

Komplexitätstheorie

Sommersemester 2020

Prof. Dr. Georg Schnitger
AG Theoretische Informatik

Johann Wolfgang Goethe-Universität Frankfurt am Main

Herzlich willkommen!!!

Kapitel 1: Einführung

(I) Der erste Teil: **(Nicht-)Deterministische** Berechnungen.

- ▶ Wie ist **Laufzeit** zu messen? Average-Case, Smoothed oder Worst-Case?
- ▶ Nichtdeterministische und alternierende Berechnungen:
 - ★ **Nichtdeterminismus**:
Rate eine Lösung, um zu zeigen dass eine Eigenschaft erfüllt werden kann.
 - ★ **co-Nichtdeterminismus**:
Verifiziere, dass *alle potentiellen* Lösungen eine Eigenschaft erfüllen.
 - ★ **Alternation**: Rate und verifiziere munter durcheinander
- ▶ Speicherplatz:
 - ★ **PSPACE-Vollständigkeit**: Die Bestimmung von Gewinnstrategien in 2-Personen Spielen, die Bestimmung von minimalen NFAs.
 - ★ Wie mächtig sind probabilistische bzw. Quantenberechnungen?
- ▶ Parallelität:
 - ★ **P-Vollständigkeit**: Probleme in P ohne superschnelle parallele Algorithmen
 - ★ Ein Zusammenhang zwischen Speicherplatz und paralleler Zeit.

Die beiden letzten Teile: Randomisierung und Quantenberechnungen

(II) Randomisierung.

- ▶ **BPP**: Was gelingt in polynomieller Zeit mit randomisierten, d.h. würfelnden Algorithmen?
- ▶ **Kryptographie**
 - ★ One-Way-Funktionen: Pseudo-Random-Generatoren, kollisionsresistente Hashfunktionen (Authentifizierung)
 - ★ Trapdoor-Funktionen: Public-Key-Kryptographie
 - ★ Gitter-Kryptographie: Verschlüsselungsmethoden, die vermutlich sicher gegen Quantenattacken sind.

(III) Quanten-Berechnungen

- ▶ **BQP**: Alle Sprachen, die Quantenrechner in polynomieller Zeit akzeptieren.
 - ★ Was bringen Quantenberechnungen im Vergleich zu randomisierten Berechnungen, d.h. ist **BQP** größer als **BPP**?
- ▶ Quanten-Suche: Grover's Algorithmus
- ▶ Effiziente Faktorisierung: Shor's Algorithmus

Worauf wird aufgebaut?

- Notwendig: Bachelor-Veranstaltungen wie etwa **ALGO 1** und **ALGO 2** für den Entwurf und die Analyse von Algorithmen.
 - ▶ Laufzeitanalyse: O , o , Ω , ω and Rekursionsgleichungen
 - ▶ Traversierung von Graphen
 - ▶ Dynamische Programmierung
 - ▶ NP-Vollständigkeit
 - ▶ Berechenbarkeit
- Spezielles mathematisches Vorwissen wird nicht verlangt, wohl aber die Fähigkeit, mathematische Beweise verstehen und führen zu können.

- (1) S. Arora und B. Barak, Computational Complexity, a Modern Approach, Cambridge University Press, 2009.
- (2) M. Sipser, Introduction to the Theory of Computation, Paperback 3rd edition, Cengage Learning, 2012.
- (3) Skript zur Vorlesung „Komplexitätstheorie“, Goethe-Universität Frankfurt.

- The blog of Scott Aaronson
- Lance Fortnow und Bill Gasarch, [Computational Complexity Blog](#)
- R.J. Lipton, [Goedel's lost letter and \$P = NP\$](#) ,
- L. Trevisan, [in theory](#)

- Die Webseite der Veranstaltung enthält alle wichtigen Informationen zur Veranstaltung wie Skript, Folien, Übungsblätter, Videos, usw.

www.thi.informatik.uni-frankfurt.de/lehre/kth/sose20.de

- ▶ Unter **Aktuelles** finden Sie die „neuesten“ Mitteilungen,
- ▶ Im **Logbuch** wird beschrieben, welche Teile vom Skript dem jeweiligen Video zu grundeliegen und weitere Referenzen werden angegeben.
- Mündliche Prüfungen: In den beiden ersten Wochen nach Semesterende.
- Übungsbetrieb: BITTE **UNBEDINGT** TEILNEHMEN!
 - ▶ Das aktuelle Übungsblatt erscheint auf der Webseite spätestens Montags.
 - ▶ Lösungen, nach 1-wöchiger Bearbeitungszeit, bitte am darauffolgenden Montag auf einer PDF-Datei schicken an holldack@em.uni-frankfurt.de.
 - ★ In LaTeX-gesetzte Lösungen wären toll.
 - ▶ Bei 50% (bzw 70%) aller Übungspunkte wird die in der mündlichen Prüfung erreichte Note um einen (bzw. zwei) Notenschritte verbessert.
- **Fragen** und **Kommentare** bitte schicken an georg.schnitger@gmail.com
 - ▶ Ich muss wissen, wo der Schuh drückt.

