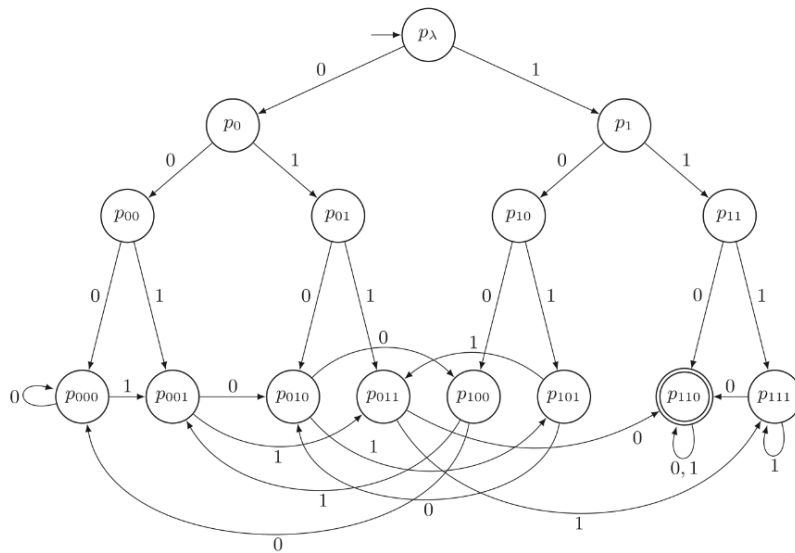


Theoretische Informatik

Cheat Sheet



Contents

1 Formale Sprachen	1	4 Berechenbarkeit	4
1.1 Algorithmische Probleme	1	4.1 Diagonalisierung	4
1.2 Kolmogorov Komplexität	1	4.2 Reduktion	4
1.3 Anwendungen der Kolmogorov Komplexität	1	4.3 Rice	4
		4.4 Kolmogorov	4
2 Endliche Automaten (EA)	2	5 Komplexität	5
2.1 Irregularität beweisen	2	5.1 Zeit & Speicher	5
2.2 Nicht-deterministische endliche Automaten (NEA)	2	5.2 O-Notation	5
		5.3 Komplexitätsklassen	5
3 Turing-Maschinen	3	5.4 Nicht-deterministische Komplexität	5
3.1 Mehrband Turing-Maschinen	3	5.5 NP-Vollständigkeit	6
3.2 Nicht-deterministische TMs	3	5.6 Klausel-Formeln	6
3.3 Sprach-Klassen	3	6 Grammatiken	7

Einleitung

Der Sinn dieses Dokuments ist, alle Resultate und Definitionen schnell auffindbar an einem Ort zu haben, z.B. für Hausaufgaben. Dieses Dokument ist keine Zusammenfassung, enthält aber einige Kommentare und intuitive Erläuterungen (Text in grau). Gute ausführliche Zusammenfassungen existieren bereits: z.B. die von Nicolas Wehrli, auf Community Solutions.

Wie immer: Keine Garantie auf Komplettheit (primär Inhalt für HS25 Mid/Enterm) oder Korrektheit.

Robin Bacher

ETH Zürich, HS25

Basierend auf:
Theoretische Informatik, J. Hromkovic
TheoInf Summary, N. Wehrli

1 Formale Sprachen

Grundlage, nötig für die Formalisierung von Algorithmen.

Def: Alphabet $\Sigma \stackrel{\text{def}}{\iff} \Sigma$ endlich und $\Sigma \neq \emptyset$
 $x \in \Sigma$ heisst Buchstabe, Zeichen, Symbol.

