

FORMALE SYSTEME

25. Vorlesung: NP-Vollständigkeit

Hannes Straß

Folien: © Markus Krötzsch, <https://iccl.inf.tu-dresden.de/web/FS2020>, CC BY 3.0 DE

TU Dresden, 20. Januar 2022

Rückblick

Komplexitätsklassen

Komplexitätsklassen sind Mengen von Sprachen, die man (grob) einteilt entsprechend der Ressourcen, die eine TM zur Entscheidung ihres Wortproblems benötigt.

deterministisch

L = LogSpace logarithmischer Speicher

P = PTime polynomielle Zeit

PSpace polynomieller Speicher

Exp = ExpTime exponentielle Zeit

nichtdeterministisch

NL = NLogSpace

NP = NPTime

NPSpace

NExp = NExpTime

Beziehungen zwischen den Klassen

NL	NP	NPSpace	NExp
L	P	PSpace	Exp

Welche Beziehungen bestehen zwischen Komplexitätsklassen?

¹Unter der Voraussetzung, dass $f(n) \geq \log(n)$.

Beziehungen zwischen den Klassen

NL NP \subseteq NPSpace NExp

L P \subseteq PSpace Exp

Welche Beziehungen bestehen zwischen Komplexitätsklassen?

- Eine $f(n)$ -zeitbeschränkte TM ist stets auch $f(n)$ -speicherbeschränkt;

¹Unter der Voraussetzung, dass $f(n) \geq \log(n)$.

Beziehungen zwischen den Klassen

$NL \subseteq NP \subseteq NPSpace \subseteq NExp$

$L \subseteq P \subseteq PSpace \subseteq Exp$

Welche Beziehungen bestehen zwischen Komplexitätsklassen?

- Eine $f(n)$ -zeitbeschränkte TM ist stets auch $f(n)$ -speicherbeschränkt;
- eine $f(n)$ -speicherbeschränkte TM macht höchstens $f(n) \cdot 2^{O(f(n))}$ Schritte.¹

¹Unter der Voraussetzung, dass $f(n) \geq \log(n)$.

Beziehungen zwischen den Klassen

$$\begin{array}{cccccc} \text{NL} & \subseteq & \text{NP} & \subseteq & \text{NPSpace} & \subseteq & \text{NExp} \\ \cup & & \cup & & \cup & & \cup \\ \text{L} & \subseteq & \text{P} & \subseteq & \text{PSpace} & \subseteq & \text{Exp} \end{array}$$

Welche Beziehungen bestehen zwischen Komplexitätsklassen?

- Eine $f(n)$ -zeitbeschränkte TM ist stets auch $f(n)$ -speicherbeschränkt;
- eine $f(n)$ -speicherbeschränkte TM macht höchstens $f(n) \cdot 2^{O(f(n))}$ Schritte.¹
- Eine DTM kann als NTM aufgefasst werden, d.h. die nichtdeterministischen Klassen sind immer mindestens ebenso stark wie die jeweiligen deterministischen.

¹Unter der Voraussetzung, dass $f(n) \geq \log(n)$.

Beziehungen zwischen den Klassen



Welche Beziehungen bestehen zwischen Komplexitätsklassen?

- Eine $f(n)$ -zeitbeschränkte TM ist stets auch $f(n)$ -speicherbeschränkt;
- eine $f(n)$ -speicherbeschränkte TM macht höchstens $f(n) \cdot 2^{O(f(n))}$ Schritte.¹
- Eine DTM kann als NTM aufgefasst werden, d.h. die nichtdeterministischen Klassen sind immer mindestens ebenso stark wie die jeweiligen deterministischen.
- DTMs können NTMs simulieren, aber das ist oft mit exponentiellen Mehrkosten verbunden.

¹Unter der Voraussetzung, dass $f(n) \geq \log(n)$.

Beziehungen zwischen den Klassen

NL		NP	\subseteq	NPSpace		NExp
UI	\hookleftarrow	UI		II	\hookleftarrow	UI
L		P		PSpace		Exp

Welche Beziehungen bestehen zwischen Komplexitätsklassen?

- Eine $f(n)$ -zeitbeschränkte TM ist stets auch $f(n)$ -speicherbeschränkt;
- eine $f(n)$ -speicherbeschränkte TM macht höchstens $f(n) \cdot 2^{O(f(n))}$ Schritte.¹
- Eine DTM kann als NTM aufgefasst werden, d.h. die nichtdeterministischen Klassen sind immer mindestens ebenso stark wie die jeweiligen deterministischen.
- DTMs können NTMs simulieren, aber das ist oft mit exponentiellen Mehrkosten verbunden.
- Der berühmte [Satz von Savitch](#) besagt, dass speicherbeschränkte NTMs durch DTMs mit nur quadratischen Mehrkosten simuliert werden können. \leadsto PSpace = NPSpace.

¹Unter der Voraussetzung, dass $f(n) \geq \log(n)$.

Die Grenzen unseres Wissens

Wir wissen:

$$L \subseteq NL \subseteq P \subseteq NP \subseteq PSpace = NPSPACE \subseteq Exp \subseteq NExp$$

- Wir wissen nicht, ob irgendeines dieser \subseteq sogar \subsetneq ist.
- Insbesondere wissen wir nicht, ob $P \subsetneq NP$ oder $P = NP$.
- Wir wissen nicht einmal, ob $L \subsetneq NP$ oder $L = NP$.

Die Grenzen unseres Wissens

Wir wissen:

$$L \subseteq NL \subseteq P \subseteq NP \subseteq PSpace = NPSPACE \subseteq Exp \subseteq NExp$$

- Wir wissen nicht, ob irgendeines dieser \subseteq sogar \subsetneq ist.
- Insbesondere wissen wir nicht, ob $P \subsetneq NP$ oder $P = NP$.
- Wir wissen nicht einmal, ob $L \subsetneq NP$ oder $L = NP$.

