Theoretische Grundlagen der Informatik Zusammenfassung

Jean-Pierre von der Heydt

12. März 2019

Inhaltsverzeichnis

1 Einführung

Diese Zusammenfassung beruht auf der Vorlesung Theoretische Grundlagen der Informatik in WS 17/18 und dem dazugehörigen Skript von Prof. Dr. Dorothea Wagner.

1.1 Alphabete und Wörter

Definition 1.1 (Definitionen zu Alphabeten und Wörtern)

- Ein Alphabet Σ ist eine endliche Menge von Zeichen.
- Ein Wort w über Σ ist eine (möglicherweise leere) Folge von Zeichen aus Σ.
- Das leere Wort wird mit ϵ symbolisiert.
- Die Menge aller Wörter über Σ wird mit Σ^* abgekürzt.
- Die Menge aller Wörter der Länge n über Σ wird mit Σ^n abgekürzt.
- Die Konkatenation zweier Wörter w_1 und w_2 wird mit $w_1 \cdot w_2$ oder $w_1 w_2$ abgekürzt.
- Die iterierte Konkatenation eines Wortes w bezweichnen wir mit $w^k := \underbrace{w \cdot \dots \cdot w}_{k \text{ mal}}$
- Lässt sich w als $w = t \cdot u \cdot v$, so heißt: $\left.\begin{array}{cc} t & \text{Präfix} \\ u & \text{Teilwort} \\ v & \text{Suffix} \end{array}\right\}$ von w

1.2 Formale Sprachen

Definition 1.2 (Definitionen zu formalen Sprachen)

- Eine formale Sprache L über Σ ist eine Teilmenge $L \subseteq \Sigma^*$
- Die Produktsprache zweier Sprachen L_1 und L_2 definieren wir als $L_1 \cdot L_2 := \{w_1 w_2 \mid w_1 \in L_1 \text{ und } w_2 \in L_2\}$
- Das Produkt von L mit sich selbst wird als $L^k := \underbrace{L \cdot \dots \cdot L}_{k \text{ mal}}$ bezeichnet. $L^0 = \{\epsilon\}$
- Der Kleene'sche Abschluss ist definiert durch $L^* := \bigcup_{i \geq 0} L^i$
- Der positive Kleenne'sche Abschluss ist definiert durch $L^+ := \bigcup_{i \geq 1} L^i$
- Die Quotientensprache bezeichnet $L_1 / L_2 := \{w \in \Sigma^* \mid \exists z \in L_2 \text{ mit } w \cdot z \in L_1\}$
- Die Komplementsprachen bezeichnet $L^c := \Sigma^* \setminus L$

1.3 Reguläre Sprachen

Definition 1.3 (Reguläre Sprache)
Eine Sprache L über Σ heißt regulär, wenn sie auf folgende Weise produziert werden kann:

- Verankerung:
  - $L = \{a\}$ mit $a \in \Sigma$
  - $L = \{\}$
  - $L = \{\epsilon\}$
- Induktion: L_1, L_2 seien regulär
  - $L = L_1 \cdot L_2$
  - $L = L_1 \cup L_2$
  - $L = L_1^*$

1.4 Kontextfreie Sprachen

Definition 1.4 (Kontextfreie Grammatik)
Eine kontextfreie Grammatik $G = (\Sigma, V, S, R)$ ist gegeben durch:

- ein endliches Alphabet Σ
- eine endliche Menge V mit $V \cap \Sigma = \{\}$ von Nichtterminalsymbolen
- ein Startsymbol $S \in V$
- eine endliche menge von Ableitungsregeln R, d.h. eine Menge von Tupeln $(I, r) \in V \times (\Sigma \cup V)^*$

Wir schreiben auch $I \to r$. $L(G) := \{w \in \Sigma^* \mid S \to^* w\}$ bezeichnet die von G erzeugte Sprache.

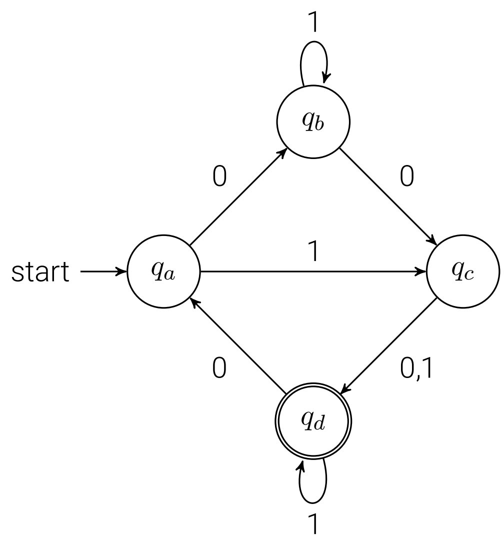
2 Endliche Automaten und reguläre Ausdrücke

2.1 Deterministische endliche Automaten

Definition 2.1 (DEA)
Ein (deterministischer) endlicher Automat $(Q, \Sigma, \delta, s, F)$ (D)EA besteht aus:

- Q, einer endlichen Menge von Zuständen
- Σ, einer endlichen Menge von Eingabesymbolen
- $\delta \colon Q \times \Sigma \to Q$, einer Übergangsfunktion
- $s \in Q$, eines Startzustand
- $F \subseteq Q$, einer Menge von Endzuständen

Beispiel 2.2
In diesem Beispiel ist $s = q_a$ und $F = \{q_d\}$. Das Eingabealphabet ist $\{0, 1\}$:



Definition 2.3
Ein EA erkennt oder akzeptiert eine Sprache L (über dem Alphabet des Automaten), wenn er nach Abarbeitung eines Wortes w genau dann in einem Endzustand ist, wenn w in L ist.

Satz 2.4
Jede reguläre Sprache wird von einem (deterministischen) endlichen Automaten akzeptiert.

2.2 Nichtdeterministische endliche Automaten

Definition 2.5 (NEA)
Ein nichtdeterministischer endlicher Automat $(Q, \Sigma, \delta, s, F)$ NEA besteht aus:

- Q, einer endlichen Menge von Zuständen
- Σ, einer endlichen Menge von Eingabesymbolen
- $\delta\colon Q \times (\Sigma \cup \{\epsilon\}) \to 2^Q$, einer Übergangsfunktion, wobei 2^Q die Potenzmenge von Q darstellt
- $s \in Q$, eines Startzustand
- $F \subseteq Q$, einer Menge von Endzuständen

Definition 2.6
Ein NEA erkennt oder akzeptiert eine Sprache L (über dem Alphabet des Automaten), wenn es genau dann eine Folge von Übergängen für ein Wort w gibt, die w akzeptiert, wenn w in L ist.

Definition 2.7
Zwei endliche Automaten heißen äquivalent, wenn sie dieselbe Sprache akzeptieren.

Satz 2.8 (Äquivalenz von NEA's und DEA's)
Zu jedem nichtdeterministischen endlichen Automaten gibt es einen äquivalenten deterministischen Automaten.

