Vorlesungsskript

Komplexitätstheorie

Wintersemester 2008/09

Prof. Dr. Johannes Köbler
Humboldt-Universität zu Berlin
Lehrstuhl Komplexität und Kryptografie

27. November 2008

Inhaltsverzeichnis

Inhaltsverzeichnis

1 Einführung **1**

2 Rechenmodelle **3**
2.1 Deterministische Turingmaschinen 3
2.2 Nichtdeterministische Berechnungen 4
2.3 Zeitkomplexität 5
2.4 Platzkomplexität 6

3 Grundlegende Beziehungen **7**
3.1 Robustheit von Komplexitätsklassen 7
3.2 Deterministische Simulationen von nichtdeterministischen Berechnungen 9
3.3 Der Satz von Savitch 10
3.4 Der Satz von Immerman und Szelepcsényi 11

4 Hierarchiesätze **15**
4.1 Diagonalisierung und die Unentscheidbarkeit des Halteproblems 15
4.2 Das Gap-Theorem 16
4.3 Zeit- und Platzhierarchiesätze 17

5 Reduktionen **19**
5.1 Logspace-Reduktionen 19
5.2 P-vollständige Probleme und polynomielle Schaltkreiskomplexität 21
5.3 NP-vollständige Probleme 23
5.4 NL-vollständige Probleme 27

1 Einführung

In der Komplexitätstheorie werden algorithmische Probleme daraufhin untersucht, welche Rechenressourcen zu ihrer Lösung benötigt werden. Naturgemäß bestehen daher enge Querbezüge zu

- Algorithmen (obere Schranken)
- Automatentheorie (Rechenmodelle)
- Berechenbarkeit (Was ist überhaupt algorithmisch lösbar?)
- Logik (liefert viele algorithmische Probleme, mit ihrer Hilfe kann auch die Komplexität von Problemen charakterisiert werden)
- Kryptographie (Wieviel Rechenressourcen benötigt ein Gegner, um ein Kryptosystem zu brechen?)

Zur weiteren Motivation betrachten wir eine Reihe von konkreten algorithmischen Problemstellungen.

Erreichbarkeitsproblem in Graphen (REACH):

Gegeben: Ein gerichteter Graph $G = (V, E)$ mit $V = \{1, \dots, n\}$ und $E \subseteq V \times V$.

Gefragt: Gibt es in G einen Weg von Knoten 1 zu Knoten n?

Zur Erinnerung: Eine Folge $(v_1, \dots, v_k)$ von Knoten heißt **Weg** in G, falls für $j = 1, \dots, k-1$ gilt: $(v_j, v_{j+1}) \in E$.

Da als Antwort nur "ja" oder "nein" möglich ist, handelt es sich um ein **Entscheidungsproblem**. Ein solches lässt sich formal durch eine Sprache beschreiben, die alle positiven (mit "ja" zu beantwortenden) Problemeingaben enthält:

$$\text{REACH} = \{G \mid \text{in } G \text{ ex. ein Weg von 1 nach } n\}.$$

Hierbei setzen wir eine Kodierung von Graphen durch Wörter über einem geeigneten Alphabet Σ voraus. Wir können G beispielsweise durch eine Binärfolge der Länge n^2 kodieren, die aus den n Zeilen der Adjazenzmatrix von G gebildet wird.

Wir entscheiden REACH durch einen Wegsuche-Algorithmus. D.h. wir markieren nach und nach alle Knoten, die vom Knoten 1 aus erreichbar sind. Dabei speichern wir alle markierten Knoten solange in einer Menge S bis auch ihre Nachbarknoten markiert sind. Genaueres ist folgendem Algorithmus zu entnehmen:

Algorithmus `suche-Weg`(G)

```
Input: Gerichteter Graph G = (V, E) mit V = {1, ..., n}
  S := {1}
  markiere Knoten 1
  repeat
    waehle einen Knoten u ∈ S
    S := S − {u}
    for all (u, v) ∈ E do
      if v ist nicht markiert then
        markiere v
        S := S ∪ {v}
  until S = ∅
  if n ist markiert then accept else reject
```

Es ist üblich, den Ressourcenverbrauch von Algorithmen (wie z.B. Rechenzeit oder Speicherplatz) in Abhängigkeit von der Größe der Problemeingabe zu messen. Falls die Eingabe aus einem Graphen besteht, kann beispielsweise die Anzahl n der Knoten (oder auch die Anzahl m der Kanten) als Bezugsgröße dienen. Genau genommen hängt die Eingabegröße davon ab, welche Kodierung wir für die Eingaben verwenden.

Komplexitätsbetrachtungen:

- REACH ist in Zeit n^3 entscheidbar.

- REACH ist nichtdeterministisch in Platz $\log n$ entscheidbar (und daher deterministisch in Platz $\log^2 n$; Satz von Savitch).

Als nächstes betrachten wir das Problem, einen maximalen Fluss in einem Netzwerk zu bestimmen.

Maximaler Fluß (MAXFLOW):

Gegeben: Ein gerichteter Graph $G = (V, E)$ mit $V = \{1, \dots, n\}$, $E \subseteq V \times V$ und einer Kapazitätsfunktion $c : E \to \mathbb{N}$.

Gesucht: Ein Fluss $f : E \to \mathbb{N}$ von 1 nach n in G, d.h.
- $\forall e \in E : f(e) \leq c(e)$ und
- $\forall v \in V - \{1, n\} : \sum_{(v,u)\in E} f(v, u) = \sum_{(u,v)\in E} f(u, v)$,

mit maximalem Wert $w(f) = \sum_{(1,v)\in E} f(1, v)$.

Da hier nach einer Lösung (Fluss) mit optimalem Wert gesucht wird, handelt es sich um ein **Optimierungsproblem** (genauer: Maximierungsproblem). Im Gegensatz hierzu wird bei vielen Entscheidungsproblemen nach der Existenz einer Lösung (mit gewissen Eigenschaften) gefragt.

Komplexitätsbetrachtungen:
- MAXFLOW ist in Zeit n^5 lösbar.
- MAXFLOW ist in Platz n^2 lösbar.

Das folgende Problem scheint zwar auf den ersten Blick nur wenig mit dem Problem MAXFLOW gemein zu haben. In Wirklichkeit entpuppt es sich jedoch als ein Spezialfall von MAXFLOW.

Perfektes Matching in bipartiten Graphen (MATCHING):

Gegeben: Ein bipartiter Graph $G = (U, V, E)$ mit $U = V = \{1, \dots, n\}$ und $E \subseteq U \times V$.

Gefragt: Besitzt G ein perfektes Matching?

Zur Erinnerung: Eine Kantenmenge $M \subseteq E$ heißt **Matching**, falls für alle Kanten $e = (u, v), e' = (u', v') \in M$ mit $e \neq e'$ gilt: $u \neq u'$ und $v \neq v'$. Gilt zudem $\|M\| = n$, so heißt M **perfekt**.

Komplexitätsbetrachtungen:
- MATCHING ist in Zeit n^3 entscheidbar.
- MATCHING ist in Platz n^2 entscheidbar.

Die bisher betrachteten Probleme können in deterministischer Polynomialzeit gelöst werden und gelten daher als effizient lösbar. Zum Schluss dieses Abschnitts betrachten wir ein Problem, für das vermutlich nur ineffiziente Algorithmen existieren.

Travelling Salesman Problem (TSP):

Gegeben: Eine symmetrische $n \times n$-Distanzmatrix $D = (d_{ij})$ mit $d_{ij} \in \mathbb{N}$.

Gesucht: Eine kürzeste Rundreise, d.h. eine Permutation $\pi \in S_n$ mit minimalem Wert $w(\pi) = \sum_{i=1}^{n} d_{\pi(i),\pi(i+1)}$, wobei wir $\pi(n + 1) = \pi(1)$ setzen.

Komplexitätsbetrachtungen:
- TSP ist in Zeit $n!$ lösbar (Ausprobieren aller Rundreisen).
- TSP ist in Platz n lösbar (mit demselben Algorithmus, der TSP in Zeit $n!$ löst).
- Durch dynamisches Programmieren[a] lässt sich TSP in Zeit $n^2 \cdot 2^n$ lösen, der Platzverbrauch erhöht sich dabei jedoch auf $n \cdot 2^n$ (siehe Übungen).

[a] Hierzu berechnen wir für alle Teilmengen $S \subseteq \{2, \dots, n\}$ und alle $j \in S$ die Länge $l(S, j)$ eines kürzesten Pfades von 1 nach j, der alle Städte in S genau einmal besucht.

2 Rechenmodelle

2.1 Deterministische Turingmaschinen

Definition 1 (**Mehrband-Turingmaschine**).
Eine ***deterministische k-Band-Turingmaschine*** *(k-***DTM** *oder einfach* **DTM***) ist ein Quadrupel* $M = (Q, \Sigma, \Gamma, \delta, q_0)$. *Dabei ist*

- Q *eine endliche Menge von* ***Zuständen***,
- Σ *eine endliche Menge von Symbolen (das* ***Eingabealphabet****) mit* $\sqcup, \triangleright \notin \Sigma$ *(*$\sqcup$ *heißt* ***Blank*** *und* $\triangleright$ *heißt* ***Anfangssymbol***,
- Γ *das* ***Arbeitsalphabet*** *mit* $\Sigma \cup \{\sqcup, \triangleright\} \subseteq \Gamma$,
- $\delta : Q \times \Gamma^k \to (Q \cup \{q_h, q_{ja}, q_{nein}\}) \times (\Gamma \times \{L, R, N\})^k$ *die* ***Überführungsfunktion*** *(*q_h *heißt* ***Haltezustand***, q_{ja} ***akzeptierender*** *und* q_{nein} ***verwerfender Endzustand***
- *und* q_0 *der* ***Startzustand***.

Befindet sich M im Zustand $q \in Q$ und stehen die Schreib-Lese-Köpfe auf Feldern mit den Inschriften $a_1, \ldots, a_k$ (a_i auf Band i), so geht M bei Ausführung der Anweisung $\delta : (q, a_1, \ldots, a_k) \mapsto (q', a'_1, D_1, \ldots, a'_k, D_k)$ in den Zustand q' über, ersetzt auf Band i das Symbol a_i durch a'_i und bewegt den Kopf gemäß D_i (im Fall $D_i = L$ um ein Feld nach links, im Fall $D_i = R$ um ein Feld nach rechts und im Fall $D_i = N$ wird der Kopf nicht bewegt).

Außerdem verlangen wir von δ, dass für jede Anweisung $(q, a_1, \ldots, a_k) \mapsto (q', a'_1, D_1, \ldots, a'_k, D_k)$ mit $a_i = \triangleright$ die Bedingung $a'_i = \triangleright$ und $D_i = R$ erfüllt ist (d.h. das Anfangszeichen $\triangleright$ darf nicht durch ein anderes Zeichen überschrieben werden und der Kopf muss nach dem Lesen von $\triangleright$ immer nach rechts bewegt werden).

Definition 2. *Eine* ***Konfiguration*** *ist ein* $(2k+1)$*-Tupel* $K = (q, u_1, v_1, \ldots, u_k, v_k) \in Q \times (\Gamma^* \times \Gamma^+)^k$ *und besagt, dass*

- q *der momentane Zustand und*
- $u_i v_i \sqcup \sqcup \cdots$ *die Inschrift des* i*-ten Bandes ist, und dass*
- *sich der Kopf auf Band* i *auf dem ersten Zeichen von* v_i *befindet.*

Definition 3. *Eine Konfiguration* $K' = (q', u'_1, v'_1, \ldots, u'_k, v'_k)$ *heißt* ***Folgekonfiguration*** *von* $K = (q, u_1, a_1 v_1, \ldots, u_k, a_k v_k)$ *(kurz:* $K \underset{M}{\longrightarrow} K'$*), falls eine Anweisung*

$$(q, a_1, \ldots, a_k) \mapsto (q', a'_1, D_1, \ldots, a'_k, D_k)$$

in δ *und* $b_1, \ldots, b_k \in \Gamma$ *existieren, so dass für* $i = 1, \ldots, k$ *jeweils eine der folgenden drei Bedingungen gilt:*

1. $D_i = N$, $u'_i = u_i$ *und* $v'_i = a'_i v_i$,
2. $D_i = L$, $u_i = u'_i b_i$ *und* $v'_i = b_i a'_i v_i$,
3. $D_i = R$, $u'_i = u_i a'_i$ *und* $v'_i = \begin{cases} \sqcup, & v_i = \varepsilon, \\ v_i, & \text{sonst}, \end{cases}$

Wir schreiben $K \underset{M}{\longrightarrow}^t K'$, falls Konfigurationen $K_0, \ldots, K_t$ existieren mit $K_0 = K$ und $K_t = K'$, sowie $K_i \underset{M}{\longrightarrow} K_{i+1}$ für $i = 0, \ldots, t-1$. Die reflexive, transitive Hülle von $\underset{M}{\longrightarrow}$ bezeichnen wir mit $\underset{M}{\longrightarrow}^*$, d.h. $K \underset{M}{\longrightarrow}^* K'$ bedeutet, dass ein $t \geq 0$ existiert mit $K \underset{M}{\longrightarrow}^t K'$.

Definition 4. *Sei* $x \in \Sigma^*$ *eine Eingabe. Die zugehörige* ***Startkonfiguration*** *ist*

$$K_x = (q_0, \varepsilon, \triangleright x, \underbrace{\varepsilon, \triangleright, \ldots, \varepsilon, \triangleright}_{(k-1)\text{-mal}}).$$

Definition 5. *Eine Konfiguration* $K = (q, u_1, v_1, \ldots, u_k, v_k)$ *mit* $q \in \{q_h, q_{ja}, q_{nein}\}$ *heißt* ***Endkonfiguration***. *Im Fall* $q = q_{ja}$ *(bzw.* $q = q_{nein}$*) heißt* K ***akzeptierende*** *(bzw.* ***verwerfende****)* ***Endkonfiguration***.

