

# Komplexitätstheorie

Sommersemester 2019

Prof. Dr. Georg Schnitger  
AG Theoretische Informatik

Johann Wolfgang Goethe-Universität Frankfurt am Main

**Herzlich willkommen!!!**

# Kapitel 1: Einführung

Der erste Teil: (Nicht-)Deterministische Berechnungen.

- Wie ist **Laufzeit** zu messen? Average-Case, Smoothed oder Worst-Case?
- Nichtdeterministische und alternierende Berechnungen:
  - ▶ **Nichtdeterminismus**: Rate eine akzeptierende Rechnung
  - ▶ **co-Nichtdeterminismus**: Verifiziere, dass alle Rechnungen akzeptieren
  - ▶ **Alternation**: Rate und verifiziere munter durcheinander
- Die Methode der Diagonalisierung: Hilft ein Mehr an Ressourcen?
- 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**
  - ▶ Welche Probleme in  $P$  besitzen keine superschnellen parallelen Algorithmen?
  - ▶ Ein Zusammenhang zwischen Speicherplatz und paralleler Zeit.

## Die beiden letzten Teile: Randomisierung und Quantenberechnungen

### (II) Randomisierung.

- ▶ **BPP**: Was gelingt in polynomieller Zeit?
- ▶ **Kryptographie**
  - ★ One-Way-Funktionen und Public-Key-Kryptographie
  - ★ Gitterkryptographie
  - ★ Natürliche Beweise: Warum ist die  $P \stackrel{?}{=} NP$ -Frage so schwierig?
- ▶ **Interaktive Beweise**
  - ★ Und wenn uns jemand eine Gewinn-Strategie für Schach verkaufen will?
  - ★ Nicht-Approximierbarkeit: Eine vollständig neue Sichtweise von **NP**.

### (III) Quanten-Berechnungen

- ▶ Quanten-Rechner: Schaltungen oder Turingmaschinen
- ▶ **BQP**: Was gelingt in polynomieller Zeit?
- ▶ Quanten-Suche
- ▶ Effiziente Faktorisierung

- Welche Eigenschaften machen ein algorithmisches Problem schwierig?
- Die Entwicklung von Methoden, um die Schwierigkeit eines Problems einschätzen zu können.
  - ▶ Komplexitätsklassen und Vollständigkeit.
  - ▶ Die Herleitung unterer Schranken.

*Um angemessene algorithmische Lösungen erarbeiten zu können, muss man die inhärente Schwierigkeit des Problems verstehen.*

# Worauf wird aufgebaut?

- **Notwendig:** Eine Bachelor-Veranstaltung wie etwa die **Theoretische Informatik 1** für den Entwurf und die Analyse von Algorithmen.
  - ▶ Laufzeitanalyse:  $O, o, \Omega, \omega$  and Rekursionsgleichungen
  - ▶ Traversierung von Graphen
  - ▶ Dynamische Programmierung
  - ▶ NP-Vollständigkeit
  - ▶ Berechenbarkeit
- **Hilfreich, aber nicht notwendig:** Eine Bachelor-Veranstaltung wie etwa die **Theoretische Informatik 2** für formale Sprachen.
- **Spezielles mathematisches Vorwissen** wird nicht verlangt, aber die Fähigkeit, mathematische Beweise verstehen und führen zu können, muss vorhanden sein.

- (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, Klausurtermine, usw.

