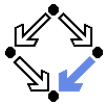
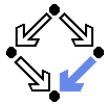
Berechenbarkeit und Komplexität
Problemkomplexität

Wolfgang Schreiner
Wolfgang.Schreiner@risc.uni-linz.ac.at

Research Institute for Symbolic Computation (RISC)
Johannes Kepler University, Linz, Austria
http://www.risc.uni-linz.ac.at



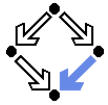
Problemkomplexität



Wir betrachten entscheidbare Probleme (rekursive Sprachen) und fragen nach der möglichen Effizienz der Entscheidung (Erkennung).

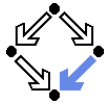
- Die rekursive Sprache L hat Zeitkomplexität $T(n)$ bzw. Raumkomplexität $S(n)$:
  - Es gibt eine Turing-Maschine mit Zeitkomplexität $T(n)$ bzw. Raumkomplexität $S(n)$, die L erkennt und immer terminiert.
- Die rekursive Sprache L hat nichtdeterministische Zeitkomplexität $T(n)$ bzw. nichtdeterministische Raumkomplexität $S(n)$:
  - Es gibt eine nichtdeterm. Turing-Maschine mit Zeitkomplexität $T(n)$ bzw. Raumkomplexität $S(n)$, die L erkennt und immer terminiert.
- $\mathrm{DTIME}(T(n))$ und $\mathrm{DSPACE}(S(n))$:
  - Die Familien der Sprachen mit Zeitkomplexität $T(n)$ und Raumkomplexität $S(n)$.
- $\mathrm{NTIME}(T(n))$ und $\mathrm{NSPACE}(S(n))$:
  - Die Familien der Sprachen mit nichtdeterministischer Zeitkomplexität $T(n)$ und nichtdeterministischer Raumkomplexität $S(n)$.

Beispiel



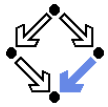
- Sprache $L := \{wcw^R \mid w \in \{0,1\}^*\}$
  - w^R ... die Umkehrung des Wortes w.
- L hat Zeitkomplexität $\mathcal{O}(n)$ und Raumkomplexität $\mathcal{O}(n)$.
  - Turing-Maschine M_1 mit zwei Bändern.
  - Kopiert zunächst den Teil der Eingabe links von c auf zweites Band.
  - Bewegt danach synchron den L/S-Kopf des ersten Bandes nach rechts und den des zweiten Bandes nach links.
  - Die gelesenen Symbole werden verglichen; sind sie gleich und ist ihre Anzahl gleich, akzeptiert M das Wort.
  - Für Eingabe der Länge n werden $n+1$ Schritte benötigt und es wird die maximale Entfernung n vom linken Ende eines Bandes erreicht.

Komplexitätsklassen



- $\mathcal{P}$ ist die Klasse aller Sprachen, die in deterministischer polynomialer Zeit erkannt werden:
  $\mathcal{P} = \bigcup_{i \in \mathbb{N}} \mathrm{DTIME}(n^i)$
- $\mathcal{NP}$ ist die Klasse aller Sprachen, die in nichtdeterministischer polynomialer Zeit erkannt werden:
  $\mathcal{NP} = \bigcup_{i \in \mathbb{N}} \mathrm{NTIME}(n^i)$
- PSPACE ist die Klasse aller Sprachen, die mit deterministischem polynomialem Raum erkannt werden:
  $\mathrm{PSPACE} = \bigcup_{i \in \mathbb{N}} \mathrm{DSPACE}(n^i)$
- NSPACE ist die Klasse aller Sprachen, die mit nichtdeterministischem polynomialem Raum erkannt werden:
  $\mathrm{NSPACE} = \bigcup_{i \in \mathbb{N}} \mathrm{NSPACE}(n^i)$

Beispiel



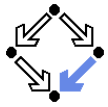
- Das Problem der Hamiltonschen Zyklen:

  Gibt es im Graphen G einen Zyklus, der jede Ecke genau einmal enthält?