Def: Wort w über $\Sigma \stackrel{\text{def}}{\iff} w = (x_1, \dots, x_n)$ endlich, $x_i \in \Sigma$.
 $\Sigma^* := \{w \mid w \text{ ist Wort über } \Sigma\}$
 $\lambda := w \text{ s.d. } |w| = 0$ (Leeres Wort)
 $\Sigma^+ := \Sigma^* \setminus \{\lambda\}$

Teilwort v von $w \stackrel{\text{def}}{\iff} \exists x, y \in \Sigma^* : w = xvy$

Präfix v von $w \stackrel{\text{def}}{\iff} \exists y \in \Sigma^* : w = vy$

Suffix v von $w \stackrel{\text{def}}{\iff} \exists x \in \Sigma^* : w = xv$

Note: Notation: $x_1x_2 \dots x_n = (x_1, x_2, \dots, x_n)$

$\text{Nummer}(x) := \sum_{i=1}^n x_i \cdot 2^{n-i}$.
 $\text{Bin}(m) :=$ kürzeste Binärkodierung von m in Σ_{Bool} .
 $\text{Bin}(0) := 0$

Def: Konkatenation $\text{Kon}(x, y) = x \cdot y = xy$
 $\forall w : \lambda \cdot w = w$
Kon ist assoziativ

Def: Reversal Für $a \in \Sigma^*$: $a^R := a_n a_{n-1} \dots a_1$

Def: Iteration Für $x \in \Sigma^*$: $x^0 := \lambda, x^1 := x, x^i := xx^{i-1}$

Def: Vorkommen von a in $w \in \Sigma^*$: $|w|_a := |\{i \mid w_i = a\}|$

Def: Kanonische Ordnung von Σ^* : Sei $<$ eine Ordnung über Σ . $u, v \in \Sigma^*$. $x, u', v' \in \Sigma^*$ und $i < j$.

$$u < v \iff |u| < |v| \quad \vee \quad |u| = |v| \wedge u = x \cdot s_i \cdot u' \wedge v = x \cdot s_j \cdot v'$$

Def: Sprache L über $\Sigma \stackrel{\text{def}}{\iff} L \subset \Sigma^*$

Def: Konkatenation von L_1, L_2 : $L^0 := L_\lambda, L^i := L^{i-1} \cdot L$

Def: Komplement $L^C := \Sigma^* \setminus L$

Def: Kleene'scher Stern $L^* := \bigcup_{i \geq 0} L^i, \quad L^+ := L^* \setminus \{\lambda\}$

Lemma: \cup ist distributiv über Sprachen:

$$L_1 L_2 \cup L_1 L_3 = L_1 (L_2 \cup L_3)$$

1.1 Algorithmische Probleme

TODO

1.2 Kolmogorov Komplexität

Ziel: Komprimierung von Wörtern, Schliessen auf Informationsdichte basierend auf Komprimierbarkeits-schwierigkeit.

Def: Kolmogorov Komplexität $\forall x \in \Sigma_{\text{bool}}^* : K(x) :=$ kürzestes Pascal-Programm für x .

Vermeidet die Festlegung auf einen spezifischen Komprimierungsalgorithmus. Buch beweist ebenfalls, dass Pascal hier *keine* Einschränkung ist.

Def: $K(x)$ von Natürlichen Zahlen: $K(n) := K(\text{Bin}(n))$

Lemma: $\exists d \forall x \in \Sigma_{\text{bool}}^* : K(x) \leq |x| + d$

$\text{Bin}(|x|)$ hat Länge $\lceil \log_2(|x| + 1) \rceil$

Lemma: $\forall n \geq 1 \exists w_n \in \Sigma_{\text{bool}}^n : K(w_n) \geq |w_n| = n$
Intuitiv: Es existieren unkomprimierbare w jeder Länge.

1.3 Anwendungen der Kolmogorov Komplexität

Def: Zufällig $\stackrel{\text{def}}{\iff} x \in \Sigma_{\text{bool}}^*$ erfüllt $K(x) \geq |x|$

$n \geq 0$ ist zufällig $\stackrel{\text{def}}{\iff} K(n) = K(\text{Bin}(n)) \geq \lceil \log_2(n + 1) \rceil - 1$

Diese Definition hat intuitiv nichts mit dem Zufallsbegriff aus der Wahrscheinlichkeit zu tun, hier geht es um den Informationsgehalt.

