Karlsruher Institut für Technologie

Theoretische Grundlagen der Informatik

Vorlesung am 18.11.2021

Torsten Ueckerdt | 18. November 2021

Letzte Vorlesung

- **Probleme**
  - Optimierungsprobleme, Optimalwertprobleme, Entscheidungsprobleme
  - Problem Π ist Klasse D_Π von Instanzen I.
- **Eingabegrößen**
  - Kodierungsschema $s\colon D_\Pi \to \Sigma^*$ über Alphabet Σ^*.
  - Kodierung $s(I)$ einer Instanz I ist ein Wort aus Σ^*.
  - Inputlänge $|s(I)|$ ist Länge des Wortes.
- **Entscheidungsprobleme**
  - Ja–Instanzen J_Π und Nein–Instanzen N_Π
  - Sprache $L[\Pi, s]$ der Kodierungen aller Ja–Instanzen
  - TM $\mathcal{M}$ löst Π wenn $\mathcal{M}$ Sprache $L[\Pi, s]$ entscheidet.
- **Laufzeiten**
  - Zeitkomplexitätsfunktion $T_\mathcal{M}(n)$ von $\mathcal{M}$ bei Eingaben der Länge n
  - Die Klasse $\mathcal{P}$: Sprachen von $\mathcal{M}$ mit $T_\mathcal{M}(n)$ polynomiell in n.
- Am Beispiel TSP: Entscheidung → Optimalwert → Optimierung

Die Nichtdeterministische Turing-Maschine (NTM)

Die deterministische TM ist von der Form $\mathcal{M} = (Q, \Sigma, \sqcup, \Gamma, s, \delta, F)$

- wobei $\delta\colon Q \times \Gamma \to Q \times \Gamma \times \{L, N, R\}$
- z.B. $\delta(q_1, a) = (q_2, b, L)$

Die nichtdeterministische TM ist von der Form $\mathcal{M} = (Q, \Sigma, \sqcup, \Gamma, s, \delta, F)$

- wobei $\delta\colon Q \times (\Gamma \cup \{\varepsilon\}) \to 2^{Q \times \Gamma \times \{L, N, R\}}$
- z.B. $\delta(q_1, a) = \{(q_2, b, L), (q_3, b, R), (q_1, a, N)\}$
  $\delta(q_2, a) = \emptyset$ oder $\delta(q_2, \varepsilon) = \{(q_2, a, L), (q_1, \sqcup, N)\}$
- Es gibt also ε-Übergänge und Wahlmöglichkeiten

Die Nichtdeterministische Turing-Maschine (NTM)

Die deterministische TM ist von der Form $\mathcal{M} = (Q, \Sigma, \sqcup, \Gamma, s, \delta, F)$

- wobei $\delta\colon Q \times \Gamma \to Q \times \Gamma \times \{L, N, R\}$
- z.B. $\delta(q_1, a) = (q_2, b, L)$

Die nichtdeterministische TM ist von der Form $\mathcal{M} = (Q, \Sigma, \sqcup, \Gamma, s, \delta, F)$

- wobei $\delta\colon Q \times (\Gamma \cup \{\varepsilon\}) \to 2^{Q \times \Gamma \times \{L,N,R\}}$
- z.B. $\delta(q_1, a) = \{(q_2, b, L), (q_3, b, R), (q_1, a, N)\}$
  $\delta(q_2, a) = \emptyset$ oder $\delta(q_2, \varepsilon) = \{(q_2, a, L), (q_1, \sqcup, N)\}$
- Es gibt also ε-Übergänge und Wahlmöglichkeiten

Eine NTM $\mathcal{M}$ akzeptiert eine Eingabe w, wenn es mindestens eine akzeptierende Abarbeitung von w gibt.

- $L_{\mathcal{M}} = \{w \in \Sigma^* : \mathcal{M} \text{ akzeptiert } w\}$

$\rightsquigarrow$ analog zu Nichtdeterminismus bei endlichen Automaten

Den Nichtdeterminismus “auslagern”

Sei $\mathcal{M}$ eine NTM und w eine Eingabe.

- Während der Abarbeitung von w gibt es zu jedem Zeitpunkt höchstens $X < \infty$ mögliche Übergänge.
- Jeder mögliche (endliche) Berechnungsweg (= Abarbeitung) ist eindeutig beschrieben durch eine (endliche) Folge von Zahlen aus $1, \ldots, X$.
- Ist diese Folge schon vorher nichtdeterministisch gegeben, so könnte die Turing-Maschine danach deterministisch arbeiten.

Den Nichtdeterminismus “auslagern”

Sei $\mathcal{M}$ eine NTM und w eine Eingabe.