Es wird aber vermutet, dass alle \subseteq eigentlich \subsetneq sind.

Bekannt ist das wiederum nur bei exponentiell großen Ressourcenunterschieden:

Es gilt:

- $NL \subsetneq PSpace$
- $P \subsetneq Exp$
- $NP \subsetneq NExp$

NP

NP-Probleme effizient lösen?

Wir haben gesehen:

$$\mathbf{SAT} \in \text{NP} \subseteq \text{PSPACE} \subseteq \text{ExpTime}$$

- Dennoch haben alle unsere Algorithmen für **SAT** bisher exponentielle Laufzeit.
- **SAT** \in NP führt nicht direkt zu einer besseren Lösung, da real existierende Computer nicht wie NTMs arbeiten.
- Andererseits spricht auch nichts dagegen, dass es einen schnelleren Algorithmus gibt.

Kann man **SAT** auch effizienter lösen?

↪ Um diese Frage anzugehen, müssen wir erst einmal NP etwas besser verstehen.

Lösungen prüfen

Eine interessante Eigenschaft von SAT:

- Es scheint schwer, die Erfüllbarkeit einer Formel zu ermitteln (z.B. können wir exponentiell viele Wertzuweisungen durchsuchen).
- Es ist einfach, Erfüllbarkeit einer Formel zu überprüfen, wenn ein Modell gegeben ist (dazu muss nur der Wahrheitswert der Formel berechnet werden).

Lösungen prüfen

Eine interessante Eigenschaft von SAT:

- Es scheint schwer, die Erfüllbarkeit einer Formel zu ermitteln (z.B. können wir exponentiell viele Wertzuweisungen durchsuchen).
- Es ist einfach, Erfüllbarkeit einer Formel zu überprüfen, wenn ein Modell gegeben ist (dazu muss nur der Wahrheitswert der Formel berechnet werden).

Ähnlich ist es bei Sudoku:

- Es ist schwer, ein Sudoku korrekt auszufüllen.
- Es ist einfach, zu prüfen, ob ein Sudoku korrekt ausgefüllt wurde.

Lösungen prüfen

Eine interessante Eigenschaft von SAT:

- Es scheint schwer, die Erfüllbarkeit einer Formel zu ermitteln (z.B. können wir exponentiell viele Wertzuweisungen durchsuchen).
- Es ist einfach, Erfüllbarkeit einer Formel zu überprüfen, wenn ein Modell gegeben ist (dazu muss nur der Wahrheitswert der Formel berechnet werden).

Ähnlich ist es bei Sudoku:

- Es ist schwer, ein Sudoku korrekt auszufüllen.
- Es ist einfach, zu prüfen, ob ein Sudoku korrekt ausgefüllt wurde.

Gibt es noch mehr Probleme, die sich so verhalten?

- Es ist schwer, eine Lösung zu finden.
- Es ist einfach, eine gegebene Lösung zu prüfen.

TMs die Lösungen überprüfen

Wir wollen diese Idee formalisieren:

Ein **polynomieller Verifikator** für eine Sprache $L \subseteq \Sigma^*$ ist eine polynomiell-zeitbeschränkte, deterministische TM \mathcal{M} , für die gilt:

- \mathcal{M} akzeptiert nur Wörter der Form $w\#z$ mit:
 - $w \in L$,
 - $z \in \Sigma^*$ ist ein **Zertifikat** polynomieller Länge (d.h. für \mathcal{M} gibt es ein Polynom p mit $|z| \leq p(|w|)$).
- Für jedes Wort $w \in L$ gibt es ein solches Wort $w\#z \in L(\mathcal{M})$.

Intuition:

- Das Zertifikat z kodiert die Lösung des Problems w , die der Verifikator lediglich nachprüft.
- Zertifikate sollten kurz sein, damit die Prüfung selbst nicht länger dauert als die Lösung des Problems.

Zertifikate werden auch **Nachweis**, **Beweis** oder **Zeuge** genannt.

Nachweis-polynomielle Sprachen

Eine Sprache L ist genau dann **nachweis-polynomiell**, wenn es für sie einen polynomiellen Verifikator gibt.

Nachweis-polynomielle Sprachen

Eine Sprache L ist genau dann **nachweis-polynomiell**, wenn es für sie einen polynomiellen Verifikator gibt.

Beispiel: SAT ist nachweis-polynomiell. Ein möglicher polynomieller Verifikator für **SAT** akzeptiert Wörter $F\#w$, wobei F die Kodierung einer erfüllbaren Formel ist und w eine Kodierung einer Wertzuweisung, die F erfüllt.

Nachweis-polynomielle Sprachen

Eine Sprache L ist genau dann **nachweis-polynomiell**, wenn es für sie einen polynomiellen Verifikator gibt.

Beispiel: SAT ist nachweis-polynomiell. Ein möglicher polynomieller Verifikator für **SAT** akzeptiert Wörter $F\#w$, wobei F die Kodierung einer erfüllbaren Formel ist und w eine Kodierung einer Wertzuweisung, die F erfüllt.

Beispiel: Jede Sprache $L \in P$ ist nachweis-polynomiell. Als Verifikator verwenden wir einfach einen polynomiell-zeitbeschränkten Entscheider für L . Das Zertifikat kann leer sein.

Beispiel: Drei-Farben-Problem

Das **Drei-Farben-Problem** besteht darin, die Knoten eines ungerichteten Graphen so mit den drei Farben rot, grün und blau zu färben, dass keine zwei benachbarten (mittels einer Kante verbundenen) Knoten die gleiche Farbe haben.



