1 Wichtige Definitionen, Sätze und Lemmas aus Kapitel 1

Alphabet, Wort, Konkatenation, Sprache, Leere Sprache,

Definition 1.1 Seien Σ_1 und Σ_2 zwei Alphabete. Eine Substitution von Σ_1 nach Σ_2 ist eine Abbildung $\delta : \Sigma_1 \to 2^{\Sigma_1^*}$ mit folgenden Eigenschaften:

- $\delta(\lambda) = \{\lambda\}$
- $\forall a \in \Sigma_1 : \delta(a)$ ist eine Sprache über Σ_2
- $\forall x, y \in \Sigma_1^* : \delta(xy) = \delta(x)\delta(y)$

Definition 1.2 Ein Entscheidungsproblem: Für ein geeignetes Alphabet Σ und gegebene Sprache $L \subseteq \Sigma^*$ ist zu entscheiden, ob $x \in L$ oder $x \notin L$.

Ein Algorithmus A löst das Entscheidungsproblem (L, Σ), wenn für alle $x \in \Sigma^*$ $A(x) = 1$, falls $x \in L$, und 0 sonst.

Wir sagen auch A erkennt L.

Definition 1.3 Ein Optimierungsproblem ist ein 6-Tupel:

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

wobei

Σ_I ist ein Alphabet (Eingabe-Alphabet)

Σ_O ist ein Alphabet (Ausgabe-Alphabet)

$L \subseteq \Sigma_I$ ist die Sprache der zulässigen Eingaben. Als zulässige Eingaben kommen nur Wörter, die eine sinnvolle Bedeutung haben, Ein $x \in L$ ist eine Problemfall von U.

$\mathcal{M}$ ist eine Funktion von L nach ${\Sigma_\delta}^*$, und für jedes $x \in L$ ist $\mathcal{M}(x)$ die Menge der zulässigen Lösungen für x.

$cost$ ist eine Funktion, $cost \bigcup_{x \in L} \mathcal{M}(x) \times L \to R^+$, Preis-Funktion genannt.

$goal \in \{Minimum, Maximum\}$

Hierzu das Beispiel mit dem TSP.

Definition 1.4 Die Kolmogorrov Komplexität eines Wortes x K(x) ist die binäre Länge des kürzesten Pascal-Programmes (Länge d. Maschinen-Programmes), das x generiert.

Idee: Falls eine Algorithmus A ein Wort x generiert, dann kann man A als eine Darstellung von x betrachten. Falls A eine kürzere Darstellung von x liefert folgt, dass A eine Kompression von x ist.

Definition 1.5 Ein Wort $x \in \Sigma^*_{bool}$ heißt zufällig, falls $K(x) \geq |x|$, d.h. x ist nicht komprimierbar.

2 Wichtige Definitionen, Sätze und Lemmas aus Kapitel 2

Definition 2.1 Ein endlicher Automat EA ist ein Quintupel

$$M = (Q, \Sigma, \delta, q_0, F)$$

wobei

1. Q ist eine endliche Menge von Zuständen
2. Σ ist ein Alphabet
3. q_0 ist der Anfangs-Zustand
4. $F \subseteq Q$ ist die Menge der Endzustände
5. $\delta : Q \times \Sigma \rightarrow Q$ ist die Übergangsfunktion

Konfiguration, Start-Konfiguration, End-Konfiguration, Berechnung, erkannte Sprache

Pumping Methode Methode der Kolmogorrov Komplexität

Definition 2.2 Ein nichtdeterministischer endlicher Automat NEA ist ein Quintupel

$$M = (Q, \Sigma, \delta, q_0, F)$$

wobei Σ, Q, q_0, F die gleiche Bedeutung wie bei einem EA haben und $\delta : Q \times \Sigma \rightarrow 2^Q$ ist die Übergangsfunktion

Satz 2.3 Zu jedem NEA M existiert ein EA M', so dass L(M)=L(M').

3 Wichtige Definitionen, Sätze und Lemmas aus Kapitel 3

Definition 3.1 Eine Turingmaschine TM ist ein 7-Tupel

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$$

wobei

1. Q ist eine endliche Menge von Zuständen
2. Σ ist das Eingabe-Alphabet. $\sqcup \notin \Sigma$
3. Γ ist das Arbeits-Alphabet. $\Sigma \subseteq \Gamma, ¢, \sqcup \in \Gamma$
4. $\delta : (Q - \{q_{acc}, q_{rej}\} \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, O\}$ ist eine Abbildung mit der Eigenschaft $\delta(q, ¢) \in Q \times ¢ \times \{R, O\} \forall q \in Q$. δ nennt man die Übergangsfunktion von M.
5. $q_0 \in Q$ ist der Anfangs-Zustand
6. $q_{acc} \in Q$ ist der akzeptierende Zustand.
7. $q_{rej} \in Q$ ist der verwerfende Zustand.