- Das Problem ist in $\mathcal{NP}$, weil es einen einfachen nichtdeterministischen Algorithmus zur Lösung gibt:
  - Wähle eine Permutation der Ecken des Graphen.
  - Überprüfe, ob die Permutation einen Hamiltonschen Zyklus bildet.
- Die Zeitkomplexität des nichtdeterministischen Algorithmus ist die Zeitkomplexität der Überprüfung der "richtigen" Permutation.
  - Alle Permutationen werden quasi "gleichzeitig" untersucht.
  - Für einen Graphen mit n Knoten hat das Überprüfen jeder Permutation Zeitkomplexität $\mathcal{O}(n)$.

Es ist nicht bekannt, ob das Problem auch in $\mathcal{P}$ ist (wahrscheinlich nicht).

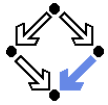
$\mathcal{P}$ und $\mathcal{NP}$



- Ein Problem in $\mathcal{NP}$:
  - Algorithmus kann zur Lösung des Problems das Ergebnis "erraten" und einen Algorithmus mit polynomialer Zeitkomplexität zur Überprüfung des Ergebnisses verwenden.
- Ein Problem in $\mathcal{P}$:
  - Algorithmus muss das Ergebnis in polynomialer Zeit konstruieren.
  - Nur Probleme in $\mathcal{P}$ sind "praktisch" lösbar.
- Offensichtlich gilt $\mathcal{P} \subseteq \mathcal{NP}$.
  - Jeder deterministische Algorithmus ist der Spezialfall eines nichtdeterministischen Algorithmus.
- Gilt $\mathcal{P} = \mathcal{NP}$?
  - Höchstwahrscheinlich nicht, das Gegenteil wäre eine Sensation.
  - Aber tatsächlich ist die Antwort unbekannt.

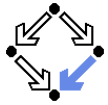
Die Frage $\mathcal{P} = \mathcal{NP}$? ist das wohl größte ungelöste theoretische Problem der Informatik.

Wissen über die Komplexitätsklassen



- Satz: $\mathcal{NP} \subseteq$ NSPACE
  - Jedes Problem, das in nicht-deterministisch polynomialer Zeit lösbar ist, ist mit nicht-deterministisch polynomialem Speicher lösbar.
    - Für jede Bewegung des L/S-Kopfes benötigt die Turing-Maschine eine Zeiteinheit; sie kann daher nicht mehr Speicher als Zeit benötigen.
- Satz: NSPACE = PSPACE
  - Jedes Problem, das mit nicht-deterministisch polynomialem Speicher lösbar ist, ist auch mit deterministisch polynomialem Speicher lösbar.
    - Beweis: Savitch, 1970.
- Also gilt: $\mathcal{P} \subseteq \mathcal{NP} \subseteq$ PSPACE = NSPACE
  - Es ist unbekannt, ob $\mathcal{P} = \mathcal{NP}$.
  - Es ist unbekannt, ob $\mathcal{NP}$ = PSPACE.

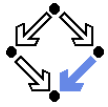
Polynom-Zeit-Reduktionen



- Die Sprache L' ist Polynom-Zeit-reduzierbar auf Sprache L:
  - Es gibt eine Turing-Maschine mit polynomialer Zeitkomplexität, die für jede Eingabe x anhält und eine Ausgabe y erzeugt, sodass $x \in L'$ genau dann wenn $y \in L$.
- Satz: Sei L' Polynom-Zeit-reduzierbar auf L. Dann gilt:
  1. $L \in \mathcal{P} \Rightarrow L' \in \mathcal{P}$.
  2. $L \in \mathcal{NP} \Rightarrow L' \in \mathcal{NP}$.