Verfahren 2.9 (Potenzmengenkonstruktion)
Wir konstruieren für jeden NEA $A := (Q, \Sigma, \delta, s, F)$ Einen DEA $\tilde{A} := (\tilde{Q}, \Sigma, \tilde{\delta}, \tilde{s}, \tilde{F})$ auf folgende Weise:

- definieren für jeden Zustand $q \in Q$ den ϵ-Abschluss
$$E(q) := \{p \in Q \mid p \text{ ist von } q \text{ durch } \epsilon\text{-Übergänge erreichbar}\}$$
- $\tilde{Q} = 2^Q$
- $\tilde{\delta} \colon \tilde{Q} \times \Sigma \to \tilde{Q}$ wobei $\tilde{\delta}(\tilde{q}, a)$ die Menge an Zuständen ist, die der NEA, startend von einem Zustand aus $\tilde{q}$ aus nach abarbeiten von beliebig vielen ϵ-Übergängen und dem Symbol a erreichen kann.
- $\tilde{s} := E(s)$
- $\tilde{F} := \left\{\tilde{q} \in \tilde{Q} \mid \tilde{q} \cap F \neq \{\}\right\}$

Satz 2.10 (NEA ohne ϵ-Übergänge)
Zu jedem NEA mit ϵ-Übergängen gibt es einen äquivalenten NEA ohne ϵ-Übergänge, der nicht mehr Zustände hat.

Verfahren 2.11 (Konstruktion für NEA ohne ϵ-Übergänge)
Für einen NEA $A := (Q, \Sigma, \delta, s, F)$ konstruieren wir einen NEA ohne ϵ-Übergänge $\tilde{A} := (\tilde{Q}, \Sigma, \tilde{\delta}, \tilde{s}, \tilde{F})$ auf folgende Weise:

- $\tilde{Q} := (Q \setminus F) \cup \tilde{F}$
- $\tilde{s} := s$
- $\tilde{\delta}(q, a)$ soll genau die Zustände enthalten, die A startend von q aus nach abarbeiten von beliebig vielen ϵ-Übergängen und dem Symbol a erreichen kann.
- $\tilde{F} := \{q \mid E(q) \cap F \neq \{\}\}$

Satz 2.12
Jede Sprache, die von einem endlichen Automaten erkannt wird, ist regulär.

Verfahren 2.13 (Konstruktion eines regulären Ausdrucks aus einem DEA)
Wir betrachten folgende Menge:

$L_{q_r,i,q_t} := \{w \in \Sigma^* \mid \text{Abarbeitung von } w \text{ führt von } q_r \text{ nach } q_t \text{ und hat nur Zwischenzustände } \{q_1, \dots, q_i\}\}$

Nun betrachten wir:

$$L_{q_r,0,q_t} = \begin{cases} \{a \in \Sigma \mid \delta(q_t, a) = q_t\} \cup \{\epsilon\} & \text{falls } r = t \\ \{a \in \Sigma \mid \delta(q_r, a) = q_t\} & \text{sonst} \end{cases}$$

Für $i > 0$ gilt dann:

$$L_{q_r,i+1,q_t} = L_{q_r,i,q_t} \cup (L_{q_r,i,q_{i+1}} \cdot (L_{q_{i+1},i,q_{i+1}})^* \cdot L_{q_{i+1},i,q_t})$$

Damit ist dann auch $L_f = L_{s,n,f}$ regulär für jedes $f \in F$.

Satz 2.14 (Pumping-Lemma für reguläre Sprachen)
Sei L eine reguläre Sprache. Dann existiert eine Zahl $n \in \mathbb{N}$, so dass für jedes Wort $w \in L$ mit $|w| > n$ eine Darstellung

$$w = uvx \text{ mit } |uv| \leq n, v \neq \epsilon,$$

existiert, bei der auch $uv^ix \in L$ ist für alle $i \in \mathbb{N}_0$.

Satz 2.15 (Verallgemeinertes Pumping-Lemma für reguläre Sprachen)
Sei L eine reguläre Sprache. Dann existiert eine Zahl $n \in \mathbb{N}$, so dass für jedes Wort $w \in L$ mit $|w| > n$ und jede Darstellung $w = tyx$ mit $|y| = n$ gilt:
Für das Teilwort y existiert eine Darstellug $y = uvz$ mit $v \neq \epsilon$ bei der auch $tuvzx \in L$ ist für alle $i \in \mathbb{N}_0$.

2.3 Minimierung von Automaten und Äquivalenzklassenautomaten

Definition 2.16 (Überflüssige Zustände)
Zustände eines (deterministischen) endlichen Automaten, die vom Anfangszustand aus nicht erreichbar sind, heißen überflüssig

Satz 2.17
Die Menge aller überflüssigen Zustände eines (deterministischen) endlichen Automaten kann in Zeit $O(|Q| \cdot |\Sigma)$ berechnet werden (DFS).

Definition 2.18 (Äquivalente Zustände)
Zwei Zustände p und q eines deterministischen endlichen Automaten heißen äquivalent($p \equiv q$), wenn für alle Wörter $w \in \Sigma^*$ gilt:

$$\delta(p, w) \in F \Leftrightarrow \delta(q, w) \in F$$

Mit $[q]$ bezeichnen wir die Äquivalenzklasse der zu q äquivalenten Zustände.

Verfahren 2.19 (Konstruktion des Äquivalenzklasseautomaten)
Zu einem DEA $A = (Q, \Sigma, \delta, s, F)$ definieren wir den Äquivalenzklasseautomaten $A^\equiv = (Q^\equiv, \Sigma^\equiv, \delta^\equiv, s^\equiv, F^\equiv)$ durch:

- $Q^\equiv := \{[q] \mid q \in Q\}$
- $\Sigma^\equiv := \Sigma$
- $\delta^\equiv([q], a) := [\delta(q, a)]$
- $s^\equiv := [s]$
- $F^\equiv := \{[f] \mid f \in F\}$

Um die Äquivalenzklassen zu bestimmen unterteile Q in immer kleinere Klassen von Zuständen, indem zuerst ein Zeuge für die nichtäquivalenz zweier Zustände der Länge 0, dann 1, dann $2\ldots$ gesucht wird. Das Verfahren bricht ab, wenn für Wörter der Länge k keine Zeugen mehr gefunden werden.
Die enstandenen Klassen sind die gesuchten Äquivalenzklassen.

Satz 2.20
Der Äquivalenzklasseautomat $A^{\equiv}$ zu A akzeptiert dieselbe Sprache wie A.

Satz 2.21
Der Äquivalenzklasseautomat $A^{\equiv}$ zu A ohne überflüssige Zustände ist minimal.

Definition 2.22 (Rechtsinvariante Äquivalenzrelation)
Eine Äquivalenzrelation R über Σ heißt rechtsinvariant, wenn für alle $x, y \in \Sigma^*$ gilt:

$$\text{falls } x\,R\,y \text{ so gilt auch } xz\,R\,yz \text{ für alle } z \in \Sigma^*$$

Der Index von R $ind(R)$ ist die Anzahl der Äquivalenzklassen von Σ^* bezüglich R.

