1.11 Definition (Klassen von Strukturen).

Mit $\left\{\begin{array}{l}\textsf{All}\\ \textsf{Fin}\\ \textsf{Ord}\\ \textsf{FinOrd}\end{array}\right\}$ bezeichnen wir die Klasse $\left\{\begin{array}{l}\textit{aller}\\ \text{aller } \textit{endlichen}\\ \text{aller } \textit{geordneten}\\ \text{aller } \textit{endlichen geordneten}\end{array}\right\}$ Strukturen.

Hierbei heißt eine Struktur $\mathfrak{A}$ *geordnet*, falls es in ihrer Signatur ein 2-stelliges Relationssymbol $<$ gibt, so dass $<^{\mathfrak{A}}$ eine *lineare Ordnung* auf dem Universum A ist, d.h. $<^{\mathfrak{A}}$ ist *transitiv*[2], *antisymmetrisch*[3] und *konnex*[4].

1.2 Einführung in die Komplexitätstheorie

In diesem Abschnitt wiederholen wir einige für diese Vorlesung wichtige Begriffe aus der Komplexitätstheorie. Für eine umfassende Einführung in die Komplexitätstheorie sei auf das Buch

> *Computational Complexity*
> von C. Papadimitriou, Addison-Wesley, 1994

verwiesen.

1.2.1 Turing-Maschinen und Berechenbarkeit

Turing-Maschinen sind von dem englischen Mathematiker Alan Turing (1912–1954) eingeführt worden.

Turing-Maschinen intuitiv:

Das hier benutzte Modell von Turing-Maschinen ist eine 1-Band Turing-Maschine mit linksseitig begrenztem Arbeitsband, bei dem die Eingabe zu Beginn auf dem Arbeitsband gespeichert ist. Es gibt Anweisungen der Form

$$(q, a, q', b, 1),$$

die folgendes besagen: Wenn die Maschine im Zustand q ist und ihr Schreib-/Lesekopf das Symbol a liest, dann kann sie in den Zustand q' wechseln, das gelesene Symbol a durch das Symbol b ersetzen und anschließend den Schreib-/Lesekopf eine Position nach rechts bewegen.
Die Maschine *hält an*, wenn sie einen Endzustand erreicht.

---

[2]D.h. für alle $a, b, c \in A$ gilt: Falls $a <^{\mathfrak{A}} b$ und $b <^{\mathfrak{A}} c$, so auch $a <^{\mathfrak{A}} c$.
[3]D.h. für alle $a, b \in A$ gilt: Falls $(a, b) \in <^{\mathfrak{A}}$, so $(b, a) \notin <^{\mathfrak{A}}$.
[4]D.h. für alle $a, b \in A$ gilt: $a <^{\mathfrak{A}} b$ oder $b <^{\mathfrak{A}} a$ oder $b = a$.

Formale Definition von Turing-Maschinen:

1.12 Definition (Turing-Maschine).
Eine *nicht-deterministische Turing-Maschine* (kurz: NTM) ist ein 6-Tupel

$$M = (Q, \Sigma, \Gamma, \Delta, q_0, F),$$

bestehend aus

- einer endlichen Menge Q von Zuständen,
- einem Arbeitsalphabet Γ mit ausgezeichnetem *Blank-Symbol* $\Box$,
- einem Eingabealphabet $\Sigma \subseteq \Gamma \setminus \{\Box\}$,
- einem Anfangszustand q_0,
- einer Menge $F = F_{\text{akz}} \dot\cup F_{\text{verw}} \subseteq Q$ von Endzuständen, die aus einer Menge F_{akz} von *akzeptierenden* und einer Menge F_{verw} von *verwerfenden* Endzuständen besteht,
- einer Übergangsrelation $\Delta \subseteq (Q \setminus F) \times \Gamma \times Q \times \Gamma \times \{-1, 0, 1\}$.