Definition 6.
*Eine DTM M **hält bei Eingabe** $x \in \Sigma^*$ **im Zustand** q (kurz: $M(x)$ hält im Zustand q), falls $q \in \{q_h, q_{ja}, q_{nein}\}$ ist und es eine Endkonfiguration $K = (q, u_1, v_1, \ldots, u_k, v_k)$ gibt mit*

$$K_x \underset{M}{\longrightarrow}^* K.$$

*Weiter definieren wir das **Resultat** $M(x)$ der Rechnung von M bei Eingabe x,*

$$M(x) = \begin{cases} \textit{ja}, & M(x) \text{ hält im Zustand } q_{ja}, \\ \textit{nein}, & M(x) \text{ hält im Zustand } q_{nein}, \\ y, & M(x) \text{ hält im Zustand } q_h, \\ \uparrow \ (\text{undefiniert}), & \text{sonst.} \end{cases}$$

Dabei ergibt sich y aus $u_k v_k$, indem das erste Symbol $\triangleright$ und sämtliche Blanks am Ende entfernt werden, d. h. $u_k v_k = \triangleright y \sqcup^i$. Für $M(x) =$ ja sagen wir auch „$M(x)$ akzeptiert“ und für $M(x) =$ nein „$M(x)$ verwirft“.

Definition 7. *Die von einer* DTM *M **akzeptierte Sprache** ist*

$$L(M) = \{x \in \Sigma^* \mid M(x) \text{ akzeptiert}\}.$$

Eine DTM, die eine Sprache L akzeptiert, darf also bei Eingaben $x \notin L$ unendlich lange rechnen. In diesem Fall heißt L **rekursiv aufzählbar** (oder **semi-entscheidbar**). Dagegen muss eine DTM, die eine Sprache L entscheidet, bei jeder Eingabe halten.

Definition 8. *Sei $L \subseteq \Sigma^*$. Eine* DTM *M **entscheidet** L, falls für alle $x \in \Sigma^*$ gilt:*

$$x \in L \Rightarrow M(x) \text{ akz.}$$
$$x \notin L \Rightarrow M(x) \text{ verw.}$$

*In diesem Fall heißt L **entscheidbar** (oder **rekursiv**).*

Definition 9. *Sei $f : \Sigma^* \to \Sigma^*$ eine Funktion. Eine* DTM *M **berechnet** f, falls für alle $x \in \Sigma^*$ gilt:*

$$M(x) = f(x).$$

*f heißt dann **berechenbar** (oder **rekursiv**).*

Aus dem Grundstudium wissen wir, dass eine nichtleere Sprache $L \subseteq \Sigma^*$ genau dann rekursiv aufzählbar ist, wenn eine rekursive Funktion $f : \Sigma^* \to \Sigma^*$ existiert, deren Bild $range(f) = \{f(x) \mid x \in \Sigma^*\}$ die Sprache L ist.

2.2 Nichtdeterministische Berechnungen

Anders als eine DTM, für die in jeder Konfiguration höchstens eine Anweisung ausführbar ist, hat eine nichtdeterministische Turingmaschine in jedem Rechenschritt die Wahl unter einer endlichen Anzahl von Anweisungen.

Definition 10. *Eine **nichtdeterministische k-Band-Turingmaschine** (kurz k-*NTM*oder einfach* NTM*) ist ein 5-Tupel $M = (Q, \Sigma, \Gamma, \delta, q_0)$, wobei Q, Σ, Γ, q_0 genau wie bei einer k-DTM definiert sind und*

$$\delta : Q \times \Gamma^k \to \mathcal{P}(Q \cup \{q_h, q_{ja} q_{nein}\} \times (\Gamma \times \{R, L, N\})^k)$$

die Eigenschaft hat, dass für $(q', a'_1, D_1, \ldots, a'_k, D_k) \in \delta(q, a_1, \ldots, a_k)$ im Fall $a_i = \triangleright$ immer $a'_i = \triangleright$ und $D_i = R$ gilt.

Die Begriffe **Konfiguration**, **Start-** und **Endkonfiguration** übertragen sich unmittelbar von DTMs auf NTMs. Der Begriff der **Folgekonfiguration** lässt sich übertragen, indem wir $\delta(q, a_1, \ldots, a_k) = (q', a'_1, D_1, \ldots, a'_k, D_k)$ durch $(q', a'_1, D_1, \ldots, a'_k, D_k) \in \delta(q, a_1, \ldots, a_k)$ ersetzen (in beiden Fällen schreiben wir auch oft

$$\delta : (q, a_1, \ldots, a_k) \mapsto (q', a'_1, D_1, \ldots, a'_k, D_k)$$

oder einfach $(q, a_1, \dots, a_k) \mapsto (q', a_1', D_1, \dots, a_k', D_k)$.

Wir werden NTMs nur zum Erkennen von Sprachen (d.h. als Akzeptoren) und nicht zum Berechnen von Funktionen benutzen.

Definition 11.

a) *Sei M eine* NTM. *Wir sagen $M(x)$* ***akzeptiert****, falls $M(x)$ nur endlich lange Rechnungen ausführt und eine akzeptierende Endkonfiguration K existiert mit $K_x \to^* K$.*

b) *Akzeptiert $M(x)$ nicht und hat $M(x)$ nur endlich lange Rechnungen, so* ***verwirft*** *$M(x)$.*

c) *Falls $M(x)$ unendlich lange Rechnungen ausführt, ist $M(x) = \uparrow$ (undefiniert).*

d) *Die von M* ***akzeptierte Sprache*** *ist*

$$L(M) = \{x \in \Sigma^* \mid M(x) \text{ akzeptiert}\}.$$

e) *M* ***entscheidet*** *$L(M)$, falls M alle Eingaben $x \notin L(M)$ verwirft.*

2.3 Zeitkomplexität

Der Zeitverbrauch $time_M(x)$ einer Turingmaschine M bei Eingabe x ist die maximale Anzahl an Rechenschritten, die M ausgehend von der Startkonfiguration K_x ausführen kann (bzw. undefiniert oder ∞, falls unendlich lange Rechnungen existieren).

Definition 12.

a) *Sei M eine TM und sei $x \in \Sigma^*$ eine Eingabe. Dann ist*

$$time_M(x) = \max\{t \geq 0 \mid \exists K : K_x \vdash^t K\}$$

die ***Rechenzeit*** *von M bei Eingabe x, wobei $\max \mathbb{N} = \infty$ ist.*

b) *Sei $t : \mathbb{N} \to \mathbb{N}$ eine monoton wachsende Funktion. Dann ist M $t(n)$****-zeitbeschränkt****, falls für alle $x \in \Sigma^*$ gilt:*

$$time_M(x) \leq t(|x|).$$

Alle Sprachen, die in (nicht-)deterministischer Zeit $t(n)$ entscheidbar sind, fassen wir in den Komplexitätsklassen

$$\mathsf{DTIME}(t(n)) = \{L(M) \mid M \text{ ist eine } t(n)\text{-zeitbeschränkte DTM}\}$$

bzw.

$$\mathsf{NTIME}(t(n)) = \{L(M) \mid M \text{ ist eine } t(n)\text{-zeitbeschränkte NTM}\}$$

zusammen. Ferner sei

$$\mathsf{FTIME}(t(n)) = \left\{ f \,\middle|\, \begin{array}{c} f \text{ wird von einer } t(n)\text{-zeitbe-} \\ \text{schränkten DTM berechnet} \end{array} \right\}.$$

Für eine Klasse F von Funktionen $t : \mathbb{N} \to \mathbb{N}$ sei $\mathsf{DTIME}(F) = \bigcup_{t \in F} \mathsf{DTIME}(f(n))$. $\mathsf{NTIME}(F)$ und $\mathsf{FTIME}(F)$ sind analog definiert. Die Klasse $n^{O(1)} + O(1)$ aller polynomiell beschränkten Funktionen bezeichnen wir mit $\text{poly}(n)$. Die wichtigsten Zeitkomplexitätsklassen sind

$$\begin{aligned}
\mathsf{LINTIME} &= \mathsf{DTIME}(O(n)) &&= \bigcup_{c \geq 1} \mathsf{DTIME}(cn + c) && \text{„Linearzeit“}, \\
\mathsf{P} &= \mathsf{DTIME}(\text{poly}(n)) &&= \bigcup_{c \geq 1} \mathsf{DTIME}(n^c + c) && \text{„Polynomialzeit“}, \\
\mathsf{E} &= \mathsf{DTIME}(2^{O(n)}) &&= \bigcup_{c \geq 1} \mathsf{DTIME}(2^{cn+c}) && \text{„Lineare Exponentialzeit“}, \\
\mathsf{EXP} &= \mathsf{DTIME}(2^{\text{poly}(n)}) &&= \bigcup_{c \geq 1} \mathsf{DTIME}(2^{n^c+c}) && \text{„Exponentialzeit“}.
\end{aligned}$$

Die Klassen $\mathsf{NP}, \mathsf{NE}, \mathsf{NEXP}$ und $\mathsf{FP}, \mathsf{FE}, \mathsf{FEXP}$ sind analog definiert.

2.4 Platzkomplexität

Zur Definition von Platzkomplexitätsklassen verwenden wir so genannte Offline-Turingmaschinen und Transducer. Diese haben die Eigenschaft, dass sie das erste Band nur als Eingabeband (also nur zum Lesen) bzw. das k-te Band nur als Ausgabeband (also nur zum Schreiben) benutzen. Der Grund für diese Einschränkungen liegt darin, sinnvolle Definitionen für Komplexitätsklassen mit einem sublinearen Platzverbrauch zu erhalten.

Definition 13. *Eine TM M heißt **Offline-TM**, falls für jede Anweisung $(q, a_1, \ldots, a_k) \mapsto (q', a_1', D_1, \ldots, a_k', D_k)$ die Implikation*

$$a_1' = a_1 \wedge a_1 = \sqcup \Rightarrow D_1 = L$$

*gilt. Gilt weiterhin immer $D_k \neq L$ und ist M eine DTM, so heißt M **Transducer**.*

Dies bedeutet, dass eine Offline-TM nicht auf das Eingabeband schreiben darf (*read-only*). Beim Transducer dient das letzte Band als Ausgabeband, auch hier können keine Berechnungen durchgeführt werden (*write-only*).

Der Zeitverbrauch $time_M(x)$ von Offline-TMs und von Transducern ist genauso definiert wie bei DTMs. Als nächstes definieren wir den Platzverbrauch einer TM als die maximale Summe aller während einer Rechnung besuchten Bandfelder.

Definition 14.

a) *Sei M eine TM und sei $x \in \Sigma^*$ eine Eingabe mit $time_M(x) < \infty$. Dann ist*

$$space_M(x) = \max\{s \geq 1 \mid \exists K = (q, u_1, v_1, \ldots, u_k, v_k) \text{ mit } K_x \vdash^* K \text{ und } s = \sum_{i=1}^{k} |u_i v_i|\}$$

*der **Platzverbrauch** von M bei Eingabe x. Für eine Offline-TM ersetzen wir $\sum_{i=1}^{k} |u_i v_i|$ durch $\sum_{i=2}^{k} |u_i v_i|$ und für einen Transducer durch $\sum_{i=2}^{k-1} |u_i v_i|$.*

b) *Sei $s : \mathbb{N} \to \mathbb{N}$ eine monoton wachsende Funktion. Dann ist M $s(n)$**-platzbeschränkt**, falls für alle $x \in \Sigma^*$*

$$space_M(x) \leq s(|x|) \text{ und } time_M(x) < \infty$$

gilt, $space_M(x)$ ist undefiniert, falls $time_M(x)$ undefiniert ist.

Alle Sprachen, die in (nicht-) deterministischem Platz $s(n)$ entscheidbar sind, fassen wir in den Komplexitätsklassen

$$\mathsf{DSPACE}(s(n)) = \left\{ L(M) \,\middle|\, \begin{array}{c} M \text{ ist eine } s(n)\text{-platzbe-} \\ \text{schränkte Offline-DTM} \end{array} \right\}$$

bzw.

$$\mathsf{NSPACE}(s(n)) = \left\{ L(M) \,\middle|\, \begin{array}{c} M \text{ ist eine } s(n)\text{-platzbe-} \\ \text{schränkte Offline-NTM} \end{array} \right\}$$

zusammen. Ferner sei

$$\mathsf{FSPACE}(s(n)) = \left\{ f \,\middle|\, \begin{array}{c} f \text{ wird von einer } t(n)\text{-platzbe-} \\ \text{schränkten DTM berechnet} \end{array} \right\}.$$

Die wichtigsten Platzkomplexitätsklassen sind

$$\begin{aligned} \mathsf{L} = \mathsf{LOGSPACE} &= \mathsf{DSPACE}(O(\log n)) \\ \mathsf{L}^c &= \mathsf{DSPACE}(O(\log^c n)) \\ \mathsf{LINSPACE} &= \mathsf{DSPACE}(O(n)) \\ \mathsf{PSPACE} &= \mathsf{DSPACE}(\mathrm{poly}(n)) \\ \mathsf{ESPACE} &= \mathsf{DSPACE}(2^{O(n)}) \\ \mathsf{EXPSPACE} &= \mathsf{DSPACE}(2^{\mathrm{poly}(n)}) \end{aligned}$$

Die Klassen $\mathsf{NL}, \mathsf{NLINSPACE}$ und $\mathsf{NPSPACE}$, sowie $\mathsf{FL}, \mathsf{FLINSPACE}$ und $\mathsf{FPSPACE}$ sind analog definiert, wobei $\mathsf{NPSPACE}$ mit PSPACE zusammenfällt (wie wir bald sehen werden).

3 Grundlegende Beziehungen

In diesem Kapitel leiten wir die wichtigsten Inklusionsbeziehungen zwischen deterministischen und nichtdeterministischen Platz- und Zeitkomplexitätsklassen her. Zuerst befassen wir uns jedoch mit Robustheitseigenschaften dieser Klassen.

3.1 Robustheit von Komplexitätsklassen

Wir zeigen zuerst, dass platzbeschränkte TMs nur ein Arbeitsband benötigen.

Lemma 15 (Bandreduktion)**.**
Sei $M = (Q, \Sigma, \Gamma, \delta, q_0)$ *eine* $s(n)$*-platzbeschränkte Offline*-DTM. *Dann ex. eine* $s(n)$*-platzbeschränkte Offline*-2-DTM M' *mit* $L(M') = L(M)$.