Konfiguration, Start-Konfiguration, Schritt, Berechnung, akzeptierende Berechnung, akzeptierte Sprache, Mehrbandturingmaschinen, NTM

Satz 3.2 Für jeden 2TM(k) existiert eine TM B, so dass L(A)=L(B)

Satz 3.3 Sei M eine NTM. Es existiert eine TM A, so dass L(M)=L(A).

4 Wichtige Definitionen, Sätze und Lemmas aus Kapitel 4

Diagonalisierung Kodierung von TM und Diagonalisierung

Satz 4.1 Es gibt aufzählbar viele TM

Satz 4.2 Es existiert eine Sprache über Σ_{bool}, die nicht rekursiv aufzählbar ist. Eine davon ist L_d, die Diagonal-Sprache

Satz 4.3 Es gibt eine TM UTM (universelle TM), so dass L(UTM)=L(M)

Lemma 4.4 Sei Σ ein Alphabet und sei L eine Sprache über Σ. Falls $L \in \mathcal{L}_R$, dann $\underline{L^C} = \Sigma^* - L \in \mathcal{L}_R$.

Definition 4.5 Das Halteproblem: entscheide für eine gegebene TM M und ein Wort x ob M auf x hält.

Satz 4.6 CHURCHsche These: TM ist die Formalisierung des Begriffes Algorithmus.

- $\mathcal{L}_R$ ist die Menge der algorithmisch entscheidbaren Sprachen
- die Funktionen, die TM-berechenbar sind, sind genau die Funktionen, die algorithmisch Berechenbar sind.

Definition 4.7 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, falls $L_2 \in \mathcal{L}_R \Rightarrow L_1 \in \mathcal{L}_R$.

L_u **ist rekursiv reduzierbar auf** L_{empty}

Korollar 4.8 Die folgenden Probleme sind nicht entscheidbar :

1. $L_{EQ} = \{< M_1 >, < M_2 > |\ L(M_1) = L(M_2)\}$
2. $L_{\subseteq} = \{< M_1 >, < M_2 > |\ L(M_1) \subseteq L(M_2)\}$

Lemma 4.9 $L_{H,\lambda}$ ist nicht rekursiv

Definition 4.10 Eine Sprache $L = \{< M > |$ M ist eine TM$\}$ heißt semantisch nicht triviales Entscheidungsproblem über TM, falls

1. $L(M_1) = L(M_2)$ für zwei TMen M_1 und M_2 impliziert $< M_1 > \in L \Leftrightarrow < M_2 > \in L$.
2. Existieren TMen M_3 und M_4, so dass $< M_3 > \in L$ und $< M_4 \notin L$.

Satz 4.11 Satz von Rice Jedes semantisch nicht-triviale Entscheidungsproblem über TMen ist unentscheidbar.

PKP, MPKP, Reduktion der beiden aufeinander, beide sind nicht entscheidbar, Simulation eines Laufs einer TM mit Hilfe des MPKP

5 Wichtige Definitionen, Sätze und Lemmas aus Kapitel 5

Definition 5.1 Sei M eine NTM und sei x ein Wort aus Σ^*. Sei $D = C_1C_2...C_k$ die Berechnung von M auf x, dann ist die Zeitkomplexität der Berechnung von M auf x $T_M(x) = k - 1$ (Anzahl der Schritte). Falls die Berechnung auf x unendlich ist, dann $T_M(x) = \infty$. Die Zeitkomplexität im schlechtesten Fall der TM die immer hält, ist die Funktion$T_M : N \to N$ definiert durch

$$T_M(n) = max\{T_M(x) \mid x \in \Sigma^m\}$$

. Dies ist die obere Grenze der Zeit, die man für jede Eingabelänge benötigt.

Definition 5.2 Sei M eine MTM (2TM(k)) $k \in N - \{0\}$, die immer hält. (d.h. irgendein Algorithmus). Sei..... eine Konfiguration. Die Speicherplatzkomplexität von C ist

$$S(C) = max\{|\alpha_i| : i = 1...k\}$$

.

