

1 Berechenbarkeit und Komplexität

Definition: Eine Turingmaschine hat die Form $M = (Q, \Sigma, \Gamma, \delta, q_0, q_s) \equiv$ (endl. Zustandsmenge, Eingabealphabet, Arbeitsalphabet, Übergangsfunktion $\delta : Q \setminus \{q_s\} \times \Gamma \rightarrow \Gamma \times \{R, L, N\} \times Q$, Anfangszustand, Stopzustand)
 $\delta(p, a) = (a', R \setminus L \setminus N, q)$ oder Turingtafel:
 $q_0, a, a', R \setminus L \setminus N, q_1$
Für $w \in \Sigma^* \setminus \text{Def}(f)$ schreibe $f(w) = \perp$ (undefiniert).

1.3 Berechenbarkeit und nicht-Berechenbarkeit

Satz: f berechenbar $\Leftrightarrow f$ Turing-Berechenbar (Beweis: \Leftarrow Klar, da TM spezielle Form eines Algorithmus, \Rightarrow nicht mathematisch nachweisbar)

Church-Turing-These: Jede berechenbare Funktion ist auch Turing-berechenbar \Rightarrow nicht bewiesen, aber wesentlich, wenn Nachweis „nicht berechenbar“.

Definition: L entscheidbar (semi-entscheidbar) gdw. es existiert ein Algorithmus, der L entscheidet (semi-entscheidet)

Satz: $L \subseteq \Sigma^*$ entscheidbar $\Leftrightarrow L$ semi-entscheidbar und $\Sigma^* \setminus L$ ist auch semi-entscheidbar. (Beweis: \Rightarrow Sei A Entscheidungsalg. für L . Bilde A' durch Einbau einer nicht term. Schleife für „nein“; \Leftarrow Seien A_1, A_2 semi-Entscheidungsverfahren für L bzw. $\Sigma^* \setminus L$, Beachte: entweder $w \in \Sigma^* \in L$ oder $w \in \Sigma^* \in \Sigma^* \setminus L$, lasse A_1 und A_2 jeweils einen Schritt laufen und teste \rightarrow Antwort)

Definition: Zu $f : \Sigma^* \rightarrow \Sigma^*$ (partiell berechenbar) definiere $G_f = \{w \# f(w) \mid w \in \text{Def}(f)\}$ „Graph von f “.

Satz: $f : \Sigma^* \rightarrow \Sigma^*$ partiell berechenbar $\Leftrightarrow G_f$ ist semi-entscheidbar (mit Semi-Entscheidungsalg. A_{G_f}). (Beweis: \Rightarrow Teste ob $v = w_0 \# w_1 \in \Sigma^*$. Wenn nicht gebe „neine“ aus, sonst starte. Lasse A_f auf w_0 laufen, speichere w_1 . Bei Termination vergleiche $f(w_0)$ mit w_1 , wenn gleich „ja“, sonst Endlosschleife; \Leftarrow Finde Berechnungsverf. für f , lasse A_{G_f} laufen (1. $\forall v$ auf Eingabe $w \# v$, 2. \forall Schrittzahlen s nach Muster: Bei Term. von A_G für $w \# v$ gebe f_v aus).

Satz: $L \subseteq \Sigma^*$ semi-entscheidbar $\Leftrightarrow L$ ist Definitionsbereich einer berechenbaren Funktion $f : \Sigma^* \rightarrow \Sigma^*$ (Beweis: \Rightarrow A semi-entscheidet $L \subseteq \Sigma^*$; Sei $a \in \Sigma$; **Definiere:** $f : \Sigma^* \rightarrow \Sigma^*$ durch $f(w) = \{aw \in L; \perp w \notin L\}$; Gilt $L = \text{Def}(f)$: Folgender Algorithmus B berechnet f : Für Eingabe w arbeite wie A , allerdings, bei Termination gebe a aus; \Leftarrow A berechnet f . L sei $\text{Def}(f)$; Gesucht: Semi-Entscheidungsalgorithmus B für L . Für B übernehme A , wobei die Ausgabe jeweils geändert ist zu „ja“)

Ein Aufzählungsalgorithmus ist ein Verfahren, das ohne Eingabe gestartet wird und sukzessiv Ausgabewörter liefert (in irgend einer Reihenfolge, eventuell mit Wiederholungen).
 $L \subseteq \Sigma^*$ heißt aufzählbar, wenn es einen Aufzählungsalgorithmus gibt, für den gilt:

Für $w \in \Sigma : w \in L \Leftrightarrow w$ wird irgendwann als Ausgabe geliefert.

