Berechenbarkeit und Komplexität
12. Vorlesung

Prof. Dr. Dietrich Kuske

FG Automaten und Logik, TU Ilmenau

Sommersemester 2024

Einführung

Berechenbarkeit

Entscheidbarkeit

Komplexitätstheorie

Zusammenfassung Berechenbarkeitstheorie

Church-Turing These

Die Funktionen, die durch Turingmaschinen bzw. While/Goto-Programme berechnet werden können, sind genau die intuitiv berechenbaren Funktionen.

Unentscheidbarkeit

Probleme, die nicht durch Turing-Maschinen gelöst werden können, sind damit prinzipiell unlösbar (wenn auch u.U. semi-entscheidbar).

Beispiele:

- die verschiedenen Versionen des Halteproblems
- Posts Korrespondenzproblem
- die Theorie der natürlichen Zahlen
- das Schnitt- und verwandte Probleme über kontextfreie Sprachen

Die zentrale Frage der Komplexitätstheorie

Welche Funktionen $\mathbb{N} \to \mathbb{N}$ können von einem effizienten Algorithmus berechnet werden?

Gilt das vielleicht für alle berechenbaren Funktionen?

vorweggenommene Antwort: nein

Für welche Sprachen L kann das Wortproblem („$w \in L$?“) von einem effizienten Algorithmus gelöst werden?

Gilt das vielleicht für alle entscheidbaren Sprachen?

vorweggenommene Antwort: nein

Zunächst: Wie zeigt man diese negativen Resultate?
Was heißt überhaupt „die Funktion $f : \mathbb{N} \to \mathbb{N}$ kann von einem effizienten Algorithmus berechnet werden“ bzw. „das Wortproblem kann von einem effizienten Algorithmus gelöst werden“?

Intuitiver Effizienzbegriff

Das Wortproblem einer Sprache L ist effizient entscheidbar, wenn es einen Algorithmus gibt, der die Antwort auf die Frage „Gehört das Wort w zu L?“ „mit geringen Ressourcen“ (Zeit, Speicherplatz, ...) bestimmt.

„mit geringen Ressourcen“ heißt hier, daß die benötigten Ressourcen nur moderat mit der Eingabelänge $|w|$ wachsen.

Komplexitätsklassen

In Algorithmenvorlesungen haben Sie zu analysieren gelernt, welche Ressourcen (Zeit, Platz) ein gegebener Algorithmus benötigt.

Hier untersuchen wir, welche Ressourcen die Lösung eines gegebenen Problems benötigt. Wir sind insbesondere an Aussagen der Gestalt „Jeder Algorithmus benötigt wenigstens ...“ interessiert.

Hierzu werden wir Probleme in Komplexitätsklassen einordnen.

Deterministische Zeitklassen

Definition

Sei $f\colon \mathbb{N} \to \mathbb{N}$ eine monotone Funktion. Die Klasse $\mathrm{TIME}(f)$ besteht aus allen Sprachen L, für die es eine Turingmaschine M gibt mit:

- M berechnet die charakteristische Funktion von L.
- Für jede Eingabe $w \in \Sigma^*$ erreicht M von der Startkonfiguration $z_0 w \square$ aus nach höchstens $f\big(|w|\big)$ Rechenschritten eine akzeptierende Haltekonfiguration
  (und gibt 0 oder 1 aus, je nachdem ob $w \notin L$ oder $w \in L$ gilt).

Poly = Menge aller Funktionen $\mathbb{N} \to \mathbb{N}$, die durch ein Polynom mit Koeffizienten aus $\mathbb{N}$ beschrieben sind.