Beispiel: Drei-Farben-Problem

Das **Drei-Farben-Problem** besteht darin, die Knoten eines ungerichteten Graphen so mit den drei Farben rot, grün und blau zu färben, dass keine zwei benachbarten (mittels einer Kante verbundenen) Knoten die gleiche Farbe haben.



Das Problem kann als Wortproblem aufgefasst werden, wenn man Graphen als Wörter kodiert (z.B. als Adjazenzliste oder als zeilenweise kodierte Adjazenzmatrix).

Das Drei-Farben-Problem ist nachweis-polynomiell. (Zertifikat: Farbzuweisung.)

Beispiel: Faktorisierung

Die **Faktorisierung** einer natürlichen Zahl n ist die Darstellung der Zahl als Produkt von natürlichen Zahlen $k_1 \cdot k_2 = n$.

Folgende Sprache ist nachweis-polynomiell:

COMPOSITE = $\{\text{bin}(n) \mid n \in \mathbb{N}, \text{es gibt } k_1, k_2 > 1 \text{ mit } n = k_1 \cdot k_2\}$

(Zertifikat: Kodierung von k_1 und k_2 , z.B. $\text{bin}(k_1)\#\text{bin}(k_2)$.)

Beispiel: Faktorisierung

Die **Faktorisierung** einer natürlichen Zahl n ist die Darstellung der Zahl als Produkt von natürlichen Zahlen $k_1 \cdot k_2 = n$.

Folgende Sprache ist nachweis-polynomiell:

$$\mathbf{COMPOSITE} = \{\text{bin}(n) \mid n \in \mathbb{N}, \text{ es gibt } k_1, k_2 > 1 \text{ mit } n = k_1 \cdot k_2\}$$

(Zertifikat: Kodierung von k_1 und k_2 , z.B. $\text{bin}(k_1)\#\text{bin}(k_2)$.)

Überraschenderweise ist auch folgende Sprache nachweis-polynomiell:

$$\mathbf{PRIMES} = \{\text{bin}(n) \mid n \in \mathbb{N}, \text{ es gibt keine } k_1, k_2 > 1 \text{ mit } n = k_1 \cdot k_2\}$$

(Zertifikat: „Primality certificate“, bekannt seit 1975.)

Beispiel: Faktorisierung

Die **Faktorisierung** einer natürlichen Zahl n ist die Darstellung der Zahl als Produkt von natürlichen Zahlen $k_1 \cdot k_2 = n$.

Folgende Sprache ist nachweis-polynomiell:

$$\mathbf{COMPOSITE} = \{\text{bin}(n) \mid n \in \mathbb{N}, \text{ es gibt } k_1, k_2 > 1 \text{ mit } n = k_1 \cdot k_2\}$$

(Zertifikat: Kodierung von k_1 und k_2 , z.B. $\text{bin}(k_1)\#\text{bin}(k_2)$.)

Überraschenderweise ist auch folgende Sprache nachweis-polynomiell:

$$\mathbf{PRIMES} = \{\text{bin}(n) \mid n \in \mathbb{N}, \text{ es gibt keine } k_1, k_2 > 1 \text{ mit } n = k_1 \cdot k_2\}$$

(Zertifikat: „Primality certificate“, bekannt seit 1975.)

(Seit 2002 wissen wir: $\mathbf{PRIMES} \in P$; Primzahlentest nach Agrawal, Kayal und Saxena.)

NP steht für „nachweis-polynomiell“ (1)

Satz: Eine Sprache L ist genau dann nachweis-polynomiell, wenn $L \in NP$.

NP steht für „nachweis-polynomiell“ (1)

Satz: Eine Sprache L ist genau dann nachweis-polynomiell, wenn $L \in NP$.

Beweis: (\Rightarrow) Sei L nachweis-polynomiell mit Verifikator \mathcal{V} . Wir erhalten eine polynomiell zeitbeschränkte NTM \mathcal{M} für L wie folgt:

NP steht für „nachweis-polynomiell“ (1)

Satz: Eine Sprache L ist genau dann nachweis-polynomiell, wenn $L \in NP$.

Beweis: (\Rightarrow) Sei L nachweis-polynomiell mit Verifikator \mathcal{V} . Wir erhalten eine polynomiell zeitbeschränkte NTM \mathcal{M} für L wie folgt:

- Für eine Eingabe w rät \mathcal{M} nichtdeterministisch ein polynomielles Zertifikat z (dabei muss \mathcal{M} die Länge von z selbst beschränken);
- anschließend führt \mathcal{M} die Berechnungen wie \mathcal{V} aus, um das geratene Zertifikat in polynomieller Zeit zu prüfen.

NP steht für „nachweis-polynomiell“ (2)

Satz: Eine Sprache L ist genau dann nachweis-polynomiell, wenn $L \in NP$.

Beweis: (\Leftarrow) Sei \mathcal{M} eine polynomiell-zeitbeschränkte NTM für L . Wir erhalten einen polynomiellen Verifikator \mathcal{V} für L wie folgt:

NP steht für „nachweis-polynomiell“ (2)

Satz: Eine Sprache L ist genau dann nachweis-polynomiell, wenn $L \in NP$.

Beweis: (\Leftarrow) Sei M eine polynomiell-zeitbeschränkte NTM für L . Wir erhalten einen polynomiellen Verifikator V für L wie folgt:

- Für jedes Wort $w \in L$ gibt es einen akzeptierenden Lauf von M .
- In jedem Schritt dieses Laufs wählt M nichtdeterministisch einen Übergang aus.
- Der Lauf ist von polynomieller Länge, da M polynomiell zeitbeschränkt ist.