Definition 2.23 (Nerode-Relation)
Für eine Sprache $L \subseteq \Sigma^*$ ist die Nerode-Relation R_L definiert durch:
für $x, y \in \Sigma^*$ ist $x\,R\,y$ genau dann, wenn $(xz \in L \Leftrightarrow yz \in L)$ für alle $z \in \Sigma^*$ gilt.
Die Nerode-Relation ist offensichtlich rechtsinvariant.

Satz 2.24 (von Nerode)
Die folgenden Aussagen sind äquivalent:

- $L \subseteq \Sigma^*$ wird von einem deterministischen endlichen Automaten erkannt.
- L ist die Vereinigung von (einigen) Äquivalenzklassen einer rechtsinvarianten Äquivalenzrelation mit endlichem Index.
- Die Nerode-Relation hat endlichen Index.

3 Turingmaschine und Berechenbarkeit

3.1 Die Turingmaschine

Definition 3.1 (Turingmaschine)
Eine deterministische Turingmaschine ((D)TM) besteht aus:

- Q, einer endlichen Zustandsmenge
- $\Sigma \cup \{\sqcup\}$, einem endlichen Eingabealphabet und Blanksymbol
- Γ, einem endlichen Bandalphabet mit $\Sigma \cup \{\sqcup\} \subseteq \Gamma$
- $s \in Q$, einem Startzustand
- $\delta\colon Q \times \Gamma \to Q \times \Gamma \times \{L, R, N\}$, einer Übergangsfunktion. Hierbei stehen L, R, N für eine Bewegung des Kopfes nach Links, Rechts oder ein Stehenbeleiben
- $F \subseteq Q$, einer Menge von Endzuständen. Diese Menge kann auch entfallen.
  Beachte: Die TM stopp, sobald sie in einen Endzustand gerät

Die TM stopp auch, wenn sie im Zustand q ein a ließt und $\delta(q, a) = (q, a, N)$ ist.

Beispiel 3.2
Der Übergang $\delta(q, a) = (p, b, L)$ wird graphisch wie folgt dargestellt:

$$\begin{tikzcd} q \arrow[r, "a|b,\ L"] & p \end{tikzcd}$$

Wenn für eine bestimmte Kombination q, a kein Übergang angegeben ist, stopp die TM in einem nicht akzeptierenden Zustand

Definition 3.3 (Definitionen zur Turingmaschine)

- Eine TM akzeptiert eine Eingabe $w \in \Sigma^*$, wenn sie nach Lesen von w in einem Zustand aus F stoppt
- Eine TM akzeptiert eine Sprache L genau dann, wenn sie ausschließlich Wörter $w \in L$ als Eingabe akzeptiert.
- Eine Sprache $L \subseteq \Sigma^*$ heißt rekursiv oder entscheidbar, wenn es eine TM gibt, die auf allen Eingaben stopp und eine Eingabe w genau dann akzeptiert, wenn $w \in L$ gilt
- Eine Sprache $L \subseteq \Sigma^*$ heißt rekursiv-aufzählbar oder semi-entscheidbar, wenn es eine TM gibt, die genau die Eingaben von w akzeptiert für die $w \in L$.
  Das Verhalten der TM für Eingaben $w \notin L$ ist damit nicht definiert.
- Eine Funktion $f\colon \Sigma^* \to \Gamma^*$ heißt berechenbar oder totalrekursiv, wenn es eine TM gibt, die bei Eingabe von $w \in \Sigma^*$ den Funktionswert $f(w) \in \Gamma^*$ ausgibt.
- Eine TM realisiert die Funktion $f\colon \Sigma^* \to \Gamma^*$, mit $f(w) =$ Ausgabe der TM, wenn sie stoppt und undefiniert sonst.

Die Chruch'sche These besagt, dass die Menge der berechenbaren Funktionen genau die Menge der im inuitiven Sinne überhaupt berechenbaren Funktionen ist.

Es gibt verschiedene Varianten der TM, die alle gleichmächtig zur TM sind:

- Mehrere Lese-/Schreibköpfe
- Mehrere Bänder
- Mehrere Lese-/Schreibköpfe für mehrere Bänder
- Mehrdimensionale Bänder

3.2 Die universelle Turingmaschine und unentscheidbare Probleme

Definition 3.4 (Gödelnummer)
Sei $M = (Q, \Sigma, \Gamma, \delta, s, F)$ eine Turingmaschine. Die Gödelnummer von M wird mit $\langle M \rangle$ bezeichnet. Es gilt $\langle M \rangle \in \{0, 1\}^*$ und $\langle M \rangle$ definiert M vollständig.
T_w sei die TM, mit der Gödelnummer w, beziehungsweise die TM, die $\{\}$ akzeptiert.

Definition 3.5 (universelle Turingmaschine)
Eine universelle Turingmaschine erhält als Eingabe ein Paar $(\langle M \rangle, w)$ und Simuliert M auf w.

Satz 3.6 (Unentscheidbarkeit der Diagonalsprache)
Die Diagonalsprache ist definiert durch:

$$L_d \coloneqq \{w_i \mid T_{w_i} \text{ akzeptiert } w_i \text{ nicht}\}$$

Die Diagonalsprache L_d ist nicht semi-entscheidbar.
L_d^c ist semi-entscheidbar.

Satz 3.7 (Unentscheidbarkeit des Halteproblem)
Das Halteproblem definiert folgende Sprache

$$\mathcal{H} \coloneqq \{wv \mid T_w \text{ hält auf der Eingabe } v\}$$

und ist semi-entscheidbar.
H^c ist nicht semi-entscheidbar.

Satz 3.8 (Unentscheidbarkeit der universellen Sprache)
Die universelle Sprache ist definiert durch

$$L_u \coloneqq \{wv \mid v \in L(T_w)\}$$

und ist nicht entscheidbar aber semi-entscheidbar.

Satz 3.9 (Satz von Rice)
Sei R die Menge der von Turingmaschinen berechenbaren Funktionen und S eine nichttriviale Teilmenge von R ($\{\} \subsetneq S \subsetneq R$). Dann ist die Sprache

$$L(S) \coloneqq \{\langle M \rangle \mid M \text{ berechnet eine Funktion aus } S\}$$

nicht entscheidbar.

