# Grundbegriffe der Theoretischen Informatik

Sommersemester 2018 - Thomas Schwentick

## Teil D: Komplexitätstheorie

## 20: **NP**: weitere Erkenntnisse

Version von: 17. Juli 2018 (12:10)

# Inhalt

# Weitere NP-vollständige Probleme (1/2)

- Es sind buchstäblich Tausende von **NP**-vollständigen Problemen bekannt
- Eine schöne Auswahl findet sich in:
  - Garey, Johnsen: Computers and Intractability: A Guide to the Theory of **NP**-Completeness, 1979

- Wir betrachten im Folgenden kurz einige weitere **NP**-vollständige Probleme

**Definition** (VERTEXCOVER)

**Gegeben:** Ungerichteter Graph $G = (V, E)$, Zahl $k$

**Frage:** Gibt es $S \subseteq V$ mit $|S| \leqslant k$, so dass jede Kante einen Knoten in $S$ hat?

**Definition** (SHORTESTSUPERSTRING)

**Gegeben:** Strings $s_1, \dots, s_m$, Zahl $k$

**Frage:** Gibt es einen String der Länge $k$, der alle Strings $s_i$ als Teilstrings enthält?

**Definition** (BINPACKING)

**Gegeben:** Gegenstände mit Größen $s_1, \dots, s_n \in \mathbb{Q}$ zwischen $0$ und $1$, Zahl $k$

**Frage:** Lassen sich die Gegenstände in $k$ Behälter mit jeweiliger Kapazität $1$ füllen?

**Definition** (INTEGERPROGRAMMINGO)

**Gegeben:** Lineares Ungleichungssystem, lineare Zielfunktion

**Frage:** Ganzzahlige Lösung des Systems mit maximalem Wert der Zielfunktion

**Definition** (INTEGERPROGRAMMING)

**Gegeben:** Lineares Ungleichungssystem

**Frage:** Gibt es eine ganzzahlige Lösung des Systems?

# Weitere NP-vollständige Probleme (2/2)

**Definition** (BUNDESLIGA)

**Gegeben:** Tabellenstand eines Turnieres, Menge restlicher Spiele, Lieblingsverein $x$

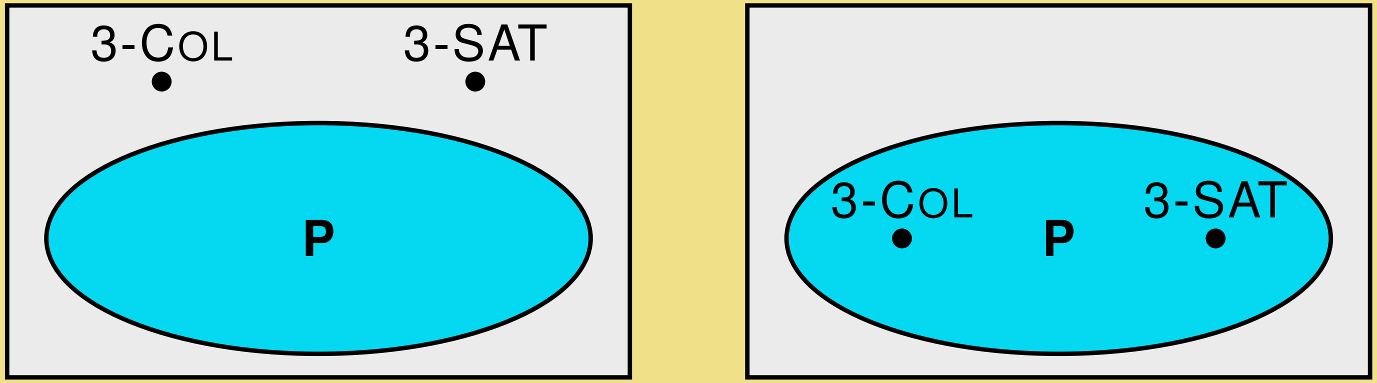
**Frage:** Kann $x$ noch Meister werden?