- Während der Abarbeitung von w gibt es zu jedem Zeitpunkt höchstens $X < \infty$ mögliche Übergänge.
- Jeder mögliche (endliche) Berechnungsweg (= Abarbeitung) ist eindeutig beschrieben durch eine (endliche) Folge von Zahlen aus $1, \ldots, X$.
- Ist diese Folge schon vorher nichtdeterministisch gegeben, so könnte die Turing-Maschine danach deterministisch arbeiten.

Dies ist die Idee hinter der **Orakel-Turing-Maschine**:

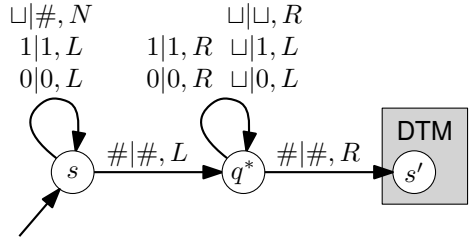
- **1. Stufe:** Es wird nichtdeterministisch vor die Eingabe auf das Band geschrieben.
- **2. Stufe:** Es wird deterministisch das gesamte Band bearbeitet.

Orakel-Turing-Maschine als NTM

- **1. Stufe:** Es wird nichtdeterministisch vor die Eingabe auf das Band geschrieben.
  - Alphabet $\Sigma = \{0, 1\}$, Startzustand $s \in Q$, Orakelzustand $q^* \in Q$, Trennzeichen $\# \in \Gamma$
  - definiere Übergange

$$\delta(s, 0) = \{(s, 0, L)\} \text{ und } \delta(s, 1) = \{(s, 1, L)\}$$
$$\delta(s, \sqcup) = \{(q^*, \#, L)\}$$
$$\delta(q^*, \sqcup) = \{(q^*, 1, L), (q^*, 0, L), (q^*, \sqcup, R)\}$$
$$\delta(q^*, \#) = \{(s', \#, R)\}$$

  - s' Startzustand für 2. Stufe



- **2. Stufe:** Es wird deterministisch das gesamte Band bearbeitet.
  - $|\delta(q, a)| = 1$ für alle $q \in Q - \{s, q^*\}$, $a \in \Gamma$
  - keine Übergänge zu s oder q^*
  - kein Entfernen oder Schreiben von $\#$

NTM und Orakel-TM

Die “klassische” nichtdeterministische Turing-Maschine:

- Übergangsfunktion δ zu Übergangsrelation erweitert
- ermöglicht Wahlmöglichkeiten und ε–Übergänge
  $\rightsquigarrow$ vergleiche endliche Automaten

Die **Orakel-Turing-Maschine**:

- äquivalentes Modell einer nichtdeterministischen Turing-Maschine
- basiert auf nichtdeterministischem Orakel und deterministischer endlichen Kontrolle
- Dies kommt der Intuition näher und wird von uns (fast ausschliesslich) verwendet werden.

NTM und Orakel-TM

Die “klassische” nichtdeterministische Turing-Maschine:

- Übergangsfunktion δ zu Übergangsrelation erweitert
- ermöglicht Wahlmöglichkeiten und ε–Übergänge
  $\rightsquigarrow$ vergleiche endliche Automaten

Die **Orakel-Turing-Maschine**:

- äquivalentes Modell einer nichtdeterministischen Turing-Maschine
- basiert auf nichtdeterministischem Orakel und deterministischer endlichen Kontrolle
- Dies kommt der Intuition näher und wird von uns (fast ausschliesslich) verwendet werden.

NTM und Orakel-TM akzeptieren ein Wort $x \in \Sigma^*$ genau dann, wenn es mindestens eine akzeptierende Berechnung gibt.

Übertragung auf Entscheidungsprobleme Π

Die Eingabe ist ein Wort aus Σ^*, zum Beispiel eine Kodierung einer Instanz $I \in D_\Pi$ des Entscheidungsproblems Π.

- **1. Stufe:** Es wird ein Orakel aus Γ^* berechnet, zum Beispiel ein Lösungsbeispiel für I, also ein Indikator, warum $I \in J_\Pi$ gelten sollte.
- **2. Stufe:** Hier wird nun dieser Lösungsvorschlag überprüft, d.h. es wird geprüft ob $I \in J_\Pi$.

Beispiel TSP

- **1. Stufe:** Es wird zum Beispiel eine zykl. Permutation $x_1 x_2 \cdots x_n$ der Knotenmenge V vorgeschlagen.
  D.h. $\boxed{(x_1, x_2, \ldots, x_n) \# G = (V, E), c, k}$ ist die Eingabe für 2. Stufe.
- **2. Stufe:** Es wird nun überprüft, ob $x_1 \to x_2 \to \cdots \to x_n$ eine Tour in $G = (V, E)$ darstellt, deren Länge bezüglich c nicht größer als k ist.