NP steht für „nachweis-polynomiell“ (2)

Satz: Eine Sprache L ist genau dann nachweis-polynomiell, wenn $L \in NP$.

Beweis: (\Leftarrow) Sei M eine polynomiell-zeitbeschränkte NTM für L . Wir erhalten einen polynomiellen Verifikator \mathcal{V} für L wie folgt:

- Für jedes Wort $w \in L$ gibt es einen akzeptierenden Lauf von M .
- In jedem Schritt dieses Laufs wählt M nichtdeterministisch einen Übergang aus.
- Der Lauf ist von polynomieller Länge, da M polynomiell zeitbeschränkt ist.

\leadsto Die Liste der Übergänge eines akzeptierenden Laufs kann als Zertifikat z dienen.

\mathcal{V} prüft, ob die durch z gegebenen Entscheidungen für die Eingabe w zu einem akzeptierenden Lauf von M führen. □

NP ist nicht symmetrisch

Es ist leicht zu sehen:

Satz: Die Klasse P ist unter Komplement abgeschlossen.

Beweis: Wenn es für L eine polynomiell-zeitbeschränkte TM \mathcal{M} gibt, dann erhält man eine TM für \bar{L} , indem man akzeptierende und nichtakzeptierende Zustände von \mathcal{M} vertauscht. □

NP ist nicht symmetrisch

Es ist leicht zu sehen:

Satz: Die Klasse P ist unter Komplement abgeschlossen.

Beweis: Wenn es für L eine polynomiell-zeitbeschränkte TM \mathcal{M} gibt, dann erhält man eine TM für \bar{L} , indem man akzeptierende und nichtakzeptierende Zustände von \mathcal{M} vertauscht. □

Für NP ist das nicht so einfach möglich. Wir haben das schon für NFAs beobachtet (eine spezielle Art von NTMs).¹

Beispiele: Es scheint kein „einfaches“ („kurzes“) Zertifikat dafür zu geben, dass eine Formel **nicht** erfüllbar ist oder dass ein Graph **keine** Dreifärbung zulässt.

¹Für ein akzeptiertes Wort kann es bei einer NTM akzeptierende **und** verwerfende Läufe geben. Nach dem Austausch der Endzustände gäbe es dann aber immer noch akzeptierende Läufe.

NP ist nicht symmetrisch

Es ist leicht zu sehen:

Satz: Die Klasse P ist unter Komplement abgeschlossen.

Beweis: Wenn es für L eine polynomiell-zeitbeschränkte TM M gibt, dann erhält man eine TM für \bar{L} , indem man akzeptierende und nichtakzeptierende Zustände von M vertauscht. □

Für NP ist das nicht so einfach möglich. Wir haben das schon für NFAs beobachtet (eine spezielle Art von NTMs).¹

Beispiele: Es scheint kein „einfaches“ („kurzes“) Zertifikat dafür zu geben, dass eine Formel **nicht** erfüllbar ist oder dass ein Graph **keine** Dreifärbung zulässt.

Die Komplexitätsklasse **coNP** enthält alle diejenigen Sprachen L , für die $\bar{L} \in NP$ gilt.

¹Für ein akzeptiertes Wort kann es bei einer NTM akzeptierende **und** verwerfende Läufe geben. Nach dem Austausch der Endzustände gäbe es dann aber immer noch akzeptierende Läufe.

NP-Vollständigkeit

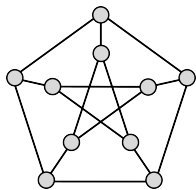
Reduktionen

Manche Probleme lassen sich auf andere zurückführen.

Reduktionen

Manche Probleme lassen sich auf andere zurückführen.

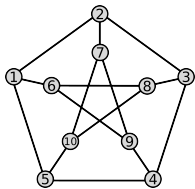
Beispiel: Das Drei-Farben-Problem ist auf **SAT** reduzierbar.



Reduktionen

Manche Probleme lassen sich auf andere zurückführen.

Beispiel: Das Drei-Farben-Problem ist auf **SAT** reduzierbar.



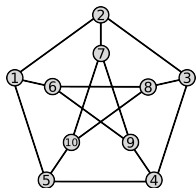
Darstellung von Farben mit Atomen:

- r_i bedeutet „Knoten i ist rot“
- g_i bedeutet „Knoten i ist grün“
- b_i bedeutet „Knoten i ist blau“

Reduktionen

Manche Probleme lassen sich auf andere zurückführen.

Beispiel: Das Drei-Farben-Problem ist auf **SAT** reduzierbar.



Darstellung von Farben mit Atomen:

- r_i bedeutet „Knoten i ist rot“
- g_i bedeutet „Knoten i ist grün“
- b_i bedeutet „Knoten i ist blau“

Färbe-Bedingungen für Knoten:

$$(r_1 \wedge \neg g_1 \wedge \neg b_1) \vee (\neg r_1 \wedge g_1 \wedge \neg b_1) \vee (\neg r_1 \wedge \neg g_1 \wedge b_1)$$

(genau eine Farbe pro Knoten)

(usw. für alle Knoten)

Färbe-Bedingungen für Kanten:

$$\neg(r_1 \wedge r_2) \wedge \neg(g_1 \wedge g_2) \wedge \neg(b_1 \wedge b_2)$$

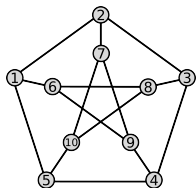
(verbundene Knoten ungleich gefärbt)

(usw. für alle Kanten)

Reduktionen

Manche Probleme lassen sich auf andere zurückführen.

Beispiel: Das Drei-Farben-Problem ist auf **SAT** reduzierbar.