M heißt *deterministisch* (kurz: M ist eine DTM), falls für alle $q \in Q \setminus F$ und $a \in \Gamma$ genau ein $q' \in Q$, $a' \in \Gamma$ und $m \in \{-1, 0, 1\}$ mit $(q, a, q', a', m) \in \Delta$ existiert. In diesem Fall schreiben wir oft

$$M = (Q, \Sigma, \Gamma, \delta, q_0, F),$$

mit *Überführungsfunktion* $\delta : (Q \setminus F) \times \Gamma \rightarrow Q \times \Gamma \times \{-1, 0, 1\}$.

1.13 Definition (Konfigurationen einer TM).

(a) Eine *Konfiguration* einer Turing-Maschine $M = (Q, \Sigma, \Gamma, \Delta, q_0, F)$ ist ein Tripel

$$c = (q, p, u) \in Q \times \mathbb{N} \times \Gamma^*$$

mit $p \leqslant |u|$. D.h. eine Konfiguration ist eine vollständige Beschreibung aller relevanten Daten über einen bestimmten Zeitpunkt während einer Berechnung. Die Konfiguration $c = (q, p, u)$ gibt an, dass die Maschine sich im Zustand q befindet, der Kopf an Position p steht und die Inschrift des Arbeitsbandes gerade u ist.

Beachte: u ist ein Wort *endlicher* Länge. Die unendlich vielen $\Box$-Symbole weiter rechts auf dem Arbeitsband werden weggelassen.

(b) Wir bezeichnen die Menge aller möglichen Konfigurationen von M mit $\mathcal{C}_M$.

(c) Die *Startkonfiguration* von M bei Eingabe w ist $C_0(w) := (q_0, 0, w\Box)$.

(d) Eine Konfiguration $C = (q, p, u)$ heißt *Endkonfiguration*, falls $q \in F$. Sie heißt *akzeptierend*, falls $q \in F_{\text{akz}}$ und *verwerfend*, falls $q \in F_{\text{verw}}$.

1.14 Definition (Lauf einer TM). Sei $M = (Q, \Sigma, \Gamma, \Delta, q_0, F)$ eine Turing-Maschine.

(a) Die Übergangsrelation Δ induziert eine (partielle) Funktion

$$\mathrm{Next}_M : \mathcal{C}_M \rightarrow \mathrm{Pot}(\mathcal{C}_M),$$

wobei für $C = (q, p, u) \in \mathcal{C}_M$ gilt:[5]

$$\mathrm{Next}_M(C) := \left\{ (q', p', u') \;\middle|\; \begin{array}{l} \text{es gibt } a, b \in \Gamma \text{ und } m \in \{-1, 0, 1\}, \text{ so dass} \\ (q, a, q', b, m) \in \Delta, p' = p{+}m, u_p = a, u'_p = b, \\ u'_i = u_i \text{ für alle } i \neq p, \text{ und } |u'| = |u|, \text{ falls} \\ p' < |u| \text{ bzw. } |u'| = |u| + 1 \text{ und } u'_{p'} = \Box, \text{ falls} \\ p' = |u|. \end{array} \right\}.$$

$\mathrm{Next}_M(C)$ enthält also alle Konfigurationen C', so dass M von Konfiguration C in einem Schritt in die Konfiguration C' übergehen kann.

Bemerkung: Ist M deterministisch und C keine Endkonfiguration, so enthält $\mathrm{Next}_M(C)$ genau eine Konfiguration, d.h. die Maschine geht von einem Berechnungszustand in einen eindeutig bestimmten nächsten Berechnungszustand über.

(b) Auf Eingabe w definiert die Turing-Maschine M einen *Berechnungsbaum* $B_M(w)$ wie folgt: Der Baum hat einen mit $C_0(w)$ beschrifteten Wurzelknoten. Ist v ein mit C beschrifteter Knoten im Baum, so hat er für jedes $C' \in \mathrm{Next}_M(C)$ einen Kindknoten mit Beschriftung C'.