Bemerkung: „berechenbar“ verlangt nur Existenz eines Algorithmus.

2 Registermaschinen und while-Programme

2.1 goto-Programme

bestehen aus $m+1$ -Zeilen, wobei in der $m+1$ -Zeile STOP steht.

Befehle: $X_i := X_i \pm 1$; if $X_i = 0$ goto l else goto l'

Konfiguration: $(m+1)$ -Tupel (Zeilennummer, Wert von X_1, \dots , Wert von X_m).

Definition: Zu goto $_m$ -Programm P und $n \leq m$ definiere die n -stellige durch P berechnete Funktion $f_P^{(n)} : \mathbb{N}^n \rightarrow \mathbb{N}$ durch: $f_P^{(n)}(x_1, \dots, x_n)$ def. $\Leftrightarrow P$ ausgehend von Konfiguration $(1, x_1, \dots, x_n, 0, \dots, 0)$ erreicht Stopkonf. (k, y_1, \dots, y_m) und in diesem Fall $f_P^{(n)}(x_1, \dots, x_n) = y_1$
 $f : \mathbb{N}^n \rightarrow \mathbb{N}$ heißt goto $_m$ -berechenbar, wenn es ein goto $_m$ -Programm P gibt mit $f_P^{(n)} = f$ goto-berechenbar, falls goto $_m$ -berechenbar für geeignete m .

2.2 while-Programme:

Arbeitet mit den Variablen X_1, \dots, X_m .

$\langle \text{prog}_m \rangle ::= X_i := 0 \mid X_i := X_j \mid X_i := X_i + 1 \mid X_i := X_i - 1 \mid X_i := X_j \text{ op } X_k$ mit $\text{op} \in \{+, \dot{-}, \cdot, \div, \text{mod}\}$ | $\langle \text{prog}_m \rangle$ | if $\langle \text{bed}_m \rangle$ then $\langle \text{prog}_m \rangle$ else $\langle \text{prog}_m \rangle$ end | while $\langle \text{bed}_m \rangle$ do $\langle \text{prog}_m \rangle$ end | loop $\langle \text{var}_m \rangle$ begin $\langle \text{prog}_m \rangle$ end: mit $1 \leq i, j, k \leq m$, $\dot{-} :=$ Ergebnis mindestens = 0, $\langle \text{bed}_m \rangle \in \{X_i > 0, X_i = 0\}$

Definition: Definiere zu while $_m$ -Programmen P eine Transformation $\llbracket P \rrbracket : \mathbb{N}^m \rightarrow \mathbb{N}^m$; $\llbracket P \rrbracket(r_1, \dots, r_m) = (s_1, \dots, s_m)$ falls P , gestartet mit (r_1, \dots, r_m) als Werten für X_1, \dots, X_m terminiert und dann als Werte von X_1, \dots, X_m die Zahlen s_1, \dots, s_m liefert.

Definition: Zu P def. wie bei goto-Program.: $f_P^{(n)}(x_1, \dots, x_n) = p_1^m; \llbracket P \rrbracket(x_1, \dots, x_n, 0, \dots, 0)^m$.

Bemerkungen: loop-Schleifen terminieren; while-Schleifen terminieren nicht immer;

Satz: „while ist nicht immer durch loop ersetzbar“; „loop ist immer durch while ersetzbar“.