Bemerkungen zur Orakel-TM

- Das Orakel kann ein beliebiges Wort aus Γ^* sein.
- Darum muss in der Überprüfungsphase (2.Stufe) geprüft werden, ob das Orakel ein zulässiges Lösungsbeispiel für die gegebene Eingabe ist.
- Ist dies der Fall, so kann die Berechnung zu diesem Zeitpunkt mit der Antwort “Ja” beendet werden.
  $\rightsquigarrow$ gehe in Zustand q_J
- Ist dies nicht der Fall, so kann die Berechnung zu diesem Zeitpunkt mit der Antwort “Nein” beendet werden.
  $\rightsquigarrow$ gehe in Zustand q_N

- Jede Orakel-TM $\mathcal{M}$ hat zu einer gegebenen Eingabe x eine unendliche Anzahl möglicher Berechnungen, eine zu jedem Orakel aus Γ^*.
- Endet mindestens eine in q_J, so wird x akzeptiert.

Zeitkomplexität für NTM

- Die **Zeit**, die eine nichtdeterministische Turing-Maschine $\mathcal{M}$ benötigt, um ein Wort $x \in L_{\mathcal{M}}$ zu akzeptieren, ist definiert als die minimale Anzahl von Schritten, die $\mathcal{M}$ in den Zustand q_J überführt.
- Die **Zeitkomplexitätsfunktion** $T_{\mathcal{M}} \colon \mathbb{Z}^+ \to \mathbb{Z}^+$ einer NTM $\mathcal{M}$ ist definiert durch

$$T_{\mathcal{M}}(n) := \max\left(\{1\} \cup \left\{m\colon \begin{array}{l}\text{es gibt ein } x \in L_{\mathcal{M}} \text{ mit } |x| = n, \text{ so dass die Zeit,}\\ \text{die } \mathcal{M} \text{ benötigt, um } x \text{ zu akzeptieren, } m \text{ ist}\end{array}\right\}\right)$$

Zeitkomplexität für NTM

- Die **Zeit**, die eine nichtdeterministische Turing-Maschine $\mathcal{M}$ benötigt, um ein Wort $x \in L_{\mathcal{M}}$ zu akzeptieren, ist definiert als die minimale Anzahl von Schritten, die $\mathcal{M}$ in den Zustand q_J überführt.
- Die **Zeitkomplexitätsfunktion** $T_{\mathcal{M}} \colon \mathbb{Z}^+ \to \mathbb{Z}^+$ einer NTM $\mathcal{M}$ ist definiert durch

$$T_{\mathcal{M}}(n) := \max\left(\{1\} \cup \left\{m\colon \begin{array}{l}\text{es gibt ein } x \in L_{\mathcal{M}} \text{ mit } |x| = n\text{, so dass die Zeit,}\\ \text{die } \mathcal{M} \text{ benötigt, um } x \text{ zu akzeptieren, } m \text{ ist}\end{array}\right\}\right)$$

Bemerkung 1

- Zur Berechnung von $T_{\mathcal{M}}(n)$ wird für jedes $x \in L_{\mathcal{M}}$ mit $|x| = n$ eine kürzeste akzeptierende Berechnung betrachtet.
- Anschließend wird von diesen kürzesten die längste bestimmt.
- Somit ergibt sich eine *worst–case* Abschätzung.

Zeitkomplexität für NTM

- Die **Zeit**, die eine nichtdeterministische Turing-Maschine $\mathcal{M}$ benötigt, um ein Wort $x \in L_{\mathcal{M}}$ zu akzeptieren, ist definiert als die minimale Anzahl von Schritten, die $\mathcal{M}$ in den Zustand q_J überführt.
- Die **Zeitkomplexitätsfunktion** $T_{\mathcal{M}} \colon \mathbb{Z}^+ \to \mathbb{Z}^+$ einer NTM $\mathcal{M}$ ist definiert durch

$$T_{\mathcal{M}}(n) := \max\left(\{1\} \cup \left\{m\colon \begin{array}{l}\text{es gibt ein } x \in L_{\mathcal{M}} \text{ mit } |x| = n, \text{ so dass die Zeit,}\\ \text{die } \mathcal{M} \text{ benötigt, um } x \text{ zu akzeptieren, } m \text{ ist}\end{array}\right\}\right)$$

Bemerkung 2

- Die Zeitkomplexität hängt nur von der Anzahl der Schritte ab, die bei einer akzeptierenden Berechnung auftreten.
- Hierbei umfasst die Anzahl der Schritte auch die Schritte der Orakelphase.
- Per Konvention ist $T_{\mathcal{M}}(n) = 1$, falls es keine Eingabe x der Länge n gibt, die von $\mathcal{M}$ akzeptiert wird.