(c) Ein *Lauf* der TM M auf Eingabe w ist ein Pfad im Berechnungsbaum, der an der Wurzel beginnt und entweder unendlich lang ist oder in einer Endkonfiguration endet (der Lauf heißt dann *endlich* bzw. *terminierend*).

Bemerkung: Ein Lauf entspricht also einer möglichen Berechnung von M auf Eingabe w. Wenn die TM nicht deterministisch ist, kann sie auf der gleichen Eingabe verschiedene Berechnungen ausführen.

(d) Ein Lauf heißt *akzeptierend* (*verwerfend*), falls er in einer akzeptierenden (verwerfenden) Endkonfiguration endet.

1.15 Definition (Sprache einer TM).

(a) Eine Turing-Maschine M *akzeptiert* eine Eingabe w, falls es (mindestens) einen akzeptierenden Lauf von M auf w gibt.
M *verwirft* eine Eingabe w, falls alle terminierenden Läufe von M auf w verwerfen.

(b) Die Sprache $L(M) := \{w \in \Sigma^* \mid M \text{ akzeptiert } w\}$ heißt die von M *akzeptierte* Sprache.

[5]Hierbei ist $u = u_0 \cdots u_\ell$ mit $u_i \in \Gamma$. Mit $|u|$ bezeichnen wir die *Länge* des Worts u, also $|u| = \ell + 1$.

(c) Eine Sprache $L \subseteq \Sigma^*$ heißt *semi-entscheidbar*, falls es eine Turing-Maschine M gibt, so dass $L = L(M)$.

(d) Eine Sprache $L \subseteq \Sigma^*$ heißt *entscheidbar*, falls es eine Turing-Maschine M gibt, so dass $L = L(M)$ und jeder Lauf von M auf jeder Eingabe w terminiert.

1.16 Bemerkungen.

(a) Man beachte die Asymmetrie in der Definition des *Akzeptierens* einer Eingabe w: Zum Akzeptieren eines Worts w muss es nur (mindestens) einen akzeptierenden Lauf geben. Zum Verwerfen des Worts w müssen hingegen alle Läufe bei Eingeabe w verwerfen oder nicht-terminierend sein.

(b) Ist M *deterministisch*, so besteht der Berechnungsbaum nur aus einem Pfad (d.h. jeder Knoten hat höchstens einen Nachfolger). Die Berechnung von M auf Eingabe w ist also eindeutig.

Es ist bekannt, dass deterministische und nicht-deterministische Turing-Maschinen genau dieselben Sprachen entscheiden können:

1.17 Satz.
Jede durch eine nicht-deterministische Turing-Maschine (semi-)entscheidbare Sprache L ist auch durch eine deterministische Turing-Maschine (semi-)entscheidbar.

(Hier ohne Beweis.)

Das Halteproblem für Turing-Maschinen:

Man kann Turing-Maschinen M und Eingaben w als Wörter $\rho(M)$ und $\rho(w)$ über einem festen, von der TM unabhängigen, Alphabet $\mathscr{A}$ so kodieren, dass sich die Berechnung von M auf w effektiv aus $\rho(M)$ und $\rho(w)$ rekonstruieren lässt (die genaue Wahl der Kodierung ist für diese Vorlesung nicht wichtig und wird deshalb hier weggelassen). Unter Verwendung einer solchen Kodierung kann man Eingaben $\rho(M)\#\rho(w)$ über dem Alphabet $\mathscr{A} \,\dot\cup\, \{\#\}$ betrachten und eine Turing-Maschine U bauen, die bei Eingabe $\rho(M)\#\rho(w)$ die Berechnung von M auf w simuliert. Eine solche Maschine U heißt *universelle Turing-Maschine*.