Beweis. Betrachte die Offline-2-DTM $M' = (Q', \Sigma, \Gamma', \delta', q'_0)$ mit $\Gamma' = \Gamma \cup (\Gamma \cup \hat{\Gamma})^{k-1}$, wobei $\hat{\Gamma}$ für jedes $a \in \Gamma$ die markierte Variante $\hat{a}$ enthält. M' hat dasselbe Eingabeband wie M, speichert aber die Inhalte von $(k-1)$ übereinander liegenden Feldern der Arbeitsbänder von M auf einem Feld ihres Arbeitsbandes. Zur Speicherung der Kopfpositionen von M werden Markierungen benutzt.

Initialisierung: In den ersten beiden Rechenschritten erzeugt M' auf ihrem Arbeitsband (Band 2) $k-1$ Spuren, die jeweils mit dem markierten Anfangszeichen $\hat{\triangleright}$ initialisiert werden:

$$K_x = (q'_0, \varepsilon, \triangleright x, \varepsilon, \triangleright) \xrightarrow[M']{} (q'_1, \triangleright, x, \triangleright, \sqcup) \xrightarrow[M']{} (q'_2, \varepsilon, \triangleright x, \triangleright, \begin{pmatrix} \hat{\triangleright} \\ \vdots \\ \hat{\triangleright} \end{pmatrix})$$

Simulation: M' simuliert einen Rechenschritt von M, indem sie den Kopf auf dem Arbeitsband soweit nach rechts bewegt, bis sie alle $(k-1)$ markierten Zeichen $a_2, \ldots, a_k$ gefunden hat. Diese speichert sie neben dem aktuellen Zustand q von M in ihrem Zustand. Während M' den Kopf wieder nach links bewegt, führt M' folgende Aktionen durch: Ist a_1 das von M' (und von M) gelesene Eingabezeichen und ist $\delta(q, a_1, a_2, \ldots, a_k) = (q', a_1, D_1, a'_2, D_2, \ldots, a'_k, D_k)$, so bewegt M' den Eingabekopf gemäß D_1, ersetzt auf dem Arbeitsband die markierten Zeichen a_i durch a'_i und verschiebt deren Marken gemäß D_i, $i = 2, \ldots, k$.

Akzeptanzverhalten: M' akzeptiert genau dann, wenn M akzeptiert.

Offenbar gilt nun $L(M') = L(M)$ und $\mathit{space}_{M'}(x) \leq \mathit{space}_M(x)$. □

In den Übungen wird gezeigt, dass die Sprache der Palindrome durch eine 2-DTM zwar in Linearzeit entscheidbar ist, eine 1-DTM hierzu jedoch Zeit $\Omega(n^2)$ benötigt. Tatsächlich lässt sich jede $t(n)$-zeitbeschränkte k-DTM M von einer 1-DTM M' in Zeit $O(t(n)^2)$ simulieren. Bei Verwendung einer 2-DTM ist die Simulation sogar in Zeit $O(t(n) \log t(n))$ durchführbar (siehe Übungen). Als nächstes wenden wir uns wichtigen Robustheitseigenschaften von Platz- und Zeitkomplexitätsklassen zu.

Satz 16 (Lineare Platzkompression und Beschleunigung)**.**
Für alle $c > 0$ *gilt*

i) $\mathsf{DSPACE}(s(n)) \subseteq \mathsf{DSPACE}(2 + cs(n))$, *(lin. space compression)*

ii) $\mathsf{DTIME}(t(n)) \subseteq \mathsf{DTIME}(2 + n + c \cdot t(n))$. *(linear speedup)*

Beweis. i) Sei $L \in \mathsf{DSPACE}(s(n))$ und sei $M = (Q, \Sigma, \Gamma, \delta, q_0)$ eine $s(n)$-platzbeschränkte Offline-k-DTM mit $L(M) = L$. Nach vorigem Lemma können wir $k = 2$ annehmen. O.B.d.A. sei $c < 1$. Wähle $m = \lceil 1/c \rceil$ und betrachte die Offline-2-DTM

$$M' = (Q \times \{1, \ldots, m\}, \Sigma, \Gamma \cup \Gamma^m, \delta', (q_0, m))$$

mit

$$\delta'((q,i),a,b) = \begin{cases} ((q',1),a,D_1,\triangleright,R), \\ \qquad \text{falls } b=\triangleright \text{ und } \delta(q,a,\triangleright)=(q',a,D_1,\triangleright,R), \\ ((q',j),a,D_1,(b_1,\ldots,b_{i-1},b'_i,b_{i+1},\ldots,b_m),D'_2), \\ \qquad \text{falls } [b=(b_1,\ldots,b_m) \text{ oder } b=\sqcup=b_1= \\ \qquad \ldots=b_m] \text{ und } \delta(q,a,b_i)=(q',a,D_1,b'_i,D_2), \end{cases}$$

wobei

$$j = \begin{cases} i, & D_2=N \\ i+1, & D_2=R, i<m \\ 1, & D_2=R, i=m \\ m, & D_2=L, i=1 \\ i-1, & D_2=L, i>1 \end{cases} \quad \text{und} \quad D'_2 = \begin{cases} L, & D_2=L, i=1 \\ R, & D_2=R, i=m \\ N, & \text{sonst} \end{cases}$$

ist. Identifizieren wir die Zustände (q_{ja}, i) mit q_{ja} und (q_{nein}, i) mit q_{nein}, so ist leicht zu sehen, dass $L(M') = L(M) = L$ gilt. Zudem gilt

$$\begin{aligned} \mathit{space}_{M'} &\le 1 + \lceil (\mathit{space}_M(x) - 1)/m \rceil \\ &\le 2 + \mathit{space}_M(x)/m \\ &\le 2 + c \cdot \mathit{space}_M(x) \qquad \text{(wegen } m = \lceil 1/c \rceil \ge 1/c). \end{aligned}$$

ii) Sei $L \in \mathsf{DTIME}(t(n))$ und sei $M = (Q, \Sigma, \Gamma, \delta, q_0)$ eine $t(n)$-zeitbeschränkte k-DTM mit $L(M) = L$, wobei wir $k \ge 2$ annehmen. Wir konstruieren eine DTM M' mit $L(M') = L$ und $\mathit{time}_{M'}(x) \le 2 + |x| + c \cdot \mathit{time}_M(x)$. M' verwendet das Alphabet $\Gamma' = \Gamma \cup \Gamma^m$ mit $m = \lceil 8/c \rceil$ und simuliert M wie folgt.

Initialisierung: M' kopiert die Eingabe $x = x_1 \ldots x_n$ in Blockform auf das zweite Band. Hierzu fasst M' je m Zeichen von x zu einem Block $(x_{im+1}, \ldots, x_{(i+1)m})$, $i = 0, \ldots, l = \lceil n/m \rceil - 1$, zusammen, wobei der letzte Block $(x_{lm+1}, \ldots, x_n, \sqcup, \ldots, \sqcup)$ mit $(l+1)m - n$ Blanks auf die Länge m gebracht wird. Sobald M' das erste Blank hinter der Eingabe x erreicht, ersetzt sie dieses durch das Zeichen $\triangleright$, d.h. das erste Band von M' ist nun mit $\triangleright x \triangleright$ und das zweite Band mit

$$\triangleright(x_1, \ldots, x_m) \ldots (x_{(l-1)m+1}, \ldots, x_{lm})(x_{lm+1}, \ldots, x_n, \sqcup, \ldots, \sqcup)$$

beschriftet. Hierzu benötigt M' genau $n+2$ Schritte. In weiteren $l + 1 = \lceil n/m \rceil$ Schritten kehrt M' an den Beginn des 2. Bandes zurück. Von nun an benutzt M' das erste Band als Arbeitsband und das zweite als Eingabeband.

Simulation: M' simuliert jeweils eine Folge von m Schritten von M in 6 Schritten:

M' merkt sich in ihrem Zustand den Zustand q von M vor Ausführung dieser Folge und die aktuellen Kopfpositionen $i_j \in \{1, \ldots, m\}$ von M innerhalb der gerade gelesenen Blöcke auf den Bändern $j = 1, \ldots, k$. Die ersten 4 Schritte verwendet M', um die beiden Nachbarblöcke auf jedem Band zu erfassen $(LRRL)$. Mit dieser Information kann M' die nächsten m Schritte von M vorausberechnen und die entsprechende Konfiguration in 2 weiteren Schritten herstellen.

Akzeptanzverhalten: M' akzeptiert genau dann, wenn M dies tut.

Es ist klar, dass $L(M') = L$ ist. Zudem gilt für jede Eingabe x der Länge $|x| = n$

$$\begin{aligned} \mathit{time}_{M'}(x) &\le n + 2 + \lceil n/m \rceil + 6 \lceil t(n)/m \rceil \\ &\le n + 2 + 7 \lceil t(n)/m \rceil \\ &\le n + 2 + 7ct(n)/8 + 7 \\ &\le n + 2 + ct(n), \text{ falls } c \cdot t(n)/8 \ge 7. \end{aligned}$$

Da das Ergebnis der Rechnung von $M(x)$ im Fall $t(n) < 56/c$ nur von konstant vielen Eingabezeichen abhängt, kann M' diese Eingaben schon während der Initialisierungsphase (durch table-lookup) in Zeit $n + 2$ entscheiden. □

Korollar 17. *Seien s, t monotone Funktionen. Dann gilt:*

i) $s(n) \geq 4 \Rightarrow \mathsf{DSPACE}(O(s(n))) = \mathsf{DSPACE}(s(n))$,

ii) $n + 2 \leq t(n) \notin O(n) \Rightarrow \mathsf{DTIME}(O(t(n))) = \mathsf{DTIME}(t(n))$,

iii) $\mathsf{DTIME}(O(n)) = \bigcap_{\varepsilon>0} \mathsf{DTIME}((1+\varepsilon)n + 2)$.

Beweis. *i)* Sei $L \in \mathsf{DSPACE}(cs(n))$ für eine Konstante $c \geq 0$. Nach vorigem Satz existiert für $c' = 1/2c$ eine Offline-k-DTM M, die L in Platz $2 + c'cs(n) = 2 + s(n)/2 \leq s(n)$ entscheidet.

ii) Sei $L \in \mathsf{DTIME}(ct(n))$ für eine Konstante $c \geq 0$. Nach vorigem Satz existiert für $c' = 1/2c$ eine DTM M, die L in Zeit $2 + n + c'ct(n) = 2 + n + t(n)/2$ entscheidet. Wegen $t(n) \notin O(n)$ existieren nur endlich viele Eingaben x mit $t(|x|) \leq 4|x| + 4$. Diese lassen sich durch einen parallel laufenden DFA in Zeit $|x| + 2$ entscheiden.

iii) Sei $L \in \mathsf{DTIME}(cn)$ für eine Konstante $c \geq 0$. Nach vorigem Satz existiert für $c' = \varepsilon/c$ eine DTM M, die L in Zeit $2+n+c'cn = 2+n+\varepsilon n$ entscheidet. □

3.2 Deterministische Simulationen von nichtdeterministischen Berechnungen

In diesem Abschnitt betrachten wir möglichst platz- und zeiteffiziente deterministische Simulationen von nichtdeterministischen Berechnungen.

Satz 18 (Beziehungen zwischen det. und nichtdet. Zeit- und Platzklassen)**.**

i) $\mathsf{NTIME}(t(n)) \subseteq \mathsf{DSPACE}(O(t(n)))$,

ii) $\mathsf{NSPACE}(s(n)) \subseteq \mathsf{DTIME}(2^{O(s(n)+\log n)})$.

Beweis. *i)* Sei $L \in \mathsf{NTIME}(t(n))$ und sei $N = (Q, \Sigma, \Gamma, \Delta, q_0)$ eine k-NTM, die L in Zeit $t(n)$ entscheidet. Weiter sei

$$d = \max_{(q,\vec{a})\in Q\times\Gamma^k} \|\delta(q, \vec{a})\|$$

der maximale Verzweigungsgrad von N. Dann ist jede Rechnung

$$K_x = K_0 \underset{N}{\rightarrow} K_1 \underset{N}{\rightarrow} \dots \underset{N}{\rightarrow} K_t$$

der Länge t von $N(x)$ eindeutig durch eine Folge $(d_1, \dots, d_t) \in \{1, \dots, d\}^t$ beschreibbar. Um N zu simulieren, generiert M auf dem Band 2 für $t = 1, 2, \dots$ der Reihe nach alle Folgen $(d_1, \dots, d_t) \in \{1, \dots, d\}^t$. Für jede solche Folge kopiert M die Eingabe auf Band 3 und simuliert die zugehörige Rechnung von $N(x)$ auf den Bändern 3 bis $k + 2$. M akzeptiert, sobald N bei einer dieser Simulationen in den Zustand q_{ja} gelangt. Wird dagegen ein t erreicht, für das alle d^t Simulationen von N im Zustand q_{nein} oder q_{h} enden, so verwirft M. Nun ist leicht zu sehen, dass $L(M) = L(N)$ und der Platzverbrauch von M durch

$$\mathit{space}_M(x) \;\leq\; \mathit{time}_N(x) + \mathit{space}_N(x) \;\leq\; (k + 1)(\mathit{time}_N(x) + 1)$$

beschränkt ist.

ii) Sei $L \in \mathsf{NSPACE}(s(n))$ und sei $N = (Q, \Sigma, \Gamma, \delta, q_0)$ eine Offline-2-NTM, die L in Platz $s(n)$ entscheidet. Fixieren wir die Eingabe x und begrenzen wir den Platzverbrauch von N durch s, so kann N

- die Köpfe des Eingabe- bzw. Arbeitsbandes auf höchstens $n + 2$ (wobei $n = |x|$) bzw. s verschiedenen Bandfeldern positionieren,
- das Arbeitsband mit höchstens $\|\Gamma\|^s$ verschiedenen Beschriftungen versehen und
- höchstens $\|Q\|$ verschiedene Zustände annehmen.