Die Klasse $\mathcal{NP}$

Die Klasse $\mathcal{NP}$ ist die Menge aller Sprachen L (Entscheidungsprobleme), für die eine nichtdeterministische Turing-Maschine existiert, deren Zeitkomplexitätsfunktion polynomial beschränkt ist, d.h. es existiert ein Polynom p mit

$$T_{\mathcal{M}}(n) \leq p(n).$$

($\mathcal{NP}$ steht für **nichtdeterministisch polynomial**.)

Die Klasse $\mathcal{NP}$

Die Klasse $\mathcal{NP}$ ist die Menge aller Sprachen L (Entscheidungsprobleme), für die eine nichtdeterministische Turing-Maschine existiert, deren Zeitkomplexitätsfunktion polynomial beschränkt ist, d.h. es existiert ein Polynom p mit

$$T_{\mathcal{M}}(n) \leq p(n).$$

($\mathcal{NP}$ steht für **nichtdeterministisch polynomial**.)

Bemerkung

- Informell ausgedrückt gehört Π zu $\mathcal{NP}$, falls Π folgende Eigenschaft hat:
  Ist die Antwort bei Eingabe eines Beispiels I von Π Ja, dann kann die Korrektheit der Antwort in polynomialer Zeit überprüft werden.

Die Klasse $\mathcal{NP}$

Die Klasse $\mathcal{NP}$ ist die Menge aller Sprachen L (Entscheidungsprobleme), für die eine nichtdeterministische Turing-Maschine existiert, deren Zeitkomplexitätsfunktion polynomial beschränkt ist, d.h. es existiert ein Polynom p mit

$$T_M(n) \leq p(n).$$

($\mathcal{NP}$ steht für **nichtdeterministisch polynomial**.)

Beispiel: TSP$\in \mathcal{NP}$:

Denn zu gegebenem $G = (V, E), c, k$ und einer festen zykl. Permutation $x_1 x_2 \cdots x_n$ von V kann in $O(|V| \cdot \log C)$ (wobei C die größte vorkommende Zahl ist) Schritten überprüft werden, ob

$$\{x_i, x_{i+1}\} \in E \text{ für } i = 1, \ldots, n-1 \quad \text{und} \quad \sum_{i=1}^{n-1} c(\{x_i, x_i + 1\}) \leq k$$

gilt.

$\mathcal{P}$ vs. $\mathcal{NP}$

Die Klasse $\mathcal{P}$ ist die Menge aller Sprachen L (Entscheidungsprobleme), für die eine deterministische Turing-Maschine existiert, deren Zeitkomplexitätsfunktion polynomial beschränkt ist.

$\rightsquigarrow$ Bei Eingabe einer Instanz I von Π kann die Existenz einer Lösung in polynomialer Zeit überprüft werden.

Große Frage:

Ist $\mathcal{P} = \mathcal{NP}$?

Die Klasse $\mathcal{NP}$ ist die Menge aller Sprachen L (Entscheidungsprobleme), für die eine nichtdeterministische Turing-Maschine existiert, deren Zeitkomplexitätsfunktion polynomial beschränkt ist.

$\rightsquigarrow$ Existiert für die Eingabe einer Instanz I von Π eine Lösung, dann kann die Korrektheit einer Lösung in polynomialer Zeit überprüft werden.

Große Frage der Theoretischen Informatik

- Trivialerweise gilt: $\mathcal{P} \subseteq \mathcal{NP}$ (Da jede DTM auch eine NTM ist.)
- **Frage:** Gilt $\mathcal{P} \subset \mathcal{NP}$ oder $\mathcal{P} = \mathcal{NP}$?
- Die Vermutung ist, dass $\mathcal{P} \neq \mathcal{NP}$ gilt.

Satz.

Alle Sprachen in $\mathcal{NP}$ sind entscheidbar.

Beweis.

- Sei L eine Sprache in $\mathcal{NP}$ und $\mathcal{M}$ eine zugehörige Orakel-TM.
- Für jedes Polynom p betrachte die folgende DTM:
  - Berechne Länge n der Eingabe.
  - Schreibe nacheinander jedes mögliche Orakelwörter der Länge höchstens $p(n)$ vor die Eingabe.
  - Überprüfe mit endlicher Kontrolle von $\mathcal{M}$ jedes Orakelwort.
    - Aber: Stoppe endliche Kontrolle nach $p(n)$ Schritten.
  - Lehne Eingabe ab, wenn alle Orakelwörter nicht akzeptiert werden.
- Mindestens eine solche DTM entscheidet L.