[www.thi.informatik.uni-frankfurt.de/lehre/kth/sose19.de](http://www.thi.informatik.uni-frankfurt.de/lehre/kth/sose19.de)

- Mündliche Prüfungen in der Woche des 22. Juli, des 29. Juli und des 7. Oktobers.
- Übungsbetrieb: BITTE **UNBEDINGT** TEILNEHMEN!
  - ▶ Die Übungsgruppe trifft sich Mittwochs von 14-16 Uhr im Magnus Hörsaal. Ausnahme: Der Ersatztermin zum 1. Mai ist Dienstag, der 30.04 um 14 Uhr im Magnus Hörsaal. Das ist auch der Termin des ersten Treffens.
  - ▶ Das aktuelle Übungsblatt erscheint spätestens Montags vor der Vorlesung.
  - ▶ Rückgabe, nach 1-wöchiger Bearbeitungszeit, ebenfalls Montags zu Beginn der Vorlesung. (Eine Rückgabe ist auch im Briefkasten neben Büro 312 möglich.)
  - ▶ Werden 50% (bzw 70%) aller Übungspunkte erreicht, dann Noten-Verbesserung um einen (bzw. zwei) Notenschritte.

- 1 Bitte helfen Sie mir durch
  - ▶ Fragen,
  - ▶ Kommentare
  - ▶ und Antworten!
- 2 Die Vorlesung kann nur durch **Interaktion** interessant werden.
  - ▶ Ich muss wissen, wo der Schuh drückt.
- 3 Sie erreichen mich außerhalb der Vorlesung im Büro 303.
  - ▶ Sprechstunde: Dienstags 10-12.
  - ▶ Kommen Sie vorbei.

Wir untersuchen algorithmische Entscheidungsprobleme nach

- der Worst-Case-Laufzeit sequentieller Algorithmen: **Zeitkomplexität**
- dem Worst-Case-Speicherplatz sequentieller Algorithmen: **Platzkomplexität**
- der Effizienz paralleler Algorithmen: **Schaltkreiskomplexität** und
- nach der Qualität approximativer Lösungen: **Approximationskomplexität**.

# Zeit-Komplexitätsklassen

Irgendein vernünftiges Rechnermodell (Turingmaschine, Registermaschinen).

- 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

Welche Verteilungen auf der Menge der Eingaben sollte man betrachten?

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

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

ein *Ensemble*. Häufig:  $D_n$  ist eine Verteilung über  $\Sigma^n$ .

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

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 (erwarteter) Laufzeit  $t_A(x)$ . 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

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

für eine Konstante  $\varepsilon > 0$ , ein Polynom  $p(n)$  und alle  $n, t \in \mathbb{N}$ .

(c) Die Komplexitätsklasse

AVGP

besteht aus allen Verteilungsproblemen  $(L; \mathcal{D})$  mit  $L = L(A)$  für einen Algorithmus  $A$  von 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  $B$  hat polynomielle erwartete Laufzeit, 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.
  - ▶ Die Laufzeit darf um den Faktor höchstens  $c^{1/\varepsilon}$  ansteigen, falls die Wahrscheinlichkeit dafür mindestens um den Faktor  $\frac{1}{c}$  abnimmt.

# 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

Die Average-Case-Reduktion ist eine partielle Ordnung mit

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

als vollständigem Problem für  $\text{AVGNP}$ .

Definiere das beschränkte Halteproblem

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

$\mathcal{V} = (V_n : n \in \mathbb{N})$  definiert „fast“ eine Gleichverteilung auf Tripeln  $(M, x, t)$ .

Leider gibt es keine vollständigen Probleme von praktischer Relevanz:  
Nur wenn  $\text{EXP} = \text{NEXP}$  gibt es Average-Case-vollständige Probleme  $(L, \mathcal{U})$ .

# Notorisch-schwierige Verteilungsprobleme

# Random- $k$ -SAT

Eingaben von  $k$ -SAT $_{n,m}$  sind KNFs  $\phi$  mit  $\leq k$  Literalen pro Klausel, den Variablen  $X_1, \dots, X_n$  und  $m$  Klauseln. Entscheide, ob  $\phi$  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  $f(n) \approx 4.2667 \cdot n$  Klauseln: Ein **Phasenübergang** 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* selbst im Mittel keine effizienten Algorithmen (Aber Vollständigkeitsergebnisse sind nicht bekannt.)

# Bounded-Distance-Decoding (BDD) für lineare Codes

- Würfle zuerst eine  $n \times m$ -Matrix  $A$  (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  $y := x^T \cdot A \oplus e$ .