D.h. ausgehend von der Startkonfiguration K_x kann N in Platz s höchstens

$$(n + 2)s\|\Gamma\|^s\|Q\| \leq c^{s+\log n}$$

verschiedene Konfigurationen erreichen, wobei c eine von N abhängige Konstante ist. Um N zu simulieren, testet M für $s = 1, 2, \ldots$, ob $N(x)$ in Platz $\leq s$ eine akzeptierende Endkonfiguration erreichen kann. Ist dies der Fall, akzeptiert M. Erreicht dagegen s einen Wert, so dass $N(x)$ keine Konfiguration der Größe s erreichen kann, verwirft M. Hierzu muss M für $s = 1, 2, \ldots, s(n)$ jeweils zwei Instanzen des Erreichbarkeitsproblems REACH in einem gerichteten Graphen mit $c^{s+\log n}$ Knoten lösen, was in Zeit $2s(n)(c^{s(n)+\log n})^{O(1)} = 2^{O(s(n)+\log n)}$ möglich ist. □

Korollar 19. $s(n) \geq \log n \Rightarrow \mathsf{NSPACE}(s(n)) \subseteq \mathsf{DTIME}(2^{O(s(n))})$.

Es gilt somit für jede monotone Funktion $s(n) \geq \log n$,

$$\mathsf{DSPACE}(s) \subseteq \mathsf{NSPACE}(s) \subseteq \mathsf{DTIME}(2^{O(s)})$$

und für jede monotone Funktion $t(n) \geq n + 2$,

$$\mathsf{DTIME}(t) \subseteq \mathsf{NTIME}(t) \subseteq \mathsf{DSPACE}(t).$$

Insbesondere erhalten wir somit die Inklusionskette

$$\begin{aligned} \mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{NPSPACE} \\ \subseteq \mathsf{EXP} \subseteq \mathsf{NEXP} \subseteq \mathsf{EXPSPACE} \subseteq \ldots \end{aligned}$$

Des weiteren impliziert Satz 16 für $t(n) \geq n + 2$ und $s(n) \geq \log n$ die beiden Inklusionen

$$\mathsf{NTIME}(t) \subseteq \mathsf{DTIME}(2^{O(t)}) \text{ und } \mathsf{NSPACE}(s) \subseteq \mathsf{DSPACE}(2^{O(s)}),$$

wovon sich letztere noch erheblich verbessern lässt, wie wir im nächsten Abschnitt sehen werden.

3.3 Der Satz von Savitch

Praktisch relevante Komplexitätsklassen werden durch Zeit- und Platzschranken $t(n)$ und $s(n)$ definiert, die sich mit relativ geringem Aufwand berechnen lassen.

Definition 20. *Eine monotone Funktion $f : \mathbb{N} \to \mathbb{N}$ heißt* ***echte*** *(engl. proper)* ***Komplexitätsfunktion****, falls es einen Transducer M gibt mit*

- $M(x) = \triangleright^{f(|x|)}$,
- $space_M(x) = O(f(|x|))$ *und*
- $time_M(x) = O(f(|x|) + |x|)$.

Beispiele für echte Komplexitätsfunktionen sind k, $\lceil \log n \rceil$, $\lceil \log^k n \rceil$, $\lceil n \cdot \log n \rceil$, $n^k + k$, 2^n, $n! \cdot \lfloor \sqrt{n} \rfloor$ (siehe Übungen).

Satz 21 (Savitch, 1970)**.**
Für jede echte Komplexitätsfunktion $s(n) \geq \log n$ gilt

$$\mathsf{NSPACE}(s) \subseteq \mathsf{DSPACE}(s^2).$$

Beweis. Sei $L \in \mathsf{NSPACE}(s)$ und sei N eine Offline-2-NTM, die L in Platz $s(n)$ entscheidet. Wie im Beweis von Satz 18 gezeigt, kann N bei einer Eingabe x der Länge n höchstens $c^{s(n)}$ verschiedene Konfigurationen einnehmen. Daher muss im Fall $x \in L$ eine akzeptierende Rechnung der Länge $\leq c^{s(n)}$ existieren. Zudem können wir annehmen, dass $N(x)$ köchstens eine akzeptierende Endkonfiguration $\hat{K}_x$ erreichen kann.

Sei $K_1, \ldots, K_{c^{s(n)}}$ eine Aufzählung aller Konfigurationen von $N(x)$ die Platz höchstens $s(n)$ benötigen. Dann ist leicht zu sehen, dass für je zwei solche Konfigurationen K, $\hat{K}$ und jede Zahl i folgende Äquivalenz gilt:

$$K \xrightarrow[N]{}^{\leq 2^i} \hat{K} \Leftrightarrow \exists K_j : K \xrightarrow[N]{}^{\leq 2^{i-1}} K_j \wedge K_j \xrightarrow[N]{}^{\leq 2^{i-1}} \hat{K}.$$

Nun können wir $N(x)$ durch folgende Offline-3-DTM $M(x)$ simulieren.

Initialisierung: $M(x)$ schreibt das Tripel $(K_x, \hat{K}_x, \lceil s(|n|)\log c\rceil)$ auf das 2. Band, wobei für das Eingabeband nur die Kopfposition, nicht jedoch die Beschriftung notiert wird (also $K_x = (q_0, 1, \varepsilon, \triangleright)$ und $\hat{K}_x = (q_{\text{ja}}, 2, \triangleright, \sqcup^{s(n)})$). Während der Simulation wird auf dem 2. Band ein Keller (*stack*) von Tripeln der Form $(K, \hat{K}, i)$ implementiert, die jeweils für die Frage stehen, ob $K \xrightarrow[N]{}^{\leq 2^i} \hat{K}$ gilt. Zur Beantwortung dieser Frage arbeitet M den Stack wie folgt ab, wobei das 3. Band zum Kopieren von Tripeln auf dem 2. Band und zur Berechnung von K_{j+1} aus K_j benutzt wird.

Simulation: Sei $(K, \hat{K}, i)$ das am weitesten rechts auf dem 2. Band stehende Tripel (also das oberste Kellerelement).

In den Fällen $K = \hat{K}$ und $i = 0$ testet M direkt, ob $K \xrightarrow[N]{}^{\leq 1} \hat{K}$ gilt und gibt die Antwort zurück.

Andernfalls fügt M für wachsendes $j = 1, 2, \ldots$ das Tripel $(K, K_j, i-1)$ hinzu und berechnet (rekursiv) die Antwort für diese Tripel.

Ist diese negativ, so wird das Tripel $(K, K_j, i-1)$ durch das nächste Tripel $(K, K_{j+1}, i-1)$ ersetzt (solange $j < c^{s(n)}$ ist, andernfalls erfährt das Tripel $(K, \hat{K}, i)$ eine negative Antwort).

Ist die Antwort auf das Tripel $(K, K_j, i-1)$ dagegen positiv, so ersetzt M das Tripel $(K, K_j, i-1)$ durch das Tripel $(K_j, \hat{K}, i-1)$ und berechnet die zugehörige Antwort. Bei einer negativen Antwort fährt M mit dem nächsten Tripel $(K, K_{j+1}, i-1)$ fort. Bei einer positiven Antwort erhält dagegen das Tripel $(K, \hat{K}, i)$ eine positive Antwort.

Akzeptanzverhalten: M akzeptiert, falls die Antwort auf das Starttripel $(K_x, \hat{K}_x, \lceil s(|n|)\log c\rceil)$ positiv ist.

Da sich auf dem 2. Band zu jedem Zeitpunkt höchstens $\lceil s(|n|)\log c\rceil$ Tripel befinden und jedes Tripel $O(s(|x|))$ Platz benötigt, besucht M nur $O(s^2(|x|))$ Felder. □

Korollar 22.

i) $\mathsf{NL} \subseteq \mathsf{L}^2$,

ii) $\mathsf{NPSPACE} = \bigcup_{k>0} \mathsf{NSPACE}(n^k) \subseteq \bigcup_{k>0} \mathsf{DSPACE}(n^{2k}) = \mathsf{PSPACE}$,

iii) $\mathsf{NPSPACE}$ *ist unter Komplement abgeschlossen,*

iv) $\mathsf{CSL} = \mathsf{NSPACE}(n) \subseteq \mathsf{DSPACE}(n^2) \cap \mathsf{E}$.

Eine weitere Folgerung aus dem Satz von Savitch ist, dass das Komplement $\overline{L}$ einer Sprache $L \in \mathsf{NSPACE}(s)$ in $\mathsf{DSPACE}(s^2)$ und somit auch in $\mathsf{NSPACE}(s^2)$ liegt. Wir werden gleich sehen, dass $\overline{L}$ sogar in $\mathsf{NSPACE}(s)$ liegt, d.h. die nichtdeterministischen Platzklassen $\mathsf{NSPACE}(s)$ sind unter Komplementbildung abgeschlossen.

3.4 Der Satz von Immerman und Szelepcsényi

Definition 23.

a) Für eine Sprache $L \in \Sigma^*$ *bezeichne* $\overline{L} = \Sigma^* - L$ *das* ***Komplement*** *von* L.

b) Für eine Sprachklasse $\mathcal{C}$ *bezeichne* $\text{co-}\mathcal{C} = \{\overline{L} | L \in \mathcal{C}\}$ *die zu* $\mathcal{C}$ ***komplementäre Sprachklasse****.*

Beispiel 24.

1) *Die zu* NP *komplementäre Klasse ist* $\text{co-}\mathsf{NP} = \{L | \overline{L} \in \mathsf{NP}\}$*. Ein Beispiel für ein* co-NP*-Problem ist* TAUT*:*

Gegeben: *Eine boolsche Formel* F *über* n *Variablen* $x_1, \ldots, x_n$*.*

Gefragt: *Ist* F *eine Tautologie, d. h. gilt* $f(\vec{a}) = 1$ *für alle Belegungen* $\vec{a} \in \{0,1\}^n$*?*

Die Frage ob NP *unter Komplementbildung abgeschlossen ist (d.h., ob* $\mathsf{NP} = \text{co-}\mathsf{NP}$ *gilt), ist ähnlich wie das* $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$*-Problem ungelöst.*

2) *Wir wir gesehen haben, impliziert der Satz von Savitch den Abschluss von* $\mathsf{NPSPACE}$ *unter Komplementbildung.*
3) *Dagegen wurde die Frage ob die Klasse* $\mathsf{CSL} = \mathsf{NSPACE}(n)$ *der kontextsensitiven Sprachen unter Komplementbildung abgeschlossen ist, erst in den 80ern gelöst (siehe Satz von Immerman und Szelepcsényi), d.h. es gilt* $\mathsf{CSL} = \text{co-}\mathsf{CSL}$.
4) *Andererseits ist* $\text{co-}\mathsf{CFL} \neq \mathsf{CFL}$. *Dies folgt aus der Tatsache, dass kontextfreie Sprachen zwar unter Vereinigung abgeschlossen sind, aber nicht unter Schnitt.* ◁

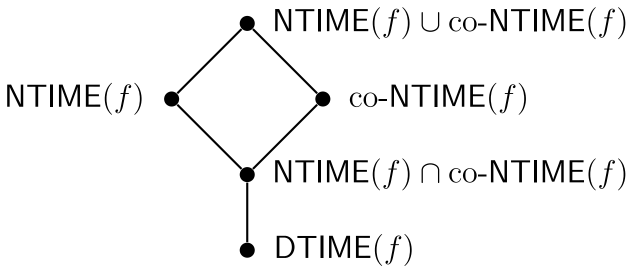
Da sich deterministische Rechnungen leicht komplementieren lassen (durch einfaches Vertauschen der Zustände q_{ja} und q_{nein}), sind deterministische Komplexitätsklassen unter Komplementbildung abgeschlossen.

Proposition 25.

i) $\text{co-}\mathsf{DSPACE}(s(n)) = \mathsf{DSPACE}(s(n))$,

ii) $\text{co-}\mathsf{DTIME}(t(n)) = \mathsf{DTIME}(t(n))$.

Damit ergibt sich folgende Inklusionsstruktur:



Dagegen lassen sich nichtdeterministische Berechnungen nicht ohne weiteres komplementieren; es sei denn, man fordert gewisse Zusatzeigenschaften.

Definition 26. *Eine* NTM N *heißt* ***strong*** *bei Eingabe* x, *falls es entweder akzeptierende oder verwerfende Rechnungen bei Eingabe* x *gibt (aber nicht beides zugleich).*

Satz 27 (Immerman und Szelepcsényi, 1987)**.**
Für jede echte Komplexitätsfunktion $s(n) \geq \log n$ *gilt*

$$\mathsf{NSPACE}(s) = \text{co-}\mathsf{NSPACE}(s)).$$

Beweis. Sei $L \in \mathsf{NSPACE}(s)$ und sei N eine $s(n)$-platzbeschränkte Offline-NTM mit $L(N) = L$. Wir konstruieren eine $O(s(n))$-platzbeschränkte Offline-NTM N' mit $L(N') = L$, die bei allen Eingaben strong ist. Hierzu zeigen wir zuerst, dass die Frage, ob $N(x)$ eine Konfiguration K in höchstens t Schritten erreichen kann, durch eine $O(s(n))$-platzbeschränkte Offline-NTM N_0 entscheidbar ist, die bei Kenntnis der Anzahl

$$r(x, t-1) = \|\{K | K_x \xrightarrow[N]{}^{\leq t-1} K\}\|$$

aller in höchstens $t-1$ Schritten erreichbaren Konfigurationen strong ist. Sei

$$L_0 = \{(x, r, t, K) | K_x \xrightarrow[N]{}^{\leq t} K\}.$$

Behauptung 28. *Es existiert eine* $O(s(n))$*-platzbeschränkte Offline-*NTM N_0 *mit* $L(N_0) = L_0$, *die auf allen Eingaben der Form* $(x, r(x, t-1), t, K)$ *strong ist.*

Beweis der Behauptung. $N_0(x, r, t, K)$ benutzt einen mit dem Wert 0 initialisierten Zähler c und rät der Reihe nach für jede Konfiguration K_i, die Platz $\leq s(|x|)$ benötigt, eine Rechnung von $N(x)$ der Länge $\leq t-1$, die in K_i endet. Falls dies gelingt, erhöht N_0 den Zähler c um 1 und testet, ob $K_i \xrightarrow[N]{}^{\leq 1} K$ gilt. Falls ja, so hält N_0 im Zustand q_{ja}. Nachdem N_0 alle Konfigurationen K_i durchlaufen hat, hält N_0 im Zustand q_{nein}, wenn c den Wert r hat, andernfalls im Zustand q_{h}.