Ackermann-Funktion: $A(0, y) = y + 1$; $A(x + 1, 0) = A(x, 1)$; $A(x + 1, y + 1) = A(x, A(x + 1, y))$; A ist total und (while-)berechenbar, jedoch nicht loop-berechenbar; **Lemma:** Zu jedem loop $_m$ -Programm P existiert Zahl k_P mit $\max(\llbracket P \rrbracket(x_1, \dots, x_m)) < A(k_P, \max(x_1, \dots, x_m))$ (Beweis durch Induktion über Aufbau der loop-Programme.)

Äquivalenzsatz: Für eine Funktion $F : \mathbb{N}^n \rightarrow \mathbb{N}$ sind folgende Aussagen äquivalent: 1. f Turing-berechenbar; 2. f goto-berechenbar; 3. f while-berechenbar (d.h. Zu jedem while $_m^0$ -Programm kann man jeweils ein goto $_m$ bzw. eine TM konstruieren (Beweis über Ind. bzw. Ringschluss))

3 Unentscheidbarkeit

3.1 Unentscheidbare Probleme:

Wortproblem (WP): Gegeben: TM M , $w \in \Sigma_{\text{bool}}^*$; Frage: Gilt $M : w \rightarrow \text{STOP}$? (Beweis: Mit CTT)

Sprache von WP: L_{WP} ist (Turing-)semi-entscheidbar!
 L aufzählbar $\Rightarrow L \leq L_{WP} \Rightarrow L_{WP}$ ist vollständig bzgl. \leq .

Halteproblem (HP): Gegeben: TM M über $\Sigma = \{0, 1\}$; Frage: $M : \varepsilon \rightarrow \text{STOP}$? ($WP \leq HP$)

Äquivalenzproblem (ÄP): Gegeben: TM M_1, M_2 über $\Sigma = \{0, 1\}$; Frage: Berechnen M_1, M_2 dieselbe Funktion $f : \Sigma^* \rightarrow \Sigma^*$? ($HP \leq \text{ÄP}$)

Dominoproblem: Ein Dominospiel ist eine Folge $D = (d_0, \dots, d_k)$ von Dominotypen d_i , jedes d_i ein Quadrupel von „Farben“ (hier Zahlen) der Form $(d_{i_1}(N), d_{i_2}(O), d_{i_3}(S), d_{i_4}(W))$ (oben beginnend im Uhrzeigersinn)

Eine Parkettierung ist eine Funktion $\pi : \mathbb{Z} \times \mathbb{Z} \rightarrow \{d_0, \dots, d_k\}$ mit $\pi(0, 0) = d_0$ und $O(\pi(i, j)) = W(\pi(i, j + 1))$ $S(\pi(i, j)) = N(\pi(i + 1, j))$ ($i, j \in \mathbb{Z} \times \mathbb{Z}$)

D ist gut: \Leftrightarrow ex. Parkettierung (von $\mathbb{Z} \times \mathbb{Z}$) mit D

Problem: Gegeben Dominospiel D ; Frage: D gut? ($HP \leq D$)

Postches Korrespondenzproblem (PCP): Gegeben: Liste $X = (x_1, \dots, x_n)$, $Y = (y_1, \dots, y_n)$ von Wörtern x_i, y_i über Alphabet Σ ; **Frage:** Hat (X, Y) eine Lösung, d.h. ex. Indexfolge i_1, \dots, i_k mit $x_{i_1} \dots x_{i_k} = y_{i_1} \dots y_{i_k}$ (erzeugtes Wort heißt Lösungswort); Modifiziertes PCP (MPCP) verlangt Indexfolge mit $i_1 = 1$. ($WP \leq \text{MPCP}$)

Halteproblem für goto $_2$ -Programme (2-Zähler-Maschinen): Gegeben: ein goto $_2$ -Programm P (Variablen $X_1, X_2, +1, -1, \text{Sprung}$)
Frage: Stoppt P , gestartet mit Wert 0 für X_1, X_2 ? ($HP \leq \text{HO}(\text{goto}_2) \leq \text{HP}(\text{goto}_4)$)

Satz: Es gibt keinen Algorithmus, der zu jeder TM M über $\Sigma = \{0, 1\}$ und zu jedem Wort $w \in \Sigma^*$ entscheidet, ob $M : w \rightarrow \text{STOP}$ (Es gibt keine TM M_0 die angesetzt auf das Paar (M, w) entscheidet ob $M : w \rightarrow \text{STOP}$ gilt.)