Komplexitätsklassen

Komplexitätsklassen für verschiedenste Ressourcen

Eine Komplexitätsklasse

zu einem Rechnermodell \mathcal{R} und (mehreren) Ressourcen

besteht aus allen Sprachen mit eingeschränktem Ressourcenverbrauch auf \mathcal{R} .

Zuerst untersuchen wir algorithmische Entscheidungsprobleme zu einem deterministischen Rechnermodell für die

- **Zeitkomplexität:**
 - ▶ die Worst-Case-Laufzeit sequentieller Algorithmen,
- **Speicherplatzkomplexität:**
 - ▶ der Worst-Case-Speicherplatz sequentieller Algorithmen,
- **Schaltkreiskomplexität:**
 - ▶ Tiefe und/oder Größe von Schaltkreisen
- sowie **Approximationskomplexität:**
 - ▶ Die erreichbare Approximationsqualität effizienter Algorithmen.

Später betrachten wir auch Komplexitätsklassen zu nichtdeterministischen, probabilistischen Rechnermodellen wie auch für Quantenrechner.

Zeit-Komplexitätsklassen

Sequentielle, deterministische Algorithmen

Irgendein vernünftiges Rechnermodell

Turingmaschinen bzw. Registermaschinen

sei gegeben.

- Für einen Algorithmus A ist
 - ▶ $L(A)$ die Menge aller akzeptierten Eingaben und
 - ▶ $\text{time}_A(w)$ die von A auf Eingabe w benötigte Schrittzahl.
- Wie soll Laufzeit gemessen werden?
 - ▶ *Average-Case-Komplexität* – also die Laufzeit im Mittel, aber bei welcher Eingabeverteilung?
 - ▶ *Worst-Case-Komplexität* – also die längste Laufzeit einer Eingabe vorgegebener Größe,
 - ▶ *Geglättete Komplexität (engl.: Smoothed Complexity)*: Erwartete Worst-Case-Laufzeit *nach* leichtem zufälligen Verrauschen der Eingabe.

Average-Case-Komplexität

Siehe auch Bogdanov, Trevisan <https://arxiv.org/abs/cs/0606037>

Welche Verteilungen auf der Menge der Eingaben sollte man betrachten?

- (a) Für jedes $n \in \mathbb{N}$ sei D_n eine Verteilung über Σ^* . Dann ist die Familie

$$\mathcal{D} = (D_n \mid n \in \mathbb{N})$$

ein *Ensemble*. Häufig: D_n ist eine Verteilung über Σ^n .

- (b) Ein Ensemble $\mathcal{D} = (D_n \mid n \in \mathbb{N})$ heißt *effizient simulierbar*, wenn

$$\text{prob}[A(n) = x] = D_n(x)$$

für einen – im Worst-Case – effizienten randomisierten Algorithmus A .

Ensembles sind für asymptotische Aussagen

also für Aussagen über Skalierbarkeit

maßgeschneidert.

Beispiel: Für die Gleichverteilung U_n auf Eingaben der Länge n ist

$$\mathcal{U} = (U_n \mid n \in \mathbb{N})$$

das *uniforme Ensemble*.

Wozu Ensembles? Um Verteilungsprobleme untersuchen zu können:

- (a) Für eine Sprache L und ein Ensemble \mathcal{D} heißt (L, \mathcal{D}) ein *Verteilungsproblem*.
- (b) *AVGNP* (häufig auch *DistNP*) ist die Klasse aller Verteilungsprobleme (L, \mathcal{D}) für eine Sprache $L \in \text{NP}$ und ein effizient simulierbares Ensemble \mathcal{D} .

AVGP

Sei $\mathcal{D} = (D_n \mid n \in \mathbb{N})$ ein Ensemble, so dass $D_n(x) > 0 \implies x \in \Sigma^n$.

(a) Sei A ein Algorithmus mit Laufzeit $t_A(x)$ für Eingabe x . Dann ist

$$\sum_{x \in \Sigma^n} D_n(x) \cdot t_A(x)$$

die erwartete Laufzeit von A bezüglich \mathcal{D} .

(b) A besitzt polynomielle erwartete Laufzeit für das Ensemble \mathcal{D} , falls es eine Konstante $\varepsilon > 0$ und ein Polynom $p(n)$ gibt, so dass

$$\text{prob}_{x \sim D_n} [t_A(x) \geq t] \leq \frac{p(n)}{t^\varepsilon}$$

für alle $n, t \in \mathbb{N}$ gilt.