Darstellung von Farben mit Atomen:

- r_i bedeutet „Knoten i ist rot“
- g_i bedeutet „Knoten i ist grün“
- b_i bedeutet „Knoten i ist blau“

Färbe-Bedingungen für Knoten:

$$(r_1 \wedge \neg g_1 \wedge \neg b_1) \vee (\neg r_1 \wedge g_1 \wedge \neg b_1) \vee (\neg r_1 \wedge \neg g_1 \wedge b_1)$$

(genau eine Farbe pro Knoten)

(usw. für alle Knoten)

Färbe-Bedingungen für Kanten:

$$\neg(r_1 \wedge r_2) \wedge \neg(g_1 \wedge g_2) \wedge \neg(b_1 \wedge b_2)$$

(verbundene Knoten ungleich gefärbt)

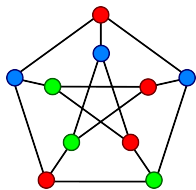
(usw. für alle Kanten)

Erfüllende Wertzuweisung \Leftrightarrow zulässige Färbung

Reduktionen

Manche Probleme lassen sich auf andere zurückführen.

Beispiel: Das Drei-Farben-Problem ist auf **SAT** reduzierbar.



Darstellung von Farben mit Atomen:

- r_i bedeutet „Knoten i ist rot“
- g_i bedeutet „Knoten i ist grün“
- b_i bedeutet „Knoten i ist blau“

Färbe-Bedingungen für Knoten:

$$(r_1 \wedge \neg g_1 \wedge \neg b_1) \vee (\neg r_1 \wedge g_1 \wedge \neg b_1) \vee (\neg r_1 \wedge \neg g_1 \wedge b_1)$$

(genau eine Farbe pro Knoten)

(usw. für alle Knoten)

Färbe-Bedingungen für Kanten:

$$\neg(r_1 \wedge r_2) \wedge \neg(g_1 \wedge g_2) \wedge \neg(b_1 \wedge b_2)$$

(verbundene Knoten ungleich gefärbt)

(usw. für alle Kanten)

Erfüllende Wertzuweisung \Leftrightarrow zulässige Färbung

Polynomielle Reduktionen

Eine Funktion $f : \Sigma^* \rightarrow \Sigma^*$ ist genau dann **polynomiell berechenbar**, wenn es eine polynomiell zeitbeschränkte TM gibt, die bei einer Eingabe w die Ausgabe $f(w)$ auf das Band schreibt und anschließend anhält.

Eine polynomiell berechenbare Funktion $f : \Sigma^* \rightarrow \Sigma^*$ ist genau dann eine **polynomielle Reduktion** von einer Sprache **L** auf eine Sprache **G**, wenn für alle Wörter $w \in \Sigma^*$ gilt:

$$w \in \mathbf{L} \quad \text{genau dann, wenn} \quad f(w) \in \mathbf{G}$$

Polynomielle Reduktionen

Eine Funktion $f : \Sigma^* \rightarrow \Sigma^*$ ist genau dann **polynomiell berechenbar**, wenn es eine polynomiell zeitbeschränkte TM gibt, die bei einer Eingabe w die Ausgabe $f(w)$ auf das Band schreibt und anschließend anhält.

Eine polynomiell berechenbare Funktion $f : \Sigma^* \rightarrow \Sigma^*$ ist genau dann eine **polynomielle Reduktion** von einer Sprache \mathbf{L} auf eine Sprache \mathbf{G} , wenn für alle Wörter $w \in \Sigma^*$ gilt:

$$w \in \mathbf{L} \quad \text{genau dann, wenn} \quad f(w) \in \mathbf{G}$$

Beispiel: Die soeben skizzierte Funktion von Graphen auf logische Formeln ist eine polynomielle Reduktion des Drei-Farben-Problems auf **SAT**.

Polynomielle Reduktionen

Eine Funktion $f : \Sigma^* \rightarrow \Sigma^*$ ist genau dann **polynomiell berechenbar**, wenn es eine polynomiell zeitbeschränkte TM gibt, die bei einer Eingabe w die Ausgabe $f(w)$ auf das Band schreibt und anschließend anhält.

Eine polynomiell berechenbare Funktion $f : \Sigma^* \rightarrow \Sigma^*$ ist genau dann eine **polynomielle Reduktion** von einer Sprache \mathbf{L} auf eine Sprache \mathbf{G} , wenn für alle Wörter $w \in \Sigma^*$ gilt:

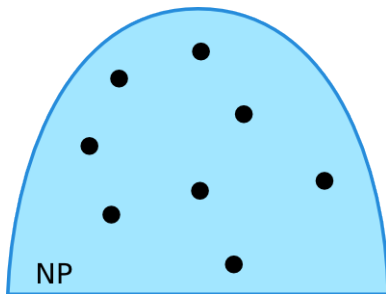
$$w \in \mathbf{L} \quad \text{genau dann, wenn} \quad f(w) \in \mathbf{G}$$

Beispiel: Die soeben skizzierte Funktion von Graphen auf logische Formeln ist eine polynomielle Reduktion des Drei-Farben-Problems auf **SAT**.

Beispiel: Wir haben das Inklusionsproblem endlicher Automaten in Vorlesung 10 auf das Leerheitsproblem reduziert. Diese Reduktion war bei deterministischen Automaten polynomiell.

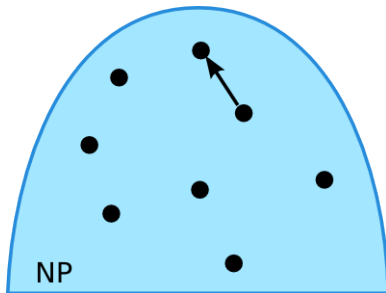
Die Struktur von NP

Idee: polynomielle Reduzierbarkeit definiert eine Art Ordnung auf Problemen.