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

# Smoothed Complexity, bzw. geglättete Komplexität

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

- (a) Die geglättete Komplexität von Algorithmus  $A$  für die Normalverteilung mit Erwartungswert 0 und Standardabweichung  $\sigma$  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  $n \in \mathbb{N}$ , für alle genügend kleinen  $\sigma$  und für geeignete  $k_1, k_2 \in \mathbb{N}$  gilt

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

Der Simplex-Algorithmus für die lineare Programmierung hat polynomielle geglättete Komplexität. Siehe auch

D.A. Spielman, S. Teng, Smoothed Analysis: an attempt to explain the behavior of algorithms in practice, Communications of the ACM, pp. 76-84, 2009.

# 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

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

- 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  $\text{NP}$  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  $\text{NEXP}$  hat einiges zu bieten.

- Eine KNF  $\alpha$  mit  $2^n$  Variablen ist *succinct*, wenn es einen Schaltkreis  $C$  mit  $2n$  Eingabebits gibt, der die Beschreibung einer Dreier-Klausel  $k$  genau dann akzeptiert, wenn  $k$  eine Klausel von  $\alpha$  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  $\text{NEXP}$  unter polynomiellen Reduktionen.

- *Existentielle Berechnungen*: Rate und verifiziere deterministisch.
  - ▶ Die Kraft des Ratens: Das KNF-Erfüllbarkeitsproblem ist einfach.
  - ▶ Die Schwäche des Ratens: Das KNF-Tautologieproblem ist schwierig.
- *Universelle Berechnungen*: Verifiziere alle Berechnungen deterministisch.
  - ▶ Die Kraft des Verifizierens: Das KNF-Tautologieproblem ist einfach.
  - ▶ Die Schwäche des Verifizierens: Das KNF-Erfüllbarkeitsproblem ist schwierig.
- *Alternierende Berechnungen* dürfen beliebig zwischen existentiellen und universellen Zuständen 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  $\Sigma_k$ -Algorithmus (bzw.  $\Pi_k$ -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  $\Sigma_k^P$  ( $\Pi_k^P$ ) besteht aus allen durch einen  $\Sigma_k$ -Algorithmus ( $\Pi_k$ -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  $\text{AP}$ , alternierende polynomielle Zeit,

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

# Alternation: Was bringt das Gedankenexperiment?

- Die Sprache

$\exists_k \text{SAT} := \{ \phi : \text{die Formel } \phi = \underbrace{Q_1}_{\exists} Q_2 \cdots Q_k \alpha - \text{ mit } k \text{ Blöcken von nur } \exists\text{- bzw. } \forall\text{-Quantoren und einer KNF } \alpha - \text{ ist wahr} \}.$

ist vollständig für  $\Sigma_k^P$  unter der polynomiellen Reduktion.

- Die Sprache aller allgemeingültigen aussagenlogischen Formeln ist vollständig für  $\Pi_1^P$  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  $\text{AP}$  wird sich als besonders wichtig erweisen. Zum Beispiel können nicht-triviale 2-Personen-Spiele optimal „in  $\text{AP}$  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 Zeit  $O(\log_2 t(n))$  pro Schritt.

- (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)
- ▶  $NP^K \subseteq \mathcal{K}$ : ✓.
  - ▶ 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  $NP^{K^*} \subseteq \mathcal{K} \subseteq P^{K^*}$ .

# 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 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 akzeptierte 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!

- ⚡ Ein Beweis von  $P \neq NP$  muss – vielleicht neben der Diagonalisierung – andere Methoden nutzen, um deterministische Algorithmen zu untersuchen.
- ⚡ Später: Auch „natürliche Beweise“ genügen für den Nachweis von  $P \neq NP$  nicht.

# Zeitkomplexität: Die wichtigen 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?
  - (\*) In welchen eingeschränkten Modellen von  $P$  und  $NP$  kann die 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.