Satz 3.10 (Eigenschaften von (semi-)entscheidbaren Sprachen)

- Die entscheidbaren Sprachen sind abgeschlossen unter Komplementbildung, Schnitt, Vereinigung und Mengendifferenz
- Die semi-entscheidbaren Sprachen sind abgeschlossen unter Schnitt und Vereinigung, aber nicht unter Komplementbildung oder Mengendifferenz
- L und L^c semi-entscheidbar genau dann, wenn L entscheidbar

Problem 3.11 (Post'sche Korrespondenzproblem)
Gegeben ist eine endliche Folge von Wortpaaren

$$K = ((x_1, y_1), \ldots, (x_n, y_n))$$

über einem endlichen Aplhabet Σ. Es gilt $x_i \neq \epsilon$ und $y_i \neq \epsilon$. Gefragt ist, ob eine endliche Folge von Indizes $i_1, \ldots, i_k \in \{1, \ldots, n\}$ existiert, so dass $x_{i_1} \ldots x_{i_k} = y_{i_1} \ldots y_{i_k}$ gilt.
Dieses Problem ist nicht entscheidbar.

4 Komplexitätsklassen

4.1 Zeitkomplexität

Definition 4.1 (Zeitkomplexitätsfunktion eine deterministischen Turingmaschine)
Für eine deterministische Turingmaschine M, die für alle Eingaben über Σ hält, ist die Zeitkomplexitätsfunktion $T_M\colon \mathbb{Z}^+ \to \mathbb{Z}^+$ definiert durch:

$$T_M(n) = \max\left\{ m \;\middle|\; \begin{array}{c} \text{es gibt eine Eingabe } x \in \Sigma^* \text{ mit } |x| = n \text{, so dass } M\ m \\ \text{Berechnungsschritte benötigt, um in einen Endzustand zu gelangen} \end{array} \right\}$$

Definition 4.2 (Die Klasse P)
Die Klasse P ist die Menge aller Sprachen, für L (Probleme) für die es eine deterministische TM gibt, deren Zeitkomplexitätsfunktion polynomial ist. Das heißt es existiert ein Polynom p mit

$$T_M(n) \leq p(n).$$

4.2 Die Nichtdeterministische Turingmaschine und die Klasse NP

Definition 4.3 (Nichtdeterministische Turingmaschine)
Die nichtdeterministische Turingmaschine (NTM) wird analog zur TM definiert hat aber zusätzlich zu der endlichen Kontrolle mit Lese-/Schreibkopf ein Orakelmodul mit eigenem Schreibkopf.
Bei der Berechnung der DTM schreibt das Orakelmodul zuerst ein beliebiges Wort aus Γ^* auf das Band und gibt anschließend die Kontrolle an den deterministischen Teil über.

Definition 4.4
Eine NTM akzeptiert ein Wort $w \in \Sigma^*$ genau dann, wenn es eine Berechnung gibt, die in einem akzeptierenden Zustand endet.
Sie akzeptiert eine Sprachen $L \subseteq \Sigma^*$ genau dann, wenn sie grade die Wörter aus L akzeptiert.

Definition 4.5 (Zeitkomplexitätsfunktion eine nichtdeterministischen Turingmaschine)
Die Zeitkomplexitätsfunktion einer nichtdeterministische Turingmaschine M $T_M\colon \mathbb{Z}^+ \to \mathbb{Z}^+$ ist definiert durch:

$$T_M(n) = \max\left(\{1\} \cup \left\{ m \;\middle|\; \begin{array}{c} \text{es gibt eine Eingabe } x \in \Sigma^* \text{ mit } |x| = n \text{, so dass } M\ m \\ \text{Berechnungsschritte benötigt, um } x \text{ zu akzeptieren} \end{array} \right\}\right)$$

Zur Berechung von $T_M(n)$ wird für jedes $x \in L_M$ mit $|x| = n$ die kürzeste akzeptierende Berechnung betrachtet und unter diesen x dann die längste Berechnung bestimmt. Die Schreibzeit des Orakelmodul spielt mit in die gesamte Berechnungszeit ein.
Die Zeitkomplexität hängt nur von der Anzahl der Schritte bei einer akzeptierenden Berechnung ab.
Bemerkung vom Autor: das ist ziemlich obskur definiert.

Definition 4.6 (Die Klasse NP)
Die Klasse NP ist die Menge aller Sprachen (Probleme) L, für die es eine nichtdeterministische Turingmaschine gibt, deren Zeitkomplexitätsfunktion polynomial beschränkt ist.
Alle Sprachen in NP sind entscheidbar.

Definition 4.7 (polynomiale Transformation)
Eine polynomiale Transformation einer Sprache $L_1 \subseteq \Sigma_1^*$ in eine Sprache $L_2 \subseteq \Sigma_2^*$ ist eine Funktion $f\colon \Sigma_1^* \to \Sigma_2^*$ mit den Eigenschaften:

- es existiert eine polynomiale TM, die f berechnet
- für alle $x \in \Sigma_1^*$ gilt: $x \in L_1 \Leftrightarrow f(x) \in L_2$

Wir schreiben dann $L_1 \propto L_2$.

Definition 4.8 (NP-vollständig)
Eine Sprache L heißt NP-vollständig, wenn $L \in NP$ und für alle $L' \in NP$ gilt $L' \propto L$.

Satz 4.9
Sei L NP-vollständig, dann gilt:

- $L \in P \Rightarrow P = NP$
- $L \notin P$, so gillt für alle NP-vollständigen Sprachen L', dass $L' \notin P$.

Definition 4.10 (NP-schwer)
Ein Suchproblem Π heißt NP-schwer, falle es eine NP-vollständige Sprache L gibt, mit $L \propto_T \Pi$. Also kann mit einem Algorithmus für Π auch L gelöst werden.

4.3 Komplementsprachen

Definition 4.11
Die Klasse NPC sei die Klasse der NP-vollständigen Sprachen/Probleme.
Die Klasse NPI ist definiert durch $NPI := NP \setminus (P \cup NPC)$.
Die Klasse $co - P$ ist die Klasse aller Sprachen $\Sigma^* \setminus L$ für $L \subseteq \Sigma^*$ und $L \in P$ (die Klasse der Komplementsprachen).
Die Klasse $co - NP$ ist die Klasse aller Sprachen $\Sigma^* \setminus L$ für $L \subseteq \Sigma^*$ und $L \in NP$

4.4 Pseudopolynomielle Algorithmen

Definition 4.12 (Pseudopolynomiell)
Sei Π ein Optimierungsproblem. Ein Algorithmus, der Π löst, heißt pseudopolynomiell, falls seine Laufzeit durch ein Polynom der Eingabegröße und der Größe der größten in der Eingabe vorkommenden Zahl beschränkt ist.

Definition 4.13 (stark NP-vollständig)
Ein Entscheidungsproblem Π heißt stark NP-vollständig, wenn das Teilproblem Π_p, in dem nur Instanzen vorkommen, bei denen die größte Zahl polynomiell durch die Eingabelänge beschränkt ist, NP-vollständig ist.

Satz 4.14
Ist Π stark NP-vollständig und $NP \neq P$, dann gibt es keinen pseudopolynomiellen Algorithmus für Π.

4.5 Approximationsalgorithmen für Optimierungsprobleme

Definition 4.15
Für $I \in D_\Pi$ bezeichne $OPT(I)$ den Wert der Optimallösung. Zu einem Algorithmus A bezeichne $A(I)$ den Wert, den A für I ausgibt.