Definition

$$
\begin{aligned}
\mathrm{P} &= \bigcup_{f \in \mathrm{Poly}} \mathrm{TIME}(f) \\
\mathrm{EXPTIME} &= \bigcup_{f \in \mathrm{Poly}} \mathrm{TIME}\left(2^{f}\right) \\
\mathrm{2EXPTIME} &= \bigcup_{f \in \mathrm{Poly}} \mathrm{TIME}\left(2^{2^{f}}\right) \\
&\vdots
\end{aligned}
$$

wegen $\mathrm{TIME}(f) \subseteq \mathrm{TIME}\left(2^{f}\right)$ gilt $\mathrm{P} \subseteq \mathrm{EXPTIME} \subseteq \mathrm{2EXPTIME} \subseteq \cdots$

Kritische Frage

Warum werden diese Klassen nicht mit Mehrband-Turingmaschinen, mit GOTO- oder mit While-Programmen definiert?

Mehrband-Turingmaschinen: Wir haben gezeigt, daß sich jede Mehrband-TM M durch eine TM M' simulieren läßt. Eine Analyse des Beweises liefert, daß jede Berechnung der Länge n von M durch eine Berechnung der Länge $\leq n^2$ von M' simuliert wird.

Konsequenz: definiert man die Klassen P, EXPTIME usw. mit Mehrband-TM, so erhält man dieselbe Klasse von Problemen.

erweiterte While-Programme: Wir erlauben auch Zuweisungen der Form $x_i := x_j + x_k$ oder $x_i := x_j \cdot x_k$.

Beispiel: $x_2 := 2$;
LOOP x_1 DO $x_2 := x_2 \cdot x_2$ END;

Zeitbedarf dieses Programms:

- Dieser Algorithmus führt x_1 viele Multiplikationen aus.
  Wenn die elementaren Anweisungen wie $x_i := x_j \cdot x_k$ eine Zeiteinheit benötigen, ist der Zeitbedarf also x_1 (uniformes Kostenmaß).
- Dieser Algorithmus berechnet die Funktion $x_1 \mapsto 2^{2^{x_1}}$. Um dieses Ergebnis niederzuschreiben, werden 2^{x_1} Bits (und damit Zeiteinheiten) benötigt.
  Wenn die elementaren Anweisungen wie $x_i := x_j \cdot x_k$ in $\log(x_j + x_k)$ Zeiteinheiten ausgeführt werden, ist der Zeitbedarf $O(2^{x_1})$ (logarithmisches Kostenmaß).

- Das logarithmische Kostenmaß ist realistischer, da jedes Bit eines Operanden „angefaßt“ werden muß (sind die behandelten Zahlen aber beschränkt, so unterscheiden sich logarithmisches und uniformes Kostenmaß nur um einen konstanten Faktor).
- Unser Beweis, daß die while-berechenbaren Funktionen genau die Turing-berechenbaren sind, verlängert die Berechnungen nur um einen polynomiellen Faktor.

Konsequenz: definiert man die Klassen P, EXPTIME usw. mit erweiterten While- oder Goto-Programmen und dem logarithmischen Kostenmaß, so erhält man dieselbe Klasse von Problemen.

Einige typische Probleme in P (1): Erreichbarkeit

Definition

REACH ist die Menge der gerichteten Graphen mit zwei ausgezeichneten Knoten s und t, in denen es einen Pfad von s nach t gibt.

Satz

REACH ist in P.

Beweis: z.B. mit Dijkstras Algorithmus □

Einige typische Probleme in P (2): Euler-Kreise

Definition

EC ist die Menge der ungerichteten Graphen, die einen Eulerkreis (d.h. einen Kreis, der jede Kante genau einmal durchläuft) enthalten.

Satz (Euler 1736)

Ein Graph (V, E) enthält einen Eulerkreis genau dann, wenn er höchstens eine Zusammenhangskomponente mit > 1 Knoten hat und jeder Knoten geraden Grad hat (d.h. jeder Knoten hat eine gerade Anzahl von Nachbarn).

Folgerung

EC ist in P, denn die genannten Bedingungen lassen sich in polynomieller Zeit prüfen.