(c) Die Komplexitätsklasse

AVGP

besteht aus allen Verteilungsproblemen $(L; \mathcal{D})$ mit $L = L(A)$ für einen Algorithmus A mit polynomieller erwarteter Laufzeit für \mathcal{D} .

Für den Algorithmus A gelte

$$\text{prob}_{x \sim D_n} [t_A(x) \geq t] \leq \frac{p(n)}{t^\varepsilon}$$

für ein Polynom p .

- Wenn $t_A(x) \leq t_B(x)^k$ für $k \in \mathbb{N}$ und Algorithmus B polynomielle erwartete Laufzeit hat, dann auch Algorithmus A .
- $t_A(x) \leq (2 \cdot p(n))^{1/\varepsilon}$ gilt mit Wahrscheinlichkeit mindestens $\frac{1}{2} \implies p(n)^{1/\varepsilon}$ spielt die Rolle einer oberen Schranke für die erwartete Laufzeit.
 - ▶ Die Wahrscheinlichkeit für eine Laufzeit größer als $(c \cdot p(n))^{1/\varepsilon}$ ist durch $\frac{1}{c}$ nach oben beschränkt.

BOUNDED-HALTING

Reduktionen zwischen Verteilungsproblemen

Verteilungsproblem (L_1, \mathcal{D}_1) ist auf (L_2, \mathcal{D}_2) Average-Case-reduzierbar

$$(L_1, \mathcal{D}_1) \leq_{\text{Avg}} (L_2, \mathcal{D}_2),$$

wenn es eine Transformation f mit den folgenden Eigenschaften für alle n gibt:

1. $f(x, n)$ ist in polynomieller Zeit (in n) durch einen deterministischen Algorithmus berechenbar, falls $D_{1,n}(x) > 0$ für Eingabe x gilt,
2. $x \in L_1 \iff f(x, n) \in L_2$ für alle Eingaben x mit $D_{1,n}(x) > 0$ und
3. es Polynome p, q gibt, s. d. für alle Eingaben y für L_2 (mit $D_{2,p(n)}(y) > 0$)

$$\sum_{x, f(x,n)=y} D_{1,n}(x) \leq q(n) \cdot D_{2,p(n)}(y).$$

Wird y gemäß D_2 zufällig gewählt, dann geschieht dies – bis auf ein Polynom – mit mindestens der Wahrscheinlichkeit mit der y „über f “ gezogen wird.

BOUNDED-HALTING

Definiere das beschränkte Halteproblem

$BH := \{ (M, x, t) : \text{die nichtdeterministische Turingmaschine } M \text{ hält auf Eingabe } x \text{ in höchstens } t \text{ Schritten} \}$

mit zugehörigem Ensemble $\mathcal{V} := (V_n : n \in \mathbb{N})$, wobei V_n „im wesentlichen“ eine „Gleichverteilung“ auf allen Tripeln (M, x, t) der Länge n definiert.

Die Average-Case-Reduktion ist eine partielle Ordnung mit

$BOUNDED-HALTING := (BH, \mathcal{V})$

als vollständigem Problem für AVG_{NP} .

Leider gibt es keine vollständigen Probleme von praktischer Relevanz:
Mgl. gibt es keine vollständigen Probleme (L, \mathcal{U}) zur Gleichverteilung.

Notorisch-schwierige Verteilungsprobleme: Random- k -SAT

Random- k -SAT

Eingaben von k -SAT $_{n,m}$ sind KNFs ϕ mit genau k Literalen pro Klausel, den Variablen X_1, \dots, X_n und m Klauseln. Entscheide, ob ϕ erfüllbar ist.

- Ziehe m Klauseln (mit k Literalen) zufällig gemäß der Gleichverteilung.
 - ▶ Wenn m „klein genug“ oder „zu groß“ ist:
Hochwahrscheinlich richtige Antwort mit *erfüllbar* bzw. *unerfüllbar*.
 - ▶ Wann wird es richtig schwer? Für $k = 3$ und $f(n, 3) \approx 4.2667n$ Klauseln:
Ein **Phasenübergang** von Erfüllbarkeit hin zur Unerfüllbarkeit findet statt.
- Für allgemeines k finde der Phasenübergang für $f(n, k)$ Klauseln statt.
 - ▶ Für Gleichverteilungen $U_{n,k}$ auf $f(n, k)$ Klauseln ist $\mathcal{U}_k := (U_{n,k} | n \in \mathbb{N})$.
 - ▶ Wie schwierig ist das Verteilungsproblem

$$\text{Random-}k\text{-SAT} := (\mathcal{U}_k, (k\text{-SAT}_{n,f(n,k)} | n \in \mathbb{N})).$$

Vermutlich erlaubt *Random k -SAT* im Mittel keine effizienten Algorithmen. Aber Vollständigkeitsergebnisse sind nicht bekannt.