1.18 Definition (Halteproblem). Das *Halteproblem* ist die Sprache

$$H := \{ \rho(M)\#\rho(w) \mid M \text{ ist eine DTM, } w \text{ eine Eingabe für } M, M \text{ terminiert auf } w \}.$$

Das *Halteproblem auf leerem Eingabewort* ist die Sprache

$$H_\varepsilon := \{ \rho(M) \mid M \text{ ist eine DTM, } M \text{ terminiert bei Eingabe } \varepsilon \}.$$

Dabei bezeichnet ε das *leere Wort*.

1.19 Satz.
Das Halteproblem H für Turing-Maschinen und das Halteproblem H_ε auf leerem Eingabewort ist nicht entscheidbar (aber semi-entscheidbar).

(Hier ohne Beweis.)

1.2.2 Komplexitätsklassen

Neben der Frage nach dem absolut Berechenbaren interessieren wir uns für Berechenbarkeit innerhalb einer gegebenen Menge von Ressourcen. Die wichtigsten Ressorcen, dessen Verfügbarkeit oft beschränkt ist, sind *Zeit* und *Platz*.

1.20 Definition.
Seien $T, S : \mathbb{N} \to \mathbb{R}_{\geqslant 0}$ monoton wachsende Funktionen, und sei M eine TM.

(a) M heißt *T-zeitbeschränkt*, falls jede Berechnung von M auf Eingaben der Länge n höchstens $T(n)$ Schritte macht. D.h. für alle $n \in \mathbb{N}$ und alle Eingabeworte w der Länge n gilt: Jeder Pfad im Berechnungsbaum $B_M(w)$ hat die Länge $\leqslant T(n)$.

(b) M heißt *S-platzbeschränkt*, falls jede Berechnung von M auf Eingaben der Länge n höchstens $S(n)$ Speicherstellen benutzt. D.h. für alle $n \in \mathbb{N}$, alle Eingabeworte w der Länge n und jeden Knoten v im Berechnungsbaum $B_M(w)$ gilt: Ist (q, p, u) die Konfiguration, mit der v beschriftet ist, so hat das Wort u die Länge $|u| \leqslant S(n)$.

1.21 Definition (Komplexitätsklassen).
Seien $S, T : \mathbb{N} \to \mathbb{R}_{\geqslant 0}$ monoton wachsende Funktionen.

(a) $\textsc{Dtime}(T)$ (bzw. $\textsc{Dspace}(S)$) ist die Klasse aller Sprachen L, für die es eine *deterministische* Turing-Maschine M gibt, die T-zeitbeschränkt (bzw. S-platzbeschränkt) ist und für die gilt: $L = L(M)$.

(b) $\textsc{Ntime}(T)$ (bzw. $\textsc{Nspace}(S)$) ist die Klasse aller Sprachen L, für die es eine *nichtdeterministische* Turing-Maschine M gibt, die T-zeitbeschränkt (bzw. S-platzbeschränkt) ist und für die gilt: $L = L(M)$.

(c)
$$\begin{aligned}
\mathrm{P} := \textsc{Ptime} &:= \bigcup_{k\in\mathbb{N}} \textsc{Dtime}(n^k), & \textsc{Pspace} &:= \bigcup_{k\in\mathbb{N}} \textsc{Dspace}(n^k),\\
\mathrm{NP} &:= \bigcup_{k\in\mathbb{N}} \textsc{Ntime}(n^k), & \mathrm{NP}\textsc{space} &:= \bigcup_{k\in\mathbb{N}} \textsc{Nspace}(n^k),\\
\textsc{Exptime} &:= \bigcup_{k\in\mathbb{N}} \textsc{Dtime}(2^{(n^k)}), & \textsc{Expspace} &:= \bigcup_{k\in\mathbb{N}} \textsc{Dspace}(2^{(n^k)}),\\
\mathrm{N}\textsc{Exptime} &:= \textstyle\bigcup_{k\in\mathbb{N}} \textsc{Ntime}(2^{(n^k)}), & \mathrm{N}\textsc{Expspace} &:= \textstyle\bigcup_{k\in\mathbb{N}} \textsc{Nspace}(2^{(n^k)}).
\end{aligned}$$

<u>Beachte:</u> Komplexitätsklassen sind keine Klassen von Turing-Maschinen sondern Klassen von Problemen (d.h. Sprachen, d.h. Mengen von Worten über einem Alphabet).