Die erweiterte Church-Turing These

P umfaßt die Klasse der effizient lösbaren Probleme.

Begründung: Für jedes „effizient lösbare“ Problem gibt es sicher einen Polynomialzeit-Algorithmus. Dieser läßt sich nach der Church-Turing These auf eine Turingmaschine übertragen. Dabei tritt wohl nur eine polynomielle Verlangsamung auf.

Deterministische Platzklassen

Definition

Sei $f\colon \mathbb{N} \to \mathbb{N}$ eine monotone Funktion. Die Klasse SPACE(f) besteht aus allen Sprachen L, für die es eine Turingmaschine M gibt mit:

- M berechnet die charakteristische Funktion von L.
- Für jede Eingabe $w \in \Sigma^*$ hat jede von der Startkonfiguration $z_0 w \square$ aus erreichbare Konfiguration höchstens die Länge $f(|w|)$.

Definition

$$\begin{aligned}
\mathrm{PSPACE} &= \bigcup_{f \in \mathrm{Poly}} \mathrm{SPACE}(f) \\
\mathrm{EXPSPACE} &= \bigcup_{f \in \mathrm{Poly}} \mathrm{SPACE}\left(2^f\right) \\
\mathrm{2EXPSPACE} &= \bigcup_{f \in \mathrm{Poly}} \mathrm{SPACE}\left(2^{2^f}\right) \\
&\vdots
\end{aligned}$$

wegen $\mathrm{SPACE}(f) \subseteq \mathrm{SPACE}\left(2^f\right)$ gilt
$\mathrm{PSPACE} \subseteq \mathrm{EXPSPACE} \subseteq \mathrm{2EXPSPACE} \subseteq \cdots$

Beobachtung

Es gilt $\mathrm{P} \overset{(1)}{\subseteq} \mathrm{PSPACE} \overset{(3)}{\subseteq} \mathrm{EXPTIME} \overset{(2)}{\subseteq} \mathrm{EXPSPACE} \overset{(4)}{\subseteq} \mathrm{2EXPTIME} \overset{(2)}{\subseteq} \cdots$

Beweis:

(1) Sei $f \in \mathrm{Poly}$. In einer Berechnung der Länge $f(n)$ können höchstens $f(n)$ viele Zellen des Bandes besucht werden, also haben die erreichbaren Konfigurationen die Länge $\leq f(n) + n$. Also gilt

$$\begin{aligned} \mathrm{P} = \bigcup_{f \in \mathrm{Poly}} \mathrm{TIME}(f) &\subseteq \bigcup_{f \in \mathrm{Poly}} \mathrm{SPACE}(f+n) \\ &\subseteq \bigcup_{g \in \mathrm{Poly}} \mathrm{SPACE}(g) = \mathrm{PSPACE}\,. \end{aligned}$$

(2) Analog werden $\mathrm{EXPTIME} \subseteq \mathrm{EXPSPACE}$, $\mathrm{2EXPTIME} \subseteq \mathrm{2EXPSPACE}$ usw. gezeigt.

(3) Seien $f \in \text{Poly}$ und $A \in \text{SPACE}(f)$. Dann existiert eine f-platzbeschränkte TM M, die χ_A berechnet, also insbes. bei jeder Eingabe terminiert. Sei c minimal mit $|\Gamma \cup Z| \leq 2^c$.
Sei w ein Eingabewort der Länge n und sei $k_0 \vdash k_1 \vdash k_2 \vdash \cdots \vdash k_m$ die Berechnung von M bei Eingabe von w. Insbes. gelten für alle $0 \leq i \leq m$:

- $|k_i| \leq f(n)$
- k_i ist akzeptierende Haltekonfiguration gdw. $i = m$, also insbes. $k_i \neq k_j$ f.a. $0 \leq i < j \leq m$.