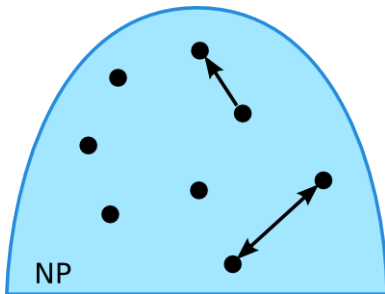
Die Struktur von NP

Idee: polynomielle Reduzierbarkeit definiert eine Art Ordnung auf Problemen.



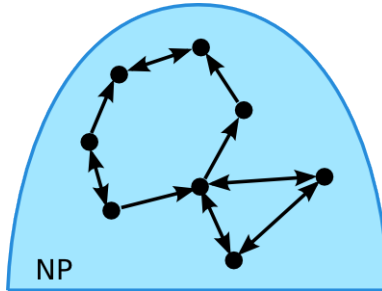
Die Struktur von NP

Idee: polynomielle Reduzierbarkeit definiert eine Art Ordnung auf Problemen.



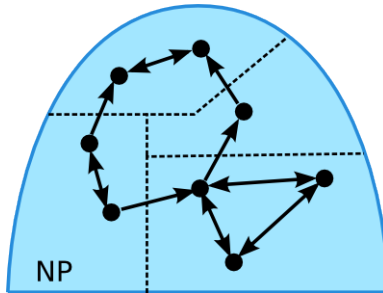
Die Struktur von NP

Idee: polynomielle Reduzierbarkeit definiert eine Art Ordnung auf Problemen.



Die Struktur von NP

Idee: polynomielle Reduzierbarkeit definiert eine Art Ordnung auf Problemen.



Das schwerste Problem in NP?



Stephen Cook



Leonid Levin



Richard Karp

Satz von Cook [1971] & Levin [1973]:

Alle Probleme in NP können polynomiell auf **SAT** reduziert werden.

Das schwerste Problem in NP?



Stephen Cook



Leonid Levin



Richard Karp

Satz von Cook [1971] & Levin [1973]:

Alle Probleme in NP können polynomiell auf **SAT** reduziert werden.

- Wer **SAT** polynomiell löst, der hat alle Probleme in NP polynomiell gelöst.

Das schwerste Problem in NP?



Stephen Cook



Leonid Levin



Richard Karp

Satz von Cook [1971] & Levin [1973]:

Alle Probleme in NP können polynomiell auf **SAT** reduziert werden.

- Wer **SAT** polynomiell löst, der hat alle Probleme in NP polynomiell gelöst.
- Es gibt in NP eine maximale Klasse, die ein praktisch interessantes Problem enthält.

Das schwerste Problem in NP?



Stephen Cook



Leonid Levin



Richard Karp

Satz von Cook [1971] & Levin [1973]:

Alle Probleme in NP können polynomiell auf **SAT** reduziert werden.

- Wer **SAT** polynomiell löst, der hat alle Probleme in NP polynomiell gelöst.
- Es gibt in NP eine maximale Klasse, die ein praktisch interessantes Problem enthält.
- Karp findet kurz darauf 21 weitere solcher Probleme (1972).
- Seitdem wurden tausende weitere entdeckt ...

NP-Schwere und NP-Vollständigkeit

Eine Sprache ist

- **NP-schwer**, wenn jede Sprache in NP polynomiell darauf reduzierbar ist;
- **NP-vollständig**, wenn sie NP-schwer ist und in NP liegt.

NP-Schwere und NP-Vollständigkeit

Eine Sprache ist

- **NP-schwer**, wenn jede Sprache in NP polynomiell darauf reduzierbar ist;
- **NP-vollständig**, wenn sie NP-schwer ist und in NP liegt.

Begrifflichkeiten: Statt „NP-schwer“ sagen wir auch „NP-hart“ (engl. „NP-hard“).

Beispiel: SAT ist NP-vollständig (Cook & Levin).

Beispiel: PRIMES ist in NP, aber **vermutlich** nicht NP-schwer. Gleiches gilt für viele Probleme in P (bei einigen ist dagegen sicher, dass sie nicht NP-schwer sind).

Beispiel: Das Halteproblem ist NP-schwer, aber sicher nicht in NP. Gleiches gilt für jedes unentscheidbare Problem.

Beispiele

Beispiel: Das Drei-Farben-Problem ist NP-vollständig.

Beispiel: Verallgemeinertes Sudoku (n^2 Zahlen in einem $n^2 \times n^2$ -Feld mit $n \times n$ -Unterfeldern) ist NP-vollständig. Das Entscheidungsproblem dabei ist, ob das gegebene, teilweise gefüllte Feld eine Lösung zulässt. Gleiches gilt für Verallgemeinerungen vieler anderer Solitärspiele, z.B. Minesweeper, Mahjong, Nonogramme).

Beispiel: Das Wortproblem für Typ-1-Sprachen (gegeben als Grammatik) ist NP-schwer aber *vermutlich* nicht in NP. Gleiches gilt für das Halteproblem von LBAs.

Beispiel: Das Problem der/des Handelsreisenden (TSP, travelling salesperson problem) ist NP-vollständig. Es besteht darin, in einem Graphen mit bewerteten Kanten einen Pfad durch alle Knoten zu finden, so dass die Summe der Kantenwerte unter einem gegebenen Maximum liegt.

Wie findet man NP-schwere Probleme?

Schwer ist, worauf sich schwere Probleme leicht zurückführen lassen:

Satz: Wenn **L** NP-schwer ist und polynomiell auf **G** reduziert werden kann, dann ist **G** auch NP-schwer.

