Technische
Universität
Dresden

Theoretische Informatik und Logik

8. Vorlesung: NP

Markus Krötzsch

Professur Wissensbasierte Systeme

TU Dresden, 11. Mai 2026

NP

Welche Komplexität hat **SAT**?

Wie schwer ist **SAT**?

Welche Komplexität hat **SAT**?

Wie schwer ist **SAT**?

Alle unsere Methoden für Erfüllbarkeitstests liefen in schlimmstenfalls exponentieller Zeit.

Welche Komplexität hat **SAT**?

Wie schwer ist **SAT**?

Alle unsere Methoden für Erfüllbarkeitstests liefen in schlimmstenfalls exponentieller Zeit.
$\rightsquigarrow$ **SAT** $\in$ ExpTime

Beim Durchsuchen der Wahrheitswertetabelle müssen wir stets nur eine Zeile im Speicher halten.

Welche Komplexität hat **SAT**?

Wie schwer ist **SAT**?

Alle unsere Methoden für Erfüllbarkeitstests liefen in schlimmstenfalls exponentieller Zeit.
$\rightsquigarrow$ **SAT** $\in$ ExpTime

Beim Durchsuchen der Wahrheitswertetabelle müssen wir stets nur eine Zeile im Speicher halten.
$\rightsquigarrow$ **SAT** $\in$ PSpace

Welche Komplexität hat **SAT**?

Wie schwer ist **SAT**?

Alle unsere Methoden für Erfüllbarkeitstests liefen in schlimmstenfalls exponentieller Zeit.
$\rightsquigarrow$ **SAT** $\in$ ExpTime

Beim Durchsuchen der Wahrheitswertetabelle müssen wir stets nur eine Zeile im Speicher halten.
$\rightsquigarrow$ **SAT** $\in$ PSpace

Wir könnten auch nichtdeterministisch eine Zeile raten und nur diese prüfen.

Welche Komplexität hat **SAT**?

Wie schwer ist **SAT**?

Alle unsere Methoden für Erfüllbarkeitstests liefen in schlimmstenfalls exponentieller Zeit.
$\rightsquigarrow$ **SAT** $\in$ ExpTime

Beim Durchsuchen der Wahrheitswertetabelle müssen wir stets nur eine Zeile im Speicher halten.
$\rightsquigarrow$ **SAT** $\in$ PSpace

Wir könnten auch nichtdeterministisch eine Zeile raten und nur diese prüfen.
$\rightsquigarrow$ **SAT** $\in$ NP

NP-Probleme effizient lösen?

Wir haben gesehen:

$$\textbf{SAT} \in \text{NP} \subseteq \text{PSpace} \subseteq \text{ExpTime}$$

- 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?

$\rightsquigarrow$ Dazu sollten 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 $\mathbf{L} \subseteq \Sigma^*$ ist eine polynomiell-zeitbeschränkte, deterministische TM $\mathcal{M}$, für die gilt:

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

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 nachweis-polynomiell wenn es für sie einen polynomiellen Verifikator gibt.

Nachweis-polynomielle Sprachen

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

Beispiel: SAT ist nachweis-polynomiell. Ein möglicher polynomiellen 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 nachweis-polynomiell wenn es für sie einen polynomiellen Verifikator gibt.

Beispiel: SAT ist nachweis-polynomiell. Ein möglicher polynomiellen 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: Die Entscheidung, ob ein gegebener Graph einen Hamilton-Pfad zulässt, ist nachweis-polynomiell. Als Zertifikat dient der entsprechende Pfad.

Nachweis-polynomielle Sprachen

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

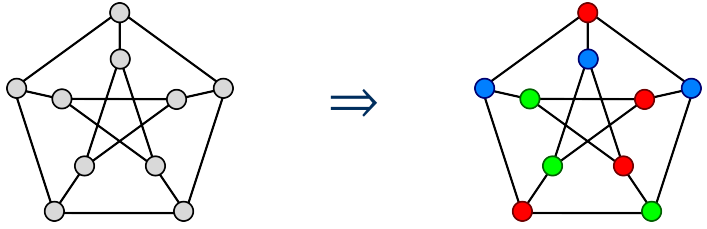
Beispiel: SAT ist nachweis-polynomiell. Ein möglicher polynomiellen Verifikator für **SAT** akzeptiert Wörter $F\#w$, wobei F die Kodierung einer erfüllbaren Formel ist und w eine Kodierung einer Wertzuweisung, die F erfüllt.

Beispiel: Die Entscheidung, ob ein gegebener Graph einen Hamilton-Pfad zulässt, ist nachweis-polynomiell. Als Zertifikat dient der entsprechende Pfad.