Da die TM M höchstens $\left(|\Gamma \cup Z|\right)^{f(n)} \leq 2^{cf(n)}$ Konfigurationen der Länge $f(n)$ hat, folgt $m \leq 2^{cf(n)}$.

$$\begin{aligned} \Rightarrow \quad \text{PSPACE} = \bigcup_{f \in \text{Poly}} \text{SPACE}(f) \quad &\subseteq \bigcup_{c \in \mathbb{N}, f \in \text{Poly}} \text{TIME}\left(2^{cf(n)}\right) \\ &\subseteq \bigcup_{g \in \text{Poly}} \text{TIME}\left(2^{g}\right) = \text{EXPTIME}. \end{aligned}$$

(4) Analog werden EXPSPACE $\subseteq$ 2EXPTIME, 2EXPSPACE $\subseteq$ 3EXPTIME usw. gezeigt. □

Einige typische Probleme in PSPACE (1): Erfüllbarkeit

Wir betrachten aussagenlogische Formeln, wie z.B.
$(x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$
(dies ist eine Formel in konjunktiver Normalform).

Solche Formeln lassen sich z.B. durch Wörter über dem Alphabet $\{a, \vee, \wedge, \neg,), (\}$ kodieren (die atomare Formel x_i wird durch das Wort a^{i+1} kodiert).

Zur Erinnerung (Logik und Logikprogrammierung): Eine aussagenlogische Formel φ ist erfüllbar, falls es eine Belegung der atomaren Formeln mit $0, 1$ gibt, so daß die gesamte Formel sich zu 1 auswertet.

Definition

SAT ist die Menge der erfüllbaren aussagenlogischen Formeln.

Beobachtung

SAT $\in$ PSPACE

Beweisidee: Sei φ eine aussagenlogische Formel, in der die atomaren Formeln $x_1, \ldots, x_n$ vorkommen.

Algorithmus: teste nacheinander jede der 2^n vielen Belegungen der atomaren Formeln mit Wahrheitswerten 0 und 1.

Platzbedarf:

- man muß die aktuell untersuchte Belegung hinschreiben (also n Bits)
- man muß jede Teilformel durch 0 oder 1 ersetzen (also $|\varphi|$)

insgesamt: Platz $O\big(|\varphi|\big)$. □

Einige typische Probleme in PSPACE (2): Hamilton-Kreise

Definition

HC ist die Menge der ungerichteten Graphen, die einen Hamiltonkreis (d.h. einen Kreis, der jeden Knoten genau einmal besucht) enthalten.

Beobachtung

HC $\in$ PSPACE

Beweisidee: Sei $G = (V, E)$ ein ungerichteter Graph und $n = |V|$.

Algorithmus: teste nacheinander die $n!$ vielen Bijektionen $\{1, 2, \dots, n\} \to V$, ob sie einen Kreis beschreiben.

Platzbedarf:

- man muß die aktuelle Bijektion hinschreiben (also $n \cdot \log(n) \leq n^2$ Bits)
- man muß nacheinander für jedes i testen, ob $\big(f(i), f(i+1)\big)$ und ob $\big(f(n), f(1)\big)$ Kanten sind (Platz $O(\log n)$)

Damit gesamter Platzbedarf: $\leq cn^2$ für ein geeignetes $c \in \mathbb{N}$. $\square$

Einige typische Probleme in PSPACE (3): 3-Färbbarkeit

Definition 3C

3C ist die Menge der ungerichteten Graphen, deren Knoten sich mit drei Farben färben lassen, so daß benachbarte Knoten unterschiedliche Farben haben.

Beobachtung

3C $\in$ PSPACE

Beweisidee: Sei $G = (V, E)$ ein ungerichteter Graph und $n = |V|$.

Algorithmus: teste nacheinander die 3^n vielen Färbungen
$V \to \{\text{rot}, \text{blau}, \text{grün}\}$.