Definition 4.16 (Absoluter Approximationsalgorithmus)
Sei Π ein Optimierungsproblem. Ein polynomialer Algorithmus A, der für jedes $I \in D_\Pi$ einen Wert $A(I)$ liefert, mit

$$|OPT(I) - A(I)| \leq K$$

für $K \in \mathbb{N}_0$ konstant, heißt absoluter Approximationsalgorithmus.
Es gibt nur wenige NP-schwere Probleme, für die diese Art von Algorithmen bekannt sind.

Definition 4.17 (Approximationsalgorithmus mit relativer Gütegarantie)
Sei Π ein Optimierungsproblem. Ein polynomialer Algorithmus A, der für jedes $I \in D_\Pi$ einen Wert $A(I)$ liefert, mit $R_A(I) \leq K$, wobei $K \geq 1$ eine Konstante, und

$$R_A(I) \coloneqq \begin{cases} \frac{A(I)}{OPT(I)} & \text{falls } \Pi \text{ Minimierungsproblem} \\ \\ \frac{OPT(I)}{A(I)} & \text{falls } \Pi \text{ Maximierungsproblem} \end{cases}$$

heißt Approximationsalgorithmus mit relativer Gütegarantie.

Definition 4.18
Zu einem polynomialen Approximationysalgorithmus A sei

$$R_A^\infty \coloneqq \{r \geq 1 \mid \text{es gibt ein } N_0 > 0, \text{ so dass } R_A(I) \leq r \text{ für alle } I \text{ mit } OPT(I) \geq N_0\}$$

Definition 4.19 (Approximationsschemata)
Ein (polynomiales) Approximationsschemata (PAS) für ein Optimierungsproblem Π ist eine Familie von Algorithmen $\{A_\epsilon \mid \epsilon > 0\}$, so dass für alle $\epsilon > 0$

- $R_{A_\epsilon} \leq 1 + \epsilon$ ist (d.h. A_ϵ ist ein ϵ-approximierender Algorithmus).
- A_ϵ polynomial in der Größe des Inputs ist.

Ein Approximationsschema $\{A_\epsilon \mid \epsilon > 0\}$ heißt vollpolynomial (FPAS) falls seine Laufzeit zudem polynomial in $\frac{1}{\epsilon}$ ist.

Satz 4.20
Sei Π ein NP-schweres Optimierungsproblem mit:

- $OPT(I) \in \mathbb{N}$ für alle $I \in D_\Pi$
- es existiert ein Polynom q mit $OPT(I) < q(\langle I \rangle)$ für alle $I \in D_\Pi$ wobei $\langle I \rangle$ die Inputlänge von I ist.

Falls $P \neq NP$, so gibt es kein FPAS $\{A_\epsilon \mid \epsilon > 0\}$ für Π.

Satz 4.21
Sei Π ein Optimierungsproblem und $\max(I)$ die größte in Π vorkommende Zahl. Wenn gilt:

- $OPT(I) \in \mathbb{N}$ für alle $I \in D_\Pi$
- es existiert ein Polynom q mit $OPT(I) \leq q(\langle I \rangle, \max(I))$

Dann hat Π genau dann ein FPAS, wenn es zu Π einen pseudopolynomialen optimalen Algorithmus gibt

5 Grammatiken und die Chomsky-Hierarchie

Definition 5.1 (Grammatik)
Eine Grammatik $G = (\Sigma, V, S, R)$ besteht auf 4 Komponenten:

- Endliches Alphabet Σ
- Endliche Menge V mit $V \cap \Sigma = \{\}$ von Variablen
- Startsymbol $S \in V$
- Endliche Menge von Ableitungsregeln R. Dabei ist eine Ableitungsregel ein Paar

  $$(l, r) \in (V \cup \Sigma)^+ \times (V \cup \Sigma)^*.$$

  Wir schreiben auch $l \to r$.

$L(G)$ bezeichnet die von G erzeugte Sprache $\{w \in \Sigma^* \mid S \to^* w\}$.

Definition 5.2 (Chomsky-Hierarchie)
Typ 0 Grammatiken ohne weitere Einschränkungen.

Typ 1 oder kontextsensitiv Grammatiken, bei denen alle Ableitungsregeln die Form habe:

- $u \to v$ mit $u \in V^+, v \in ((V \cup \Sigma) \setminus \{S\})^+$ und $|u| \leq |v|$, oder
- $S \to \epsilon$

Typ 2 oder kontextfrei Grammatiken, bei denen alle Ableitungsregeln die Form haben:

- $A \to v$ mit $A \in V$ und $v \in (V \cup \Sigma)^*$

Typ 3 oder rechtslinear Grammatiken, bei denen alle Ableitungsregeln die Form haben:

- $A \to v$ mit $A \in V$ und $v = \epsilon$ oder $v = aB$ mit $a \in \Sigma, B \in V$

Satz 5.3
Die Chomsky-Hierarchie ist echt. Also $L_3 \subset L_2 \subset L_1 \subset L_0$, wobei L_i Klasse der durch Typ-i-Grammatiken erzeugten Sprachen.

5.1 Chomsky-0-Grammatiken und rekursiv aufzählbare Sprachen

Satz 5.4
Falls L rekursiv-aufzählbar (semi-entscheidbar) ist, so gibt es eine Chomsky-0-Grammatik G mit $L(G) = L$.

Satz 5.5
Die von Typ-0-Grammatiken G erzeugten Sprachen sind rekursiv-aufzählbar.

5.2 Chomsky-1-Grammatiken bzw. kontextsensitive Sprachen

Definition 5.6 (DTAPE und NTAPE)
$DTAPE(s(n))$ ist die Klasse der Sprachen L, für die es eine DTM mit Platzbedarf $s(n)$ (bei Eingabelänge n) gibt, die L akzeptiert.
$NTAPE(s(n))$ ist die Klasse der Sprachen L, für die es eine NTM mit Platzbedarf $s(n)$ (bei Eingabelänge n) gibt, die L akzeptiert.
Es gilt $NTAPE(n) = NTAPE(f(n))$ für eine lineare Funktion f. Es ist offen, ob $NTAPE(n) = DTAPE(n)$.

Satz 5.7
Die Klasse der von Chomsky-1-Grammatiken erzeugten Sprachen stimmt mit der Klasse $NTAPE(n)$ überein.

5.3 Chomsky-2-Grammatiken bzw. kontextfreie Sprachen und Syntaxbäume

Definition 5.8 (eindeutige Grammatik)
Eine kontextfreie Grammatik G heißt eindeutig, wenn es für jedes Wort $w \in L(G)$ genau einen Syntaxbaum gibt. Eine kontextfreie Sprache L heißt eindeutig, wenn es eine eindeutige Grammatik G mit $L(G) = G$ gibt. Ansonsten heißt L inhärent mehrdeutig.
Bemerke: Zu jedem Syntaxbaum gehören mehrere Ableitungen zu jeder Ableitung aber nur ein Syntaxbaum.