Zwei weitere wichtige Platzkomplexitätsklassen sind die Klassen $\textsc{Logspace}$ und $\mathrm{N}\textsc{Logspace}$ aller Probleme, die durch Turing-Maschinen gelöst werden können, deren Platzverbrauch bei Eingaben der Länge n nur höchstens von der Größe $\mathcal{O}(\log n)$ ist. Da $\log n < n$ ist, heißt das, dass das Arbeitsband viel zu kurz ist um das ganze Eingabewort zu enthalten.

Zur Definition der Klassen LOGSPACE und NLOGSPACE werden daher Turing-Maschinen benutzt, die ein separates *Eingabeband* besitzen, dessen Kopf nur zum Lesen des Eingabeworts, nicht aber zum Schreiben genutzt werden kann. Neben diesem Eingabeband gibt es ein (herkömmliches) *Arbeitsband*, das zu Beginn der Berechnung leer ist. Für eine monoton wachsende Funktion $S : \mathbb{N} \to \mathbb{R}_{\geqslant 0}$ schreiben wir

$$\textsc{Dspace}'(S) \quad \text{bzw.} \quad \textsc{Nspace}'(S),$$

um die Klassen aller Sprachen L zu bezeichnen, für die es eine deterministische bzw. nichtdeterministische Turing-Maschine M mit separatem Eingabeband gibt (von dem gelesen, auf das aber nicht geschrieben werden kann), so dass bei Eingaben der Länge n auf dem *Arbeitsband* von M zu jedem Zeitpunkt der Berechnung höchstens $S(n)$ Speicherstellen benutzt werden.

1.22 Definition (LOGSPACE **und** NLOGSPACE)**.**

$$\begin{aligned} \textsc{Logspace} &:= \bigcup_{k \in \mathbb{N}} \textsc{Dspace}'(k \cdot \log n), \\ \textsc{NLogspace} &:= \bigcup_{k \in \mathbb{N}} \textsc{Nspace}'(k \cdot \log n). \end{aligned}$$

1.23 Proposition. *Seien $S, T : \mathbb{N} \to \mathbb{R}_{\geqslant 0}$ monoton wachsende Funktionen. Es gilt:*

$$\begin{aligned} \textsc{Dtime}(T) &\subseteq \textsc{Ntime}(T), \qquad & \textsc{Dspace}(S) &\subseteq \textsc{Nspace}(S), \\ \textsc{Dtime}(T) &\subseteq \textsc{Dspace}(T), \qquad & \textsc{Ntime}(T) &\subseteq \textsc{Nspace}(T). \end{aligned}$$

Beweis: klar. □

1.24 Korollar. LOGSPACE $\subseteq$ NLOGSPACE *und* P $\subseteq$ NP $\subseteq$ NPSPACE.

1.25 Satz. *Für jede monoton wachsende Funktion $T : \mathbb{N} \to \mathbb{R}_{\geqslant 0}$ mit $T(n) \geqslant \log n$ gilt:*

$$\textsc{Dtime}(T) \subseteq \textsc{Ntime}(T) \subseteq \textsc{Nspace}(T) \subseteq \textsc{Dtime}(2^{\mathcal{O}(T)}).$$

Beweisidee: Die ersten beiden Inklusionen sind klar, siehe Proposition 1.23. Zum Beweis der dritten Inklusion sei M eine T-platzbeschränkte NTM. Jede Konfiguration der Turing-Maschine, die während eines Laufs von M bei einer Eingabe der Länge n auftritt, kann durch ein Wort der Länge $\mathcal{O}(T(n))$ kodiert werden. Insbesondere gibt es höchstens $2^{\mathcal{O}(T(n))}$ viele solche Konfigurationen. Die (nichtdeterministische) Maschine M kann daher durch eine *deterministische* Turing-Maschine M' simuliert werden, die bei Eingabe eines Worts w auf ihrem Arbeitsband eine Liste aller Konfigurationen erzeugt, die M bei Eingabe w erreichen kann. □

1.26 Korollar. NLOGSPACE $\subseteq$ P *und* NPSPACE $\subseteq$ EXPTIME.

Frage: Welche Inklusionen sind strikt? – Kann man z.B. mit exponentieller Zeit *mehr* Probleme lösen als mit polynomieller Zeit?