Platzbedarf:

- aktuelle Abbildung (also $O(n)$ Bits)
- Man muß nacheinander für jede Kante (i, j) testen, ob $f(i) \neq f(j)$ (Platz $O(\log n)$)

Damit gesamter Platzbedarf: $\leq cn$ für ein geeignetes $c \in \mathbb{N}$. $\square$

Zusammenfassung: typische Probleme

Die Algorithmen hatten die Form: „Teste alle ...“, wobei

- es exponentiell viele Kandidaten gab und
- jeder einzelne Kandidat in polynomieller Zeit getestet werden konnte.

nichtdeterministischer „Algorithmus“:

1. rate einen Kandidaten
2. teste diesen.

Dieser läuft dann in Polynomialzeit.

Um diese Beobachtung zu formalisieren, führen wir jetzt nichtdeterministische Turingmaschinen ein.

Nichtdeterministische Turingmaschinen

Definition (vgl. Folie 5.12)

Eine nichtdeterministische Turingmaschine (NTM) ist ein 7-Tupel $M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$, wobei

- Z die endliche Menge der Zustände,
- Σ das Eingabealphabet,
- Γ mit $\Gamma \supseteq \Sigma$ und $\Gamma \cap Z = \emptyset$ das Arbeits- oder Bandalphabet,
- $z_0 \in Z$ der Startzustand,
- $\delta\colon Z \times \Gamma \to \mathcal{P}\big(Z \times \Gamma \times \{L, N, R\}\big)$ die Überführungsfunktion,
- $\square \in \Gamma \backslash \Sigma$ das Leerzeichen oder Blank und
- $E \subseteq Z$ die Menge der Endzustände ist.

Konfigurationen, Berechnungsschritte, Haltekonfigurationen, akzeptierende Haltekonfigurationen werden analog zu Turingmaschinen definiert.

Definition

Sei M NTM. Die von M akzeptierte Sprache ist

$$L(M) = \{w \in \Sigma^* \mid \text{es gibt akzept. Haltekonf. } k \text{ mit } z_0 w\Box \vdash_M^* k\}.$$

Bemerkung:

- Gilt $w \in L(M)$, so existieren u.U. trotzdem
  - nicht akzeptierende Haltekonfigurationen k mit $z_0 w\Box \vdash_M^* k$ und
  - unendliche Berechnungen von $z_0 w\Box$ aus.
- NTM akzeptieren Sprachen, berechnen aber keine Funktionen!

Determinisierbarkeit von NTM

Satz

Zu jeder nichtdeterministischen Turingmaschine gibt es eine Turingmaschine, die dieselbe Sprache akzeptiert.

Beweis:

Sei $M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$ eine NTM, d.h.
$\delta \colon Z \times \Gamma \to \mathcal{P}\big(Z \times \Gamma \times \{L, N, R\}\big)$.

Idee: Wir konstruieren eine TM M_d, die bei Eingabe $x \in \Sigma^*$ systematisch nach einer erfolgreichen Berechnung von M sucht.

Sei $\# \notin Z \cup \Gamma$ ein neues Symbol.

Eine erfolgreiche Berechnung von M mit Eingabe x ist ein Wort der Form

$$k_0 \# k_1 \# \cdots k_{m-1} \# k_m$$

mit folgenden Eigenschaften:

(1) $k_0, k_1, \ldots, k_m \in \Gamma^* Z \Gamma^+$ sind Konfigurationen von M,

(2) $k_0 = z_0 x \square$ ist Initialkonfiguration bei Eingabe von x,

(3) $k_i \vdash_M k_{i+1}$ gilt für alle $0 \leq i < m$ und

(4) $k_m \in \square^* E \Gamma^+$ ist akzeptierende Haltekonfiguration.

Offensichtlich gilt $x \in L(M)$ genau dann, wenn eine erfolgreiche Berechnung von M mit Eingabe x existiert.