Theorem: (2.2) \exists Programm A_L welches $(\Sigma_{\text{bool}}, L)$ löst $\implies \forall n \geq 1 : K(z_n) \leq \lceil \log_2(n + 1) \rceil + c$

$L \subset \Sigma_{\text{bool}}^*$. $z_n := n$ -tes Wort bzgl. kan. Ordnung. $(\Sigma_{\text{bool}}, L)$ ist ein Entscheidungsproblem.

Vereinfacht häufig Beweise zur Kolmogorov Komplexität stark.

Theorem: $\lim_{n \rightarrow \infty} \frac{\text{Prim}(n)}{n / \ln(n)} = 1$

$\text{Prim}(x) :=$ Anzahl Primzahlen kleiner x . Intuitiv: Anzahl Primzahlen wächst gleich schnell wie Anzahl Zahlen.

2 Endliche Automaten (EA)

Def: Endlicher Automat $M := (Q, \Sigma, \delta, q_0, F)$

Zustände Q (endlich)
 Eingabealphabet Σ (Alphabet)
 Anfangszustand $q_0 \in Q$
 Akzeptierende Zustände $F \subseteq Q$
 Übergangsfunktion $\delta : Q \times \Sigma \rightarrow Q$

$\delta(q, a) = p \implies$ im Zustand q bei Eingabe a , gehe zu p .

Darstellungsformen: goto-Programm, gerichteter Graph

Def: Konfiguration von M : $(q, w) \in Q \times \Sigma^*$

Intuitiv: M hat Zustand q und liest noch den Suffix w

Def: Akzeptierte Sprache $L(M) := \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \in F\}$

Intuitiv: Menge aller Wörter, die M akzeptiert

Klasse regulärer Sprachen: $\mathcal{L}_{EA} := \{L(M) \mid M \text{ ist EA}\}$

Def: Klasse $Kl[p] := \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) = p\}$

Klassen bilden eine Partition von Σ^* . Ähnlich zu Äquivalenzklassen aus DM.

Lemma: $L(M) = \bigcup_{p \in F} Kl[p]$

(q, w) Endkonfiguration $\stackrel{\text{def}}{\iff} (q, w) \in Q \times \{\lambda\}$

Def: Schritt $:= \overline{}_M \subseteq (Q \times \Sigma^*) \times (Q \times \Sigma^*)$

wobei $(q, w) \overline{}_M (p, x) \stackrel{\text{def}}{\iff} w = ax \wedge a \in \Sigma \wedge \delta(q, a) = p$
 Intuitiv: Übergangsfunktion auf M im Zustand q anwenden, bei a .

Def: Berechnung $C := C_0, \dots, C_n$ s.d. C_i Konfiguration

wobei $\forall i \leq n-1 : C_i \overline{}_M C_{i+1}$ gilt

Def: Relationen bzgl. endlichen Automaten

$(q, w) \overline{}_M^* (p, u) \stackrel{\text{def}}{\iff} \exists \text{ Berechnung in } M \text{ von } (q, w) \text{ zu } (p, u).$

Die formelle Definition ist sehr lang. (S.54)

$\hat{\delta}(q, w) = p \stackrel{\text{def}}{\iff} (q, w) \overline{}_M^* (p, \lambda)$

Intuitiv: Wenn M in Zustand q Wort w liest, endet M in p

Lemma: $\forall \odot \in \{\cup, \cap, -\} \exists M : L(M) = L(M_1) \odot L(M_2)$

Für alle EA M_1, M_2 über Σ

Wegen diesem Lemma ist es möglich, EAs aus Teilautomaten zu bauen

2.1 Irregularität beweisen

Einige Ansätze um Aussagen der Art $L \notin \mathcal{L}_{EA}$ zu beweisen:

Lemma: (Direkt via Zustände) Sei $A = (Q, \Sigma, \delta_A, q_0, F)$, $x \neq y \in \Sigma^*$:

$$\exists p \in Q : \underbrace{(q_0, x) \overline{}_A^* (p, \lambda) \wedge (q_0, y) \overline{}_A^* (p, \lambda)}_{\delta_A(q_0, x) = \delta_A(q_0, y) = p \text{ und } x, y \in Kl[p]} \implies \forall z \in \Sigma^*, \exists r \in Q : xz \in L(A) \iff yz \in L(A)$$

Intuitiv: Wenn man für zwei (auch unterschiedliche!) Eingaben die selbe Konfiguration erreicht, ist der weitere Verlauf identisch. Dieses Lemma formalisiert die intuitiv klare "Gedächtnislosigkeit" von EAs, d.h. dass ein EA keinen Speicher (ausser dem aktuellen Zustand) besitzt.

Lemma: (Pumping) für $L \in \mathcal{L}_{EA}$: $\exists n_0 \in \mathbb{N}$ s.d. $\forall w \in \Sigma^*$ mit $|w| \geq n_0 : \exists x, y, z : w = yxz$ und:

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

Intuitiv: Alle Wörter länger als n_0 lassen sich als $w = yxz$ zerlegen: wenn w (nicht) akzeptiert wird müssen alle anderen $w_k = yx^kz$ auch (nicht) akzeptiert werden. Ein n_0 welches diese Zerlegung erlaubt existiert immer, wenn L regulär ist.

Theorem: (Kolmogorov) Sei $L \subseteq (\Sigma_{\text{bool}})^*$ regulär. Sei $L_x = \{y \in (\Sigma_{\text{bool}})^* \mid xy \in L\}$, y_n das n -te Wort in L_x

$$\exists c \quad \forall x, y \in (\Sigma_{\text{bool}})^* : K(y_n) \leq \lceil \log_2(n+1) \rceil + c$$

Intuitiv: Suffixe von Wörtern einer regulären Sprache besitzen eine kleine Kolmogorov-Komplexität. Man versucht meistens eine unendliche Menge unterschiedlicher y_1 zu finden, was dann einen Widerspruch bildet zu diesem Satz.

2.2 Nicht-deterministische endliche Automaten (NEA)

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

Übergangsfunktion $\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$

Intuitiv: δ gibt alle möglichen Zustände, statt nur Einen.

Def: Relationen bzgl. NEAs

$\hat{\delta}(q, \lambda) := \{q\} \quad \forall q \in Q$

$\hat{\delta}(q, wa) := \{p \in Q \mid \exists r \in \hat{\delta}(q, w) : p \in \delta(r, a)\}$

Intuitiv: $\hat{\delta}$ gibt alle möglichen Endzustände, statt nur Einen.

Def: Akzeptierte Sprache in NEAs

$$L(M_{\text{NEA}}) := \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset\}$$

Intuitiv: Alle Wörter mit möglichen Berechnungsweg zu $q \in F$.

D.h. Akzeptierte Wörter müssen nicht immer akzeptiert werden.

Theorem: (3.2) Potenzmengen Konstruktion

$$\mathcal{L}_{EA} = \mathcal{L}_{\text{NEA}}$$

Intuitiv: Für jeden NEA gibt es einen äquivalenten EA.

3 Turing-Maschinen

Eine Formalisierung des Begriffs "Algorithmus".