Definition 5.3 Für jede Funktion $F : N \to R^+$:

$$O(F(n)) = \{r : N \to R^+ | \exists n_o \in N \exists c \in N \forall n \geq n_0 : r_n \leq c \cdot f(n)\}$$

Für jedes $r \in O(F(n))$ sagen wir, dass r asymptotisch nicht schneller als F wächst.

Für jede Funktion $G : N \to R^+$:

$$\Omega(G(n)) = \{q : N \to R^+ | \exists n_o \in N \exists d \in N \forall n \geq n_0 : q(n) \geq \frac{1}{d} \cdot g(n)\}$$

Für jedes $r \in Omega(G(n))$ sagen wir, dass r asymptotisch mindestens genauso schnell wie G wächst.

Definition 5.4 Seien $\mathcal{A}$ und $\mathcal{B}$ Maschinenmodelle (Algorithmen-Modelle). Wir sagen, dass $\mathcal{A}$ auf $\mathcal{B}$ polynomialzeitreduzierbar ist : SCHREIBWEISE.

Satz 5.5 These der sequentiellen Berechnungen

Alle vernünftigen(bzgl.Zeitkomplexitätsmessung) Maschinenmodelle (Algorithmenmodelle) sind polynomialzeitäquivalent. Der Begriff der polynomialzeit Berechnungen ist also robust in der Komplexitätstheorie bzgl. der Wahl des Berechnungsmodelles.

Definition 5.6 Fundamentale deterministische Komplexitätsklassen Für alle Funktionen $f, g : N \to R^+$:

- $TIME(f) = \{L(M) \mid M \text{ ist eine NTM mit } T_M(n) \in O(f(n))\}$
- $SPACE(g) = \{L(A) \mid A \text{ ist eine NTM mit } S_A(n) \in O(g(n))\}$
- $TIME - SPACE(f, g) = \{L(B) \mid B \in TIME(f) \wedge B \in SPACE(g)\}$
- $DLOG = SPACE(log_2(n))$
- $P = \bigcup_{c \in N} TIME(n^c)$
- $EXPTIME = \bigcup_{d \in N} TIME(2^{n^d})$
- $PSPACE = \bigcup_{c \in N} SPACE(n^c)$

Lemma 5.7 Für jede Funktion $t : N \to R^+ gilt TIME(t(n)) \subseteq SPACE(t(n))$

Korollar 5.8

$$P \subseteq PSPACE$$

s-konstruierbarkeit, Zeit-konstruierbarkeit, Platz-konstruierbarkeit

$DLOG \subseteq P$**,** $PSPACE \subseteq EXPTIME$

Nichtdeterministische Turingmaschinen

Definition 5.9 Nichtdeterministische Komplexitätsklassen

für alle $f, g : N \to R^+$

- $NTIME(f) = \{L(M) \mid M \text{ ist NTM mit } T_M(n) \in O(f(n))\}$

- $NSPACE(g) = \{L(N) \mid M \text{ ist NTM mit } S_M(n) \in O(g(n))\}$
- $NLOG = NSPACE(log_2(n))$
- $NP = \bigcup_{c \in N} NTIME(n^c)$
- $NPSPACE = \bigcup_{c \in N} NSPACE(n^c)$

Lemma 5.10 Für alle platzkonstruierbaren Funktionen t,s :

$$NTIME(t) \subseteq NSPACE(t)$$

$$NSPACE(s) \subseteq \bigcup_{c \in N} NTIME(c^{s(n)})$$

Satz 5.11 Für jede platzkonstruierbare Funktion $t : N \to R^+$

- $TIME(t(n)) \subseteq NTIME(t(n))$
- $SPACE(t(n)) \subseteq NSPACE(t(n))$
- $NTIME(t(n)) \subseteq SPACE(t(n)) \subseteq TIME(c^t(n))$

$$NP \subseteq PSPACE$$

$$DLOG \subseteq NLOG \subseteq P \subseteq NP \subseteq PSPACE \subseteq EXPTIME$$

Definition 5.12 Sei $L \subseteq \Sigma^*$ eine Sprache. Sie $p : N \to N$ eine Funktion. Wir sagen, dass ein Algorithmus (eine TM) ein p-Verifizierer ist, $V(A) = L$, falls A mit folgenden Eigenschaften auf Eingaben aus $\Sigma^* \times \Sigma^*_{bool}$ arbeitet.