Notorisch-schwierige Verteilungsprobleme: Bounded-Distance-Decoding

Bounded-Distance-Decoding (BDD) für lineare Codes

- Würfle zuerst eine $n \times m$ -Matrix $A \in \mathbb{Z}_2^{n \cdot m}$ (mit $m = c \cdot n$ und $c > 1$) wie auch eine „Nachricht“ $x \in \mathbb{Z}_2^n$ zufällig gemäß der Gleichverteilung aus.
- Ein Fehlervektor $e \in \mathbb{Z}_2^m$ mit **relativ wenigen** Einsen wird ebenfalls zufällig ausgewürfelt.
- Rekonstruiere x aus $y := x^T \cdot A \oplus e$. Mit anderen Worten,
 - ▶ Die zufällige, binäre Nachricht x wird mit Hilfe der Matrix A verschlüsselt und
 - ▶ das Ergebnis $z := x^T \cdot A$ wird „leicht“ mit dem Fehlervektor e gestört.

Anscheinend ist das Entschlüsselungsproblem

„Bestimme die ursprüngliche Nachricht x bei gegebenem y “

für zufällige Verschlüsselungsmatrizen A und Fehlervektoren e selbst „im Mittel“ sehr schwierig.

Schwierige Verteilungsprobleme (wie etwa (BDD)) sind wichtig für die Kryptographie
⇒ Siehe später die *Gitter-Kryptographie*

Smoothed Complexity, bzw. geglättete Komplexität

siehe auch: Tim Roughgarden, *Beyond Worst-Case Analysis*, lectures 17, 18,
<http://timroughgarden.org/w17/l/117.pdf>,
<http://timroughgarden.org/w17/l/118.pdf>

Eine Familie von Funktion $f_n : [-1, 1]^n \rightarrow \mathbb{R}$ ist für Eingabe x auszuwerten. Die Eingabe x wird durch die Normalverteilung zu $x + g$ verrauscht.

- (a) Die geglättete Komplexität von Algorithmus A für die Normalverteilung mit Erwartungswert 0 und Standardabweichung σ ist

$$\text{Smoothed}_A^\sigma(n) = \max_{x \in [-1, 1]^n} \mathbb{E}_g [t_A(x + g)].$$

- (b) A hat **polynomielle geglättete Komplexität**, wenn für alle genügend großen Eingabelängen $n \in \mathbb{N}$, für alle genügend kleinen σ und für geeignete $k_1, k_2 \in \mathbb{N}$ gilt

$$\text{Smoothed}_A^\sigma(n) = \mathcal{O}\left(\frac{n^{k_1}}{\sigma^{k_2}}\right).$$

Smoothed Complexity: Killerapplikationen

- 1 Die ***k*-Means-Methode** für das Cluster-Problem hat polynomielle geglättete Komplexität Arthur, Manthey und Röglin, Smoothed Analysis of the *k*-Means-Method, Journal of the ACM, 2011.
- 2 Die **2-OPT-Heuristik** für das Traveling Salesman Problem hat polynomielle geglättete Komplexität Tim Roughgarden, Lecture 17.
- 3 Der **Simplex-Algorithmus** für die lineare Programmierung (mit der „Shadow-Vertex-Regel“ als Pivot-Regel) hat polynomielle geglättete Komplexität Tim Roughgarden, Lecture 17.
- 4 Ein ***n*-dimensionales binäres Optimierungsproblem** wird durch eine Lösungsmenge $L \subseteq \{0, 1\}^n$ und nicht-negative Koeffizienten $v_1, \dots, v_n \in \mathbb{R}$ beschrieben. Es ist eine Lösung $x \in L$ gesucht, die $\sum_{i=1}^n v_i \cdot x_i$ maximiert.
 - ▶ Das Rucksackproblem ist ein binäres Optimierungsproblem.
 - ▶ Das Rucksackproblem besitzt einen pseudo-polynomiellen Algorithmus und einen Algorithmus mit polynomieller geglätteter Komplexität.

Ein binäres Optimierungsproblem besitzt genau dann pseudo-polynomielle Algorithmen, wenn es eine polynomielle geglättete Komplexität besitzt.

Worst-Case-Komplexitätsklassen

Determinismus

- 1 Algorithmus A führe auf Eingabe $x \in \Sigma^*$ genau $t_A(x)$ Schritte aus. Dann ist

$$t_A(n) := \max\{t_A(x) : x \in \Sigma^n\}$$

die Worst-Case-Laufzeit von A für Eingabelänge n .