Pseudocode für $N_0(x, r, t, K)$

```
c := 0
for each Konfiguration K_i do
  rate eine Rechnung α der Laenge ≤ t − 1 von N(x)
  if α endet in K_i then
    c := c + 1
    if K_i →_N^{≤1} K then
      halte im Zustand q_ja
if c = r then
  halte im Zustand q_nein
else
  halte im Zustand q_h
```

Da N_0 genau dann eine akzeptierende Rechnung hat, wenn eine Konfiguration K_i mit $K_x \xrightarrow[N]{}^{\le t-1} K_i$ und $K_i \xrightarrow[N]{}^{\le 1} K$ existiert, ist klar, dass N_0 die Sprache L_0 entscheidet. Da N_0 zudem $O(s(n))$-platzbeschränkt ist, bleibt nur noch zu zeigen, dass N_0 bei Eingaben der Form $x_0 = (x, r(x, t-1), t, K)$ strong ist, also $N_0(x_0)$ genau im Fall $x_0 \notin L_0$ eine verwerfende Endkonfiguration erreichen kann.

Um bei Eingabe x_0 eine verwerfende Endkonfiguration zu erreichen, muss N_0 $r = r(x, t-1)$ Konfigurationen K_i finden, für die zwar $K_x \xrightarrow[N]{}^{\le t-1} K_i$ aber nicht $K_i \xrightarrow[N]{}^{\le 1} K$ gilt. Dies bedeutet jedoch, dass K von keiner der $r(x, t-1)$ in $t-1$ Schritten erreichbaren Konfigurationen in einem Schritt erreichbar ist und somit x_0 tatsächlich nicht zu L_0 gehört. Die Umkehrung folgt analog. □

Betrachte nun folgende NTM N', die für $t = 1, 2, \ldots$ die Anzahl $r(x, t)$ der in höchstens t Schritten erreichbaren Konfigurationen in der Variablen r berechnet (diese Technik wird induktives Zählen, engl. *inductive counting*, genannt) und mit Hilfe dieser Anzahlen im Fall $x \notin L$ verifiziert, dass keine der erreichbaren Konfigurationen akzeptierend ist.

Pseudocode für $N'(x)$

```
1  t := 0
2  r := 1
3  repeat
4    t := t + 1
5    r⁻ := r
6    r := 0
7    for each Konfiguration K_i do
8      simuliere N_0(x, r⁻, t, K_i)
9      if N_0 akzeptiert then
10       r := r + 1
11       if K_i ist akzeptierende Endkonfiguration then
12         halte im Zustand q_ja
13     if N_0 haelt im Zustand q_h then
14       halte im Zustand q_h
15 until (r = r⁻)
16 halte im Zustand q_nein
```

Behauptung 29. *Im t-ten Durchlauf der repeat-Schleife wird r^- in Zeile 5 auf den Wert $r(x, t-1)$ gesetzt. Folglich wird N_0 von N' in Zeile 8 nur mit Eingaben der Form $(x, r(x, t-1), t, K_i)$ aufgerufen.*

Beweis der Behauptung. Wir führen Induktion über t:

$t = 1$: Im ersten Durchlauf der repeat-Schleife erhält r^- den Wert $1 = r(x, 0)$.

$t \rightsquigarrow t + 1$: Da r^- zu Beginn des $t+1$-ten Durchlaufs auf den Wert von r gesetzt wird, müssen wir zeigen, dass r im t-ten Durchlauf auf $r(x, t)$ hochgezählt wird. Nach Induktionsvoraussetzung wird N_0 im t-ten Durchlauf nur mit Eingaben der Form $(x, r(x, t-1), t, K_i)$ aufgerufen. Da N_0 wegen Beh. 1 auf all diesen Eingaben strong ist und keine dieser Simulationen im Zustand q_h endet (andernfalls würde N' sofort stoppen), werden alle in $\le t$ Schritten erreichbaren Konfigurationen K_i als solche erkannt und somit wird r tatsächlich auf den Wert $r(x,t)$ hochgezählt. $\square$

Behauptung 30. *Bei Beendigung der repeat-Schleife in Zeile 15 gilt* $r = r^- = \|\{K | K_x \underset{N}{\rightarrow}^* K\}\|$.

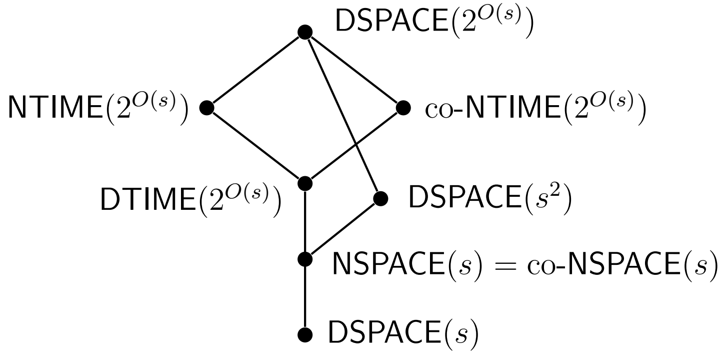
Beweis der Behauptung. Wir wissen bereits, dass im t-ten Durchlauf der repeat-Schleife r den Wert $r(x,t)$ und r^- den Wert $r(x,t-1)$ erhält. Wird daher die repeat-Schleife nach t_e Durchläufen verlassen, so gilt $r = r^- = r(x,t_e) = r(x,t_e-1)$.
Angenommen $r(x,t_e) < \|\{K|K_x \underset{N}{\rightarrow}^* K\}\|$. Dann gibt es eine Konfiguration K, die für ein $t' > t_e$ in t' Schritten, aber nicht in t_e Schritten erreichbar ist. Betrachte eine Rechnung $K_x = K_0 \underset{N}{\rightarrow} K_1 \underset{N}{\rightarrow} \ldots \underset{N}{\rightarrow} K_{t'} = K$ minimaler Länge, die in K endet. Dann gilt $K_x \underset{N}{\rightarrow}^{t_e} K_{t_e}$, aber nicht $K_x \underset{N}{\rightarrow}^{\leq t_e - 1} K_{t_e}$ und daher folgt $r(x,t_e) > r(x,t_e-1)$. Widerspruch! $\square$

Da N' offenbar die Sprache L in Platz $O(s(n))$ entscheidet, bleibt nur noch zu zeigen, dass N' bei allen Eingaben strong ist. Wegen Behauptung 30 hat $N'(x)$ genau dann eine verwerfende Rechnung, wenn im letzten Durchlauf der repeat-Schleife alle erreichbaren Konfigurationen K als solche erkannt werden und darunter keine akzeptierende Endkonfiguration ist. Dies impliziert $x \notin L$. Umgekehrt ist leicht zu sehen, dass $N'(x)$ im Fall $x \notin L$ eine verwerfende Rechnung hat. $\square$

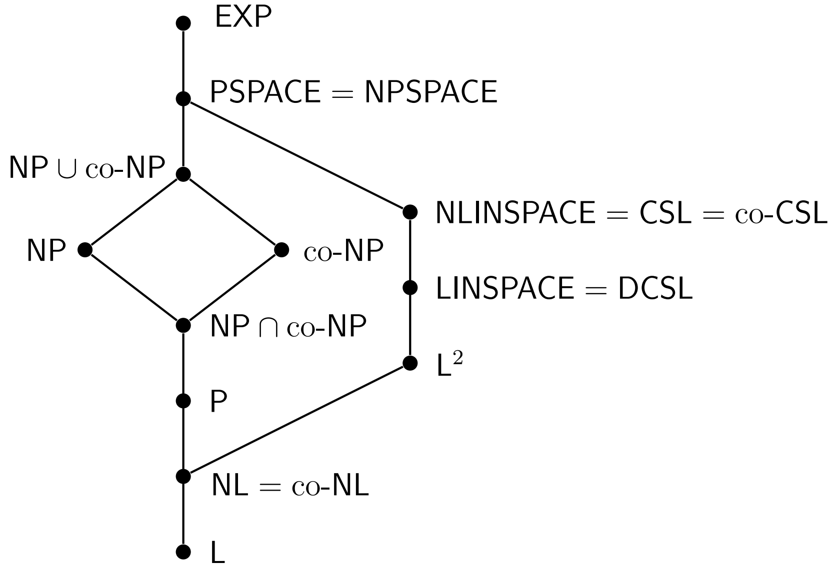
Korollar 31.

1. $\mathsf{NL} = \text{co-}\mathsf{NL}$,
2. $\mathsf{CSL} = \mathsf{NLINSPACE} = \text{co-}\mathsf{CSL}$.

Damit ergibt sich folgende Inklusionsstruktur für (nicht)deterministische Platz- und Zeitklassen:



Angewandt auf die wichtigsten bisher betrachteten Komplexitätsklassen erhalten wir folgende Inklusionsstruktur:



Eine zentrale Fragestellung der Komplexitätstheorie ist, welche dieser Inklusionen echt sind. Dieser Frage gehen wir im nächsten Kapitel nach.

4 Hierarchiesätze

4.1 Diagonalisierung und die Unentscheidbarkeit des Halteproblems

Wir benutzen folgende Kodierung (Gödelisierung) von 1-DTMs $M = (Q, \Sigma, \Gamma, \delta, q_0)$. O.B.d.A. sei $Q = \{q_0, q_1, \ldots, q_m\}$, $\{0, 1, \#\} \subseteq \Sigma$ und $\Gamma = \{a_1, \ldots, a_l\}$ (also z.B. $a_1 = \sqcup$, $a_2 = \triangleright$, $a_3 = 0$, $a_4 = 1$ etc.). Dann kodieren wir jedes $\alpha \in Q \cup \Gamma \cup \{q_{\text{h}}, q_{\text{ja}}, q_{\text{nein}}, L, R, N\}$ wie folgt durch eine Binärzahl $c(\alpha)$ der Länge $b = \lceil \log_2(\|Q\| + \|\Gamma\| + 6) \rceil = \lceil \log_2(m + l + 7) \rceil$:

α	$c(\alpha)$
q_i, $i = 0, \ldots, m$	$bin_b(i)$
a_j, $j = 1, \ldots, l$	$bin_b(m + j)$
$q_{\text{h}}, q_{\text{ja}}, q_{\text{nein}}, L, R, N$	$bin_b(m + l + 1), \ldots, bin_b(m + l + 6)$

M wird nun durch eine Folge von Binärzahlen, die durch # getrennt sind, kodiert:

$$\begin{array}{c} c(q_0)\#c(a_1)\#c(p_{0,1})\#c(b_{0,1})\#c(D_{0,1})\# \\ c(q_0)\#c(a_2)\#c(p_{0,2})\#c(b_{0,2})\#c(D_{0,2})\# \\ \vdots \\ c(q_m)\#c(a_l)\#c(p_{m,l})\#c(b_{m,l})\#c(D_{m,l})\# \end{array}$$

wobei

$$\delta(q_i, a_j) = (p_{i,j}, b_{i,j}, D_{i,j})$$

für $i = 1, \ldots, m$ und $j = 1, \ldots, l$ ist. Kodieren wir die Zeichen $0, 1, \#$ binär (z.B. $0 \mapsto 00$, $1 \mapsto 11$, $\# \mapsto 10$), so gelangen wir zu einer Binärkodierung von M. Diese Kodierung lässt sich auch auf k-DTM's und k-NTM's erweitern. Die Kodierung einer TM M bezeichnen wir mit $\langle M \rangle$. Ein Paar (M, x) bestehend aus einer TM M und einer Eingabe $x \in \{0, 1\}^*$ kodieren wir durch das Wort $\langle M, x \rangle = \langle M \rangle \# x$.

Definition 32. *Das **Halteproblem** ist*

$$H = \{\langle M, x \rangle | M \textit{ ist eine DTM, die bei Eingabe } x \textit{ hält}\}.$$

Satz 33. *H ist rekursiv aufzählbar, aber nicht entscheidbar.*

Beweis. Es ist klar, dass H rekursiv aufzählbar ist, da es eine (universelle) TM U gibt, die bei Eingabe $\langle M, x \rangle$ die Berechnung von $M(x)$ simuliert und genau dann akzeptiert, wenn $M(x)$ hält.
Unter der Annahme, dass H entscheidbar ist, ist auch die Sprache

$$D = \{\langle M \rangle | M \text{ ist eine DTM, die die Eingabe } \langle M \rangle \text{ verwirft}\} \quad (*)$$

entscheidbar. Sei also M_d eine Turingmaschine, die D entscheidet,

$$L(M_d) = D \quad (**).$$

Dann verhält sich M_d „komplementär" zur Diagonalen der Matrix, deren Eintrag in Zeile M und Spalte $\langle M \rangle$ das Resultat von $M(\langle M \rangle)$ angibt.

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$
M_1	**ja**	$\uparrow$	nein	nein	$\cdots$
M_2	nein	$\boldsymbol{\uparrow}$	nein	$\uparrow$	$\cdots$
M_3	ja	$\uparrow$	**nein**	$\uparrow$	$\cdots$
M_4	$\uparrow$	nein	$\uparrow$	**ja**	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$
M_d	**nein**	**nein**	**ja**	**nein**	$\cdots$

Folglich kann keine Zeile dieser Matrix mit M_d übereinstimmen:

$$\begin{array}{lclcl} \langle M_d\rangle \in D & \overset{(*)}{\Rightarrow} & M_d(\langle M_d\rangle) = \text{nein} & \overset{(**)}{\Rightarrow} & \langle M_d\rangle \notin D \ \lightning \\ \langle M_d\rangle \notin D & \overset{(*)}{\Rightarrow} & M_d(\langle M_d\rangle) \neq \text{nein} & \overset{(**)}{\Rightarrow} & \langle M_d\rangle \in D \ \lightning \end{array}$$

□

Satz 34. *Für jede rekursive Funktion $f : \mathbb{N} \longrightarrow \mathbb{N}$ existiert eine rekursive Sprache $D_f \notin \mathsf{DTIME}(f(n))$.*

Beweis. Wir definieren

$$D_f = \{\langle M\rangle \mid M(\langle M\rangle) \text{ verwirft nach } \leq f(|\langle M\rangle|) \text{ Schritten}\} \quad (*)$$

Offensichtlich ist D_f entscheidbar. Unter der Annahme, dass $D_f \in \mathsf{DTIME}(f(n))$ ist, existiert eine $f(n)$-zeitbeschränkte DTM M_d, die D_f entscheidet, d.h.

$$L(M_d) = D \quad (**)$$

Dies führt jedoch auf einen Widerspruch:

$$\begin{array}{lclcl} \langle M_d\rangle \in D_f & \overset{(*)}{\Rightarrow} & M_d(\langle M_d\rangle) \text{ verw.} & \overset{(**)}{\Rightarrow} & \langle M_d\rangle \notin D_f \ \lightning \\ \langle M_d\rangle \notin D_f & \overset{(*,**)}{\Rightarrow} & M_d(\langle M_d\rangle) \text{ akz.} & \overset{(**)}{\Rightarrow} & \langle M_d\rangle \in D_f \ \lightning \end{array}$$

□

Eine interessante Frage ist nun, wieviel Zeit eine DTM benötigt um die Sprache D_f zu entscheiden. Im nächsten Abschnitt werden wir sehen, dass D_f i.a. sehr hohe Komplexität haben kann.