Def: Turing Maschine $M := (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$

Zustände	Q (endlich)
Eingabealphabet	Σ (Alphabet)
Arbeitsalphabet	Γ s.d. $\Sigma \subset \Gamma, \Gamma \cap Q = \emptyset$
Anfangszustand	$q_0 \in Q$
Akzeptierende Zustände	$F \subseteq Q$
Übergangsfunktion	$\delta : Q \times \Sigma \rightarrow Q$

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

Beispiel: $\epsilon w_1 q a w_2 \in \text{Konf}(M)$ heisst:

M in Zustand q , hat Kopf bei $|w_1| + 1$ auf a . Bandinhalt: $\epsilon w_1 a w_2$

Def: Äquivalenz TMs A, B s.d. $\Sigma_A = \Sigma_B$:

1. $x \in L(A) \iff x \in L(B)$
2. A hält nicht auf $x \iff B$ hält nicht auf x

D.h. $\neg(L(A) = L(B)) \implies A, B$ äquivalent).

Theorem: (Church'sche These) Turing-Maschinen formalisieren tatsächlich das intuitive Konzept "Algorithmus".

Paraphrasiert, bedeutet dass das Modell der TMs (vermutlich) alle möglichen Algorithmen abbildet.

3.1 Mehrband Turing-Maschinen

Def: Mehrband TMs

1. Endliche Kontroll-logik
2. Endliches Eingabeband
3. k nach rechts unendliche Arbeitsbänder

Die Formelle Definition im Skript ist sehr lang.

Intuitiv bleiben alle Definitionen gleich, akzeptieren aber nun k Bänder.

Lemma: $\forall \text{ TM } A : \exists \text{ MTM } B \text{ s.d. } A, B \text{ äquivalent.}$

Lemma: $\forall \text{ MTM } B : \exists \text{ TM } A \text{ s.d. } A, B \text{ äquivalent.}$

Theorem: TMs und MTMs sind äquivalente Modelle.

D.h. es existiert immer eine äquivalente Maschine im jeweils anderen Modell.

3.2 Nicht-deterministische TMs

Definitionen analog zu TMs, $w \in L(\text{NTM})$ falls irgendeine akzeptierende Berechnung existiert.

Theorem: $\forall \text{ NTM } M : \exists \text{ TM } A \text{ s.d.}$

1. $L(M) = L(A)$
2. A hält immer falls M keine unendlichen Berechnungen hat

D.h. auch NTMs sind konzeptuell äquivalent zu regulären TMs.

3.3 Sprach-Klassen

Relevant für Kapitel 5.

Def: Rekursiv aufzählbar $\mathcal{L}_{\text{RE}} := \{L(M) \mid M \text{ ist TM}\}$

Def: Rekursiv entscheidbar $\mathcal{L}_{\text{R}} := \{L(M) \mid M \text{ ist TM, hält immer}\}$

Lemma: $L \in \mathcal{L}_{\text{RE}} \wedge L^c \in \mathcal{L}_{\text{RE}} \iff L \in \mathcal{L}_{\text{R}}$

Sehr nützlich für Beweise der Form $L \notin \mathcal{L}_{\text{RE}}$.

4 Berechenbarkeit

Methoden zur Klassifizierung Algorithmischer Lösbarkeit.

4.1 Diagonalisierung

DM Repetition.

Def: Mächtigkeit

$$\begin{aligned} |A| \leq |B| &\stackrel{\text{def}}{\iff} \exists f : A \rightarrow B \text{ injektiv} \\ |A| = |B| &\stackrel{\text{def}}{\iff} |A| \leq |B| \wedge |B| \leq |A| \\ |A| < |B| &\stackrel{\text{def}}{\iff} |A| \leq |B| \wedge \neg |B| \leq |A| \end{aligned}$$

Lemma: $A \subset B \implies |A| \leq |B|$

Lemma: \leq ist Transitiv.

Def: Abzählbarkeit $\stackrel{\text{def}}{\iff} |A| = |\mathbb{N}| \vee A \text{ endlich}$

Lemma: $\forall \Sigma : \Sigma^*$ ist abzählbar

Intuitiv: Da Σ endlich ist.