- 2 Für die Funktion $t : \mathbb{N} \rightarrow \mathbb{N}$ ist

$$\text{DTIME}(t) := \{L \subseteq \Sigma^* : \text{es gibt einen deterministischen Algorithmus } A \text{ mit } L(A) = L \text{ und } t_A = \mathcal{O}(t)\}.$$

- 3 Die Klasse P besteht aus den in polynomieller Laufzeit erkennbaren Sprachen,

$$P := \bigcup_{k \in \mathbb{N}} \text{DTIME}(n^k).$$

Die Sprachklassen mit exponentieller Laufzeit sind

$$E := \bigcup_{k \in \mathbb{N}} \text{DTIME}(2^{k \cdot n})$$

und

$$\text{EXP} := \bigcup_{k \in \mathbb{N}} \text{DTIME}(2^{n^k}).$$

Nichtdeterminismus

- 1 Der nichtdeterministische Algorithmus A führe auf Eingabe $x \in \Sigma^*$ im Worst-Case genau $nt_A(x)$ Schritte aus. Dann ist

$$nt_A(n) := \max\{nt_A(x) : x \in \Sigma^n\}$$

die nichtdeterministische Worst-Case-Laufzeit von A für Eingabelänge n .

- 2 Für die Funktion $t : \mathbb{N} \rightarrow \mathbb{N}$ ist

$$\text{NTIME}(t) := \{L \subseteq \Sigma^* : \text{es gibt einen nichtdeterministische Algorithmus } A \text{ mit } L(A) = L \text{ und } nt_A = \mathcal{O}(t)\}.$$

- 3 Die Klasse NP besteht aus den in polynomieller Laufzeit erkennbaren Sprachen,

$$\text{NP} := \bigcup_{k \in \mathbb{N}} \text{NTIME}(n^k).$$

Die Sprachklassen zu exponentieller Laufzeit sind

$$\text{NE} := \bigcup_{k \in \mathbb{N}} \text{NTIME}(2^{k \cdot n}) \text{ und } \text{NEXP} := \bigcup_{k \in \mathbb{N}} \text{NTIME}(2^{n^k}).$$

- (a) Eine Sprache L_1 ist genau dann auf eine Sprache L_2 *polynomiell-reduzierbar* ($L_1 \leq_p L_2$), wenn für alle Eingaben x :

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

mit einem effizienten deterministischen Algorithmus T .

- (b) Die Sprache K ist genau dann *vollständig* für eine Komplexitätsklasse \mathcal{K} (unter *polynomiellen Reduktionen*), wenn
- ▶ $K \in \mathcal{K}$ und
 - ▶ $L \leq_p K$ für alle Sprachen $L \in \mathcal{K}$ gilt.

NP ist eine „erfolgreiche“ Klasse, denn sie besitzt viele interessante vollständige Probleme. Aber auch NEXP hat einiges zu bieten.

- Wir sagen, dass ein Schaltkreis C mit $3(n + 1)$ Eingabebits eine 3-KNF α mit 2^n Variablen *beschreibt*, wenn C die Kodierung einer Dreier-Klausel k genau dann akzeptiert, wenn k eine Klausel von α ist.
- **SUCCINCT-3-SAT** besteht aus allen Schaltkreisen C , so dass die von C repräsentierte KNF erfüllbar ist.
 - ▶ Achtung: Ein Schaltkreis der Größe $\text{poly}(n)$ kann eine KNF mit 2^n Variablen beschreiben!

SUCCINCT-3-SAT ist vollständig für NEXP unter polynomiellen Reduktionen.

Alternierende Berechnungen

- *Existentielle Berechnungen*: Rate eine Berechnung und verifiziere deterministisch.
 - ▶ Die Kraft des Ratens: Das Erfüllbarkeitsproblem ist einfach.
 - ▶ Die Schwäche des Ratens: Das Tautologieproblem bleibt schwierig.
- *Universelle Berechnungen*: Erzeuge Berechnungen „parallel“ und verifiziere alle Berechnungen deterministisch.
 - ▶ Die Kraft des Verifizierens: Das Tautologieproblem ist einfach.
 - ▶ Die Schwäche des Verifizierens: Das Erfüllbarkeitsproblem bleibt schwierig.
- *Alternierende Berechnungen* dürfen beliebig zwischen existentiellen und universellen Berechnungen hin und her wechseln.