Beweis: Folgt direkt aus der Definition, da die Komposition von zwei polynomiellen Reduktionen ebenfalls eine polynomielle Reduktion ist. □

Wie findet man NP-schwere Probleme?

Schwer ist, worauf sich schwere Probleme leicht zurückführen lassen:

Satz: Wenn **L** NP-schwer ist und polynomiell auf **G** reduziert werden kann, dann ist **G** auch NP-schwer.

Beweis: Folgt direkt aus der Definition, da die Komposition von zwei polynomiellen Reduktionen ebenfalls eine polynomielle Reduktion ist. □

Technik zum Beweis von NP-Schwere:

- Suche ein bekanntes NP-schweres Problem, welches dem untersuchten Problem möglichst ähnlich ist.
 - Gib eine polynomielle Reduktion an und zeige, dass sie korrekt ist.
- ↪ Oft nicht schwer, wenn man bereits viele NP-schwere Probleme kennt – aber wo fängt man an?

Cook/Levin: Beweisskizze (1)

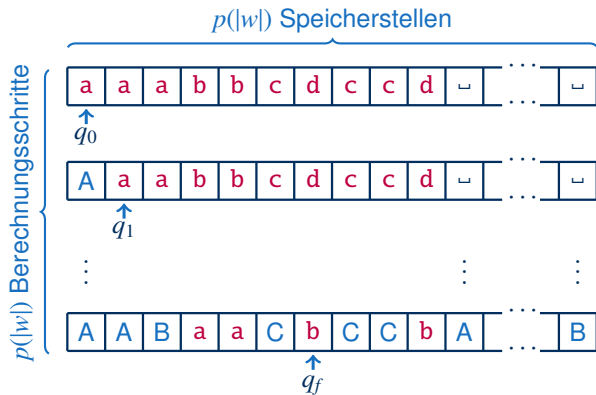
Wie stellen wir TM-Berechnungen in aussagenlogisch dar?

Idee:

- Jedes Problem in NP ist durch eine konkrete NTM \mathcal{M} in einer Zeit lösbar, die durch ein konkretes Polynom p gegeben ist.
 - Für eine Eingabe $|w|$ kennen wir also die maximale Zahl an Schritten $p(|w|)$ dieser Berechnung.
- ↪ Ein Lauf ist eine Folge von maximal $p(|w|)$ Konfigurationen, die jeweils maximal $p(|w|)$ Speicherstellen verwenden.

Ansatz: Kodiere \mathcal{M} und w so in Aussagenlogik, dass die möglichen Läufe genau den erfüllenden Wertzuweisungen entsprechen.

Cook/Levin: Beweisskizze (2)



Wir können diesen Lauf mit aussagenlogischen Atomen kodieren:

- $Q_{q,s}$: „TM ist in Schritt s in Zustand q “
- $P_{i,s}$: „TM-Kopf ist in Schritt s an Position i “
- $S_{a,i,s}$: „Das Band enthält in Schritt s Zeichen a an Position i “

Cook/Levin: Beweisskizze (3)

Wir können diesen Lauf mit aussagenlogischen Atomen kodieren:

- $Q_{q,s}$: „TM ist in Schritt s in Zustand q “
- $P_{i,s}$: „TM-Kopf ist in Schritt s an Position i “
- $S_{a,i,s}$: „Das Band enthält in Schritt s Zeichen a an Position i “

→ Es sind nur polynomiell viele Atome nötig.

Mit Hilfe von aussagenlogischen Formeln kann man alle Bedingungen ausdrücken, die für einen korrekten Lauf der NTM gelten müssen.

Die Übergangsrelation wird z.B. mit Formeln der folgenden Form kodiert:

$$(Q_{q,s} \wedge P_{i,s} \wedge S_{a,i,s}) \rightarrow \bigvee_{\substack{\langle q',b,D \rangle \in \delta(q,a), \\ i' = D(i)}} (S_{b,i,s+1} \wedge Q_{q',s+1} \wedge P_{i',s+1})$$

(für alle $s \in \{1, \dots, p(|w|)\}$ und Positionen $i \in \{1, \dots, p(|w|)\}$).

P-Schwere

Man kann Schwere und Vollständigkeit auch für andere Klassen definieren. Bei schwächeren Klassen muss man allerdings sicherstellen, dass die Reduktion selbst nicht schon so viele Ressourcen benötigt wie die Lösung des Problems!

Eine Sprache ist genau dann P-schwer, wenn jede Sprache in P mit logarithmischem Speicherbedarf darauf reduzierbar ist. Sie ist P-vollständig, wenn sie zudem in P liegt.

P-Schwere

Man kann Schwere und Vollständigkeit auch für andere Klassen definieren. Bei schwächeren Klassen muss man allerdings sicherstellen, dass die Reduktion selbst nicht schon so viele Ressourcen benötigt wie die Lösung des Problems!

Eine Sprache ist genau dann P-schwer, wenn jede Sprache in P mit logarithmischem Speicherbedarf darauf reduzierbar ist. Sie ist P-vollständig, wenn sie zudem in P liegt.

- P-vollständige Probleme sind die schwersten polynomiell lösbaren Probleme.
- Sie gelten als inhärent seriell, d.h. als nicht gut parallelisierbar (bisher nicht bewiesen).

P-Schwere

Man kann Schwere und Vollständigkeit auch für andere Klassen definieren. Bei schwächeren Klassen muss man allerdings sicherstellen, dass die Reduktion selbst nicht schon so viele Ressourcen benötigt wie die Lösung des Problems!