Es gibt eine Turingmaschine M', die bei Eingabe von $x\$w \in \Sigma^*\$(Z \cup \Gamma \cup \{\#\})^*$ feststellt, ob w eine erfolgreiche Berechnung von M mit Eingabe x ist:

Hierzu muss M' lediglich die vier Eigenschaften (1)–(4) überprüfen.

Die zu konstruierende TM M_d geht systematisch der Reihe nach alle Wörter $w \in \left(Z \cup \Gamma \cup \{\#\}\right)^*$ durch und überprüft jedesmal (mittels M'), ob w eine erfolgreiche Berechnung von M mit Eingabe x ist.

“Systematisch der Reihe nach” kann hier z.B. mittels einer längenlexikographischen Ordnung realisiert werden.

Sei zunächst $\leq$ eine beliebige lineare Ordnung auf dem Alphabet $\Omega = Z \cup \Gamma \cup \{\#\}$.

Die zu $\leq$ gehörende längenlexikographische Ordnung $\leq_{llex}$ auf Ω^* ist wie folgt definiert:

Für $u, v \in \Omega^*$ gilt $u \leq_{llex} v$ genau dann, wenn

- $|u| < |v|$ oder
- $|u| = |v|$ und es gibt $x, y, z \in \Omega^*, a, b \in \Omega$ mit $u = xay,\ v = xbz,\ a \leq b$.

Grobstruktur der Turingmaschine M_d:

(1) Initialisiere hinter der Eingabe x auf dem Band ein Wort $w \in \left(Z \cup \Gamma \cup \{\#\}\right)^*$ mit ε

(2) Überprüfe mittels M', ob w eine erfolgreiche Berechnung von M mit Eingabe x ist.
Falls ja, gehe in eine akzeptierende Haltekonfiguration über, sonst gehe zu (3)

(3) Inkrementiere w, d.h. überschreibe w mit dem längenlexikographisch nächsten Wort (d.h. w wird durch das kleinste Wort w' mit $w \prec_{llex} w'$ ersetzt und dieses kleinste Wort existiert!).

(4) Gehe zu (2).

dann gilt:

$x \in L(M)$
gdw. es erfolgreiche Berechnung von M mit Eingabe x gibt
gdw. M' in Schritt (2) irgendwann Erfolg meldet
gdw. M_d in eine akzeptierende Haltekonfiguration gerät

sonst wird Sprung in (4) unendlich oft ausgeführt, d.h. M_d hält nicht. $\square$

Folgerung (Ergänzung des Satzes auf Folie 8.16)

Eine Sprache ist genau dann semi-entscheidbar, wenn sie von einer nichtdeterministischen Turingmaschine akzeptiert wird.

Beweis:

$$\begin{array}{lcl} L \text{ semi-entscheidbar} & \iff & L \text{ von TM akzeptiert} \\ & \iff & L \text{ von NTM akzeptiert} \end{array}$$

$\square$

Nichtdeterministische Zeitklassen

Definition

Sei $f : \mathbb{N} \to \mathbb{N}$ eine monotone Funktion. Die Klasse NTIME(f) besteht aus allen Sprachen L, für die es eine nichtdeterministische Turingmaschine M gibt mit:

- M akzeptiert L.
- Für jede Eingabe $w \in \Sigma^*$ hält M auf jeden Fall nach $f(|w|)$ vielen Schritten.

Definition

$$\begin{aligned}
\mathrm{NP} &= \bigcup_{f \in \mathrm{Poly}} \mathrm{NTIME}(f) \\
\mathrm{NEXPTIME} &= \bigcup_{f \in \mathrm{Poly}} \mathrm{NTIME}\left(2^{f}\right) \\
\mathrm{2NEXPTIME} &= \bigcup_{f \in \mathrm{Poly}} \mathrm{NTIME}\left(2^{2^{f}}\right) \\
&\vdots
\end{aligned}$$

wegen $\mathrm{NTIME}(f) \subseteq \mathrm{NTIME}\left(2^{f}\right)$ gilt $\mathrm{NP} \subseteq \mathrm{NEXPTIME} \subseteq \mathrm{2NEXPTIME} \subseteq \cdots$

Lemma

NP $\subseteq$ PSPACE, NEXPTIME $\subseteq$ EXPSPACE, 2NEXPTIME $\subseteq$ 2EXPSPACE usw.