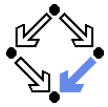
  - Beweis von (1): Sei $L \in \mathcal{P}$. Seien $p_1(n)$ eine polynomiale Zeitschranke für die Reduktion von L' auf L und $p_2(n)$ eine polynomiale Zeitschranke für die Erkennung von L. Für gegebenes $x \in L'$ der Länge n berechnet man y in Zeit $p_1(n)$, die Länge von y ist daher auch beschränkt durch $p_1(n)$. Der Test $y \in L$ kann also in Zeit $p_2(p_1(n))$ ausgeführt werden. Für die Entscheidung $x \in L'$ ist daher $p_1(n) + p_2(p_1(n))$ eine polynomiale Zeitschranke.
  - Beweis von (2): analog.

log-Raum-Reduktionen



- Ein log-Raum Übersetzer:
  - Eine immer haltende Turing-Maschine mit einem Eingabeband (nur zu lesen), einem Ausgabeband (nur zu schreiben), auf dem der LS-Kopf niemals nach links bewegt wird, und einem Arbeitsband, auf dem für eine Eingabe der Länge n nicht mehr als $\log n$ Zellen verwendet werden.
- Die Sprache L' ist log-Raum-reduzierbar auf Sprache L:
  - Es gibt einen log-Raum-Übersetzer, der für jede Eingabe x eine Ausgabe y erzeugt, sodass $x \in L'$ genau dann wenn $y \in L$.
- Satz: Sei L' log-Raum-reduzierbar auf L. Dann gilt:
  1. $L \in \mathcal{P} \Rightarrow L' \in \mathcal{P}$.
  2. $L \in \text{DSPACE}(\log^k n) \Rightarrow L' \in \text{DSPACE}(\log^k n)$.
  - Beweis von (1): Es genügt zu zeigen, dass die Reduktion von L' auf L nicht mehr als polynomiale Zeit benötigt (siehe letzten Beweis). Das Produkt P aus Anzahl der Zustände, Arbeitsbandinhalte und Positionen der L/S-Köpfe auf Eingabeband und Arbeitsband ist eine Oberschranke für die Anzahl der Rechenschritte: $P = |Q| \cdot |\Gamma|^{\log n} \cdot n \cdot \log n \leq |Q| \cdot n^{\log |\Gamma|} \cdot n \cdot n = |Q| \cdot n^{2+\log |\Gamma|}$

Vollständige Probleme

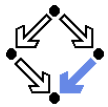


Um zeigen, dass $\mathcal{P} \neq \mathcal{NP}$, sollte man als Kandidaten für ein Element von $\mathcal{NP} \backslash \mathcal{P}$ ein möglichst "schwieriges" Problem in $\mathcal{NP}$ heranziehen.

- Eine Sprache L ist schwierig für eine Klasse $\mathcal{C}$ von Sprachen bezüglich Polynom-Zeit/log-Raum-Reduktion:
  - Jede Sprache in $\mathcal{C}$ ist Polynom-Zeit/log-Raum-reduzierbar auf L.
- L ist vollständig für $\mathcal{C}$ bezüglich Polynom-Zeit/log-Raum-Reduktion:
  - $L \in \mathcal{C}$ und L ist schwierig für $\mathcal{C}$ bezüglich P-Zeit/log-Raum-Reduktion.
- L ist $\mathcal{NP}$-vollständig ($\mathcal{NP}$-schwierig):
  - L ist vollständig/schwierig für $\mathcal{NP}$ bezüglich Polynom-Zeit-Reduktion.
- L ist PSPACE-vollständig (PSPACE-schwierig):
  - L ist vollständig/schwierig für PSPACE bezüglich Pol.-Zeit-Reduktion.

Wäre ein einziges $\mathcal{NP}$-vollständiges Problem in polynomialer Zeit lösbar, wäre $\mathcal{P} = \mathcal{NP}$; dies ist also höchst unwahrscheinlich.

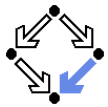
Das Erfüllbarkeits-Problem



- Ein Boolescher Ausdruck E:
  - $x_1, x_2, \ldots$, T, F, $\neg E_1$, $E_1 \wedge E_2$, $E_1 \vee E_2$.
- Ein Boolescher Ausdruck E ist erfüllbar:
  - Die Variablen in E können so durch die Konstanten T oder F ersetzt werden, dass E den Wert T erhält.
    - $(x_1 \vee x_2) \wedge (\neg x_3 \vee x_1)$ ist erfüllbar für $x_1 := T$, $x_2 := F$.
    - $x_1 \wedge \neg x_2 \wedge (\neg x_1 \vee x_2)$ ist nicht erfüllbar.
- Das Erfüllbarkeitsproblem:
  Ist der Boolesche Ausdruck E erfüllbar?
  - Für E mit n Variablen in exponentieller Zeit entscheidbar durch Überprüfung aller 2^n möglichen Variablenbelegungen.
- Die Sprache L_{SAT} des Erfüllbarkeitsproblems:
  - $L_{\text{SAT}} := \{\text{code}(E) \mid E \text{ ist erfüllbar}\}$
    - code(E) ... der Text von E, wobei jede Variable x_i durch die Binärdarstellung von i ersetzt wurde.
- Satz: L_{SAT} ist $\mathcal{NP}$-vollständig.
  - Beweis (Cook, 1971): siehe Skriptum.