4.2 Das Gap-Theorem

Satz 35 (Gap-Theorem)**.**
Es gibt eine rekursive Funktion $f : \mathbb{N} \to \mathbb{N}$ mit

$$\mathsf{DTIME}(2^{f(n)}) = \mathsf{DTIME}(f(n)).$$

Beweis. Wir definieren $f(n) \geq n + 2$ so, dass für jede DTM M gilt:

$$\forall x \in \{0,1\}^* : \mathit{time}_M(x) \leq 2^{f(|x|)} \Rightarrow$$

$$\text{für fast alle } x \in \{0,1\}^* : \mathit{time}_M(x) \leq f(|x|).$$

Betrachte hierzu das Prädikat

$$P(k,t) : t \geq k + 2 \text{ und für } i = 1, \ldots, k \text{ gilt:}$$

$$\forall x \in \{0,1\}^k : \mathit{time}_{M_i}(x) \notin [t + 1, 2^t].$$

Da für jedes n alle $t \geq \max\{\mathit{time}_{M_i}(x) < \infty | 1 \leq i \leq n, x \in \{0,1\}^n\}$ das Prädikat $P(n,t)$ erfüllen, können wir nun $f(n)$ wie folgt induktiv definieren:

$$f(n) = \begin{cases} 2, & n = 0, \\ \min\{t \geq f(n-1) + n | P(n,t)\}, & n > 0. \end{cases}$$

Da P entscheidbar ist, ist f rekursiv. Um zu zeigen, dass jede Sprache $L \in \mathsf{DTIME}(2^{f(n)})$ bereits in $\mathsf{DTIME}(f(n))$ enthalten ist, sei M_k eine beliebige $2^{f(n)}$-zeitbeschränkte DTM mit $L(M_k) = L$. Dann muss M_k alle Eingaben $x \in \{0,1\}^*$ mit $|x| \geq k$ in Zeit $\mathit{time}_{M_k}(x) \leq f(n)$ ($n = |x|$) entscheiden, da andernfalls $P(n, f(n))$ verletzt wäre. Folglich ist $L \in \mathsf{DTIME}(f(n))$, da die endlich vielen Eingaben x mit $|x| < k$ durch table-lookup in Zeit $|x| + 2$ entscheidbar sind. □

Es ist leicht zu sehen, dass der Beweis des Gap-Theorems für jede rekursive Funktion g eine rekursive Zeitschranke f liefert, so dass $\mathsf{DTIME}(g(f(n))) = \mathsf{DTIME}(f(n))$ ist. Folglich ist D_f nicht in Zeit $g(f(n))$ entscheidbar.

4.3 Zeit- und Platzhierarchiesätze

Wie der folgende Satz zeigt, ist D_f für jede echte Komplexitätsfunktion f mit einem relativ geringen Mehraufwand entscheidbar. Da die Rechenressourcen bei praktisch relevanten Komplexitätsklassen durch eine echte Komplexitätsfunktion f beschränkt sind, lassen sich daher mit Hilfe von D_f die wichtigsten deterministischen Zeitkomplexitätsklassen trennen.

Satz 36. *Falls $f(n) \geq n$ eine echte Komplexitätsfunktion ist, dann gilt*

$$D_f \in \mathsf{DTIME}(nf^2(n)) - \mathsf{DTIME}(f(n)).$$

Beweis. Betrachte folgende 4-DTM M':

Initialisierung: M' überprüft bei Eingabe x zuerst, ob x die Kodierung $\langle M \rangle$ einer k-DTM $M = (Q, \Sigma, \Gamma, \delta, q_0)$ ist. Falls ja, erzeugt M' die Startkonfiguration K_x von M bei Eingabe $x = \langle M \rangle$, wobei sie die Inhalte von k übereinander liegenden Feldern der Bänder von M auf ihrem 2. Band in je einem Block von kb, $b = \lceil \log_2(\|Q\| + \|\Gamma\| + 6) \rceil$, Feldern speichert und den aktuellen Zustand von M und die gerade gelesenen Zeichen auf ihrem 3. Band notiert. Hierfür benötigt M' Zeit $O(kb \cdot |x|) = O(|x|^2)$. Abschließend erzeugt M' auf dem 4. Band den String $1^{f(|x|)}$ in Zeit $O(f(|x|))$.

Simulation: M' simuliert jeden Rechenschritt von M wie folgt: Zunächst inspiziert M' die auf dem 1. Band gespeicherte Kodierung von M, um die durch den Inhalt des 3. Bandes bestimmte Aktion von M zu ermitteln. Diese führt sie sodann auf dem 2. Band aus und aktualisert dabei auf dem 3. Band den Zustand und die gelesenen Zeichen von M. Schließlich vermindert M' noch auf dem 4. Band die Anzahl der Einsen um 1. Insgesamt benötigt M' für die Simulation eines Rechenschrittes von M Zeit $O(k \cdot f(|M|)) = O(|M| \cdot f(|M|)))$.

Akzeptanzverhalten: M' bricht die Simulation ab, sobald M stoppt oder der Zähler auf Band 4 den Wert 0 erreicht. M' hält genau dann im Zustand q_{ja}, wenn die Simulation von M im Zustand q_{nein} endet.

Nun ist leicht zu sehen, dass M' $O(n \cdot f(n)^2)$-zeitbeschränkt ist und die Sprache D_f entscheidet. □

Korollar 37. *(Zeithierarchiesatz)*
Falls $f(n) \geq n$ eine echte Komplexitätsfunktion ist, gilt

$$\mathsf{DTIME}(n \cdot f(n)^2) - \mathsf{DTIME}(f(n)) \neq \emptyset$$

Korollar 38.

$$\mathsf{P} \subsetneq \mathsf{E} \subsetneq \mathsf{EXP}$$

Beweis.

$$\begin{aligned}
\mathsf{P} = \bigcup_{c>0} \mathsf{DTIME}(n^c + c) &\subseteq \mathsf{DTIME}(2^n) \\
&\subsetneq \mathsf{DTIME}(n2^{2n}) \subseteq \mathsf{E} = \bigcup_{c>0} \mathsf{DTIME}(2^{cn}) \subseteq \mathsf{DTIME}(2^{n^2}) \\
&\subsetneq \mathsf{DTIME}(n2^{2n^2}) \subseteq \bigcup_{c>0} \mathsf{DTIME}(2^{n^c+c}) = \mathsf{EXP}
\end{aligned}$$

□

Aus dem Beweis von Satz 36 können wir weiterhin die Existenz einer universellen TM folgern.

Korollar 39. *Es gibt eine universelle 3-DTM U, die bei Eingabe $\langle M, x \rangle$ eine Simulation von M bei Eingabe x durchführt und dasselbe Ergebnis liefert:*

$$U(\langle M, x \rangle) = M(x)$$

Hierbei können wir annehmen, dass U verwirft, falls die Eingabe keine zulässige Kodierung eines Paares (M, x) mit $x \in \Sigma^*$ darstellt.

Bemerkung 40. *Mit Hilfe einer aufwändigeren Simulationstechnik von* k-DTMs *durch eine* 2-DTM *in Zeit* $O(f(n) \cdot \log f(n))$ *lässt sich folgende schärfere Form des Zeithierarchiesatzes beweisen:*

Sei f *eine echte Komplexitätsfunktion und gelte*

$$\liminf_{n\to\infty} \frac{g(n) \cdot \log g(n)}{f(n)} = 0.$$

Dann ist

$$\mathsf{DTIME}(f(n)) \backslash \mathsf{DTIME}(g(n)) \neq \emptyset.$$

Für $g(n) = n^2$ erhalten wir beispielsweise die echten Inklusionen $\mathsf{DTIME}(g(n)) \subsetneq \mathsf{DTIME}(f(n))$ für die Funktionen $f(n) = n^3$, $n^2 \log^2 n$ und $n^2 \log n \log\log n$. In den Übungen zeigen wir, dass die Inklusion

$$\mathsf{DTIME}(n^k) \subsetneq \mathsf{DTIME}(n^k \log^a n)$$

tatsächlich für alle $k \geq 1$ und $a > 0$ echt ist. Für Platzklassen erhalten wir sogar eine noch feinere Hierarchie (siehe Übungen).

Satz 41 (Platzhierarchiesatz)**.** *Sei* f *eine echte Komplexitätsfunktion und gelte*

$$\liminf_{n\to\infty} \frac{g(n)}{f(n)} = 0.$$

Dann ist

$$\mathsf{DSPACE}(f(n)) \backslash \mathsf{DSPACE}(g(n)) \neq \emptyset.$$

Damit lässt sich für Zeitschranken $g(n) \leq f(n)$ die Frage, ob die Inklusion von $\mathsf{DSPACE}(g(n))$ in $\mathsf{DSPACE}(f(n))$ echt ist, eindeutig beantworten: Sie ist genau dann echt, wenn $\liminf_{n\to\infty} g(n)/f(n) = 0$ ist, da andernfalls $f(n) = O(g(n))$ ist und somit beide Klassen gleich sind.

Korollar 42.

$$\mathsf{L} \subsetneq \mathsf{L}^2 \subsetneq \mathsf{LINSPACE} \subseteq \mathsf{NLINSPACE} \subsetneq \mathsf{PSPACE} \subsetneq \mathsf{ESPACE} \subsetneq \mathsf{EXPSPACE}.$$

Durch Kombination der Beweistechnik von Satz 41 mit der Technik von Immerman und Szelepcsényi erhalten wir auch für nichtdeterministische Platzklassen eine sehr fein abgestufte Hierarchie.

Satz 43 (nichtdeterministischer Platzhierarchiesatz)**.** *Sei* f *eine echte Komplexitätsfunktion und gelte*

$$\liminf_{n\to\infty} \frac{g(n)}{f(n)} = 0.$$

Dann ist

$$\mathsf{NSPACE}(f(n)) \backslash \mathsf{NSPACE}(g(n)) \neq \emptyset.$$

Ob sich auch der Zeithierarchiesatz auf nichtdeterministische Klassen übertragen lässt, ist dagegen nicht bekannt. Hier lässt sich jedoch folgender Satz beweisen.

Satz 44 (nichtdeterministischer Zeithierarchiesatz)**.** *Sei* f *eine echte Komplexitätsfunktion und gelte*

$$g(n+1) = o(f(n)).$$

Dann ist

$$\mathsf{NTIME}(g(n)) \subsetneq \mathsf{NTIME}(f(n)).$$

Der nichtdeterministische Zeithierarchiesatz liefert für langsam wachsende Zeitschranken eine feinere Hierarchie als der deterministische Zeithierarchiesatz. So ist $\mathsf{NTIME}(n^k)$ zum Beispiel für jede Funktion h mit $\liminf_{n\to\infty} h(n) = \infty$ echt in der Klasse $\mathsf{NTIME}(n^k h(n))$ enthalten, da $(n+1)^k = O(n^k) = o(n^k h(n))$ ist.

Für schnell wachsende Zeitschranken liefert dagegen der deterministische Zeithierarchiesatz eine feinere Hierarchie. So ist zum Beispiel die Klasse $\mathsf{DTIME}(2^{2^n})$ echt in $\mathsf{DTIME}(h(n)2^n 2^{2^n})$ enthalten, während wir für $\mathsf{NTIME}(2^{2^n})$ nur eine Separierung von der Klasse $\mathsf{NTIME}(h(n)2^{2^{n+1}}) = \mathsf{NTIME}(h(n)2^{2^n}2^{2^n})$ erhalten.

5 Reduktionen

5.1 Logspace-Reduktionen

Oft können wir die Komplexitäten zweier Probleme A und B vergleichen, indem wir die Frage, ob $x \in A$ ist, auf eine Frage der Form $y \in B$ zurückführen. Lässt sich y leicht aus x berechnen, so kann jeder Algorithmus für B in einen Algorithmus für A verwandelt werden, der vergleichbare Komplexität hat.

Definition 45. *Seien A und B Sprachen über einem Alphabet Σ. A ist auf B* ***logspace-reduzierbar*** *(in Zeichen: $A \leq_m^{log} B$ oder einfach $A \leq B$), falls eine Funktion $f \in \mathsf{FL}$ existiert, so dass für alle $x \in \Sigma^*$ gilt,*

$$x \in A \Leftrightarrow f(x) \in B.$$

Lemma 46. $\mathsf{FL} \subseteq \mathsf{FP}$.