Definition 5.9 (Chromsky-Normalform)
Eine kontextfreie Grammatik ist in Chromsky-Normalform, wenn alle Regeln von der Form:

- $A \to BC$ oder
- $A \to a$

sind, mit $A, B, C \in V$ und $a \in \Sigma$.
Für kontextfreie Sprachen, die ϵ enthalten, lässt sich die Grammatik durch $S' \to \epsilon$ und $S' \to S$ ergänzen. Dies wird als erweiterte Chomsky-Normalform bezeichnet.
Jede kontextfreie Grammatik kann in eine Grammatik in erweiterte Chomsky-Normalform überführt werden.

Verfahren 5.10 (Umwandlung in Chomsky-Normalform)
Schritt 1: Ersetze in allen Regeln, bei denen auf der rechten Seite ein Terminalsymbol $a \in \Sigma$ nicht alleine steht durch $Y_a \in V$ und füge die Regel $Y_a \to a$ hinzu.

Schritt 2: Ersetze Regeln der Art $A \to B_1 \dots B_m$ mit $m > 2$ durch die Regeln.

$$\begin{aligned} A &\to B_1 C_1 \\ C_i &\to B_{i+1} C_{i+1} \text{ für } 1 \le i \le m-3 \\ C_{m-2} &\to B_{m-1} B_m \end{aligned}$$

Schritt 3: Finde die Menge V' aller Variablen A für die gilt $A \to^* \epsilon$. Streiche alle Regeln $A \to \epsilon$. Für $A \to BC$ wird die Regel $A \to C$ hinzugefügt, falls $B \in V'$ (analog für $C \in V'$).

Schritt 4: Zur Ersetzung von Kettenregeln $A \to B$. Finde zuerst alle Kreise

$$A_1 \to A_2 \to \dots \to A_r \to A_1$$

und ersetzte alle A_i durch A_1.
Für Regel $A \to B$ und jede Regeln $B \to b$ füge Regel $A \to b$ hinzu und lösche $A \to B$.

Satz 5.11 (Cocke-Younger-Kasami Algorithmus (CYK Algorithmus))
Das Wortproblem zu kontextfreien Grammatiken in Chomsky-Normalform kann in polynomieller Zeit gelöst werden (Abhängig von Wortgröße und Anzahl der Ableitungsschritte).

Verfahren 5.12 (Cocke-Younger-Kasami Algorithmus (CYK Algorithmus))
Es wird dynamische Programmierung verwendet.
Sei $w = w_1 \dots w_n$ und $V_{ij} \subseteq V$ die Menge an Variablen A, so dass $A \to^* w_i \dots w_j$ impliziert. Baue eine Tabelle mit wachsendem $l := j - i$ auf beginnend mit $l = 0$.
Für $l = 0$ ist $V_{ii} = \{A \mid a \to w_i\}$.
Für $l > 0$ muss das zu untersuchende Wort $w_i \dots w_j$ für ein k mit $i \le k < j$ aufgeteilt werden in $w_i \dots w_k$ und $w_{k+1} \dots w_j$. Anschließend wird untersucht ob eine Ableitungsregeln $A \to BC$ existiert mit $B \to^* w_i \dots w_k$ und $C \to^* w_{k+1} \dots w_j$.

Satz 5.13 (Pumping-Lemma für kontextfreie Sprachen)
Für jede kontextfreie Sprache L gibt es eine Konstante $n \in \mathbb{N}$, so dass sich jedes Wort $z \in L$ mit $|z| \ge n$ so als $z = uvwxy$ schreiben lässt, dass $|vx| \ge 1, |vwx| \le n$ und für alle $I \ge 1$ das Wort $uvwxy \in L$ ist.

Satz 5.14 (Ogden's Lemma)
Für jede kontextfreie Sprache L gibt es eine Konstante $n \in \mathbb{N}$, so dass sich jedes Wort $z \in L$ mit $|z| \ge n$ gilt:
Wenn wir in z mindestens n Buchstaben markieren, so lässt sich z so als $z = uvwxy$ schreiben, dass von den mindestens n markierten Buchstaben mindestens einer zu vx und höchstens n zu vwx gehören und für alle $I \ge 1$ das Wort $uvwxy \in L$ ist.

Definition 5.15 (Nutzlose Variablen)
Sei G eine kontextfreie Grammatik. Eine Variable A heißt nutzlos, falls es keine Ableitung $S \to^* w$ gibt, mit $w \in \Sigma^*$, in der A vorkommt.

Satz 5.16
Für eine kontextfreie Grammatik kann die Menge der nutzlosen Variable in polynomialer Laufzeit berechnet werden.

Verfahren 5.17 (Bestimmen von nutzlose Variablen)
Berechne zuerst $V' \subseteq V$, mit $A \in V'$ genau dann, wenn es $w \in \Sigma^*$ gibt mit $A \to^* w$
Berechne anschließend $V'' \subseteq V'$ aller $A \in V'$, für die es eie Ableitung $S \to^* \alpha A \beta$ mit $\alpha\beta \in (V' \cup \Sigma)^*$ oder $A = S$.
V'' ist die Menge der nützlichen Variablen.

Satz 5.18
Für eine kontextfreie Grammatik G kann in polynomialer Zeit berechnet werden, ob $L(G) = \{\}$ oder $L(G)$ endlich.

Satz 5.19
Die Klasse der kontextfreie Sprachen ist abgeschlossen bzgl. Vereinigung, Konkatenation und Kleenschen Abschluss
Die Klasse der kontextfreie Sprachen ist nicht abgeschlossen bzgl. Komplementbildung und Durchschnitt

Definition 5.20 (Greibach-Normalform)
Eine kontexfreie Grammatik G ist in Greibach-Normalform, wenn allle Ableitungsregeln der Form

$$A \to a\alpha \text{ mit } A \in V, a \in \Sigma \text{ und } \alpha \in V^*$$

sind.
Jede kontexfreie Grammatik G, für die $L(G)$ das leere Wort nicht enthält, kann in eine Greibach-Normalform überführt werden.

Definition 5.21 (Kellerautomat(NPDA bzw. PDA))
Ein (nichtdeterministischer) Kellerautomat besteht aus $(Q, \Sigma, \Gamma, q_0, Z_0, \delta, F)$,wobei

- Q endliche Zustandsmenge
- Σ endliches Eingabealphabet
- Γ endliches STACK-Alphabet
- $q_0 \in Q$ Anfangszustand
- $Z_0\Gamma$ Initialisierung des STACK
- $\delta\colon Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \to 2^{Q \times \Gamma^*}$, d.h. $\delta(q, a, Z) \subseteq \{(q, \gamma) \mid q \in Q, \gamma \in \Gamma^*\}$

Ein PDA akzeptiert ein $w \in \Sigma^*$ durch leeren Stack, wenn es eine zulässige Folge von Konfigurationen aus der Anfangskonfiguration (q_0, w, Z_0) in eine Endkonfiguration (q, ϵ, ϵ) gibt.
Ein PDA akzeptiert ein $w \in \Sigma^*$ durch einen akzeptierenden Endzustand, wenn es eine zulässige Folge von Konfigurationen aus der Anfangskonfiguration (q_0, w, Z_0) in eine Konfiguration (q, ϵ, γ) mit $q \in F$ und $\gamma \in \Gamma^*$ gibt.