Eine Sprache ist genau dann P-schwer, wenn jede Sprache in P mit logarithmischem Speicherbedarf darauf reduzierbar ist. Sie ist P-vollständig, wenn sie zudem in P liegt.

- P-vollständige Probleme sind die schwersten polynomiell lösbaren Probleme.
- Sie gelten als inhärent seriell, d.h. als nicht gut parallelisierbar (bisher nicht bewiesen).

Beispiel: Die Erfüllbarkeit von aussagenlogischen Horn-Formeln (**HornSAT**) ist ein P-vollständiges Problem.

Deterministisch vs. Nichtdeterministisch

Wir haben bereits bemerkt, dass $P \subseteq NP$ gilt, da DTMs spezielle NTMs sind.

Bis heute ist nicht bekannt, ob die Umkehrung dieser Inklusion gilt oder nicht.

- Intuitiv gefragt: „Wenn es einfach ist, eine mögliche Lösung für ein Problem zu prüfen, ist es dann auch einfach, eine zu finden?“
- Übertrieben: „Can creativity be automated?“ (Wigderson, 2006)
- Das Problem ist seit über 40 Jahren ungelöst;
- es ist eines der größten offenen Probleme der Informatik/Mathematik unserer Zeit.
- Das Clay Mathematics Institute in Cambridge (Massachusetts) hat im Jahr 2000 ein Preisgeld von 1.000.000 USD für die Lösung ausgelobt. („Millenium Problem“)

Status von P vs. NP

Viele Menschen glauben, dass $P \neq NP$ gilt.

- Hauptargument: „Wenn $P = NP$ gälte, dann müsste jemand inzwischen einen polynomiellen Algorithmus für ein NP-vollständiges Problem gefunden haben.“
- „This is, in my opinion, a very weak argument. The space of algorithms is very large and we are only at the beginning of its exploration.“ (Moshe Vardi, 2002)
- Ein weiterer Grund für die verbreitete Meinung: Menschen finden es schwer, NP-Probleme zu lösen und können sich schwer vorstellen, wie man sie vereinfachen sollte – eventuell „human chauvinistic bravado“ (Zeilenberger, 2006).
- Es gibt noch bessere Argumente, aber keines mehr als eine Intuition.

Status von P vs. NP (2)

Verschiedene Ergebnisse sind denkbar:

Status von P vs. NP (2)

Verschiedene Ergebnisse sind denkbar:

- $P = NP$ könnte durch einen nichtkonstruktiven Beweis gezeigt werden.

Status von P vs. NP (2)

Verschiedene Ergebnisse sind denkbar:

- $P = NP$ könnte durch einen nichtkonstruktiven Beweis gezeigt werden.
- Die Antwort könnte unabhängig von den Standardaxiomen der Mathematik (ZFC) sein.

Status von P vs. NP (2)

Verschiedene Ergebnisse sind denkbar:

- $P = NP$ könnte durch einen nichtkonstruktiven Beweis gezeigt werden.
- Die Antwort könnte unabhängig von den Standardaxiomen der Mathematik (ZFC) sein.
- Selbst wenn $P \neq NP$ gilt, ist weiterhin unklar, ob NP-schwere Probleme wirklich exponentielle Zeit benötigen – es gibt viele andere super-polynomielle Funktionen . . .

Status von P vs. NP (2)

Verschiedene Ergebnisse sind denkbar:

- $P = NP$ könnte durch einen nichtkonstruktiven Beweis gezeigt werden.
- Die Antwort könnte unabhängig von den Standardaxiomen der Mathematik (ZFC) sein.
- Selbst wenn $P \neq NP$ gilt, ist weiterhin unklar, ob NP-schwere Probleme wirklich exponentielle Zeit benötigen – es gibt viele andere super-polynomielle Funktionen . . .
- Das Problem könnte für immer ungelöst bleiben.

Status von P vs. NP (2)

Verschiedene Ergebnisse sind denkbar:

- $P = NP$ könnte durch einen nichtkonstruktiven Beweis gezeigt werden.
- Die Antwort könnte unabhängig von den Standardaxiomen der Mathematik (ZFC) sein.
- Selbst wenn $P \neq NP$ gilt, ist weiterhin unklar, ob NP-schwere Probleme wirklich exponentielle Zeit benötigen – es gibt viele andere super-polynomielle Funktionen . . .
- Das Problem könnte für immer ungelöst bleiben.

Über 100 „Beweise“ zeigen, dass $P = NP$ wahr/falsch/beides/keines der beiden ist:

<https://www.win.tue.nl/~gwoegi/P-versus-NP.htm>

Most researchers think that . . .

Häufig wird im Bezug auf P vs. NP angeführt, dass die meisten Expert:innen an die Ungleichheit glauben. Das wurde zumindest punktuell bestätigt:

Ergebnisse einer Umfrage unter 152 Expert:innen [Gasarch 2012]:

- $P \neq NP$: 126 (83%)
- $P = NP$: 12 (9%)
- Weiß nicht/ist mir egal: 7 (4%)
- Unabhängig von ZFC: 5 (3%)
- „I don't **want** it to be equal.“: 1 (0.6%)

Aber Expert:innen haben sich auch schon früher geirrt . . .

Zusammenfassung und Ausblick

NP ist die Klasse der **nachweis-polynomiellen Probleme**.

NP-vollständige Probleme sind die schwersten Probleme in NP: Wenn man eines davon effizient löst, dann kann man alle Probleme in NP effizient lösen.

SAT ist **NP-vollständig** und **HornSAT** ist **P-vollständig**.

Offene Fragen:

- Haben Sie sich ausreichend auf die Prüfung vorbereitet?
- Haben Sie noch inhaltliche Fragen? (Wir werden Konsultationstermine anbieten.)
- Gilt $P = NP$?