Bemerkung: Algorithmus über beliebige TM wäre selbst durch TM darstellbar und würde somit über sich selbst Auskunft geben.

Entscheidungsproblem: Ein Entscheidungsproblem hat die Form $P = (\text{Inst}_P, \text{Pos}_P)$, wobei Inst_P die Instanzenmenge von P ist Pos_P die Menge der Instanzen, die die positive Antwort (ja) verlangen.

Definition: Für Probleme $P = (Inst_P, Pos_P), Q = (Inst_Q, Pos_Q)$ def. $P \leq Q$ („ P auf Q reduzierbar“), falls ex. eine berechenbare Funktion $f : Inst_P \rightarrow Inst_Q$ mit für jedes $x \in Inst_P : x \in Pos_P \Leftrightarrow f(x) \in Pos_Q$ (Alternativ: Gelte $P \leq Q$. Wenn Q entscheidbar, dann P entscheidbar.)

Zusatz: Ein TM-Problem ist **nichttrivial**, wenn weder $Pos_P = Inst_P$ noch $Pos_P = \emptyset$. Ein TM-Problem heißt **semantisch**, wenn die Mitgliedschaft einer TM M in Pos_P nur von der durch M berechneten Funktion abhängt. (d.h. M_1, M_2 äquivalent $\Rightarrow M_1 \in Pos_P \Leftrightarrow M_2 \in Pos_P$)

Satz von Rice: Jedes nichttriviale semantische TM-Problem ist unentscheidbar.

4 Komplexitätstheorie: Grundlagen

4.1 Zeitkomplexität

Definition: Zu $g : \mathbb{N} \rightarrow \mathbb{N}$ sei $O(g)$ die Klasse der Funktion $f : \mathbb{N} \rightarrow \mathbb{N}$ mit $\exists c > 0 \exists n_0 \forall n \geq n_0 f(n) \leq c \cdot g(n)$ (Typische Funktionen $g : n, \log(n), n \log(n), n^2, 2^n, \dots$)

Definition: TM M terminiere für jede Eingabe w . M heißt $O(g(n))$ -zeitbeschränkt, falls folgende Funktion f zu $O(g(n))$ gehört: $f(n) = \max\{m \mid \text{ex. } w \text{ mit } |w| = n, M \text{ auf } w \text{ läuft } m \text{ Schritte}\}$

Offline-TM: Eine Offline-TM hat 2 Bänder (Eingabeband zum Lesen und Arbeitsband zum Lesen und Schreiben) und Anweisungen folgender Form: $p a b b' L/N/R L/N/R q$.

4.2 Klassen P und NP

Definition: $P =$ Klasse der durch polynomial zeitbeschränkte TM entscheidbaren Sprachen. (Äquivalente Bedingung: ex. $k \geq 0$ so dass $M O(n^k)$ -zeitbeschränkt ist.)

These von Cobham und Edmonds: P enthält (ziemlich) genau die algorithmischen Probleme, die praktisch lösbar sind. (Argumente: Erfahrung mit polynomial zeitbeschränkten Algorithmen; Qualitätssprung gegenüber exponentiellen Laufzeiten; Bezug auf Turingmaschinen ist unwesentlich (Präziser: Liegt ein polynomial zeitbeschränkter Algorithmus für irgendein sequentielles Standard-Rechnermodell vor, so ex. dafür auch polynomial zeitbeschränkte TM.)) **wichtig?**