- [Bernholt, Gülich, Hofmeister, Schmitt 99] zeigen:
  - in **P** mit 2-Punkte-Regel
  - **NP**-vollständig mit 3-Punkte-Regel

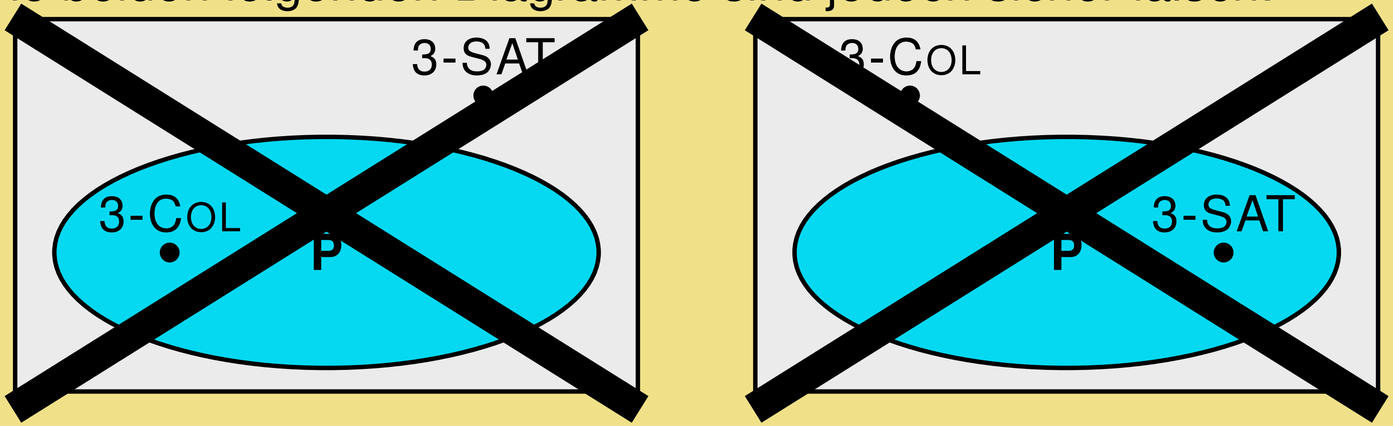
# Alles oder nichts...

- Aus Satz 18.6 folgt für 3-SAT und 3-COL:
  - Entweder sind beide in polynomieller Zeit lösbar oder
  - beide sind **nicht** in polynomieller Zeit lösbar

- Eines der beiden folgenden Diagramme ist also gültig:




- Die beiden folgenden Diagramme sind jedoch sicher falsch:




- Entsprechende Aussagen gelten für beliebige andere Paare **NP**-vollständiger Probleme

# NP-Vollständigkeit: weitere Anmerkungen

- **NP**-vollständige Probleme sind in einem gewissen Sinne sehr ähnlich:
  - Sind $L_1$ und $L_2$ **NP**-vollständig so gilt $L_1 \leqslant_p L_2$ und $L_2 \leqslant_p L_1$

- Aber **NP**-vollständige Probleme haben im Detail durchaus unterschiedliche Eigenschaften

- Manche lassen sich gut approximieren, andere nicht

- Für KNAPSACKO gibt es einen Algorithmus, der bei Eingabe $(I, \epsilon)$ in polynomieller Zeit in $|I|$ und $1/\epsilon$ eine Lösung berechnet, die mindestens einen $(1-\epsilon)$-Anteil des optimalen Wertes liefert

- TSPO lässt sich im Gegensatz dazu im Allgemeinen gar nicht approximieren

  ☞ falls **P** $\neq$ **NP**

- Manche **NP**-vollständige Probleme lassen sich effizient lösen, falls nur ein gewisser Parameter des Problems nicht zu groß wird:
  - VERTEXCOVER lässt sich beispielsweise leicht in Zeit $\mathcal{O}(2^k n)$ lösen

- Andere **NP**-vollständige Probleme haben durch ausgefeilte Algorithmen für viele in der Praxis auftretende Eingaben ihren Schrecken verloren:
  - Inzwischen lässt sich für viele große „praktisch relevante“ KNF-Formeln die Erfüllbarkeit effizient testen
  - In manchen Anwendungen werden sogar polynomiell lösbare (aber praktisch ungünstige) Probleme durch Reduktion auf SAT und Anwendung eines effizienten SAT-Solvers gelöst

- Mehr darüber gibt es in der Vorlesung *Komplexitätstheorie* zu erfahren
  - (fast) immer im Sommersemester

  ☞ wohl nicht in 2019

# Inhalt

# Untere Schranken für NP-schwierige Probleme

- Wir kennen viele **NP**-schwierige Probleme
- Wir vermuten, dass sie alle nicht in polynomieller Zeit lösbar sind
- Eine bestimmte (vermutete) untere Schranke für die Laufzeit von Lösungsalgorithmen impliziert die **NP**-Schwierigkeit aber nicht
- Lässt sich über untere Schranken für **NP**-schwierige Probleme etwas Genaueres sagen?