1. $T_A(w, x) \leq p(|w|)$ für jede Eingabe $(w, x) \in \Sigma^* \times \{0, 1\}^*$.
2. Für jeden $w \in L$ existiert ein $x \in \{0, 1\}^*$ mit $|x| \leq p(|w|)$, so dass $(w, x) \in L(A)$ also A akzeptiert (w,x). x nennt man den Beweis oder das Zertifikat der Behauptung $''w \in L''$.
3. Für jedes $x \notin L, (y, z) \notin L(A)$ für alle $z \in \{0, 1\}^*$.

Falls $p(n) \in O(n^k)$ für ein $k \in N$, dann sagen wir, dass A ein Polynomialzeitverfizierer ist.

$VP = \{V(A) \mid$ **A ist Polynomialzeitverifzierer** $\}$

Satz 5.13 $NP = VP$

Definition 5.14 NP-Vollständigkeit Ausgangssituation

1. es gibt mehrere tausend praktisch interessant Probleme in NP, für die kein Polynomialzeit-Algorithmus bekannt ist.
2. Keine Beweis-Methode ist vorhanden, mit der man zeigen könnte, dass ein Problem aus NP keinen polynomiellen Algorithmus besitzt.

Neuer Ansatz

Entwicklung einer Methode, die unter der Voraussetzung ¶ $\subset NP$ das zeigen der Nicht-Existenz von polynomiellen Algorithmen für konkrete Aufgaben aus NP ermöglicht.

Warum ist die Annahme $P \subset NP$ vernünftig ?

1. **Ein theoretische Grund:** NP=VP und man glaubt nicht, dass die Verifikation eines Beweises genauso schwierig ist, wie die Herstellung eines Beweises sein sollte.
2. **Ein Praktische Grund:** -Erfahrung- mehr als 3000 Probleme in NP sind bekannt, für die die besten Algorithmen exponentielle Laufzeiten haben. Man glaubt nicht, bei so großen Bemühungen, dass diese Lage nur der menschlichen Unfähigkeit zuzuschreiben ist.

Idee: Wir wollen die schwersten Probleme in NP definieren auf eine solche Weise, dass falls $P \subset NP$ ist, jedes der schwersten Probleme aus NP in $NP - P$ liegt.

Definition 5.15 Eine Sprache ist NP-schwer, falls für alle $U \in NP, U \leq_p L$. Eine Sprache L ist NP-vollständig. falls

1. $L \in NP$
2. L ist NP-schwer.

Satz 5.16 **Satz von Cook** SAT ist NP vollständig.

Lemma 5.17 Seien L_1 und L_2 zwei Sprachen. Falls $L_1 \leq_p L_2$ und L_1 ist NP-schwer, dann ist L_2 auch NP-schwer.

$SAT \leq_p CLIQUE$
$CLIQUEQ \leq_p VC$
$SAT \leq_p 3SAT$

Definition 5.18 NPO ist die Klasse der Optimierungsprobleme wobei $U = (\Sigma_i, \Sigma_O, L, \Updownarrow, cost, goal) \in NPO$, falls folgende Bedingungen erfüllt sind:

1. $L \in P$ d.h. es kann effizient verifiziert werden, ob ein $x \in \Sigma_I^*$ eine zulässige Eingabe von U ist.
2. $\exists P_n$ Polynom, so dass
   (a) für jedes $x \in L$ und jedes $y \in \mathcal{M}(x) : |y| \leq P_n(|x|)$. d.h. die Größe jeder Lösung ist polynomiell in der Eingabegröße
   (b) Es existiert ein polynomialzeit-Algorithmus A, der für jedes $y \in \Sigma_O^*$ und jedes $x \in L$ mit $|y| \leq P_n(|x|)$ entscheidet, ob $y \in \mathcal{M}(x)$ ist oder nicht.
3. Die Funktion cost kann man in polynomieller Zeit berechnen.

Lemma 5.19 Falls ein Optimierungsproblem $U \in PO$, dann $Lang_U \in P$

Satz 5.20 Sei $U \in NPO$. Falls $Lang_U$ NP-schwer ist und $P \neq NP$, dann $U \notin PO$

Lemma 5.21 MAX-SAT ist NP-schwer

Lemma 5.22 MAX-CL ist NP-schwer

6 Wichtige Defintionen, Sätze und Lemmas aus Kapitel 6

Approximationsgüte, Bsp: Makespan-Scheduling, Greedy Algorithmus

Randomisierte Algorithmen, Wahrscheinlichkeitsverteilung, Erwartungswert, one-sided-error, Bsp: Primzahltest

7 Wichtige Definitionen, Sätze und Lemmas aus Kapitel 7

Einwegfunktionen, Bsp: Primfaktorzerlegung

RSA, Kommunikationsprotokolle