Beispiel: Jede Sprache $\mathbf{L} \in \mathrm{P}$ 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 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 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 $f_1 \cdot f_2 = n$.

Folgende Sprache ist nachweis-polynomiell:

$$\textbf{COMPOSITE} = \{\text{bin}(n) \mid \text{es gibt } f_1, f_2 > 1 \text{ mit } n = f_1 \cdot f_2\}$$

(Zertifikat: Kodierung von f_1 und f_2)

Beispiel: Faktorisierung

Die Faktorisierung einer natürlichen Zahl n ist die Darstellung der Zahl als Produkt von natürlichen Zahlen $f_1 \cdot f_2 = n$.

Folgende Sprache ist nachweis-polynomiell:

$$\textbf{COMPOSITE} = \{\text{bin}(n) \mid \text{es gibt } f_1, f_2 > 1 \text{ mit } n = f_1 \cdot f_2\}$$

(Zertifikat: Kodierung von f_1 und f_2)

Überraschender Weise ist auch folgende Sprache nachweis-polynomiell:

$$\textbf{PRIMES} = \{\text{bin}(n) \mid \text{es gibt keine } f_1, f_2 > 1 \text{ mit } n = f_1 \cdot f_2\}$$

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

Beispiel: Faktorisierung

Die Faktorisierung einer natürlichen Zahl n ist die Darstellung der Zahl als Produkt von natürlichen Zahlen $f_1 \cdot f_2 = n$.

Folgende Sprache ist nachweis-polynomiell:

$$\textbf{COMPOSITE} = \{\text{bin}(n) \mid \text{es gibt } f_1, f_2 > 1 \text{ mit } n = f_1 \cdot f_2\}$$

(Zertifikat: Kodierung von f_1 und f_2)

Überraschender Weise ist auch folgende Sprache nachweis-polynomiell:

$$\textbf{PRIMES} = \{\text{bin}(n) \mid \text{es gibt keine } f_1, f_2 > 1 \text{ mit } n = f_1 \cdot f_2\}$$

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

(Seit 2002 wissen wir: **PRIMES** $\in$ P $\rightsquigarrow$ Primzahlentest nach Agrawal, Kayal und Saxena)

NP steht für „nachweis-polynomiell“ (1)

Satz: Eine Sprache **L** ist genau dann nachweis-polynomiell wenn $\mathbf{L} \in \mathrm{NP}$.

NP steht für „nachweis-polynomiell“ (1)

Satz: Eine Sprache **L** ist genau dann nachweis-polynomiell wenn $\mathbf{L} \in \mathrm{NP}$.

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 $\mathbf{L} \in \mathrm{NP}$.

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 $\mathbf{L} \in \mathrm{NP}$.

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 $\mathbf{L} \in \mathrm{NP}$.

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

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

NP steht für „nachweis-polynomiell“ (2)

Satz: Eine Sprache **L** ist genau dann nachweis-polynomiell wenn $\mathbf{L} \in \mathrm{NP}$.

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

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

$\rightsquigarrow$ Die Liste der Übergänge eines akzeptierenden Laufs kann als Zertifikat dienen

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

NP ist nicht symmetrisch

Es ist leicht zu sehen:

Satz: Die Klasse P ist unter Komplement abgeschlossen.

Beweis: Wenn es für $\mathbf{L}$ eine polynomiell-zeitbeschränkte TM $\mathcal{M}$ gibt, dann erhält man eine TM für $\overline{\mathbf{L}}$ 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 $\overline{\mathbf{L}}$ 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).*

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

* Grund: für 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 $\mathcal{M}$ gibt, dann erhält man eine TM für $\overline{\mathbf{L}}$ 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).*

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

Die Klasse aller Sprachen **L**, für die $\overline{\mathbf{L}} \in$ NP gilt, heißt coNP.

Jede nichtdeterministische Klasse kann komplementiert werden: coNL, coNExp,...
Jede deterministische Klasse ist ihr eigenes Komplement (Beweis wie bei P).

* Grund: für 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.

In NP oder nicht?

Vermutung: coNP $\neq$ NP, d.h. Komplemente von „typischen“ Problemen in NP sind nicht in NP.

Aber: Es gibt viele Probleme in coNP $\cap$ NP. Zum Beispiel ist P $\subseteq$ coNP $\cap$ NP.

In NP oder nicht?

Vermutung: coNP $\neq$ NP, d.h. Komplemente von „typischen“ Problemen in NP sind nicht in NP.