Beweis: Sei $A \in$ NP.
Dann existieren $f \in$ Poly und f-zeitbeschränkte NTM M, die A akzeptiert. Die NTM M benötigt also höchstens den Platz $f(n) + n$. Sei weiter M_d die im Beweis von Folie 12.28 konstruierte äquivalente TM. Sie benötigt maximal den Platz $f(n) \cdot (f(n) + n) \in$ Poly. Also gilt $A \in$ PSPACE, d.h. NP $\subseteq$ PSPACE.

Analog kann NEXPTIME $\subseteq$ EXPSPACE, 2NEXPTIME $\subseteq$ 2EXPSPACE usw. gezeigt werden. □

Nichtdeterministische Platzklassen

Definition

Sei $f : \mathbb{N} \to \mathbb{N}$ eine monotone Funktion. Die Klasse NSPACE(f) besteht aus allen Sprachen L, für die es eine nichtdeterministische Turingmaschine M gibt mit:

- M akzeptiert L.
- Für jede Eingabe $w \in \Sigma^*$ folgt $|k| \leq f(|w|)$ aus $z_0 w \Box \vdash_M^* k$.

Satz von Kuroda (1964)

Sei L eine Sprache. Dann sind äquivalent

1. L ist kontextsensitiv (d.h. vom Typ 1)
2. $L \in \mathrm{NSPACE}(n)$

ohne Beweis.

Definition

$$\begin{aligned}
\text{NPSPACE} &= \bigcup_{f \in \text{Poly}} \text{NSPACE}(f) \\
\text{NEXPSPACE} &= \bigcup_{f \in \text{Poly}} \text{NSPACE}\left(2^f\right) \\
\text{2NEXPSPACE} &= \bigcup_{f \in \text{Poly}} \text{NSPACE}\left(2^{2^f}\right) \\
&\vdots
\end{aligned}$$

wegen $\text{NSPACE}(f) \subseteq \text{NSPACE}\left(2^f\right)$ gilt
$\text{NPSPACE} \subseteq \text{NEXPSPACE} \subseteq \text{2NEXPSPACE} \subseteq \cdots$

Satz von Savitch (1970)

Für jede super-lineare monotone Funktion $f : \mathbb{N} \to \mathbb{N}$ gilt $\mathsf{NSPACE}\big(f(n)\big) \subseteq \mathsf{SPACE}\Big(\big(f(n)\big)^2\Big)$.

Damit haben wir die folgende Struktur der Komplexitätsklassen:

$$\begin{array}{c}
\vdots \\
\mathsf{EXPSPACE} = \mathsf{NEXPSPACE} \\
| \\
\mathsf{NEXPTIME} \\
| \\
\mathsf{EXPTIME} \\
| \\
\mathsf{PSPACE} = \mathsf{NPSPACE} \\
| \\
\mathsf{NP} \\
| \\
\mathsf{P}
\end{array}$$

Zusammenfassung 12. Vorlesung

in dieser Vorlesung neu

- Komplexitätstheorie: Einteilung der entscheidbaren Probleme danach, welche Ressourcen für ihre Lösung nötig sind.
  genauer: Definition von Komplexitätsklassen
- sehr viele natürliche Probleme liegen in PSPACE
- nichtdet. Turingmaschinen, deren Determinisierung

kommende Vorlesung

- die genannten natürlichen Probleme aus PSPACE liegen sogar in NP
- und SAT ist „NP-vollständig“ (d.h. „maximal schwer in NP“)