- Ohne Annahme von **P** $\neq$ **NP** gibt es bisher keine besseren unteren Schranken als $\Omega(n \log n)$
- Aus **P** $\neq$ **NP** ließe sich die untere Schranke $n^{\omega(1)}$ schließen
- Gibt es präzisere Vermutungen?

- Zur Vereinfachung der Notation schreiben wir $f(n) = \hat{O}(g(n))$ für zwei Funktionen $f, g$, falls es eine Konstante $c$ gibt, so dass für alle $n \geqslant 1$ gilt: $f(n) \leqslant g(n)n^c$

# Obere Schranken für SAT

**Definition ($k$-SAT)**

**Gegeben:** AL-Formel $\varphi$ in konjunktiver Normalform mit $n$ Variablen, $m$ Klauseln, und $\leqslant k$ Literalen je Klausel

**Frage:** Gibt es eine erfüllende Belegung für $\varphi$?

- Wie effizient lässt sich $k$-SAT lösen?

- Brute Force:
  - Probiere alle Belegungen aus
  - Aufwand: $\hat{\mathcal{O}}(2^n)$

- Beste Algorithmen für 3-SAT haben ungefähre Laufzeit $1,3^n$ ☞ [Makino, Tamaki, Yamamoto 2013]

- Beste Algorithmen für $k$-SAT haben Laufzeit $\hat{\mathcal{O}}(2^{(1-c_k)n})$, mit $c_k = \theta(\frac{1}{k})$ ☞ [Paturi,Pudlak,Saks,Zane 1998]

- Vermutung: viel besser geht es nicht

# ETH & SETH

- Sei für jedes $k > 0$, $s_k$ das Infimum aller $\delta$, für die es einen Lösungsalgorithmus für $k$-SAT mit Zeitschranke $\hat{\mathcal{O}}(2^{\delta n})$ gibt

- **Exponential Time Hypothesis (ETH)**:
  - $s_3 > 0$

- **Strong Exponential Time Hypothesis (SETH)**:
  - $\lim\limits_{k\to\infty} s_k = 1$

- Klar: wenn ETH gilt, ist **P** $\neq$ **NP**
- Ob auch die Umkehrung gilt, ist ungewiss

- ETH und SETH sind der Versuch die Schwierigkeit von SAT in konkrete untere Schranken zu fassen

- ETH wird von vielen Forschern für plausibel gehalten
- Prinzipiell könnte aber **P** $\neq$ **NP** gelten und ETH trotzdem falsch sein, falls es beispielsweise einen 3-SAT-Algorithmus mit Laufzeit $n^{\log n}$ geben sollte

# SETH und effizient lösbare Probleme

- Die Annahme von ETH oder SETH impliziert andere untere Schranken für konkrete **NP**-vollständige Probleme

- Erstaunlicherweise folgt aus SETH auch, dass sich die besten Algorithmen für einige *effizient lösbare* Probleme nicht wesentlich verbessern lassen

**Definition (RegExWordProblem)**

**Gegeben:** Gegeben RE $\alpha$ und Wort $w$

**Frage:** Ist $w \in L(\alpha)$ ?

- Die besten Algorithmen für RegExWordProblem haben die Laufzeit $\mathcal{O}(|\alpha||w|)$
- Es ist unbekannt, ob es deutlich bessere Algorithmen gibt