Ein alternierender Algorithmus A besitzt eine Menge

Q_{\exists} *existentieller Zustände*

und eine Menge

Q_{\forall} *universeller Zustände.*

Der **Berechnungsbaum** $\mathcal{B}_A(x)$ für eine Eingabe x :

- Die Wurzel von $\mathcal{B}_A(x)$ ist mit der Anfangskonfiguration beschriftet.
 - ▶ Eine Konfiguration besteht aus dem aktuellen Zustand und dem aktuellen Speicherinhalt.
- Wenn Knoten v von $\mathcal{B}_A(x)$ mit Konfiguration k beschriftet ist, dann hat v für jede in einem Schritt erreichbare Konfiguration k' ein Kind, das mit k' beschriftet ist.
- Knoten v von $\mathcal{B}_A(x)$ ist existentiell bzw. universell, wenn „sein“ Zustand q_v existentiell bzw. universell ist.

Wann akzeptiert der alternierende Algorithmus A eine Eingabe x ?

- Ein Blatt v von $\mathcal{B}_A(x)$ ist genau dann akzeptierend, wenn die Konfiguration von v akzeptierend ist, und ansonsten verwerfend.
- Der Knoten v von $\mathcal{B}_A(x)$ besitze den Zustand q_v .
 - ▶ Wenn q_v ein *existentieller* Zustand ist, dann ist v genau dann akzeptierend, wenn *mindestens* ein Kind von v akzeptierend ist.
 - ▶ Wenn q_v ein *universeller* Zustand ist, dann ist v genau dann akzeptierend, wenn *alle* Kinder von v akzeptierend sind.

Wir sagen, dass A die Eingabe x genau dann akzeptiert, wenn die Wurzel von $\mathcal{B}_A(x)$ akzeptierend ist und definieren

$$L(A) := \{x : A \text{ akzeptiert } x\}.$$

A ist ein Σ_k -Algorithmus (bzw. Π_k -Algorithmus): Der Anfangszustand ist existentiell (bzw. universell) und jeder in der Wurzel von $\mathcal{B}_A(x)$ beginnende Weg alterniert höchstens $k - 1$ Mal zwischen existentiellen und universellen Zuständen.

Alternation: Komplexitätsklassen

Die Funktion $t : \mathbb{N} \rightarrow \mathbb{N}$ sei gegeben.

- (a) Der alternierende Algorithmus A benötigt höchstens Zeit $t(n)$, wenn die Tiefe von $\mathcal{B}_A(x)$ für Eingaben x der Länge n höchstens $t(n)$ beträgt.

$$\text{ATIME}(t) := \{L \subseteq \Sigma^* \mid \text{es gibt einen alternierenden Algorithmus } A \text{ mit } L(A) = L \text{ und } A \text{ benötigt } \mathcal{O}(t) \text{ Schritte}\}.$$

- (b) Die Klasse Σ_k^P (Π_k^P) besteht aus allen durch einen Σ_k -Algorithmus (Π_k -Algorithmus) mit polynomieller Laufzeit erkennbaren Sprachen.

► $\Sigma_1^P = \text{NP}$ und $\Pi_1^P = \text{coNP}$.

- (c) Die **polynomielle Hierarchie**,

$$\text{PH} := \bigcup_{k \in \mathbb{N}} \Sigma_k^P.$$

- (d) Die Klasse AP , alternierende polynomielle Zeit,

$$\text{AP} := \bigcup_{k \in \mathbb{N}} \text{ATIME}(n^k).$$

Alternation: Was bringt das Gedankenexperiment?

- Die Sprache

$$\exists_k \text{SAT} := \{ \phi : \text{die Formel } \phi = \underbrace{Q_1}_{\exists} Q_2 \cdots Q_k \alpha \text{ ist wahr, wobei } Q_i \text{ ein Block von nur } \exists\text{- bzw- } \forall\text{-Quantoren ist und } \alpha \text{ eine KNF ist} \}.$$

ist vollständig für Σ_k^P unter der polynomiellen Reduktion.

- Die Sprache aller Tautologien, also aller allgemeingültigen aussagenlogischen Formeln ist vollständig für Π_1^P unter der polynomiellen Reduktion.
- Die Sprache *Minimale-NFA* besteht aus allen NFAs N mit minimaler Zustandszahl. Dann ist

$$\text{Minimale-NFA} \in \Sigma_2^P \cap \Pi_2^P.$$

- Die Klasse AP wird sich als besonders wichtig erweisen. Zum Beispiel können nicht-triviale 2-Personen-Spiele optimal „in AP gespielt werden“.

Die Methode der Diagonalisierung, Eine Zeithierarchie

Können Berechnungen **mehr Probleme** lösen,
wenn **mehr Zeit** zur Verfügung steht?

- Wir benutzen die *Diagonalisierungsmethode* von Cantor.
 - ▶ Cantor hat diese Methode erstmalig angewandt um zu zeigen, dass die Menge der reellen Zahlen überabzählbar groß ist.
 - ▶ In der Informatik wird die Diagonalisierung z. B. für den Nachweis der Unentscheidbarkeit der Diagonalsprache oder des Halteproblems benutzt.
- Was ist zu tun?

