}_M(x) = \min\{\mathrm{Space}_M(C) \,|\, C \text{ ist eine akzeptierende Berechnung von } M \text{ auf } x\}.$$

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

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

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

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

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

> **Aus diesem Satz folgt direkt $\mathrm{NP} \subseteq \mathrm{PSPACE}$**

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

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

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.

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

"

> **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

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

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:

- $\mathrm{SAT} := \{\Phi \mid \Phi \text{ ist eine erfüllbare Formel in KNF}\}$
- $\mathrm{3SAT} := \{\Phi \mid \Phi \text{ ist eine erfüllbare 3KNF Formel, d.h. } \leq 3 \text{ Literale pro Klausel}\}$
- $\mathrm{CLIQUE} := \{(G,\, k) \mid G \text{ ist ein ungerichteter Graph, der eine } k\text{-Clique enthält}\}$
- $\mathrm{VC} := \{(G,\, k) \mid G \text{ ist ein ung. Graph mit einer Knotenüberdeckung } \leq k\}$

Lemma: Wir stellen die folgenden Ungleichungen:

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

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.*
2. Σ_T ist ein Alphabet, genannt *Terminalalphabet.*

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

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

"

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,

"

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

"

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

"

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

"

Eine Folge von Ableitungsschritten

"

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

"

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:

"

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

"

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

"

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:

"

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

"

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

10.5 Allgemeine Grammatiken und Turingmaschinen

Satz: $\mathcal{L}_{RE} = \mathcal{L}_0$, d.h. die Menge der rekursiv aufzählbaren Sprachen ist genau die gleiche Menge wie die Menge aller Sprachen, die durch Grammatiken generiert werden können.