# Theoretische Grundlagen der Informatik

## Vorlesung am 1.12.2022

Torsten Ueckerdt | 1. Dezember 2022

# Letzte Vorlesung

- **Probleme**
  - Optimierungsprobleme, Optimalwertprobleme, Entscheidungsprobleme
  - Problem $\Pi$ ist Klasse $D_\Pi$ von Instanzen $I$.
- **Eingabegrößen**
  - Kodierungsschema $s\colon D_\Pi \to \Sigma^*$ über Alphabet $\Sigma^*$.
  - Kodierung $s(I)$ einer Instanz $I$ ist ein Wort aus $\Sigma^*$.
  - Inputlänge $|s(I)|$ ist Länge des Wortes.
- **Entscheidungsprobleme**
  - Ja–Instanzen $J_\Pi$ und Nein–Instanzen $N_\Pi$
  - Sprache $L[\Pi, s]$ der Kodierungen aller Ja–Instanzen
  - TM $\mathcal{M}$ löst $\Pi$ 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 $n$.
- **Jetzt:** Am Beispiel TSP: Entscheidung $\to$ Optimalwert $\to$ Optimierung

# Schwierigkeit von Entscheidungs- und Optimierungsproblem

**Satz.**

Falls es einen Algorithmus $\mathcal{A}$ gibt, der das Entscheidungsproblem des TSP in polynomialer Zeit löst, so gibt es auch einen Algorithmus, der das Optimierungsproblem in polynomialer Zeit löst.

Entscheidungsproblem TSP:

Gibt es eine Tour in $G, c$ mit Länge $\leq k$?

Algorithmus $\mathcal{A}$:

$\rightsquigarrow \mathcal{A}(G, c, k) =$ “ja”

$\rightsquigarrow \mathcal{A}(G, c, k) =$ “nein”

# Schwierigkeit von Entscheidungs- und Optimierungsproblem

Karlsruher Institut für Technologie

**Satz.**

Falls es einen Algorithmus $\mathcal{A}$ gibt, der das Entscheidungsproblem des TSP in polynomialer Zeit löst, so gibt es auch einen Algorithmus, der das Optimierungsproblem in polynomialer Zeit löst.

**Beweis:** Algorithmus, der das Optimierungsproblem löst.

Input: $G = (V, E)$, $c_{ij} = c(\{i, j\})$ für $i, j \in V := \{1, \dots, n\}$, Algorithmus $\mathcal{A}$

Output: $d_{ij}$ $(1 \leq i, j \leq n)$, so dass alle bis auf $n$ der $d_{ij}$-Werte den Wert $1 + \max\{c_{ij} \mid 1 \leq i, j \leq n\}$ haben. Die restlichen $n$ $d_{ij}$-Werte haben den Wert $c_{ij}$ und geben genau die Kanten einer optimalen Tour an.

Entscheidungsproblem TSP:

Gibt es eine Tour in $G, c$ mit Länge $\leq k$?

Algorithmus $\mathcal{A}$:

$\rightsquigarrow \mathcal{A}(G, c, k) =$ “ja”

$\rightsquigarrow \mathcal{A}(G, c, k) =$ “nein”

# Algorithmus OPT-TOUR (als Beweis) 1/2

```
berechne m := max{c_ij | 1 ≤ i, j ≤ n};
setze L(ow) := 0 und H(igh) := n · m;                // L ≤ OPT ≤ H
Solange H − L > 1 gilt, führe aus:                   // binäre Suche nach OPT
    Falls A(G, c, ⌈½(H + L)⌉) = "nein",              // OPT > ⌈½(H + L)⌉
        setze L := ⌈½(H + L)⌉ + 1;
    Sonst                                            // OPT ≤ ⌈½(H + L)⌉
        setze H := ⌈½(H + L)⌉;
Falls A(G, c, L) = "nein"                            // hier gilt H − L ≤ 1
    setze OPT := H;
Sonst
    setze OPT := L;
```

**Entscheidungsproblem TSP:**

Gibt es eine Tour in $G, c$ mit Länge $\leq k$?

**Algorithmus $\mathcal{A}$:**

$\rightsquigarrow \mathcal{A}(G, c, k) =$ "ja"

$\rightsquigarrow \mathcal{A}(G, c, k) =$ "nein"

# Algorithmus OPT-TOUR (als Beweis) 2/2

Wir kennen den Optimalwert *OPT* und finden jetzt eine optimale Tour.