Definition: $L \in NP : \Leftrightarrow$ ex. polynomial entsch. Relation $R \subseteq \Sigma^* \times \Gamma^*$ (Relation $R \subseteq \Sigma^* \times \Gamma^*$ heißt polynomial entscheidbar, falls TM von Konf. $q_0 u \sqcup v$ die Entscheidung „ $(u, v) \in R$ “ liefert in Laufzeit $P(|u| + |v|)$ mit Polynom P) und Polynom $q(n)$, so dass $w \in L \Leftrightarrow \exists v (|v| \leq q(|w|) \wedge (w, v) \in R)$; „Teste Mitgliedschaft von w in L mit Probieren aller Lösungskandidaten v , deren Länge polynomial ind

$|w|$ beschränkt.“ (alternative Def.: $L \in NP : \Leftrightarrow$ ex. polynomial zeitbeschränkte NTM, die w akz. gdw. $w \in L$.)

Bem: Ist R durch $p(n)$ -Zeitbeschr. TM entscheidbar, so ist Überprüfung von (w, v) in Zeit $\underbrace{p(|w|) + q(|v|)}_{\text{möglich}}$ also polynomial in $|w|$

Definition: Nichtdeterministische TM (NTM) ist wie zuvor definiert, kann jedoch für $(q, a) \in$ Zustandsmenge $Q \times$ Arbeitsalphabet Γ mehrere Turingzeilen enthalten. Zu geg. Konfiguration K existieren dann gegebenenfalls mehrere Folgekonfigurationen. (NTM heißt $t(n)$ -zeitbeschränkt, falls jeder Zweig des Konfigurationsbaums endet nach $\leq t(n)$ Schritten, jeweils für Eingabe w der Länge n .)

Primzahlproblem $\in NP$: Zahl Primzahl?

COLOR(3) $\in NP$: Gegeben: Endl. unger. Graph $G = (V, E)$; Frage: Ex. 3-Färbung von G , d.h. $c : V \rightarrow \{1, 2, 3\}$ mit $c(u) \neq c(v)$ für $(u, v) \in E$; Probieralgorithmus testet alle 3^n viele Farbverteilungen c und überprüfe jeweils, ob korrekte Färbung vorliegt. (Angabe NTM oder $SAT(3) \leq P$ COLOR(3))

SAT $\in NP$: Gegeben: aussagenlogischer Ausdruck $\varphi (\neg, \wedge, \vee, Xi$ (i Binärdarstellung)); häufig nur KNF ($\phi = c_1 \wedge \dots \wedge c_m$)); Frage: ist φ erfüllbar? (Angabe polynomialzeitbeschränkte NTM)

SAT(3) $\in NP$: Gegeben: KNF-Ausdruck φ mit je 3 Literalen pro Klausel; Frage: Ist φ erfüllbar? (Transform. KNF in Graph)

CLIQUE $\in NP$: Gegeben: Graph G , Zahl k ; Frage: Existiert Clique der Größe k in G d.h. eine Menge von k Knoten, die paarweise untereinander durch Kanten verbunden sind. ($SAT(3) \leq P$ CLIQUE)

CLIQUE(k) $\in P \neg$ **CLIQUE**: Gegeben: Graph G ; Frage: Ex. Clique der Größe k in G ?

5 NP-Vollständigkeit

Problem kodiert durch Sprache L : Gegeben: $w \in \Sigma^*$; Frage: $w \in L$?

Vergleich der Komplexität von Sprachen $L_1 \subseteq \Sigma_1^*, L_2 \subseteq \Sigma_2^* : L_1 \leq L_2$ (L_1 polynomzeit-reduzierbar auf L_2): \Leftrightarrow ex. eine polynomzeitberechenbare Funktion $f : \Sigma_1^* \rightarrow \Sigma_2^*$ mit $w \in L_1 \Leftrightarrow f(w) \in L_2$; $f : \Sigma_1^* \rightarrow \Sigma_2^*$ polynomzeitberechenbar gdw. es ex. polynomzeitbeschr. TM die f berechnet.