Große Frage der Theoretischen Informatik

- Trivialerweise gilt: $\mathcal{P} \subseteq \mathcal{NP}$ (Da jede DTM auch eine NTM ist.)
- **Frage:** Gilt $\mathcal{P} \subset \mathcal{NP}$ oder $\mathcal{P} = \mathcal{NP}$?
- Die Vermutung ist, dass $\mathcal{P} \neq \mathcal{NP}$ gilt.
- Dazu betrachten wir Probleme, die zu den schwersten Problemen in $\mathcal{NP}$ gehören.
- Dabei ist am schwersten im folgenden Sinne gemeint:
- Wenn ein schwerstes $\mathcal{NP}$-Problem trotzdem in $\mathcal{P}$ liegt, so kann man folgern, dass alle $\mathcal{NP}$-Probleme in $\mathcal{P}$ liegen, d.h. $\mathcal{P} = \mathcal{NP}$.
- Diese schwersten $\mathcal{NP}$-Probleme sind also Kandidaten, um $\mathcal{P}$ und $\mathcal{NP}$ zu trennen.
- Es wird sich zeigen, dass alle diese schwersten $\mathcal{NP}$-Probleme im Wesentlichen gleich schwer sind.

NP-Vollständigkeit für Sprachen

Definition.

Eine polynomiale Transformation einer Sprache $L_1 \subseteq \Sigma_1^*$ in eine Sprache $L_2 \subseteq \Sigma_2^*$ ist eine Funktion $f\colon \Sigma_1^* \to \Sigma_2^*$ mit den Eigenschaften:

- es existiert eine polynomiale deterministische Turing-Maschine, die f berechnet;
- für alle $x \in \Sigma_1^*$ gilt: $\quad x \in L_1 \Leftrightarrow f(x) \in L_2$.

Wir schreiben dann $L_1 \propto L_2$ (L_1 ist polynomial transformierbar in L_2).

Eine Sprache L heißt **$\mathcal{NP}$–vollständig**, falls gilt:

- $L \in \mathcal{NP}$ und
- für alle $L' \in \mathcal{NP}$ gilt $L' \propto L$.

NP-Vollständigkeit für Entscheidungsprobleme

Definition.

Ein Entscheidungsproblem Π_1 ist polynomial transformierbar in das Entscheidungsproblem Π_2, wenn eine Funktion $f\colon D_{\Pi_1} \to D_{\Pi_2}$ existiert mit folgenden Eigenschaften:

- f ist durch einen polynomialen Algorithmus berechenbar;
- $\forall\, I \in D_{\Pi_1}\colon \quad I \in J_{\Pi_1} \iff f(I) \in J_{\Pi_2}$.

Wir schreiben dann $\Pi_1 \propto \Pi_2$.

Ein Entscheidungsproblem Π heißt **$\mathcal{NP}$-vollständig**, falls gilt:

- $\Pi \in \mathcal{NP}$ und
- für alle $\Pi' \in \mathcal{NP}$ gilt $\Pi' \propto \Pi$.

Transitivität der poly. Transformation

Eine polynomiale Transformation einer Sprache $L_1 \subseteq \Sigma_1^*$ in eine Sprache $L_2 \subseteq \Sigma_2^*$ ist eine Funktion $f\colon \Sigma_1^* \to \Sigma_2^*$ mit den Eigenschaften:

- es existiert eine polynomiale deterministische Turing-Maschine, die f berechnet;
- für alle $x \in \Sigma_1^*$ gilt: $\quad x \in L_1 \Leftrightarrow f(x) \in L_2$.

Wir schreiben dann $L_1 \propto L_2$ (L_1 ist polynomial transformierbar in L_2).

Lemma.

Die Relation $\propto$ ist transitiv, d.h. aus $L_1 \propto L_2$ und $L_2 \propto L_3$ folgt $L_1 \propto L_3$.

Beweis. Die Hintereinanderausführung zweier polynomialer Transformationen ist wieder eine polynomiale Transformation.

Beobachtung

Korollar.

Falls $L_1, L_2 \in \mathcal{NP}$, $L_1 \propto L_2$ und L_1 $\mathcal{NP}$-vollständig, dann ist auch L_2 $\mathcal{NP}$-vollständig.

Bedeutung.

Um also zu zeigen, dass ein Entscheidungsproblem Π $\mathcal{NP}$-vollständig ist, gehen wir folgendermaßen vor. Wir beweisen:

- $\Pi \in \mathcal{NP}$ und
- $\Pi' \propto \Pi$ für ein bekanntes $\mathcal{NP}$-vollständiges Problem Π'.

Problem.

- Wir haben noch kein "bekanntes $\mathcal{NP}$-vollständiges Problem".
- Das erste $\mathcal{NP}$-vollständige Problem ist das Erfüllbarkeitsproblem SAT (satisfiability).