Aber: Es gibt viele Probleme in coNP $\cap$ NP. Zum Beispiel ist P $\subseteq$ coNP $\cap$ NP.

Primzahl (= $\overline{\textbf{Zusammengesetzte Zahl}}$)

Gegeben: Eine natürliche Zahl $z > 1$

Frage: Gibt es eine keine natürliche Zahlen $p, q > 1$ mit $p \cdot q = z$?

In NP oder nicht?

Vermutung: coNP $\neq$ NP, d.h. Komplemente von „typischen“ Problemen in NP sind nicht in NP.

Aber: Es gibt viele Probleme in coNP $\cap$ NP. Zum Beispiel ist P $\subseteq$ coNP $\cap$ NP.

Primzahl (= $\overline{\textbf{Zusammengesetzte Zahl}}$)

Gegeben: Eine natürliche Zahl $z > 1$

Frage: Gibt es eine keine natürliche Zahlen $p, q > 1$ mit $p \cdot q = z$?

Seit 1975 ist bekannt: **Primzahl** $\in$ NP, also **Primzahl** $\in$ NP $\cap$ coNP
(Zertifikat: „Primality certificate“)

In NP oder nicht?

Vermutung: coNP ≠ NP, d.h. Komplemente von „typischen“ Problemen in NP sind nicht in NP.

Aber: Es gibt viele Probleme in coNP ∩ NP. Zum Beispiel ist P ⊆ coNP ∩ NP.

Primzahl (= $\overline{\textbf{Zusammengesetzte Zahl}}$)

Gegeben: Eine natürliche Zahl $z > 1$

Frage: Gibt es eine keine natürliche Zahlen $p, q > 1$ mit $p \cdot q = z$?

Seit 1975 ist bekannt: **Primzahl** ∈ NP, also **Primzahl** ∈ NP ∩ coNP
(Zertifikat: „Primality certificate“)

Seit 2002 ist bekannt: **Primzahl** ∈ P
(Primzahlentest nach Agrawal, Kayal und Saxena)

Randbemerkung: Ist Kryptografie sicher?

Das Wirkprinzip asymmetrischer Verschlüsselungsverfahren:

- Es ist leicht, zwei Zahlen zu multiplizieren
- Es ist schwer, eine Zahl in ihre Faktoren zu zerlegen

Aber seit 2002 wissen wir: Man kann in polynomieller Zeit entscheiden, ob es Faktoren mit $p \cdot q = z$ gibt.

Haben Agrawal, Kayal und Saxena die Kryptografie geknackt?

Randbemerkung: Ist Kryptografie sicher?

Das Wirkprinzip asymmetrischer Verschlüsselungsverfahren:

- Es ist leicht, zwei Zahlen zu multiplizieren
- Es ist schwer, eine Zahl in ihre Faktoren zu zerlegen

Aber seit 2002 wissen wir: Man kann in polynomieller Zeit entscheiden, ob es Faktoren mit $p \cdot q = z$ gibt.

Haben Agrawal, Kayal und Saxena die Kryptografie geknackt?

Nein:

- Es ist leicht zu entscheiden, ob eine Zahl echte Faktoren hat
- Aber es ist dennoch schwer die Faktoren zu bestimmen
  (glauben wir zumindest . . .)

Faktorisierung

(Nicht)Existenz von Faktoren (**Primzahl**) erfasst nicht wirklich, wie schwer Faktorisierung ist

$\rightsquigarrow$ Wie kann man das komplexitätstheoretisch ausdrücken?

Faktorisierung

(Nicht)Existenz von Faktoren (**Primzahl**) erfasst nicht wirklich, wie schwer Faktorisierung ist

$\rightsquigarrow$ Wie kann man das komplexitätstheoretisch ausdrücken?

Faktor-7

Gegeben: Eine natürliche Zahl $z > 1$

Frage: Hat z einen Primfaktor, der mit der Ziffer 7 endet?

Faktorisierung

(Nicht)Existenz von Faktoren (**Primzahl**) erfasst nicht wirklich, wie schwer Faktorisierung ist

$\rightsquigarrow$ Wie kann man das komplexitätstheoretisch ausdrücken?

Faktor-7

Gegeben: Eine natürliche Zahl $z > 1$

Frage: Hat z einen Primfaktor, der mit der Ziffer 7 endet?

Ein interessantes Entscheidungsproblem:

- **Faktor-7** erfordert (vermutlich) die Kenntnis der Primfaktoren, nicht nur die Bestimmung deren Existenz

Faktorisierung

(Nicht)Existenz von Faktoren (**Primzahl**) erfasst nicht wirklich, wie schwer Faktorisierung ist

$\rightsquigarrow$ Wie kann man das komplexitätstheoretisch ausdrücken?