Entwerfe einen Algorithmus mit Laufzeit $O(t(n))$, der sich von allen Algorithmen mit Laufzeit $O(t(n)/\log_2 t(n))$ unterscheidet.

Problem: Simuliere eine Turingmaschine, solange die Zeitschranke $t(n)$ nicht überschritten ist.

Lösung: $t : \mathbb{N} \rightarrow \mathbb{N}$ heißt *zeitkonstruierbar*, falls $t(n) \geq n \cdot \log_2 n$ und falls es eine det. TM gibt, die für jede Eingabe x die Binärdarstellung von $t(|x|)$ in Zeit höchstens $O(t(|x|))$ berechnet.

Zeitkontrolle:

- ▶ Zur Berechnung der Binärdarstellung von $t(n)$ steht Zeit $O(t)$ und damit exponentielle Zeit in der Länge der Binärdarstellung von t zur Verfügung.
- ▶ Initialisiere einen Zähler mit Wert $t(n)$ in Zeit $O(t(n))$.
- ▶ Halte den Zähler stets in der Nähe des Kopfes.
⇒ Zeitkontrolle in $O(\log_2 t(n))$ Schritten pro Simulationsschritt.

- (a) Die Funktion t sei zeitkonstruierbar.
Dann ist $\text{DTIME}(o(\frac{t}{\log_2 t}))$ eine echte Teilmenge von $\text{DTIME}(t)$.
- (b) P ist eine echte Teilmenge von $E = \bigcup_{k \in \mathbb{N}} \text{DTIME}(2^{k \cdot n})$.
- (c) SUCCINCT-3-SAT liegt nicht in P .

Baue eine Turingmaschine M^ mit Laufzeit $O(t)$, die sich von allen Maschinen M mit Laufzeit $o(\frac{t}{\log_2 t})$ unterscheidet.*

- 1 Der Wecker wird gestellt: M^* bestimmt die Länge n der Eingabe w und speichert die Binärdarstellung von t in einem Zähler ab.
/* Dies ist mit Laufzeit $O(t(n))$ möglich, da t zeitkonstruierbar ist. */
- 2 Wenn $w \neq \langle M \rangle 0^k$ für eine Turingmaschine M ist, verwirft M^* .
/* $\langle M \rangle$ bezeichnet die Gödelnummer – also die Programmierung – der Turingmaschine M . */
- 3 M^* simuliert M auf der Eingabe $\langle M \rangle 0^k$ und verwirft, wenn die Simulation mehr als $t(n)$ Schritte benötigt.
- 4 M^* akzeptiert (bzw. verwirft) w , wenn M verwirft (bzw. akzeptiert).

Orakel-Berechnungen

Berechnungswelten

Warum ist die $P \stackrel{?}{=} NP$ Frage immer noch unbeantwortet?

- ? Vielleicht, weil $P \neq NP$ zwar wahr, aber *nicht beweisbar* ist?
- ? Vielleicht, weil $P = NP$ in einigen „*Berechnungswelten*“ sogar wahr ist?

Sei $A \subseteq \Sigma^*$ eine Sprache.

(a) Eine TM M mit Orakel A besitzt ein zusätzliches *Orakelband*.

- ▶ Wenn das Orakelband mit der Eingabe $w\#$ beschrieben ist, dann wird in einem einzigen Berechnungsschritt mitgeteilt, ob w zur Sprache A gehört.
- ▶ Die Beschriftung des Orakelbands benötigt andererseits eine Laufzeit proportional zur Länge der Anfrage.

(b) Sei \mathcal{K} eine durch die Beschränkung einer Ressource
wie Laufzeit oder Speicherplatz

definierte Komplexitätsklasse. Dann ist \mathcal{K}^A entsprechend zu definieren, wobei jetzt Fragen an das Orakel A zugelassen sind.

Gibt es Berechnungswelten A, B mit $P^A = NP^A$ und $P^B \neq NP^B$?

- (a) Wenn $A \in P$, dann ist $P^A = P$.
- (b) Sei \mathcal{K} eine Komplexitätsklasse mit $\mathcal{K}^K = \mathcal{K}$ für alle Sprachen $K \in \mathcal{K}$. Wenn K^* vollständig für \mathcal{K} unter der polynomiellen Reduktion ist und $NP \subseteq \mathcal{K}$ gilt, dann ist

$$P^{K^*} = NP^{K^*} = \mathcal{K}$$

- (a) $P^A \subseteq P$, denn eine Turingmaschine mit Orakel A kann Orakelanfragen (mit nur polynomiellem Mehraufwand) auch selbst beantworten.
- (b)
- ▶ $NP^K \subseteq \mathcal{K}^K = \mathcal{K}$ für alle Sprachen $K \in \mathcal{K}$ ✓.
 - ▶ Die Beziehung $\mathcal{K} \subseteq P^{K^*}$ folgt aus der \mathcal{K} -Vollständigkeit von K^* :
 - ★ Für jede Sprache $L \in \mathcal{K}$ gibt es eine effiziente deterministische TM M mit

$$w \in L \Leftrightarrow M(w) \in K^*.$$

- ▶ Also $NP^{K^*} \subseteq \mathcal{K} \subseteq P^{K^*}$ und die Behauptung folgt.