Weitere abzählbare Mengen: $\mathbb{Z}, \mathbb{N}^k, \mathbb{Q}, \text{KodTM}$

Überabzählbare Mengen: $\mathbb{R}, [0, 1], \mathcal{P}((\Sigma_{\text{bool}})^*)$

Theorem: $|\text{KodTM}| \leq \mathcal{P}((\Sigma_{\text{bool}})^*)$

D.h. existieren unendliche viele nicht rekursiv aufzählbare Sprachen.

4.2 Reduktion

Ansatz für Beweise von Aussagen der Form $L \in \mathcal{L}_R$ oder $L \notin \mathcal{L}_R$.

Def: Rekursive Reduzierbarkeit

$$L_1 \leq_R L_2 \stackrel{\text{def}}{\iff} L_2 \in \mathcal{L}_R \implies L_1 \in \mathcal{L}_R$$

D.h. L_2 zu lösen, bedeutet auch L_1 zu lösen.

Def: Eingabe-zu-Eingabe Reduzierbarkeit

$$L_1 \leq_{EE} L_2 \stackrel{\text{def}}{\iff} \exists M \text{ (TM)} : \exists f_M : \Sigma_1^* \rightarrow \Sigma_2^* \text{ s.d. } x \in L_1 \iff f_M(x) \in L_2$$

D.h. Es existiert eine TM M , die eine Abbildung f_M darstellt, mit welcher man L_1 via L_2 direkt bestimmen kann.

Lemma: $L_1 \leq_{EE} L_2 \implies L_1 \leq_R L_2$

Gilt nicht umgekehrt!

Lemma: \leq_{EE} ist Transitiv.

Lemma: $\forall L \subseteq \Sigma^* : L \leq_R L^C \wedge L^C \leq_R L$

Also $L \in \mathcal{L}_R \iff L^C \in \mathcal{L}_R$

Lemma: $\mathcal{L}_R \subsetneq \mathcal{L}_{RE}$

Def: Universelle Sprache

$$L_U := \{\text{Kod}(M)\#w \mid w \in \Sigma_{\text{bool}}^* \wedge w \in L(M)\}$$

Theorem: $L_U \in \mathcal{L}_{RE}$ aber $L_U \notin \mathcal{L}_R$

D.h. man kann nicht (in endlicher Zeit) prüfen, ob M ein Wort w akzeptiert.

Def: Halteproblem

$$L_H := \{\text{Kod}(M)\#x \mid x \in \Sigma_{\text{bool}}^* \wedge M \text{ hält auf } x\}$$

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

D.h. man kann nie wissen, ob eine Turingmaschine anhalten wird.

Def: $L_{\text{Empty}} := \{\text{Kod}(M) \mid L(M) = \emptyset\}$

Theorem: $(L_{\text{Empty}})^C \in \mathcal{L}_{RE}$ aber $(L_{\text{Empty}})^C \notin \mathcal{L}_R$

Def: Äquivalenzproblem

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

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

D.h. man kann nicht 2 TMs auf Äquivalenz prüfen, in endlicher Zeit.

TODO: reformat Def, Theorems as a table for languages

4.3 Rice

Ansatz für Beweise von Aussagen der Form $L \notin \mathcal{L}_R$, für $L \subseteq \text{KodTM}$.

Def: Semantisch nicht-triviales Entscheidungsproblem über TMs

$$L \subseteq \text{KodTM} : \underbrace{(\exists \text{ TM } M_1 : \text{Kod}(M_1) \in L)}_{L \neq \emptyset} \wedge \underbrace{(\exists \text{ TM } M_2 : \text{Kod}(M_2) \notin L)}_{L \neq \text{KodTM}} \wedge \underbrace{(\forall \text{ TM } A, B : L(A) = L(B) \implies (A \in L \iff B \in L))}_{L \text{ behandelt semantisch gleiche TMs gleich}}$$

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

Theorem: $L_{H, \lambda} \notin \mathcal{L}_R$

Theorem: Satz von Rice: Alle sem. nicht-triv. Probleme L sind unentscheidbar. ($L \notin \mathcal{L}_R$)

D.h. es reicht aus zu zeigen, dass $L \subseteq \text{KodTM}$ die Bedingungen oben erfüllt, um $L \notin \mathcal{L}_R$ zu zeigen.