**Satz 20.1 [Williams 05]**

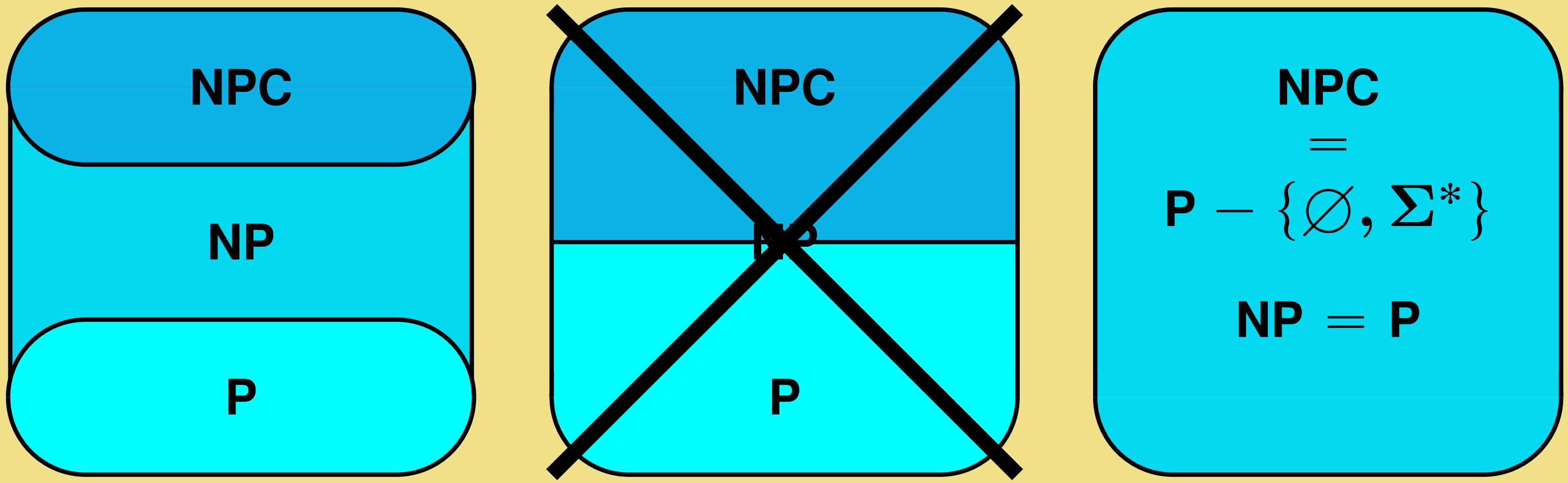
- Falls SETH wahr ist, kann RegExWordProblem für kein $\epsilon > 0$ und $c$ in Zeit $\mathcal{O}((|\alpha||w|)^{1-\epsilon}(\log n)^c)$ gelöst werden

# Inhalt

# Die innere Struktur von NP (1/2)

- Empirische Erfahrung: für die meisten Probleme $L$ in **NP** lässt sich eine der beiden folgenden Aussagen zeigen:
  (1) $L \in$ **P**
  (2) $L$ ist vollständig für **NP** ☞ in **NPC**

- Lässt sich das vielleicht allgemein beweisen?
- Anders gefragt: welche der folgenden Illustrationen der Struktur von **NP** könnten der Realität entsprechen?




- Es gilt: **NP** ist **nicht** die disjunkte Vereinigung von **P** und **NPC**
- Die Frage, welche der beiden verbleibenden Möglichkeiten die Richtige ist, ist gerade die Frage, ob **P** $=$ **NP** gilt

# Die innere Struktur von NP (2/2)

**Weitere Bemerkungen**

- Die Relation $\leqslant_p$ induziert eine Äquivalenzrelation $\equiv_p$ durch:
$$L \equiv_p L' \overset{\text{def}}{\Leftrightarrow} L \leqslant_p L' \text{ und } L' \leqslant_p L$$

- Die Sprachen aus **P** (außer $\varnothing$ und $\Sigma^*$) bilden die „einfachste" Äquivalenzklasse von **NP**
- Die **NP**-vollständigen Sprachen bilden die „schwierigste" Äquivalenzklasse innerhalb von **NP**

- Es gilt: falls **P** $\neq$ **NP** gibt es noch unendlich viele weitere $\equiv_p$-Klassen

- Das bekannteste Problem in **NP**, von dem vermutet wird, dass es weder in **P** noch **NP**-schwierig ist, ist das Graph-Isomorphismus-Problem
- Es ist für zwei Graphen mit $n$ Knoten in Zeit $n^{\mathcal{O}(\log n)}$ lösbar