Hierarchiesätze:

1.27 Definition (zeit- bzw. platzkonstruierbare Funktionen).
Seien $S, T : \mathbb{N} \to \mathbb{R}_{\geqslant 0}$ monoton wachsende Funktionen.

(a) T heißt *zeitkonstruierbar*, falls es eine DTM gibt, die auf Eingaben der Länge $n \in \mathbb{N}$ genau $T(n)$ Schritte macht und dann hält. (Dabei ist egal, was M berechnet.)

(b) S heißt *platzkonstruierbar*, falls es eine DTM gibt, die auf Eingaben der Länge $n \in \mathbb{N}$ irgendwann terminiert und im Laufe der Berechnung auf ihrem Arbeitsband genau $S(n)$ Speicherzellen benutzt.

Bemerkung: Zeit- bzw. platzkonstruierbare Funktionen sind in dem Sinne "schön", dass ihre Komplexität nicht höher ist als ihre Werte. Die meisten üblicherweise verwendeten Funktionen, beispielsweise jedes Polynom sowie die Funktionen $n!$, 2^n und $2^{(n^k)}$, sind zeit- und platzkonstruierbar. Die Funktion $k \cdot \log n$ ist platzkonstruierbar (aber nicht zeitkonstruierbar).

1.28 Theorem (Zeithierarchiesatz). *Seien $T, t : \mathbb{N} \to \mathbb{R}_{\geqslant 0}$ monoton wachsende Funktionen mit $t \cdot \log t = o(T)$, T zeitkonstruierbar und $T(n) \geqslant n$ für alle $n \in \mathbb{N}$. Dann gilt:*

$$\textsc{Dtime}(t) \subsetneqq \textsc{Dtime}(T).$$

(Hier ohne Beweis.)

1.29 Korollar. P $\subsetneqq$ EXPTIME.

Analog zum Zeithierarchiesatz gibt es auch einen Platzhierarchiesatz:

1.30 Theorem (Platzhierarchiesatz). *Seien $S, s : \mathbb{N} \to \mathbb{R}_{\geqslant 0}$ monoton wachsende Funktionen mit $s = o(S)$, S platzkonstruierbar und $S(n) \geqslant \log n$ für alle $n \in \mathbb{N}$. Dann gilt:*

$$\textsc{Dspace}(s) \subsetneqq \textsc{Dspace}(S).$$

(Hier ohne Beweis.)

1.31 Korollar. LOGSPACE $\subsetneqq$ PSPACE $\subsetneqq$ EXPSPACE.

Komplementabschluss nichtdeterministischer Komplexitätsklassen:

Frage: Wenn $L \subseteq \Sigma^*$ eine Sprache aus $\textsc{Ntime}(T)$ oder $\textsc{Nspace}(S)$ ist, welche Ressourcen benötigt man, um die Sprache $\overline{L} := \Sigma^* \setminus L$ (d.h. das Komplement von L) zu entscheiden?

1.32 Definition. Sei $\mathscr{L}$ eine Komplexitätsklasse. Wir definieren die Komplementklasse

$$\text{co-}\mathscr{L} \;:=\; \{\overline{L} \mid L \in \mathscr{L}\}$$

als die Klasse aller Sprachen, deren Komplement in $\mathscr{L}$ liegt.

Beispielsweise besteht die Klasse co-NP aus allen Sprachen, deren Komplement in NP liegt.