4.4 Kolmogorov

Theorem: Unlösbarkeit von Kolmogorov: Das Problem, $\forall x \in (\Sigma_{\text{bool}})^*$ die Komplexität $K(x)$ zu berechnen, ist unlösbar.

Ein Alternativer Ansatz um Unlösbarkeit zu zeigen, unabhängig von Diagonalisierung.

5 Komplexität

Eine Formalisierung der "Schwierigkeit" von Algorithmisch lösbaren Problemen.

5.1 Zeit & Speicher

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

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

Intuitiv: Die Komplexität im Schlechtesten Fall einer Eingabe der Länge n

Wobei: M immer hält, $x \in \Sigma^*$ und $D = C_1 C_2 \dots C_k$ Die Berechnung von M auf x

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

Intuitiv: Die Länge des längsten Arbeitsbandes in M , bei der Konfiguration C .

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

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

Wobei M eine k -Band MTM, $C = (q, x, i, \alpha_1, i_1, \dots, \alpha_k, i_k)$ eine Konfiguration.

Lemma: $\forall k\text{-MTM } A : \exists \text{ äquivalente 1-MTM } B : \text{Space}_B(n) \leq \text{Space}_A(n)$

Lemma: $\forall k\text{-MTM } A : \exists \text{ äquivalente } k\text{-MTM } B : \text{Space}_B(n) \leq \frac{\text{Space}_A(n)}{2} + 2$

Intuitiv: Die Speicherkomplexität von M lässt sich für jedes $d \in \mathbb{N}$ um den Faktor d verkleinern. Das selbe gilt für $\text{Time}_M(n)$.

Theorem: $\exists(L, \Sigma_{\text{bool}}) \forall \text{ MTM } A \text{ s.d. } L(A) = L : \exists \text{ MTM } B \text{ s.d. } L(B) = L \text{ und } \text{Time}_B(n) \leq \log_2(\text{Time}_A(n))$

Intuitiv: Es gibt Probleme, wobei wir einen Lösungsalgorithmus unendlich oft signifikant verbessern können. D.h. macht es keinen Sinn allgemein von einem "Besten Algorithmus" für ein Problem zu reden.

5.2 O-Notation

Def: $\mathcal{O}(f(n)) := \{r : \mathbb{N} \rightarrow \mathbb{R}^+ \mid \exists n_0 \in \mathbb{N}, \exists c \in \mathbb{N} \text{ s.d. } \forall n \geq n_0 : r(n) \leq c \cdot f(n)\}$

Def: $\Omega(f(n)) := \{r : \mathbb{N} \rightarrow \mathbb{R}^+ \mid \exists n_0 \in \mathbb{N}, \exists c \in \mathbb{N} \text{ s.d. } \forall n \geq n_0 : r(n) \geq \frac{1}{c} \cdot f(n)\}$

Def: $\Theta(f(n)) := \mathcal{O}(f(n)) \cap \Omega(f(n))$

Def: $\mathcal{o}(f(n)) := \{r : \mathbb{N} \rightarrow \mathbb{R}^+ \mid \lim_{n \rightarrow \infty} \frac{r(n)}{f(n)} = 0\}$

Intuitiv: f wächst asymptotisch schneller als r .