# Inhalt

# Ein effizienter (?) Algorithmus für KNAPSACKO

- KNAPSACKO lässt sich mit *dynamischer Programmierung* lösen

- Gegeben: $w_1, \ldots, w_m, g_1, \ldots, g_m, G$
  - OBdA: $w_1 \geqslant w_i$ für alle $i \in \{1, \ldots, m\}$

  ➡ der optimale Wert ist $\leqslant mw_1$

- Es bezeichne $\underline{A(i,w)}$ das minimale Gewicht einer Menge $I \subseteq \{1, \ldots, i\}$ von Gegenständen mit Gesamtwert $w$

- **Ansatz:** Berechne induktiv, für jedes $i \leqslant m$ und jedes $w \leqslant w_1$, den Wert $A(i,w)$
- Der optimale Wert ist dann das maximale $w$, das $A(m,w) \leqslant G$ erfüllt

- **Entscheidende Beobachtung**: $A(i,w)$ ist das Minimum der beiden folgenden Werte:
  - $A(i-1,w)$

    ✎ Das entspricht dem Weglassen von Gegenstand $i$
  - $A(i-1,w-w_i)+g_i$

    ✎ Das entspricht dem Hinzunehmen von Gegenstand $i$

➡ Die Werte $A(i,w)$ lassen sich leicht berechnen, wenn die Werte $A(i-1,j)$ schon für alle $j$ berechnet sind

- Initialisierung:
  - $A(0,0) = 0$
  - $A(0,w) = \infty$, für alle $w > 0$

- Dynamische Programmierung:

  $m^2w_1$ Tabelleneinträge berechnen
- Aufwand: $\mathcal{O}(m^2w_1)$

# Pseudo-polynomiell vs. stark NP-vollständig (1/2)

- Der Algorithmus für KNAPSACKO hat also Laufzeit $\mathcal{O}(m^2 w_1)$
- Ist das ein Polynomialzeit-Algorithmus? **Nein!**

- Ein Polynomialzeit-Algorithmus hat eine Laufzeitschranke der Form $n^k$ für Eingaben der Länge $n$

- Länge einer (binär kodierten) KNAPSACKO-Eingabe:
$$\log(G) + \sum_{i=1}^{m} (\log w_i + \log g_i)$$
✎ Abweichungen durch etwas andere Kodierungen sind hier irrelevant

- Die Laufzeitschranke $\mathcal{O}(m^2 w_1)$ ist polynomiell in $m$ und dem Wert von $w_1$

- Die Laufzeitschranke des KNAPSACKO-Algorithmus ist aber nicht polynomiell in der *Länge der Kodierung* von $w_1$:
  - Die Laufzeitschranke lässt sich schreiben als $\mathcal{O}(m^2 2^{\log w_1})$
  - Sie ist also exponentiell in der Länge der Eingabe

- Einen Algorithmus, der polynomiell in den *Größen* (statt: der Länge der Kodierung) der vorkommenden Zahlen ist, nennen wir *pseudo-polynomiell*

# Pseudo-polynomiell vs. stark NP-vollständig (2/2)

- Umgekehrt heißt ein Problem, das **NP**-vollständig bleibt, selbst wenn die Größe der vorkommenden Zahlen durch ein Polynom in der Länge der Eingabe beschränkt sind, *stark* **NP**-*vollständig*

- Beispiel: TSP ist stark **NP**-vollständig:
  - es ist auch schon für Eingaben **NP**-vollständig, bei denen alle Abstände den Wert $1$ oder $2$ haben und $k = |V|$

- Gäbe es für ein stark **NP**-vollständiges Problem einen pseudo-polynomiellen Algorithmus, wäre **P** $=$ **NP**

✎ Es ist also kein Wunder, dass in der Reduktion 3-SAT $\leqslant_p$ KNAPSACK die verwendeten Zahlen so groß sind

✎ Zu beachten: bei der genauen Formalisierung der Begriffe „pseudo-polynomiell" und „stark **NP**-vollständig" ist etwas Vorsicht geboten:
  - Es müsste sauber definiert werden, was „vorkommende Zahlen" bedeutet...