Das Problem SAT (satisfiability)

Sei $U = \{u_1, \ldots, u_m\}$ eine Menge von booleschen Variablen.
Es heißen $u_i, \overline{u_i}$ Literale.
Eine Wahrheitsbelegung für U ist eine Funktion $t\colon U \to \{\texttt{wahr}, \texttt{falsch}\}$.
Eine Klausel ist ein Boolescher Ausdruck der Form

$$y_1 \vee \ldots \vee y_s \quad \text{mit} \quad y_i \in \{u_1, \ldots, u_m\} \cup \{\overline{u_1}, \ldots, \overline{u_m}\} \text{ Literale}$$

Problem SAT

Gegeben: Menge U von Variablen, Menge C von Klauseln über U.
Frage: Existiert eine Wahrheitsbelegung von U, so dass jede Klausel in C erfüllt wird?

Beispiel:
$U = \{u_1, u_2\}$ mit $C = \{u_1 \vee \overline{u_2},\ \overline{u_1} \vee u_2\}$ ist Ja-Instanz von SAT.
Wahrheitsbelegung $t(u_1) = t(u_2) = \texttt{wahr}$ erfüllt alle Klauseln in C.

Weitere Beispiele für SAT-Instanzen

Erfüllbar (Ja-Instanz):

$$U = \{a, b, c, d, e\}, C = \{c \vee \overline{d},\ \ \overline{a} \vee b \vee \overline{c} \vee d \vee e,\ \ \overline{c} \vee d\}$$

eine Lösung: $t(a) = \texttt{falsch}, t(b) = t(c) = t(d) = t(e) = \texttt{wahr}$

Nicht erfüllbar (Nein-Instanz):

$$U = \{a, b, c\}, C = \{a \vee b,\ \ \overline{a},\ \ \overline{b} \vee c,\ \ \overline{c}\}$$

a	b	c	$a \vee b$	$\overline{a}$	$\overline{b} \vee c$	$\overline{c}$
wahr	wahr	wahr	wahr	falsch	wahr	falsch
⋮	⋮	⋮	⋮	⋮	⋮	⋮
falsch	falsch	falsch	falsch	wahr	wahr	wahr

Der Satz von Cook (Steven Cook, 1971)

- Problem Π ist $\mathcal{NP}$-vollständig wenn
  - $\Pi \in \mathcal{NP}$ und
  - $\Pi' \propto \Pi$ für alle $\Pi' \in \mathcal{NP}$

Satz von Cook.

SAT ist $\mathcal{NP}$-vollständig.

Der Satz von Cook (Steven Cook, 1971)

- Problem Π ist $\mathcal{NP}$-vollständig wenn
  - $\Pi \in \mathcal{NP}$ und
  - $\Pi' \propto \Pi$ für alle $\Pi' \in \mathcal{NP}$

Satz von Cook.

SAT ist $\mathcal{NP}$-vollständig.

Beweis: Nächste Vorlesung!

Der Satz von Cook (Steven Cook, 1971)

- Problem Π ist $\mathcal{NP}$-vollständig wenn
  - $\Pi \in \mathcal{NP}$ und
  - ~~$\Pi' \propto \Pi$ für alle $\Pi' \in \mathcal{NP}$~~
  - $\Pi' \propto \Pi$ für ein $\mathcal{NP}$-vollständiges Π'

Satz von Cook.

SAT ist $\mathcal{NP}$-vollständig.

Beweis: Nächste Vorlesung!

Der Satz von Cook (Steven Cook, 1971)

- Problem Π ist $\mathcal{NP}$-vollständig wenn
  - $\Pi \in \mathcal{NP}$ und
  - ~~$\Pi' \propto \Pi$ für alle $\Pi' \in \mathcal{NP}$~~
  - $\Pi' \propto \Pi$ für ein $\mathcal{NP}$-vollständiges Π'

Satz von Cook.

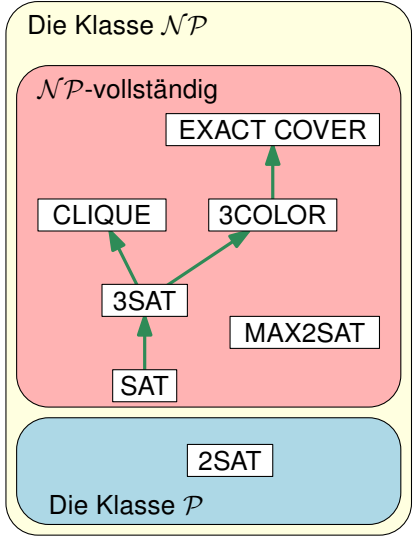
SAT ist $\mathcal{NP}$-vollständig.

Beweis: Nächste Vorlesung!