Es gibt ein Orakel A mit $P^A \neq NP^A$

Konstruiere ein Orakel A , so dass die Sprache

$$L_A = \{w \mid \exists x \in A (|x| = |w|)\}$$

zu NP^A , nicht aber zu P^A gehört.

- Offensichtlich gilt $L_A \in NP^A$ für jedes Orakel A .
 - ▶ Für Eingabe w rate einen String x gleicher Länge und frage, ob $x \in A$.
 - ▶ Akzeptiere genau dann, wenn die Antwort positiv ist.
- Sei M_k eine beliebige Aufzählung aller deterministischen Orakel-Turingmaschinen, wobei M_k in Zeit höchstens $k \cdot n^k$ rechne.
 - ▶ Um $L_A \notin P^A$ zu garantieren, stellen wir sicher, dass sich M_k und L_A auf einer Eingabe $w = 1^{n_k}$ unterscheiden.
 - ▶ Wir nehmen an, dass wir dieses Ziel bereits für M_1, \dots, M_{k-1} erreicht haben:
 - ★ $\{w_1, \dots, w_m\}$ sei die Menge der während der Berechnung irgendeiner Maschine M_i auf Eingabe 1^{n_i} ($1 \leq i \leq k-1$) an das Orakel A gestellten Anfragen.
 - ★ Wie ist n_k zu definieren und wie soll das Orakel die von M_k auf Eingabe 1^{n_k} gestellten Fragen beantworten?

$$L_A = \{w \mid \exists x \in A (|x| = |w|)\}$$

- Simuliere M_k auf der Eingabe 1^{n_k} . (Es gelte $n_k > \max\{|w_1|, \dots, |w_m|\}$ und $2^{n_k} > k \cdot n_k^k$.)
 - ▶ Wenn M_k eine Anfrage y aus $\{w_1, \dots, w_m\}$ stellt, dann antwortet A konsistent.
 - ▶ Ist die Anfrage y hingegen neu, dann antwortet A mit **nein**.
- Wenn M_k die Eingabe 1^{n_k} akzeptiert:
 - ▶ Erzwinge $1^{n_k} \notin L_A$ durch Ausschluss **aller** Worte der Länge n_k für A .
- Wenn M_k die Eingabe 1^{n_k} verwirft:
 - ▶ M_k rechnet in Zeit höchstens $k \cdot n^k < 2^n$. Es gibt also ein Wort u der Länge n_k , das von M_k (für Eingabe 1^{n_k}) **nicht** nachgefragt wurde.
 - ▶ Definiere A so, dass u das einzige zu A gehörende Wort der Länge n_k ist.

- Unser Beweis der Zeit-Hierarchie „*relativiert*“, gilt also in jeder Berechnungswelt:

Im Beweis der Zeithierarchie fragen wir nicht nach, ob die simulierte Turingmaschine ein Orakelband besitzt.

- Aber die Aussage $P \neq NP$ gilt *nicht* in jeder Berechnungswelt!

- ⚡ Diagonalisierungsmethoden allein sind für einen Beweis von $P \neq NP$ unzureichend.
- ⚡ Später: Auch „natürliche Beweise“ genügen für den Nachweis von $P \neq NP$ nicht.

Zeitkomplexität: Wichtige Fragestellungen

- ? $P = NP$
 - (*) In welchem Ausmaß kann die Methode der Diagonalisierung benutzt werden? Achtung: *Orakel-Berechnungen*!
 - (*) Ist die Frage möglicherweise mit heutigen Methoden nicht beantwortbar? Später: *Natürliche Beweise* existieren nicht.
 - (*) In welchen eingeschränkten Modellen von P und NP kann die Frage beantwortet werden?
- ? Um wie viel größer ist die Berechnungskraft von randomisierten oder Quanten-Algorithmen im Vergleich zu deterministischen Algorithmen?
 - ▶ Können Quanten-Algorithmen NP -vollständige Probleme effizient lösen?
- ? Wie sehen Querbezüge zwischen parallelen und sequentiellen Zeitklassen, zwischen Zeit- und Speicherplatzklassen aus?
- ? Für die Approximationskomplexität wichtiger Optimierungsprobleme wird eine neue Sichtweise von NP benötigt.