Faktor-7

Gegeben: Eine natürliche Zahl $z > 1$

Frage: Hat z einen Primfaktor, der mit der Ziffer 7 endet?

Ein interessantes Entscheidungsproblem:

- **Faktor-7** erfordert (vermutlich) die Kenntnis der Primfaktoren, nicht nur die Bestimmung deren Existenz
- **Faktor-7** $\in$ NP

Faktorisierung

(Nicht)Existenz von Faktoren (**Primzahl**) erfasst nicht wirklich, wie schwer Faktorisierung ist

$\rightsquigarrow$ Wie kann man das komplexitätstheoretisch ausdrücken?

Faktor-7

Gegeben: Eine natürliche Zahl $z > 1$

Frage: Hat z einen Primfaktor, der mit der Ziffer 7 endet?

Ein interessantes Entscheidungsproblem:

- **Faktor-7** erfordert (vermutlich) die Kenntnis der Primfaktoren, nicht nur die Bestimmung deren Existenz
- **Faktor-7** $\in$ NP: Zertifikat ist die Liste aller Primfaktoren*

* Kontrollfrage: Wieso kann man polynomiell verifizieren, dass dies wirklich Primfaktoren sind?

Faktorisierung

(Nicht)Existenz von Faktoren (**Primzahl**) erfasst nicht wirklich, wie schwer Faktorisierung ist

$\rightsquigarrow$ Wie kann man das komplexitätstheoretisch ausdrücken?

Faktor-7

Gegeben: Eine natürliche Zahl $z > 1$

Frage: Hat z einen Primfaktor, der mit der Ziffer 7 endet?

Ein interessantes Entscheidungsproblem:

- **Faktor-7** erfordert (vermutlich) die Kenntnis der Primfaktoren, nicht nur die Bestimmung deren Existenz
- **Faktor-7** $\in$ NP: Zertifikat ist die Liste aller Primfaktoren*
- **Faktor-7** $\in$ coNP

* Kontrollfrage: Wieso kann man polynomiell verifizieren, dass dies wirklich Primfaktoren sind?

Faktorisierung

(Nicht)Existenz von Faktoren (**Primzahl**) erfasst nicht wirklich, wie schwer Faktorisierung ist

$\rightsquigarrow$ Wie kann man das komplexitätstheoretisch ausdrücken?

Faktor-7

Gegeben: Eine natürliche Zahl $z > 1$

Frage: Hat z einen Primfaktor, der mit der Ziffer 7 endet?

Ein interessantes Entscheidungsproblem:

- **Faktor-7** erfordert (vermutlich) die Kenntnis der Primfaktoren, nicht nur die Bestimmung deren Existenz
- **Faktor-7** $\in$ NP: Zertifikat ist die Liste aller Primfaktoren*
- **Faktor-7** $\in$ coNP: Zertifikat ist die Liste aller Primfaktoren*

* Kontrollfrage: Wieso kann man polynomiell verifizieren, dass dies wirklich Primfaktoren sind?

Faktorisierung

(Nicht)Existenz von Faktoren (**Primzahl**) erfasst nicht wirklich, wie schwer Faktorisierung ist

$\rightsquigarrow$ Wie kann man das komplexitätstheoretisch ausdrücken?

Faktor-7

Gegeben: Eine natürliche Zahl $z > 1$

Frage: Hat z einen Primfaktor, der mit der Ziffer 7 endet?

Ein interessantes Entscheidungsproblem:

- **Faktor-7** erfordert (vermutlich) die Kenntnis der Primfaktoren, nicht nur die Bestimmung deren Existenz
- **Faktor-7** $\in$ NP: Zertifikat ist die Liste aller Primfaktoren*
- **Faktor-7** $\in$ coNP: Zertifikat ist die Liste aller Primfaktoren*
- Aber niemand konnte bisher zeigen, dass **Faktor-7** $\in$ P
  vermutlich gilt **Faktor-7** $\notin$ P

* Kontrollfrage: Wieso kann man polynomiell verifizieren, dass dies wirklich Primfaktoren sind?

Zusammenfassung und Ausblick

NP entspricht der Klasse der nachweis-polynomiellen Probleme

Die Klasse der Komplemente von NP-Problemen ist coNP

Polynomielle Reduktionen erlauben uns, die Schwere von Problemen zu vergleichen

Was erwartet uns als nächstes?

- Mehr NP
- Pseudopolynomielle Probleme
- Komplexität jenseits von NP