**Für** $i = 1 \dots n$ führe aus
  **Für** $j = 1 \dots n$ führe aus
    setze $R := c_{ij}$; // merke Länge der Kante *ij*
    setze $c_{ij} := m + 1$; // mache Kante *ij* zu lang
    **Falls** $\mathcal{A}(G, c, OPT) =$ “nein”, // Kante *ij* in opt. Tour
      setze $c_{ij} := R$; // Kante *ij* wie vorher
    setze $d_{ij} := c_{ij}$;

Entscheidungsproblem TSP:

Gibt es eine Tour in $G, c$ mit Länge $\leq k$?

Algorithmus $\mathcal{A}$:

$\rightsquigarrow \mathcal{A}(G, c, k) =$ “ja”
$\rightsquigarrow \mathcal{A}(G, c, k) =$ “nein”

# Bemerkungen zum Algorithmus

Die Schleife der binären Suche bricht ab, und danach ist die Differenz $H - L$ gleich 1 oder 0, denn:

- Solange $H - L > 1$, ändert sich bei jedem Schleifendurchlauf einer der Werte $H, L$:
  - Für $H - L > 1$ gilt, dass $L \neq \left\lceil \frac{1}{2}(H + L) \right\rceil + 1$ und $H \neq \left\lceil \frac{1}{2}(H + L) \right\rceil$ ist.
- Die Differenz $H - L$ verkleinert sich mit jedem Durchlauf
- Da $H$ und $L$ ganzzahlig sind, tritt der Fall $H - L \leq 1$ ein.
- Zu jedem Zeitpunkt gilt $H - L \geq 0$:
  - $H - L = 0$ ist möglich, wenn zum Beispiel $L$ auf $\lceil \frac{1}{2}(H + L) \rceil + 1$ erhöht wird und vorher $H - L = 2$ oder $H - L = 3$ war.

# Laufzeit des Algorithmus

- $\mathcal{A}(G, c, k)$ wird (für verschiedene $k$) etwa $\log(n \cdot m)$–mal aufgerufen.
- $\mathcal{A}(G, c, OPT)$ wird etwa $n^2$–mal aufgerufen.
- Es finden also $\mathcal{O}(n^2 + \log(nm))$ Aufrufe von $\mathcal{A}$ statt.
- Die Inputlänge ist $\mathcal{O}(n^2 \cdot \max \langle c_{ij} \rangle) = \mathcal{O}(n^2 \cdot \max \log c_{ij})$.
- Da $\mathcal{A}$ polynomiell ist, ist dies also auch OPT-TOUR.

# Zwischenstand Komplexitätstheorie

- **Probleme**
  - Optimierungsprobleme, Optimalwertprobleme, Entscheidungsprobleme
  - Problem $\Pi$ ist Klasse $D_\Pi$ von Instanzen $I$.
- **Eingabegrößen**
  - Kodierungsschema $s\colon D_\Pi \longrightarrow \Sigma^*$ über Alphabet $\Sigma^*$.
  - Kodierung $s(I)$ einer Instanz $I$ ist ein Wort aus $\Sigma^*$.
  - Inputlänge $|s(I)|$ ist Länge des Wortes.
- **Entscheidungsprobleme**
  - Ja–Instanzen $J_\Pi$ und Nein–Instanzen $N_\Pi$
  - Sprache $L[\Pi, s]$ der Kodierungen aller Ja–Instanzen
  - TM $\mathcal{M}$ löst $\Pi$ 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 $n$.
- **Gerade:** Am Beispiel TSP: Entscheidung $\longrightarrow$ Optimalwert $\longrightarrow$ Optimierung

# Zwischenstand Komplexitätstheorie

- **Probleme**
  - Optimierungsprobleme, Optimalwertprobleme, Entscheidungsprobleme
  - Problem $\Pi$ ist Klasse $D_\Pi$ von Instanzen $I$.
- **Eingabegrößen**

**Testen Sie sich!**

  - Kodierungsschema $s\colon D_\Pi \to \Sigma^*$ über Alphabet $\Sigma^*$.
  - Kodierung $s(I)$ einer Instanz $I$ ist ein Wort aus $\Sigma^*$.
  - Inputlänge $|s(I)|$ ist Länge des Wortes.
- **Entscheidungsprobleme**
  - Ja–Instanzen $J_\Pi$ und Nein–Instanzen $N_\Pi$
  - Sprache $L[\Pi, s]$ der Kodierungen aller Ja–Instanzen
  - TM $\mathcal{M}$ löst $\Pi$ 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 $n$.
- **Gerade:** Am Beispiel TSP: Entscheidung $\longrightarrow$ Optimalwert $\longrightarrow$ 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 $\varepsilon$-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 $\varepsilon$-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**:

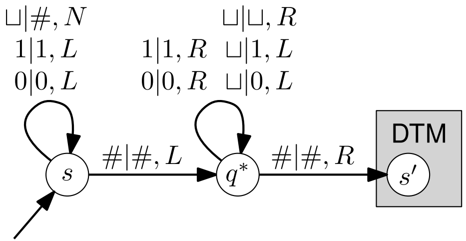
- **1. Stufe:** Es wird nichtdeterministisch vor die Eingabe auf das Band geschrieben.
- **2. Stufe:** Es wird deterministisch das gesamte Band bearbeitet.

# 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 $\delta$ zu Übergangsrelation erweitert
- ermöglicht Wahlmöglichkeiten und $\varepsilon$–Übergänge
  $\rightsquigarrow$ vergleiche endliche Automaten

Die **Orakel-Turing-Maschine**:

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

# NTM und Orakel-TM

Die “klassische” nichtdeterministische Turing-Maschine:

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

Die **Orakel-Turing-Maschine**:

- äquivalentes Modell einer nichtdeterministischen Turing-Maschine
- basiert auf nichtdeterministischem Orakel und deterministischer endlicher 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 $\Pi$

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

- **1. Stufe:** Es wird ein Orakel aus $\Gamma^*$ 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 $\Gamma^*$ 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 $\Gamma^*$.
- Endet mindestens eine in $q_J$, so wird $x$ akzeptiert.

# Zeitkomplexität für NTM

Karlsruher Institut für Technologie

- 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** $T_\mathcal{M} : \mathbb{Z}^+ \to \mathbb{Z}^+$ 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** $T_{\mathcal{M}}: \mathbb{Z}^+ \to \mathbb{Z}^+$ einer NTM $\mathcal{M}$ ist definiert durch

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

**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** $T_{\mathcal{M}} : \mathbb{Z}^+ \to \mathbb{Z}^+$ 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 $\Pi$ zu $\mathcal{NP}$, falls $\Pi$ folgende Eigenschaft hat:
  Ist die Antwort bei Eingabe eines Beispiels $I$ von $\Pi$ 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

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

($\mathcal{NP}$ 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 $\Pi$ kann die Existenz einer Lösung in polynomialer Zeit überprüft werden.

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 $\Pi$ eine Lösung, dann kann die Korrektheit einer Lösung in polynomialer Zeit überprüft werden.

Große Frage:

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

# 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 Orakelwort der Länge höchstens $p(n)$ vor die Eingabe.
  - Überprüfe für jedes Orakelwort mit endlicher Kontrolle von $\mathcal{M}$.
    - Aber: Stoppe endliche Kontrolle nach $p(n)$ Schritten.
  - Lehne Eingabe ab, wenn kein Orakelwort von der endlichen Kontrolle akzeptiert wurde.
- 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 $f\colon \Sigma_1^* \longrightarrow \Sigma_2^*$ 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 $\Pi_1$ ist polynomial transformierbar in das Entscheidungsproblem $\Pi_2$, wenn eine Funktion $f\colon D_{\Pi_1} \to D_{\Pi_2}$ existiert mit folgenden Eigenschaften:

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

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

Ein Entscheidungsproblem $\Pi$ 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 $f\colon \Sigma_1^* \longrightarrow \Sigma_2^*$ 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 $L_1$ ist $\mathcal{NP}$-vollständig, dann ist auch $L_2$ $\mathcal{NP}$-vollständig.

**Bedeutung.**

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

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

**Hindernis.**

- 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 $\Pi$ 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 $\Pi$ 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 $\Pi$ 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 $\Pi'$

**Satz von Cook.**

SAT ist $\mathcal{NP}$-vollständig.

**Beweis:** Nächste Vorlesung!

# Der Satz von Cook (Steven Cook, 1971)

- Problem $\Pi$ 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 $\Pi'$

**Satz von Cook.**

SAT ist $\mathcal{NP}$-vollständig.

**Beweis:** Nächste Vorlesung!

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