Klar ist, dass *deterministische* Komplexitätsklassen unter Komplementbildung abgeschlossen sind – dazu vertauscht man einfach akzeptierende und verwerfende Endzustände einer DTM. Es gilt also insbesondere $\mathrm{P} = \text{co-}\mathrm{P}$, $\textsc{Pspace} = \text{co-}\textsc{Pspace}$ und $\textsc{Logspace} = \text{co-}\textsc{Logspace}$.
Bloßes Vertauschen von akzeptierenden und verwerfenden Endzuständen liefert bei *nichtdeterministischen* Turing-Maschinen in der Regel nicht das gewünschte Ergebnis. In der Tat ist die Frage, ob *nichtdeterministische Zeitkomplexitätsklassen* unter Komplementbildung abgeschlossen sind, ein offenes Forschungsproblem. Man vermutet, dass

$$\text{co-NP} \neq \text{NP}$$

gilt (woraus insbesondere $\mathrm{P} \neq \mathrm{NP}$ folgen würde), kann dies bisher aber nicht beweisen.

Erstaunlicherweise konnte das Problem für nichtdeterministische *Platz*klassen gelöst werden:

1.33 Theorem (Satz von Immerman und Szelepcsényi).
Sei $S : \mathbb{N} \to \mathbb{R}_{\geqslant 0}$ platzkonstruierbar und $S(n) \geqslant \log n$ für alle $n \in \mathbb{N}$. Dann gilt:

$$\textsc{Nspace}(S) \;=\; \text{co-}\textsc{Nspace}(S).$$

(Hier ohne Beweis.)

1.34 Korollar. $\textsc{NLogspace} = \text{co-}\textsc{NLogspace}$.

Frage: Wie verhalten sich nichtdeterministische zu deterministischen Komplexitätsklassen? Gilt zum Beispiel

$$\mathrm{P} = \mathrm{NP}\ ?$$

Wiederum ist das Problem für Zeitklassen offen, während es für Platzklassen weitgehend gelöst ist:

1.35 Theorem (Satz von Savitch).
Sei $S : \mathbb{N} \to \mathbb{R}_{\geqslant 0}$ platzkonstruierbar und $S(n) \geqslant \log n$ für alle $n \in \mathbb{N}$. Dann gilt:

$$\textsc{Nspace}(S) \;\subseteq\; \textsc{Dspace}(S^2).$$

(Hier ohne Beweis.)

1.36 Korollar. $\textsc{Pspace} = \textsc{NPspace}$ *und* $\textsc{Expspace} = \textsc{NExpspace}$.

Offen bleibt weiterhin, ob

$$\textsc{Logspace} \;=\; \textsc{NLogspace}\ ?$$

Insgesamt erhält man die in Abbildung 1.1 dargestellte Inklusionsstruktur der Komplexitätsklassen.