5.3 Komplexitätsklassen

$f, g : \mathbb{N} \rightarrow \mathbb{R}^+$

Def: $\text{TIME}(f) := \{L(M) \mid M \text{ s.d. } \text{Time}_M(n) \in \mathcal{O}(f(n))\}$

Def: $\text{SPACE}(f) := \{L(M) \mid M \text{ s.d. } \text{Space}_M(n) \in \mathcal{O}(f(n))\}$

Def: $\text{P} := \bigcup_{c \in \mathbb{N}} \text{TIME}(n^c)$

Def: $\text{PSPACE} := \bigcup_{c \in \mathbb{N}} \text{SPACE}(n^c)$

Def: $\text{EXPTIME} := \bigcup_{d \in \mathbb{N}} \text{TIME}(2^{n^d})$

Lemma: $\forall t : \mathbb{N} \rightarrow \mathbb{R}^+ : \text{TIME}(t(n)) \subseteq \text{SPACE}(t(n))$

Lemma: $\text{DLOG} \subseteq \text{P} \subseteq \text{PSPACE} \subseteq \text{EXPTIME}$

TODO: Konstruierbarkeit

5.4 Nicht-deterministische Komplexität

M := Nicht deterministische (M)TM. $C = C_1 \dots C_m$ ist eine akzeptierende Berechnung auf x .

Def: $\text{Time}_M(x) := \text{Länge kürzester akzept. Berechnung für } x$.

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

Def: $\text{Space}_M(C) := \max\{\text{Space}_M(C_i) \mid i \leq m\}$

Def: $\text{Space}_M(x) := \min\{C \mid C \text{ akzeptiert } x\}$

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

Def: Komplexitätsklassen: **NTIME**, **NSPACE**, **NLOG**, **NP**, **NSPACE** analog zur deterministischen Definition.

Theorem: **NP** = **VP** (Polynomielle Verifizierer)

D.h. ein polynomieller Verifizierer für L beweist direkt, dass L in **NP** ist.

5.5 NP-Vollständigkeit

Unter der Annahme: $P \subsetneq NP$, kann man Beweise der Form $L \notin P$ machen.

Def: Polynomielle Reduzierbarkeit

$L_1 \leq_p L_2 \stackrel{\text{def}}{\iff} \exists \text{ polynomielle } M \text{ s.d. } \forall x \in (\Sigma_1)^* : x \in L_1 \iff M(x) \in L_2$

Intuitiv: EE-Reduktion, muss aber polynomielle Zeitkomplexität haben.

Def: NP-Schwer L s.d. $\forall L' \in NP : L' \leq_p L$

L NP-Schwer bedeutet *nicht*, dass L in NP ist.

Def: NP-Vollständigkeit L s.d. $L \in NP$ und NP-Schwer

Lemma: $\exists L : L \in P \wedge \text{NP-Schwer} \implies P = NP$

Ein NP-Schweres Problem polynomiell zu lösen bedeutet alle NP-Probleme polynomiell zu lösen.

Lemma: $L_1 \leq_p L_2 \implies (L_1 \text{ NP-schwer} \implies L_2 \text{ NP-schwer})$

D.h. Mit P-Reduktionen kann man beweisen, dass L_2 NP-Schwer ist.

Def: SAT $:= \{x \in (\Sigma_{\text{logic}})^* \mid x \text{ kodiert erfüllbare Formel in KNF}\}$

Theorem: (Cook) SAT ist NP-vollständig.

Der Beweis ist sehr lang. Im Endeffekt bedeutet dies, Boole'sche Formeln sind enorm ausdrucksstark.

Weitere NP-Schwere Probleme: SAT-Variationen (3SAT, E3SAT), Clique, Vertex-Cover, Dominating Sets

5.6 Klausel-Formeln

Nützliche Gleichungen für Beweise mit KNF-Formeln

6 Grammatiken

Wurden in HS25 nur kurz angesprochen.

Def: Grammatik $G := (\Sigma_N, \Sigma_T, P, S)$

Nicht-Terminale Σ_N

Terminale Σ_T

s.d. $\Sigma_N \cap \Sigma_T = \emptyset$

Startsymbol $S \in \Sigma_N$

Ableitungsregeln $P \subseteq \Sigma^* \Sigma_N \Sigma^* \times \Sigma^*$

Wobei $\Sigma := \Sigma_N \cup \Sigma_T$