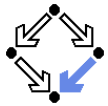
Das bekannteste $\mathcal{NP}$-vollständige Problem.

Andere $\mathcal{NP}$-vollständige Probleme

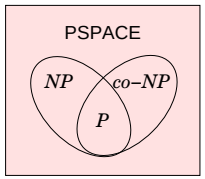


- Das Erfüllbarkeitsproblem für konjunktive Normalformen (CNFs).
  - Boolesche Ausdrücke $(\alpha_{11} \vee \alpha_{12} \vee \ldots) \wedge (\alpha_{21} \vee \alpha_{22} \vee \ldots) \vee \ldots$
    - Jedes α_{ij} eine Konstante, Variable oder negierte Variable.
  - Auch mit nur 3 Literalen pro Klause.
- Das Problem der linearen Programmierung:
  Gibt es für ganzzahlige Matrix A, Vektoren c, d und Zahl B einen Vektor x, sodass $A \cdot x \leq d$ und $c \cdot x \geq B$?
- Das chromatische Zahlenproblem:
  Kann Graph G mit k Farben so gefärbt werden, dass keine zwei benachbarten Knoten dieselbe Farbe haben?
- Das Problem des Handlungsreisenden:
  Gibt es in gewichtetem Graphen G einen Hamiltonschen Zyklus mit Gewicht kleiner gleich k?
- Das Partitionierungsproblem:
  Gibt es zu einer Liste von ganzen Zahlen $i_1, i_2, \ldots, i_k$ eine Unterliste, deren Summe $\frac{1}{2}(i_1 + i_2 + \ldots + i_k)$ ist?
- Das Problem der Hamiltonschen Zyklen.

Die Komplexitätsklasse co-$\mathcal{NP}$



- co$-\mathcal{NP} := \{L \mid \overline{L} \in \mathcal{NP}\}$
  - Die Klasse derjenigen Sprachen, deren Komplement in $\mathcal{NP}$ ist.
    - Diejenigen Eigenschaften, bei denen die Entscheidung der *Negation* der Eigenschaft nicht-deterministisch polynomiale Zeit benötigt.
- Frage: co-$\mathcal{NP} \neq \mathcal{NP}$?
  - Antwort ist nicht bekannt.
  - Wenn ja, dann würde $\mathcal{P} \neq \mathcal{NP}$ gelten.
    - Bekannt: $\mathcal{P} = \{L \mid \overline{L} \in \mathcal{P}\}$, d.h. $\mathcal{P} \subseteq$ co$-\mathcal{NP}$.



- Satz: co-$\mathcal{NP} \neq \mathcal{NP}$ gilt genau dann wenn für alle $\mathcal{NP}$-vollständigen Probleme S gilt, dass $\overline{S} \notin \mathcal{NP}$.
  - Für kein einziges $\mathcal{NP}$-vollständiges Problem S ist aber bekannt, ob $\overline{S} \notin \mathcal{NP}$.
    - Gäbe es ein Problem mit $\overline{S} \in \mathcal{NP}$, dann würde $\mathcal{P} = \mathcal{NP}$ gelten.
    - Um beispielsweise die Nicht-Erfüllbarkeit eines Booleschen Ausdrucks zu entscheiden, scheint also auch ein nicht-deterministischer Algorithmus mehr als polynomiale (exponentielle) Zeit zu benötigen.

Wahrscheinlich ist das Nicht-Erfüllbarkeitsproblem nicht einmal in $\mathcal{NP}$.