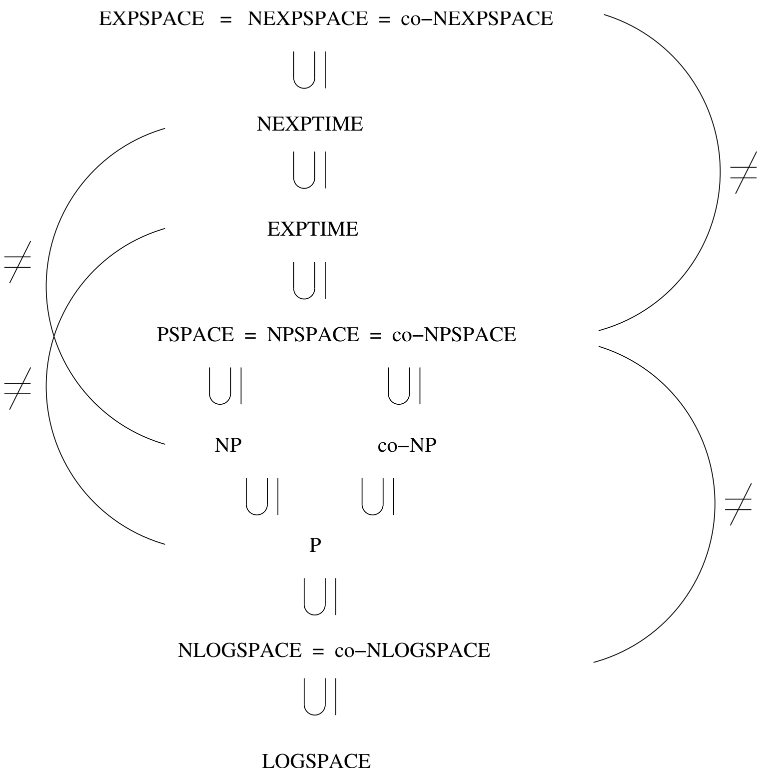

Abbildung 1.1: Inklusionsstruktur der Komplexitätsklassen.

1.2.3 Reduktionen und NP-Vollständigkeit

1.37 Definition (Polynomialzeit-Reduktionen).
Seien Σ_1 und Σ_2 endliche Alphabete, und sei $A \subseteq \Sigma_1^*$ und $B \subseteq \Sigma_2^*$.
Eine *Polynomialzeit-Reduktion* (kurz: Reduktion) von A auf B ist eine (deterministisch) in polynomieller Zeit berechenbare Funktion $f : \Sigma_1^* \to \Sigma_2^*$, so dass für alle $w \in \Sigma_1^*$ gilt:

$$w \in A \iff f(w) \in B.$$

Existiert eine Polynomialzeitreduktion f von A auf B, so sagen wir: A *ist polynomiell reduzierbar auf* B (kurz: $A \leqslant_p B$ bzw. genauer $f : A \leqslant_p B$).

Bemerkung: Der Begriff *Polynomialzeit-Reduktion* ist so definiert, dass folgendes gilt: Falls $A \leqslant_p B$ und $B \in \text{PTIME}$, so auch $A \in \text{PTIME}$. Umgekehrt heißt das: Wenn $A \leqslant_p B$ und $A \notin \text{PTIME}$, so auch $B \notin \text{PTIME}$.
Anschaulich bedeutet $A \leqslant_p B$, dass A "höchstens so schwer" wie B ist.

1.38 Definition (Vollständigkeit). Sei $\mathscr{L}$ eine Komplexitätsklasse und sei $B \subseteq \Sigma^*$.

(a) Das Problem B heißt $\mathscr{L}$*-hart*, falls für alle Probleme $A \in \mathscr{L}$ gilt: $A \leqslant_p B$.

(b) B heißt $\mathscr{L}$*-vollständig*, falls $B \in \mathscr{L}$ und B $\mathscr{L}$-hart ist.

In einem gewissen Sinn sind die vollständigen Probleme die "schwersten" Probleme einer Komplexitätsklasse. Insbesondere gilt für jedes NP-*vollständige* Problem B: Falls B in P liegt, so liegt *jedes* Problem aus NP bereits in P, d.h. es gilt NP = P.
Beispiele für NP-vollständige Probleme sind:

SAT (aussagenlogisches Erfüllbarkeitsproblem)
Eingabe: Eine aussagenlogische Formel α.
Frage: Gibt es eine Variablenbelegung, die α erfüllt?

TSP (Travelling Salesman Problem)
Eingabe: Eine Distanzmatrix D von Städten und eine Distanz k.
Frage: Gibt es eine Rundreise, die jede Stadt genau einmal besucht und bei der insgesamt höchstens die Distanz k zurückgelegt wird?

CLIQUE
Eingabe: Ein ungerichteter Graph $G = (V, E)$ und eine Zahl $k \in \mathbb{N}$.
Frage: Gibt es in G eine Clique[6] der Größe k?

[6] d.h. eine Menge $V' \subseteq V$ so dass für alle $u, v \in V'$ mit $u \neq v$ gilt: $E(u, v)$