Satz 5.22 (Äquivalenz des Akzeptanzverhalten von PDAs)
Zu einem PDA, der eine Sprache L durch einen akzeptierenden Endzustand akzeptiert, kann ein PDA konstruiert werden, der L mit leerem STACK akzeptiert und umgekehrt.

Satz 5.23
Die klasse der von nichtdeterministischen Kellerautomaten akzeptierten Sprachen ist gleich der Klasse der kontextfreien Sprachen.

5.4 Chomsky-3-Grammatiken und reguläre Sprachen

Satz 5.24
Die Klasse der von endlichen Automaten akzeptieren Sprachen ist genau die Klasse von Chomsky-3-Grammatiken erzeugten Sprachen

5.5 Unentscheidbare Probleme für kontextfreie Grammatiken

Satz 5.25
Das Problem für kontextfreie Grammatiken G_1 und G_2 zu entscheiden, ob $L(G_1 \cap G_2 = \{\}$ ist, ist nicht entscheidbar.

Satz 5.26
Das Problem für eine kontextfreie Grammatiken G_1 zu entscheiden, ob sie eindeutig ist, ist nicht entscheidbar.

Definition 5.27 (Sprache der korrekten Rechenwege einer TM)
Die Sprache B_M der korrekten Rechenwege einer TM M besteht aus allen Worten

$$w_1 \# w_2^R \# w_3 \# w_4^R \dots w_n^R \#, \text{ falls } n \text{ gerade und}$$

$$w_1 \# w_2^R \# w_3 \# w_4^R \dots w_n \#, \text{ falls } n \text{ ungerade}$$

wobei die w_i aufeinanderfolgende Konfigurationen von M sind.

Satz 5.28
Für alle Turingmaschinen M ist B_M der Durchschnitt zweier Sprachen $L_1 = L(G_1)$ und $L_2 = L(G_2)$, wobei G_1 und G_2 kontextfreie Grammatiken sind.

5.6 Zusammenfassung

Typ	$w \in L(G)$	$L(G) = \{\}$	$L(G)$ endlich	$L(G_1) = L(G_2)$	$L(G_1) \cap L(G_2) = \{\}$
0	✗	✗	✗	✗	✗
1	✓	✗	✗	✗	✗
2	✓	✓	✓	✗	✗
3	✓	✓	✓	✓	✓

Tabelle 1: Entscheidbarkeit für Grammatiken der Chomsky-Hierarchie

Typ	$L(G)^c$	$L(G_1) \cdot L(G_2)$	$L(G_1) \cup L(G_2)$	$L(G_1) \cap L(G_2)$	$L(G)^*$
0	✗	✓	✓	✓	✓
1	✓	✓	✓	✓	✓
2	✗	✓	✓	✗	✓
3	✓	✓	✓	✓	✓

Tabelle 2: Abgeschlossenheit für Grammatiken der Chomsky-Hierarchie

6 Informationstheorie

6.1 Quellkodierung

Wir betrachten $\Sigma = \{1, \ldots, n\}$ eine Menge von Zeichen mit Wahrscheinlichkeit $\{p_1, \ldots, p_n\}$. X (Zufallsvariable/Informationsquelle) liefert $i \in \Sigma$ mit Wahrscheinlichkeit p_i.

Definition 6.1 (Information)
Die Information einer Wahrscheinlichkeit p ist

$$I \coloneqq \log_2\left(\frac{1}{p}\right) = -\log_2(p).$$

Definition 6.2 (Entropie)
Die Entropie einer diskreten Zufallsvariable X mit Wahrscheinlichkeiten $p(x)$ für $x \in X$ ist definiert durch

$$H(X) \coloneqq \sum_{x \in X} p(x) \log_2\left(\frac{1}{p(x)}\right)$$

7 Probleme

7.1 Definitionen und Begriffe

Definition 7.1 (Problemvarianten)
Zu den meisten Problemen gibt es verschiedene Problemvarianten.

- Optimierungsproblem: Gesucht ist die optimale Lösung eines gegebenen Problems.
- Optimalwertproblem: Gesucht ist der Wert der optimalen Lösung.
- Entscheidungsproblem: Es soll entschieden Werden, ob es Lösungen gibt, die besser als ein gegebener Wert k sind.

Definition 7.2 (Kodierungsschema)
Ein Kodierungsschema ordnet jeder Probleminstanz eines Problems eine Zeichenkette über dem Alphabet Σ zu. Die Inputlänge eines Problembeispiels ist die Anzahl der Symbole der Kodierung.

Definition 7.3 (äquivalente Kodierungsschemata)
Zwei Kodierungsschemata heißen äquivalent bezüglich eines Problems Π, wenn sie sich gegenseitig durch Polynome abschätzen lassen für alle Instanzen von Π.

Definition 7.4
Zu einem Entscheidungsproblem Π bezeichnen wird die Menge an Probleminstanzen als D_Π, die Teilmenge der Ja-Instanzen mit $J_\Pi \subseteq D_\Pi$ und der Nein-Instanzen mit $N_\Pi \subseteq D_\Pi$.

Definition 7.5
Die zu einem Problem Π und einem Kodierungsschema s zugehörige Sprache ist

$$L[\Pi, s] \coloneqq \{x \in \Sigma^* \mid x \text{ ist Kodierung einer Ja-Instanz von } \Pi \text{ unter } s\}$$

Eine deterministische Turingmaschine M löst ein Entscheidungsproblem Π unter der Kodierung s, wenn M für jede Eingabe hält und $L_M = L[\Pi, s]$.

Definition 7.6 (Suchproblem)
Ein Suchproblem Π wird beschrieben durch eine Menge von Probleminstanzen D_Π und für $I \in D_\Pi$ die Menge $S_\Pi(I)$ aller Lösungen von I.
Die Lösung eines Suchproblems besteht in der Angabe einer Lösung aus $S_\Pi(I)$.

Definition 7.7 (Aufzählungsproblem)
Ein Aufzählungsproblem Π wird beschrieben durch eine Menge von Probleminstanzen D_Π und für $I \in D_\Pi$ die Menge $S_\Pi(I)$ aller Lösungen von I.
Die Lösung eines Aufzählungsproblem besteht in der Angabe der Kardinalität von $S_\Pi(I)$ also $|S_\Pi(I)|$.