# Inhalt

# Die Welt der Komplexitätstheorie

- Wir haben in Teil D drei Komplexitätsklassen betrachtet: **P**, **NP** und **EXPTIME**
- Alle algorithmischen Probleme, die wir in Teil D betrachtet haben, sind in **EXPTIME**, die meisten davon in **NP**
- Es stellen sich offensichtliche Fragen:
- Sind alle entscheidbaren Probleme in **EXPTIME**?
- Sind alle „natürlichen“ entscheidbaren Probleme in **EXPTIME**?
- Gibt es außer den drei Klassen noch andere relevante Komplexitätsklassen?
- Haben andere Klassen auch vollständige Probleme?

# Rechenzeit vs. Speicherplatz

- Für manche Berechnungen ist Speicherplatz eine kritischere Ressource als Rechenzeit
- Die algorithmische Schwierigkeit mancher algorithmischen Probleme lässt sich besser durch ihren Speicherplatzverbrauch als durch ihre Rechenzeit charakterisieren

**Definition (KorrekterAutomat)**

**Gegeben:** Endlicher Automat $\mathcal{A}$, regulärer Ausdruck $\alpha$

**Frage:** Erfüllt der Automat $\mathcal{A}$ die Spezifikation $\alpha$?

- Dieses Problem ist vermutlich nicht in **NP**
- Es ist aber vollständig für die Komplexitätsklasse **PSPACE**, die alle Probleme enthält, die für Eingaben der Größe $n$ mit Speicherplatz $p(n)$, für ein Polynom $p$, gelöst werden können

- Es gilt: **NP** $\subseteq$ **PSPACE** $\subseteq$ **EXPTIME**

# Polynomielle Hierarchie (1/2)

- Zur Erinnerung:
  - Eine Sprache $L$ ist genau dann semientscheidbar, wenn es eine entscheidbare Sprache $L'$ gibt, so dass für alle Strings $w \in \Sigma^*$ gilt:
    $$w \in L \iff \exists x_1 : (w, x_1) \in L',$$
    wobei der Existenzquantor über alle $\Sigma^*$ -Strings quantifiziert

**Satz 20.2**

- Eine Sprache $L$ ist genau dann in **NP**, wenn es eine Sprache $L' \in$ **P** und ein Polynom $p$ gibt, so dass für alle Strings $w \in \Sigma^*$ gilt:
  $$w \in L \iff \exists^p x_1 : (w, x_1) \in L',$$
  wobei der modifizierte Existenzquantor $\exists^p$ über alle $\Sigma^*$-Strings der Länge $p(|w|)$ quantifiziert

- Analog zur arithmetischen Hierarchie gibt es auch hier eine Hierarchie von vermutlich nicht in polynomieller Zeit entscheidbaren Problemen

# Polynomielle Hierarchie (2/2)

- Die **polynomielle Hierarchie** besteht aus allen Sprachen $L$, für die es ein $k \in \mathbb{N}$, eine Sprache $L' \in$ **P** und ein Polynom $p$ gibt, so dass gilt:
  $w \in L \iff$
  $\exists^p x_1 \forall^p x_2 \exists^p x_3 \cdots \forall^p x_k :$
  $(w, x_1, x_2, \ldots, x_k) \in L'$,
  wobei alle Quantoren über die $\Sigma^*$-Strings der Länge $p(|w|)$ quantifizieren

- Die Klasse der Probleme, die sich für ein bestimmtes $k$ auf diese Weise charakterisieren lassen, wird $\Sigma_k^p$ genannt
  - Analog: $\Pi_k^p$, wenn der erste Quantor in der Charakterisierung ein Allquantor ist