Beweis. Sei $f \in \mathsf{FL}$ und sei M ein logarithmisch platzbeschränkter Transducer (kurz: FL-Transducer), der f berechnet. Da M bei einer Eingabe der Länge n nur $O(\log n)$ verschieden Konfigurationen einnehmen kann, ist M dann auch polynomiell zeitbeschränkt. □

Beispiel 47. *Wir reduzieren das Hamiltonkreisproblem auf das Erfüllbarkeitsproblem* Sat *für aussagenlogische Formeln.*

Hamiltonkreisproblem (Ham):

Gegeben: *Ein Graph $G = (V, E)$.*

Gefragt: *Hat G einen Hamiltonkreis?*

Erfüllbarkeitsproblem für boolesche Formeln (Sat):

Gegeben: *Eine boolesche Formel F über n Variablen.*

Gefragt: *Ist F erfüllbar?*

Hierzu benötigen wir eine Funktion $f \in \mathsf{FL}$, die einen Graphen $G = (V, E)$ so in eine Formel $f(G) = F_G$ transformiert, dass F_G genau dann erfüllbar ist, wenn G hamiltonsch ist. Wir konstruieren F_G über den Variablen $x_{1,1}, \ldots, x_{n,n}$, wobei $x_{i,j}$ für die Aussage steht, dass Knoten $j \in V = \{1, \ldots, n\}$ in der Rundreise an i-ter Stelle besucht wird. Betrachte nun folgende Klauseln.

a) *An der i-ten Stelle wird mindestens ein Knoten besucht:*

$$x_{i,1} \vee x_{i,2} \vee \ldots \vee x_{i,n},\ i = 1, \ldots, n.$$

b) *An der i-ten Stelle wird höchstens ein Knoten besucht:*

$$\neg x_{i,j} \vee \neg x_{i,k},\ i = 1, \ldots, n,\ 1 \leq j < k \leq n.$$

c) *Jeder Knoten j wird mindestens einmal besucht:*

$$x_{1,j} \vee \ldots \vee x_{n,j},\ j = 1, \ldots, n.$$

d) *Für $(i, j) \notin E$ wird Knoten j nicht unmittelbar nach Knoten i besucht:*

$$\neg x_{1,i} \vee \neg x_{2,j}, \ldots, \neg x_{n-1,i} \vee \neg x_{n,j}, \neg x_{n,i} \vee \neg x_{1,j},\ (i, j) \notin E.$$

Die Klauseln in a) und b) stellen sicher, dass die Relation $\pi = \{(i, j) \mid x_{i,j} = 1\}$ eine Funktion $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$ ist. Bedingung c) besagt, dass π surjektiv (und damit auch bijektiv) ist, und d) sorgt dafür, dass der durch π beschriebene Kreis entlang der Kanten von G verläuft. Bilden wir daher $F_G(x_{1,1}, \ldots, x_{n,n})$ als Konjunktion dieser

$$n + n\binom{n}{2} + n + n\left[\binom{n}{2} - \|E\|\right] = O(n^3)$$

Klauseln, so ist leicht zu sehen, dass die Reduktionsfunktion f in FL berechenbar ist und G genau dann einen Hamiltonkreis besitzt, wenn F_G erfüllbar ist. ◁

Ein zentraler Begriff in der Komplexitätstheorie ist die Vollständigkeit einer Sprache für eine Komplexitätsklasse.

Definition 48.

*a) Sei $\mathcal{C}$ eine Sprachklasse. Eine Sprache L heißt $\mathcal{C}$-**hart** (bzgl. $\leq$), falls für alle Sprachen $A \in \mathcal{C}$ gilt, $A \leq L$.*

*b) Eine $\mathcal{C}$-harte Sprache, die zur Klasse $\mathcal{C}$ gehört, heißt $\mathcal{C}$-**vollständig**.*

*c) $\mathcal{C}$ heißt **abgeschlossen** unter $\leq$, falls gilt:*

$$B \in \mathcal{C}, A \leq B \Rightarrow A \in \mathcal{C}.$$

Lemma 49.

1. *Die $\leq_m^{log}$-Reduzierbarkeit ist reflexiv und transitiv.*
2. *Die Klassen $\mathsf{L}, \mathsf{NL}, \mathsf{NP}, \text{co-}\mathsf{NP}, \mathsf{PSPACE}, \mathsf{EXP}$ und $\mathsf{EXPSPACE}$ sind unter $\leq$ abgeschlossen.*
3. *Sei L vollständig für eine Klasse $\mathcal{C}$, die unter $\leq$ abgeschlossen ist. Dann gilt*
$$\mathcal{C} = \{A \mid A \leq L\}.$$

Beweis. Siehe Übungen. □

Definition 50. *Ein **boolescher Schaltkreis** c mit n Eingängen ist eine Folge $(g_1, \ldots, g_m)$ von **Gattern***

$$g_l \in \{0, 1, x_1, \ldots, x_n, (\neg, j), (\wedge, j, k), (\vee, j, k)\}$$

mit $1 \leq j, k < l$. Der am Gatter g_l berechnete Wert bei Eingabe $a = a_1 \cdots a_n$ ist induktiv wie folgt definiert.

g_l	0	1	x_i	$(\neg, j)$	$(\wedge, j, k)$	$(\vee, j, k)$
$g_l(a)$	0	1	a_i	$1 - g_j(a)$	$g_j(a)g_k(a)$	$g_j(a) + g_k(a) - g_j(a)g_k(a)$

*Der Schaltkreis c berechnet die boolesche Funktion $c(a) = g_m(a)$. Er heißt **erfüllbar**, wenn es eine Eingabe $a \in \{0,1\}^n$ mit $c(a) = 1$ gibt.*

Bemerkung: Die Anzahl der Eingänge eines Gatters g wird als **Fan-in** von g bezeichnet, die Anzahl der Ausgänge (also die Anzahl der Gatter, die g als Eingabe benutzen) als **Fanout**. Boolesche Formeln entsprechen also den booleschen Schaltkreisen mit (maximalem) Fanout 1 und umgekehrt.

Ähnlich wie bei booleschen Formeln sind auch für Schaltkreise die beiden folgenden Entscheidungsprobleme von Interesse.

Auswertungsproblem für boolesche Schaltkreise (CirVal):

Gegeben: Ein boolescher Schaltkreis c mit n Eingängen und eine Eingabe $a \in \{0,1\}^n$.

Gefragt: Ist der Wert von $c(a)$ gleich 1?

Erfüllbarkeitsproblem für boolesche Schaltkreise (CirSat):

Gegeben: Ein boolescher Schaltkreis c mit n Eingängen.

Gefragt: Ist c erfüllbar?

Im folgenden Beispiel führen wir die Lösung des Erreichbarkeitsproblems in gerichteten Graphen auf die Auswertung von booleschen Schaltkreisen zurück.

Beispiel 51. *Für die Reduktion* Reach $\leq$ CirVal *benötigen wir eine Funktion $f \in \mathsf{FL}$ mit der Eigenschaft, dass für alle Graphen G gilt:*

$$G \in \textsc{Reach} \Leftrightarrow f(G) \in \textsc{CirVal}.$$

Der Schaltkreis $f(G)$ besteht aus den Gattern

$$g_{i,j,k'} \text{ und } h_{i,j,k} \text{ mit } 1 \leq i,j,k \leq n \text{ und } 0 \leq k' \leq n,$$

wobei die Gatter $g_{i,j,0}$ für $1 \leq i,j \leq n$ die booleschen Konstanten

$$g_{i,j,0} = \begin{cases} 1, & i = j \text{ oder } (i,j) \in E, \\ 0, & \text{sonst} \end{cases}$$

sind und für $k = 1, 2, \ldots, n$ gilt,

$$\begin{aligned} h_{i,j,k} &= g_{i,k,k-1} \wedge g_{k,j,k-1}, \\ g_{i,j,k} &= g_{i,j,k-1} \vee h_{i,j,k}. \end{aligned}$$

Dann folgt

$$\begin{aligned} g_{i,j,k} = 1 &\Leftrightarrow \text{es existiert in } G \text{ ein Pfad von } i \text{ nach } j\text{, der nur Zwischenknoten } l \le k \text{ durchläuft,} \\ h_{i,j,k} = 1 &\Leftrightarrow \text{es existiert in } G \text{ ein Pfad von } i \text{ nach } j\text{, der den Knoten } k\text{, aber keinen Knoten } l > k \text{ durchläuft.} \end{aligned}$$

Wählen wir also $g_{1,n,n}$ als Ausgabegatter, so liefert der aus diesen Gattern aufgebaute Schaltkreis c genau dann den Wert 1, wenn es in G einen Weg von Knoten 1 zu Knoten n gibt. Es ist auch leicht zu sehen, dass die Reduktionsfunktion f in FL *berechenbar ist.* ◁

Der in Beispiel 51 konstruierte Schaltkreis hat Tiefe $2n$. In den Übungen werden wir sehen, dass sich Reach auch auf die Auswertung eines Schaltkreises der Tiefe $O(\log^2 n)$ reduzieren lässt. Als nächstes leiten wir Vollständigkeitsresultate für CirVal und CirSat her.

5.2 P-vollständige Probleme und polynomielle Schaltkreiskomplexität

Satz 52. CirVal *ist* P*-vollständig.*

Beweis. Es ist leicht zu sehen, dass CirVal $\in$ P ist. Um zu zeigen, dass CirVal hart für P ist, müssen wir für jede Sprache $L \in$ P eine Funktion $f \in FL$ finden, die L auf CirVal reduziert, d.h. es muss für alle Eingaben x die Äquivalenz $x \in L \Leftrightarrow f(x) \in$ CirVal gelten.

Zu $L \in$ P existiert eine 1-DTM $M = (Q, \Sigma, \Gamma, \delta, q_0)$, die L in Zeit $n^c + c$ entscheidet. Wir beschreiben die Rechnung von $M(x)$, $|x| = n$, durch

eine Tabelle $T = (T_{i,j})$, $(i,j) \in \{1, \ldots, n^c + c\} \times \{1, \ldots, n^c + c + 2\}$, mit

$$T_{i,j} = \begin{cases} (q_i, a_{i,j}), & \text{nach } i \text{ Schritten besucht } M \text{ das } j\text{-te Bandfeld,} \\ a_{i,j}, & \text{sonst,} \end{cases}$$

wobei q_i der Zustand von $M(x)$ nach i Rechenschritten ist und $a_{i,j}$ das nach i Schritten an Position j befindliche Zeichen auf dem Arbeitsband ist. $T = (T_{i,j})$ kodiert also in ihren Zeilen die von $M(x)$ der Reihe nach angenommenen Konfigurationen. Dabei

- überspringen wir jedoch alle Konfigurationen, bei denen sich der Kopf auf dem ersten Bandfeld befindet (zur Erinnerung: In diesem Fall wird der Kopf sofort wieder nach rechts bewegt) und
- behalten die in einem Schritt $i < n^c + c$ erreichte Endkonfiguration bis zum Zeitpunkt $i = n^c + c$ bei.

Da M in $n^c + c$ Schritten nicht das $(n^c + c + 2)$-te Bandfeld erreichen kann, ist $T_{i,1} = \triangleright$ und $T_{i,n^c+c+2} = \sqcup$ für $i = 1, \ldots, n^c + c$. Außerdem nehmen wir an, dass M bei jeder Eingabe x auf dem zweiten Bandfeld auf einem Blank hält, d.h. es gilt

$$x \in L \Leftrightarrow T_{n^c+c,2} = (q_{\text{ja}}, \sqcup).$$

Da T nicht mehr als $\|\Gamma\| + \|Q \times \Gamma\|$ verschiedene Tabelleneinträge besitzt, können wir jeden Eintrag $T_{i,j}$ durch eine Bitfolge $t_{i,j,1} \cdots t_{i,j,m}$ der Länge $m = \lceil \log_2 \|\Gamma\| + \|Q \times \Gamma\| \rceil$ kodieren.

Da der Eintrag $T_{i,j}$ im Fall $i \in \{2, \ldots, n^c+c\}$ und $j \in \{2, \ldots, n^c+c+1\}$ eine Funktion $T_{i,j} = g(T_{i-1,j-1}, T_{i-1,j}, T_{i-1,j+1})$ der drei Einträge $T_{i-1,j-1}$, $T_{i-1,j}$ und $T_{i-1,j+1}$ ist, existieren für $k = 1, \ldots, m$ Schaltkreise c_k mit

$$t_{i,j,k} = c_k(t_{i-1,j-1,1} \cdots t_{i-1,j-1,m}, t_{i-1,j,1} \cdots t_{i-1,j,m}, t_{i-1,j+1,1} \cdots t_{i-1,j+1,m}).$$

Die Reduktionsfunktion f liefert nun bei Eingabe x folgenden Schaltkreis c_x mit 0 Eingängen.

- Für jeden der $n^c+c+2+2(n^c+c-1) = 3(n^c+c)$ Randeinträge $T_{i,j}$ mit $i = 1$ oder $j \in \{1, n^c+c+2\}$ enthält c_x m konstante Gatter $c_{i,j,k} = t_{i,j,k}$, $k = 1, \dots, m$, die diese Einträge kodieren.
- Für jeden der $(n^c + c - 1)(n^c + c)$ übrigen Einträge $T_{i,j}$ enthält c_x für $k = 1, \dots, m$ je eine Kopie $c_{i,j,k}$ von c_k, deren $3m$ Eingänge mit den Ausgängen der Schaltkreise $c_{i-1,j-1,1} \cdots c_{i-1,j-1,m}, c_{i-1,j,1} \cdots c_{i-1,j,m}, c_{i-1,j+1,1} \cdots c_{i-1,j+1,m}$ verdrahtet sind.
- Als Ausgabegatter von c_x fungiert das Gatter $c_{n^c+c,2,1}$, wobei wir annehmen, dass das erste Bit der Kodierung von $(q_{\text{ja}}, \sqcup)$ eine Eins und von $(q_{\text{nein}}, \sqcup)$ eine Null ist.

Nun lässt sich induktiv über $i = 1, \dots, n^c+c$ zeigen, dass die von den Schaltkreisen $c_{i,j,k}$, $j = 1, \dots, n^c+c$, $k = 1, \dots, m$ berechneten Werte die Tabelleneinträge $T_{i,j}$, $j = 1, \dots, n^c+c$, kodieren. Wegen

$$x \in L \Leftrightarrow T_{n^c+c,2} = (q_{\text{ja}}, \sqcup) \Leftrightarrow c_x = 1$$

folgt somit die Korrektheit der Reduktion. Außerdem ist leicht zu sehen, dass f in logarithmischem Platz berechenbar ist, da ein $O(\log n)$-platzbeschränkter Transducer existiert, der bei Eingabe x

- zuerst die $3(n^c+c)$ konstanten Gatter von c_x ausgibt und danach
- die $m(n^c + c - 1)(n^c + c)$ Kopien der Schaltkreise $c_1, \dots, c_k$ erzeugt und diese Kopien richtig verdrahtet. □

Eine leichte Modifikation des Beweises von Satz 52 liefert folgendes Resultat.