- **Polynomiale Transformation** $\Pi' \propto \Pi$
  - Instanzen von $\Pi' \longrightarrow$ Instanzen von Π
  - in polynomialer Zeit berechenbar
  - Ja-Instanz von $\Pi' \longrightarrow$ Ja-Instanz von Π
    Nein-Instanz von $\Pi' \longrightarrow$ Nein-Instanz von Π

Der Plan

- 3SAT ist $\mathcal{NP}$-vollständig
  $\rightsquigarrow$ 3SAT $\in \mathcal{NP}$
  $\rightsquigarrow$ SAT $\propto$ 3SAT
- 2SAT ist in $\mathcal{P}$
- MAX2SAT ist $\mathcal{NP}$-vollständig
  $\rightsquigarrow$ Übung
- CLIQUE ist $\mathcal{NP}$-vollständig
  $\rightsquigarrow$ CLIQUE $\in \mathcal{NP}$
  $\rightsquigarrow$ 3SAT $\propto$ CLIQUE



- 3COLOR ist $\mathcal{NP}$-vollständig
  $\rightsquigarrow$ 3COLOR $\in \mathcal{NP}$
  $\rightsquigarrow$ 3SAT $\propto$ 3COLOR
- EXACT COVER ist $\mathcal{NP}$-vollständig
  $\rightsquigarrow$ EXACT COVER $\in \mathcal{NP}$
  $\rightsquigarrow$ 3COLOR $\propto$ EXACT COVER

Das Problem 3SAT

Karlsruher Institut für Technologie

Problem 3SAT

Gegeben: Menge U von Variablen
Menge C von Klauseln über U
wobei jede Klausel genau drei Literale enthält

Frage: Existiert eine erfüllende Wahrheitsbelegung für C?

Das Problem 3SAT

Problem 3SAT

Gegeben: Menge U von Variablen
Menge C von Klauseln über U
wobei jede Klausel genau drei Literale enthält

Frage: Existiert eine erfüllende Wahrheitsbelegung für C?

Satz.

Das Problem 3SAT ist $\mathcal{NP}$-vollständig.

Beweis: $\mathcal{NP}$-Vollständigkeit von 3SAT

3SAT $\in \mathcal{NP}$:

- Es existiert eine nichtdeterministische Turing-Maschine mit polynomialer Zeitkomplexitätsfunktion die in q_J hält bei Ja-Instanzen.

Beweis: $\mathcal{NP}$-Vollständigkeit von 3SAT

3SAT $\in \mathcal{NP}$:

- Es existiert eine nichtdeterministische Turing-Maschine mit polynomialer Zeitkomplexitätsfunktion die in q_J hält bei Ja-Instanzen.
- Wir konstruieren eine Orakel-Turing-Maschine:
  - Das Orakel ist eine Wahrheitsbelegung $t\colon U \to \{\texttt{wahr}, \texttt{falsch}\}$.
  - Die endliche Kontrolle überprüft, ob jede Klausel in C durch t erfüllt ist.
  - Wenn alle Klauseln erfüllt, gehe in q_J.
  - Wenn (mindestens) eine Klausel nicht erfüllt, gehe in q_N.
  - Laufzeit ist linear in der Größe der eingegebenen Klauselmenge C.

Beweis: $\mathcal{NP}$-Vollständigkeit von 3SAT

3SAT $\in \mathcal{NP}$:

- Es existiert eine nichtdeterministische Turing-Maschine mit polynomialer Zeitkomplexitätsfunktion die in q_J hält bei Ja-Instanzen.
- Wir konstruieren eine Orakel-Turing-Maschine:
  - Das Orakel ist eine Wahrheitsbelegung $t\colon U \to \{\texttt{wahr}, \texttt{falsch}\}$.
  - Die endliche Kontrolle überprüft, ob jede Klausel in C durch t erfüllt ist.
  - Wenn alle Klauseln erfüllt, gehe in q_J.
  - Wenn (mindestens) eine Klausel nicht erfüllt, gehe in q_N.
  - Laufzeit ist linear in der Größe der eingegebenen Klauselmenge C.
- Für eine feste Wahrheitsbelegung t kann in polynomialer Zeit $O(|C|)$ überprüft werden, ob t alle Klauseln aus C erfüllt.

Beweis: $\mathcal{NP}$-Vollständigkeit von 3SAT

SAT $\propto$ 3SAT:

- Wir geben eine polynomiale Transformation f von SAT zu 3SAT an.
- Gegeben sei eine SAT-Instanz I

Wir konstruieren eine 3SAT-Instanz $f(I)$ indem wir jede Klausel c in I einzeln auf Klausel(n) $f(c)$ in $f(I)$ abbilden:

- Besteht die Klausel $c = x_1$ aus **einem** Literal,
  so wird c auf $x_1 \vee x_1 \vee x_1$ abgebildet.
- Besteht die Klausel $c = x_1 \vee x_2$ aus **zwei** Literalen,
  so wird c auf $x_1 \vee x_2 \vee x_1$ abgebildet.
- Besteht die Klausel c aus **drei** Literalen,
  so wird c auf sich selbst abgebildet.