- $\Sigma_1^p =$ **NP**, $\Pi_1^p =$ **co-NP**

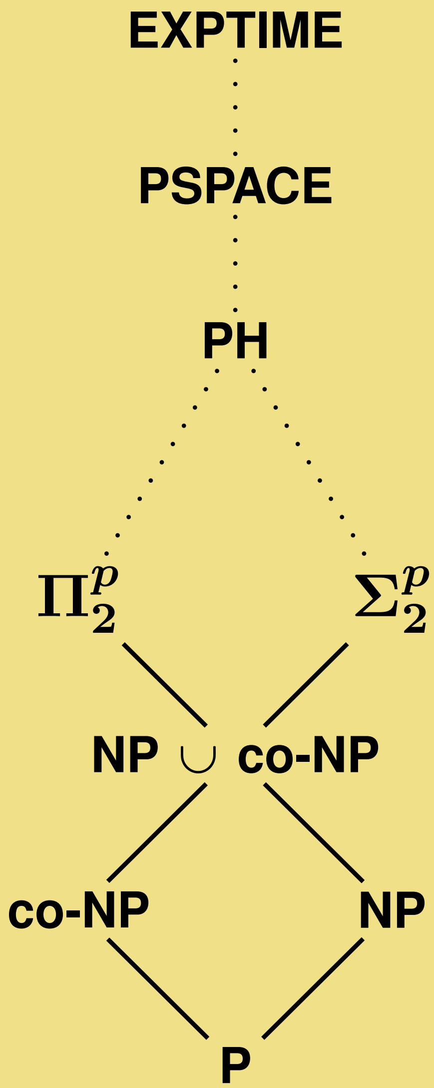
**Definition ( UCQ-CONT)**

**Gegeben:** Select-From-Where-Anfragen $q_1, q_2$ (mit UNION)

**Frage:** Liefert $q_1$ für alle passenden relationalen Datenbanken ein Teilergebnis von $q_2$?

- UCQ-CONT ist vollständig für $\Pi_2^p$

# Übersicht



# Weit jenseits des Effizienten

- Ein **regulärer Ausdruck mit Negation** darf neben den üblichen Operatoren auch einen Negationsoperator $\neg$ verwenden
- Die Semantik-Definition wird dann erweitert durch:
  – $L(\neg\alpha) \stackrel{\text{def}}{=} \Sigma^* - L(\alpha)$

- Zum Beispiel bezeichnet $\neg((a+b)^*ab(a+b)^*)$ die Menge aller Strings, in denen *nicht* $ab$ als Teilstring vorkommt

**Definition (RENEGEMPTY)**

**Gegeben:** Regulärer Ausdruck $\alpha$ mit Negation

**Frage:** Ist $L(\alpha) = \varnothing$?

- Mit den Methoden aus Teil A der Vorlesung ist es nicht schwer zu zeigen:
  – RENEGEMPTY ist entscheidbar

- Eine Funktion $T : \mathbb{N} \to \mathbb{R}$ ist **(1-fach) exponentiell beschränkt**, wenn $T(n) = 2^{\mathcal{O}(p(n))}$ für ein Polynom $p$
- Eine Funktion ist **$k$-fach exponentiell beschränkt**, wenn $T(n) = 2^{\mathcal{O}(g)}$, für eine $(k-1)$-fach exponentiell beschränkte Funktion

- Eine Funktion heißt **elementar**, wenn sie, für irgendein $k \in \mathbb{N}$, $k$-fach exponentiell beschränkt ist

**Satz [Stockmeyer, Meyer 73]**

- Es gibt keinen Algorithmus für RENEGEMPTY mit einer elementaren Laufzeitschranke

# Zusammenfassung

- Die **NP**-vollständigen Probleme sind die schwierigsten Probleme in **NP**:
  - Jedes **NP**-Problem lässt sich polynomiell auf jedes **NP**-vollständige Problem reduzieren
- Es gibt Tausende von **NP**-vollständigen Problemen
- Bessere Algorithmen für die Auswertung regulärer Ausdrücke könnten bessere Algorithmen für SAT nach sich ziehen
- Pseudo-polynomielle Algorithmen lösen Probleme in polynomieller Zeit in Abhängigkeit von der Größe der vorkommenden Zahlen
- Für stark **NP**-vollständige Probleme lässt sich die **NP**-Vollständigkeit ohne die Verwendung „großer Zahlen“ zeigen
- Es gibt viele algorithmische Probleme, die für andere Klassen als **P** und **NP** vollständig sind

# Literaturhinweise

- Angegebene Bücher


- Thorsten Bernholt, Alexander Gülich, Thomas Hofmeister, and Niels Schmitt. Football elimination is hard to decide under the 3-point-rule. In *MFCS*, pages 410–418, 1999
- Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time: Preliminary report. In *Proceedings of the 5th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1973, Austin, Texas, USA*, pages 1–9, 1973
- Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005