Definition 7.8 (Reduzierbarkeit für Suchprobleme)
Zu einem Suchproblem Π sei die Relation R_Π gegeben:

$$R_\Pi := \{(x, c) \mid x \in D_\Pi, s \in S_\Pi(x)\}$$

Eine Funktion $f\colon \Sigma^* \to \Sigma^*$ realisiert eine Relation R, wenn für alle $x \in \Sigma^*$ gilt:

$$f(x) = \begin{cases} \epsilon & \nexists y \in \Sigma^* \setminus \{\epsilon\} : (x, y) \in R \\ y & \text{sonst, mit beliebigem } y\colon (x, y) \in R \end{cases}$$

Ein Algorithmus löst das durch R_Π beschriebene Suchproblem Π, wenn er eine Funktion berechnet, die R_Π berechnet.

7.2 Beispiele

Problem 7.9 (Post'sche Korrespondenzproblem)
Gegeben ist eine endliche Folge von Wortpaaren

$$K = ((x_1, y_1), \ldots, (x_n, y_n))$$

über einem endlichen Alphabet Σ. Es gilt $x_i \neq \epsilon$ und $y_i \neq \epsilon$.
Gesucht ist, eine endliche Folge von Indizes $i_1, \ldots, i_k \in \{1, \ldots, n\}$, so dass $x_{i_1} \ldots x_{i_k} = y_{i_1} \ldots y_{i_k}$ gilt.
Dieses Problem ist nicht entscheidbar.

Problem 7.10 (Traveling Salesman Problem (TSP))
Gegeben sei ein vollständiger Graph $G = (V, E)$, sowie eine Kostenfunktion $c\colon E \to \mathbb{Z}^+$.
Gesucht ist eine Tour, die alle Elemente aus V enthält und minimale Gesamtlänge hat.
Das TSP ist NP-vollständig
Für das TSP mit Dreiecksungleichung existiert ein Approximationsalgorithmus A mit $R_A \leq 2$ für alle Instanzen I.

Problem 7.11 (SAT)
Gegeben ist eine Menge U von Variablen, und eine Menge C von Klauseln über U.
Gesucht ist eine Wahrheitsbelegung von U, so dass C erfüllt wird. Also in jeder Klausel mindestens eine Variable Wahr ist.
SAT ist ist NP-vollständig.

Problem 7.12 (3SAT)
Gegeben ist eine Menge U von Variablen, und eine Menge C von Klauseln über U, wobei jede Klausel genau drei Literale enthält.
Gesucht ist eine Wahrheitsbelegung von U, so dass C erfüllt wird. Also in jeder Klausel mindestens eine Variable Wahr ist.
3SAT ist ist NP-vollständig.

Problem 7.13 (2SAT)
Gegeben ist eine Menge U von Variablen, und eine Menge C von Klauseln über U, wobei jede Klausel genau zwei Literale enthält.
Gesucht ist eine Wahrheitsbelegung von U, so dass C erfüllt wird. Also in jeder Klausel mindestens eine Variable Wahr ist.
Im Gegensatz zu 3SAT ist 2SAT$\in P$.

Problem 7.14 (Max2SAT)
Gegeben ist eine Menge U von Variablen, und eine Menge C von Klauseln über U, wobei jede Klausel genau zwei Literale enthält und eine Zahl $K \in \mathbb{N}$
Gesucht ist eine Wahrheitsbelegung von U, so dass mindestens K Klauseln aus C erfüllt wird.
Max2SAT ist NP-vollständig.

Problem 7.15 (CLIQUE)
Gegeben ist ein Graph $G = (V, E)$ und ein Parameter $K \leq |V|$
Gesucht ist eine Clique $V' \subseteq V$ der Größe K, sodass für alle $i, j \in V', i \neq j$ gilt: $\{i, j\} \in E$. CLIQUE ist NP-vollständig.
Falls $P \neq NP$ existiert kein absoluter Approximationsalgorithmus für CLIQUE.

Problem 7.16 (COLOR)
Gegeben ist ein Graph $G = (V, E)$ und ein Parameter $K \in \mathbb{N}$
Gesucht ist eine Knotenfärbung von G mit höchstens K Farben, so dass je zwei adjazente Knoten verschiedene Farben besitzen.
3COLOR bezeichnet das Problem COLOR mit festem $K = 3$. 3COLOR ist NP-vollständig.
Falls $P \neq NP$, dann existiert kein relativer Approximationsalgorithmus A für COLOR mit $R_A^\infty < \frac{4}{3}$.

Problem 7.17 (SUBSET SUM)
Gegeben ist eine endliche Menge M, eine Gewichtsfunktion $w\colon M \to \mathbb{N}_0$ und $K \in \mathbb{N}_0$.
Gesucht ist eine Teilmenge $M' \subseteq M$ mit

$$\sum_{a \in M'} w(a) = K.$$

SUBSET SUM ist NP-vollständig.

Problem 7.18 (PARTITION)
Gegeben ist eine endliche Menge M und eine Gewichtsfunktion $w\colon M \to \mathbb{N}_0$.
Gesucht ist eine Teilmenge $M' \subseteq M$ mit

$$\sum_{a \in M'} w(a) = \sum_{a \in M \setminus M'} w(a)$$

PARTITION ist NP-vollständig.

Problem 7.19 (KNAPSACK)
Gegeben ist eine endliche Menge M, eine Gewichtsfunktion $w\colon M \to \mathbb{N}$ und eine Kostenfunktion $c\colon M \to \mathbb{N}_0$ und $W, C \in \mathbb{N}_0$.
Gesucht ist eine Teilmenge $M' \subseteq M$ mit

$$\sum_{a \in M'} w(a) \leq W \text{ und } \sum_{a \in M'} c(a) \geq C$$

KNAPSACK ist NP-vollständig.
Falls $P \neq NP$ existiert kein absoluter Approximationsalgorithmus für KNAPSACK. Es gibt einen einfachen Greedy-Algorithmus A, mit $R_A(I) \leq 2$ für alle Instanzen I.
Für KNAPSACK existiert ein FPAS. Es verwendet dynamische Programmierung mit skalierten Eingabegrößen.

Problem 7.20 (Hamilton-Kreis)
Gegeben ist ein ungerichteter, ungewichteter Graph $G = (V, E)$.
Gesucht ist ein Hamilton-Kreis in G. Ein Hamilton-Kreis ist eine Permutation π auf V, so dass $\{\pi(n), \pi(1)\} \in E$ und $\{\pi(i), \pi(i-1)\} \in E$ für $1 \leq i \leq n-1$.

Problem 7.21 (Subgraphisomorphie)
Gegeben sind Graphen G und H wobei H weniger Knoten als G besitzt.
Es wird danach gefragt, ob H isomorph zu einem Subgraphen von G ist.
Subgraphisomorphie ist NP-vollständig.

Problem 7.22 (Graphisomorphie)
Gegeben sind Graphen G und H mit gleich vielen Knoten.
Es wird danach gefragt, ob G und H zueinander isomorph sind.
Graphisomorphie ist ein Kandidat für ein Problem aus NPI. Vielleicht ist es auch in P.

„Da ich den kürzesten Weg durch die Vorlesung gefunden habe, sind wir jetzt schon fertig.“