Beweis: $\mathcal{NP}$-Vollständigkeit von 3SAT

Wir konstruieren eine 3SAT-Instanz $f(I)$ indem wir jede Klausel c in I einzeln auf Klausel(n) $f(c)$ in $f(I)$ abbilden:

- Besteht die Klausel $c = x_1 \vee \cdots \vee x_k$ aus $k > 3$ Literalen, bilde c wie folgt ab:
- Führe $k - 3$ neue Variablen $y_{c,3}, \ldots, y_{c,k-1}$ ein.
- Bilde c auf die folgenden $k - 2$ Klauseln ab:

$x_1 \vee x_2 \vee y_{c,3}$	"Falls $x_1, x_2 = \texttt{falsch} \rightsquigarrow y_{c,3}$ muss $\texttt{wahr}$"
$\overline{y_{c,3}} \vee x_3 \vee y_{c,4}$	"$y_{c,3} = \texttt{wahr}, x_3 = \texttt{falsch} \rightsquigarrow y_{c,4} = \texttt{wahr}$"
$\vdots$	$\vdots$
$\overline{y_{c,k-2}} \vee x_{k-2} \vee y_{c,k-1}$	"$y_{c,k-2} = \texttt{wahr}, x_{k-2} = \texttt{falsch} \rightsquigarrow y_{c,k-1} = \texttt{wahr}$"
$\overline{y_{c,k-1}} \vee x_{k-1} \vee x_k$	"$y_{c,k-1} = \texttt{wahr} \rightsquigarrow x_{k-1}$ oder x_k muss $\texttt{wahr}$"

- Diese Klauseln lassen sich in Zeit $O(|C| \cdot |U|)$ konstruieren.

Beweis: $\mathcal{NP}$-Vollständigkeit von 3SAT

Noch zu zeigen:

- I ist erfüllbar $\Leftrightarrow$ $f(I)$ ist erfüllbar

I ist erfüllbar $\Rightarrow$ $f(I)$ ist erfüllbar

- Sei die SAT-Instanz I erfüllbar.
- Wir setzen eine erfüllende Wahrheitsbelegung t von I auf $f(I)$ fort.
  - Also wenn es Literal x in I und $f(I)$ gibt, lasse $t(x)$ wie gehabt.
- Wir untersuchen jede Klausel $c = x_1 \vee \cdots \vee x_k$ in I einzeln.
- Da c von t erfüllt ist, gilt für mindestens ein i, dass $t(x_i) = \texttt{wahr}$.
- Fall $k \leq 3$: Damit ist auch Klausel c in $f(I)$ erfüllt.
- Fall $k > 3$: Setze für $j = 3, \ldots, k-1$

$$t(y_{c,j}) = \begin{cases} \texttt{wahr} & \text{falls } t(x_i) = \texttt{wahr} \text{ für (mind.) ein } i \geq j \\ \texttt{falsch} & \text{falls } t(x_i) = \texttt{falsch} \text{ für alle } i \geq j \end{cases}$$

- Diese Erweiterung erfüllt alle Klauseln in $f(I)$, die zu c gehören.

I ist erfüllbar $\Leftarrow$ $f(I)$ ist erfüllbar

- Sei t eine erfüllende Wahrheitsbelegung von $f(I)$.
  - Nehme für jedes Literal x in I die Belegung $t(x)$ wie in $f(I)$.
- Wir untersuchen jede Klausel $c = x_1 \vee \cdots \vee x_k$ in I einzeln.
- Fall $k \leq 3$: Dann ist Klausel c auch in $f(I)$, also durch t erfüllt.
- Fall $k > 3$: Alle Klauseln in $f(I)$ zu Klausel c in I sind erfüllt:

$x_1 \vee x_2 \vee y_{c,3}$

Falls $t(x_1), t(x_2) = \texttt{falsch}$, dann $t(y_{c,3}) = \texttt{wahr}$.

$\overline{y_{c,j}} \vee x_j \vee y_{c,j+1}$

Falls $t(y_{c,j}) = \texttt{wahr}, t(x_j) = \texttt{falsch}$, dann $t(y_{c,j+1}) = \texttt{wahr}$.

$\overline{y_{c,k-1}} \vee x_{k-1} \vee x_k$

Falls $t(y_{c,k-1}) = \texttt{wahr}$, dann $t(x_{k-1})$ oder $t(x_k) = \texttt{wahr}$.

- Also ist $t(x_i) = \texttt{wahr}$ für (mind.) ein i, und demnach c erfüllt durch t.

Das Problem 2SAT

Problem 2SAT

Gegeben: Menge U von Variablen
Menge C von Klauseln über U
wobei jede Klausel genau zwei Literale enthält

Frage: Existiert eine erfüllende Wahrheitsbelegung für C?

Satz.

Das Problem 2SAT liegt in $\mathcal{P}$.

Beweis: Übung