Bem.: $L_1 \leq_p L_2, L_2 \in P \Rightarrow L_1 \in P$ (Beweis: M_2 entscheide L_2 in $q(n)$ -zeitbeschr.. Sei M_f die TM, die f in $p(n)$ -z.b. berechnet. Stelle aus $w \in \Sigma_1^*$ mit $M_f f(w)$ her, entscheide mit M_2 , wegen Korrektheit von oben, fertig (Zeit: $p(|w|) + q(|w| + p(|w|))$)

Definition: L_0 NP-vollständig: $\Leftrightarrow : L_0 \in NP$; für alle $L \in NP : L \leq_p L_0$

5.1 Beweis von NP-Vollst.:

Um zu zeigen, dass L_0 NP-vollst., genügt es, zu zeigen: $L_0 \in NP$; Für geeingete NP-vollst. Sprache L_1 gilt: $L_1 \leq_p L_0$

5.2 Platzkomplexität

Idee für TM: Zähle die Anzahl der besuchten Felder während einer Berechnung.

Unterscheide Feldzugriffe zwecks Lesen der Eingabe und Feldzugriffen zwecks Rechnung.

Offline-Turingmaschine (Lese-) Eingabeband

Notationen DLOG = Klasse der Sprachen die durch DTM an Platz $O(\log(n))$ entschieden werden können. ($L = \{a^i b^j \mid i > 0\} \in$ DLOG); NLOG analog für NTM. Sprache der Kodierung des Erreichbarkeitsproblems gehört zu NLOG; (D/N)SPACE analog für Polyom $p(n)$ an Stelle $\log(n)$

Satz: Eine $f(n)$ -platzbeschränkte Offline-TM kann durch eine $2^{O(f)}$ -zeitbeschr. TM simuliert werden. Eine $f(n)$ -zeitbeschränkte TM kann durch eine $f(n)$ -platzbeschränkte (Offline-)TM simuliert werden.

Eine polynomial $p(n)$ -platzbeschr. Offline-NTM kann durch eine $O(p^2(n))$ -platzbeschr. Offline-DTM simuliert werden.

Bem. $DLOG \subseteq NLOG \subseteq P \subseteq NP \subseteq PSPACE$ (nur bekannt: $DLOG \subseteq PSPACE$)

Zum Problem QBF Quantifizierte Boolesche Formeln (in pränex-Normalform) haben die Form $Q_1 x_1 Q_2 x_2 \dots Q_n x_n \varphi(x_1, \dots, x_n)$ mit $Q_i \in \{\exists \forall\}$ aussagenlog. Formel. $\exists x_i \psi$ besagt "es ex. Wahrheitswert (0 oder 1) so dass ψ wahr wird, wenn dieser für x_i eingesetzt wird." $\exists x_i \psi(\dots x_i \dots) \Leftrightarrow \psi(\dots 0 \dots) \vee \psi(\dots 1 \dots)$ Analog für \forall mit \wedge an Stelle von \vee . (keine Belegung nötig, Formel selber wahr oder falsch) **Problem:** Gegeben: QBF ψ ; Frage: Ist ψ wahr?

5.3 Approximationsalgorithmen

Rucksackproblem: Gegeben: $w_1, \dots, w_n, b \in \mathbb{N}_+$; Gesucht: $I \subseteq \{1, \dots, n\}$ mit $\sum_{i \in I} w_i$ möglichst groß, mit Bedingung $\sum_{i \in I} w_i \leq b$ (Spezialfall: $\underline{RS} \in NP$: Gegeben: w_1, \dots, w_n, b ; Frage: ex. $I \subseteq \{1, \dots, n\}$ mit $\sum_{i \in I} w_i = b$? ($SAT(3) \leq_p \underline{RS}$))

Definition: Ein Polynomzeit-Approximationsschema (PTAS (polyn. time. approx. scheme.)) liefert für jedes $\epsilon > 0$ einen Algorithmus der zu Eingabe x eine Lösung M liefert mit $\frac{c(M_0)}{c(M)} \leq 1 + \epsilon$ (mit M_0 optimale Lösung), polynomzeitbeschr. in $|x|$. Erlaubte Laufzeit: $|x|^{\frac{1}{\epsilon}}$