Korollar 53. *Sei $L \subseteq \{0,1\}^*$ eine beliebige Sprache in P. Dann existiert eine Funktion $f \in \mathsf{FL}$, die bei Eingabe 1^n einen Schaltkreis c_n mit n Eingängen berechnet, so dass für alle $x \in \{0,1\}^n$ gilt:*

$$x \in L \quad \Leftrightarrow \quad c_n(x) = 1.$$

Die Turingmaschine ist ein **uniformes** Rechenmodell, da alle Instanzen eines Problems von einer einheitlichen Maschine entschieden werden. Im Gegensatz hierzu stellen Schaltkreise ein **nichtuniformes** Berechnungsmodell dar, da für jede Eingabegröße n ein anderer Schaltkreis c_n verwendet wird. Um im Schaltkreis-Modell eine unendliche Sprache entscheiden zu können, wird also eine unendliche Folge c_n, $n \geq 0$, von Schaltkreisen benötigt. Probleme, für die Schaltkreisfamilien polynomieller Größe existieren, werden zur Klasse PSK zusammengefasst.

Definition 54.

a) *Eine Sprache L über dem Binäralphabet $\{0,1\}$ hat **polynomielle Schaltkreiskomplexität** (kurz: $L \in \mathsf{PSK}$), falls es eine Folge von booleschen Schaltkreisen c_n, $n \geq 0$, mit n Eingängen und $n^{O(1)} + O(1)$ Gattern gibt, so dass für alle $x \in \{0,1\}^*$ gilt:*

$$x \in L \Leftrightarrow c_{|x|}(x) = 1.$$

b) *Eine Sprache L über einem Alphabet $\Sigma = \{a_0, \dots, a_{k-1}\}$ hat **polynomielle Schaltkreiskomplexität** (kurz: $L \in \mathsf{PSK}$), falls die Binärkodierung*

$$bin(L) = \{bin(x_1) \cdots bin(x_n) \mid x_1 \cdots x_n \in L\}$$

von L polynomielle Schaltkreiskomplexität hat. Hierbei kodieren wir a_i durch die m-stellige Binärdarstellung $bin(i) \in \{0,1\}^m$ von i, wobei $m = \max\{1, \lceil \log_2 k \rceil\}$ ist.

Korollar 55 (Savage 1972)**.**
Es gilt $\mathsf{P} \subseteq \mathsf{PSK}$.

Ob auch alle NP-Sprachen polynomielle Schaltkreiskomplexität haben, ist ein berühmtes offenes Problem. Gelingt es nämlich, für ein NP-Problem superpolynomielle untere Schranken für die Schaltkreisgröße zu zeigen, so folgt mit dem Resultat von Savage $\mathsf{P} \neq \mathsf{NP}$. Wir werden später sehen, dass auch die Annahme $\mathsf{NP} \subseteq \mathsf{PSK}$ schwerwiegende Konsequenzen hat. Selbst für EXP ist die Inklusion in PSK offen. Dagegen zeigt ein einfaches Diagonalisierungsargument, dass in EXPSPACE Sprachen mit superpolynomieller Schaltkreiskomplexität existieren.

Zudem ist nicht schwer zu sehen, dass die Inklusion $\mathsf{P} \subseteq \mathsf{PSK}$ echt ist. Hierzu betrachten wir Sprachen über einem einelementigen Alphabet.

Definition 56. *Eine Sprache $T \subseteq \{0,1\}^*$ heißt **tally** (kurz: $T \in$ Tally), falls jedes Wort $x \in T$ die Form $x = 1^n$ hat.*

Es ist leicht zu sehen, dass alle tally Sprachen polynomielle Schaltkreiskomplexität haben.

Proposition 57. $\mathsf{Tally} \subseteq \mathsf{PSK}$.

Andererseits wissen wir aus der Berechenbarkeitstheorie, dass es tally Sprachen T gibt, die nicht einmal rekursiv aufzählbar sind (etwa wenn T das Komplement des Halteproblems unär kodiert). Folglich sind in PSK beliebig schwierige Sprachen (im Sinne der Berechenbarkeit) enthalten.

Korollar 58. $\mathsf{PSK} \not\subseteq \mathsf{RE}$.

5.3 NP-vollständige Probleme

Wir wenden uns nun der NP-Vollständigkeit von CirSat zu. Hierbei wird sich folgende Charakterisierung von NP als nützlich erweisen.

Definition 59. *Sei $B \subseteq \Sigma^*$ eine Sprache und sei p ein Polynom.*

*a) B heißt **p-balanciert**, falls B nur Strings der Form $x\#y$ mit $|y| = p(|x|)$ enthält.*

b) Die Sprache $\exists B$ ist definiert durch

$$\exists B = \{x \in \Sigma^* \mid \exists y \in \{0,1\}^* : x\#y \in B\}.$$

c) Für eine Sprachklasse $\mathcal{C}$ sei

$$\exists \mathcal{C} = \{\exists B \mid B \in \mathcal{C} \textit{ ist polynomiell balanciert}\}.$$

Jeder String $y \in \{0,1\}^*$ mit $x\#y \in B$ wird auch als **Zeuge** (engl. *witness, certificate*) für die Zugehörigkeit von x zu $\exists B$ bezeichnet.

Satz 60 (Zeugen-Charakterisierung von NP)**.**
Es gilt $\mathsf{NP} = \exists\mathsf{P}$.

Beweis. Zu jeder NP-Sprache $A \subseteq \Sigma^*$ existiert eine NTM M, die A in Zeit $p(n)$ für ein Polynom p entscheidet. Dabei können wir annehmen, dass jede Konfiguration höchstens zwei Folgekonfigurationen hat, die entsprechend der zugehörigen Anweisungen angeordnet sind. Folglich lässt sich jede Rechnung von $M(x)$ durch einen Binärstring y der Länge $p(n)$ eindeutig beschreiben. Das Ergebnis der durch y beschriebenen Rechnung von $M(x)$ bezeichnen wir mit $M_y(x)$. Nun ist leicht zu sehen, dass

$$B = \{x\#y \mid |y| = p(|x|) \text{ und } M_y(x) = \text{ja}\}$$

eine p-balancierte Sprache in P mit $L = \exists B$ ist.

Gilt umgekehrt $A = \exists B$ für eine p-balancierte Sprache $B \in \mathsf{P}$, dann kann A in Polynomialzeit durch eine NTM M entschieden werden, die bei Eingabe x einen String $y \in \{0,1\}^{p(|x|)}$ geeigneter Länge rät und testet, ob $x\#y \in B$ ist. Diese Vorgehensweise von nichtdeterministischen Algorithmen wird im Englischen auch als *"guess and verify"* bezeichnet. $\square$

Satz 61. CirSat *ist* NP*-vollständig.*

Beweis. Es ist leicht zu sehen, dass CirSat $\in$ NP ist. Um zu zeigen, dass CirSat hart für NP ist, müssen wir für jede Sprache $L \in \mathsf{NP}$ eine Funktion $f \in FL$ finden, die L auf CirSat reduziert, d.h. es muss für alle Eingaben x die Äquivalenz $x \in L \Leftrightarrow f(x) \in$ CirSat gelten.

Im Beweis von Satz 60 haben wir gezeigt, dass für jede NP-Sprache $A \subseteq \Sigma^*$ eine p-balancierte Sprache $B \in \mathsf{P}$ mit $A = \exists B$ existiert,

$$x \in A \Leftrightarrow \exists y \in \{0,1\}^{p(|x|)} : x\#y \in B.$$

Sei $m = \lceil \log_2 \|\Sigma \cup \{\#\}\| \rceil$. Da B in P entscheidbar ist, existiert nach Korollar 53 eine FL-Funktion f, die für 1^n einen Schaltkreis c_n mit $m(n+1)+p(n)$ Eingängen berechnet, so dass für alle $x \in \Sigma^*$, $x = x_1 \cdots x_n$, und $y \in \{0,1\}^{p(n)}$ gilt:

$$x\#y \in B \Leftrightarrow c_n(bin_m(x_1)\cdots bin_m(x_n)bin_m(\#)y) = 1.$$

Betrachte nun die Funktion g, die bei Eingabe x den Schaltkreis c_x ausgibt, der sich aus c_n dadurch ergibt, dass die ersten $m(n+1)$ Input-Gatter durch konstante Gatter mit den durch $bin_m(x_1)\cdots bin_m(x_n)bin_m(\#)$ vorgegebenen Werten ersetzt werden. Dann ist auch g in FL berechenbar und es gilt für alle Eingaben x, $|x| = n$,

$$\begin{aligned} x \in A \quad &\Leftrightarrow \quad \exists y \in \{0,1\}^{p(n)} : x\#y \in B \\ &\Leftrightarrow \quad \exists y \in \{0,1\}^{p(n)} : c_n(bin_m(x_1)\cdots bin_m(x_n)bin_m(\#)y) = 1 \\ &\Leftrightarrow \quad \exists y \in \{0,1\}^{p(n)} : c_x(y) = 1 \\ &\Leftrightarrow \quad c_x \in \textsc{CirSat}. \end{aligned}$$

□

Als nächstes zeigen wir, dass auch Sat NP-vollständig ist, indem wir CirSat auf Sat reduzieren. Tatsächlich können wir CirSat sogar auf ein Teilproblem von Sat reduzieren.

Definition 62. *Eine Boolesche Formel F über den Variablen $x_1,\ldots,x_n$ ist in* ***konjunktiver Normalform*** *(kurz KNF), falls F eine Konjunktion*

$$F = \bigwedge_{i=1}^{m} C_i$$

von Disjunktionen $C_i = \bigvee_{j=1}^{k_i} l_{ij}$ von ***Literalen*** *$l_{ij} \in \{x_1,\ldots,x_n,\bar{x}_1,\ldots,\bar{x}_n\}$ ist. Hierbei verwenden wir $\bar{x}$ als abkürzende Schreibweise für $\neg x$. Gilt $k_i \le k$ für $i = 1,\ldots,m$, so heißt F in* ***k-KNF****.*

Eine Disjunktion $C = \bigvee_{j=1}^{k} l_j$ von Literalen wird auch als **Klausel** bezeichnet. Klauseln werden meist als Menge $C = \{l_1,\ldots,l_k\}$ der zugehörigen Literale und KNF-Formeln als Menge $F = \{C_1,\ldots,C_m\}$ ihrer Klauseln dargestellt.

Erfüllbarkeitsproblem für *k*-KNF Formeln (*k*-Sat):

Gegeben: Eine Boolesche Formel in k-KNF.

Gefragt: Ist F erfüllbar?

Beispiel 63. *Die Formel $F = (x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee x_3) \wedge (x_2 \vee \bar{x}_3 \vee x_4)$ ist in 3-KNF und lässt sich in Mengennotation durch $F = \{\{x_1,\bar{x}_2\},\{\bar{x}_1,x_3\},\{x_2,\bar{x}_3,x_4\}\}$ beschreiben. F ist offensichtlich erfüllbar, da in jeder Klausel ein positives Literal vorkommt.* ◁

Satz 64. 3-Sat *ist* NP*-vollständig.*

Beweis. Es ist leicht zu sehen, dass 3-Sat $\in \mathsf{NP}$ ist. Um 3-Sat als hart für NP nachzuweisen, reicht es aufgrund der Transitivität von $\le$ CirSat auf 3-Sat zu reduzieren.

Idee: Wir transformieren einen Schaltkreis $c = \{g_1,\ldots,g_m\}$ mit n Eingängen in eine 3-KNF-Formel F_c mit $n+m$ Variablen $x_1,\ldots,x_n,y_1,\ldots,y_m$, wobei y_i den Wert des Gatters g_i widergibt.

Konkret enthält F_c für jedes Gatter g_i folgende Klauseln:

Gatter g_i	zugeh. Klauseln	Semantik
0	$\{\bar{y}_i\}$	$y_i = 0$
1	$\{y_i\}$	$y_i = 1$
x_j	$\{\bar{y}_i, x_j\}, \{\bar{x}_j, y_i\}$	$y_i \leftrightarrow x_j$
$(\neg, j)$	$\{\bar{y}_i, \bar{y}_j\}, \{y_j, y_i\}$	$y_i \leftrightarrow \bar{y}_j$
$(\wedge, j, k)$	$\{\bar{y}_i, y_j\}, \{\bar{y}_i, y_k\}, \{\bar{y}_j, \bar{y}_k, y_i\}$	$y_i \leftrightarrow y_j \wedge y_k$
$(\vee, j, k)$	$\{\bar{y}_j, y_i\}, \{\bar{y}_k, y_i\}, \{\bar{y}_i, y_j, y_k\}$	$y_i \leftrightarrow y_j \vee y_k$

Außerdem fügen wir noch die Klausel $\{y_m\}$ zu F_c hinzu. Nun ist leicht zu sehen, dass für alle $x \in \{0,1\}^n$ die Äquivalenz

$$c(x) = 1 \Leftrightarrow \exists y \in \{0,1\}^m : F_c(x, y) = 1$$

gilt. Dies bedeutet jedoch, dass der Schaltkreis c und die 3-KNF-Formel F_c erfüllbarkeitsäquivalent sind, d.h.

$$c \in \textsc{CirSat} \Leftrightarrow F_c \in 3\text{-}\textsc{Sat}.$$

Zudem ist leicht zu sehen, dass die Reduktion $c \mapsto F_c$ in FL berechenbar ist. □

3-Sat ist also nicht in Polynomialzeit entscheidbar, außer wenn $\mathsf{P} = \mathsf{NP}$ ist. Am Ende dieses Abschnitts werden wir sehen, dass dagegen 2-Sat effizient entscheidbar ist. Zunächst betrachten wir folgende Variante von 3-Sat.

Not-All-Equal-SAT (NaeSat):

Gegeben: Eine Formel F in 3-KNF.

Gefragt: Existiert eine Belegung für F, unter der in jeder Klausel beide Wahrheitswerte angenommen werden?

Satz 65. NaeSat $\in \mathsf{NPC}$.

Beweis. NaeSat $\in \mathsf{NP}$ ist klar. Wir zeigen CirSat $\leq$ NaeSat durch eine leichte Modifikation der Reduktion $C(x_1, \ldots, x_n) \mapsto F_c(x_1, \ldots, x_n, y_1, \ldots, y_m)$ von CirSat auf 3-Sat:

Sei $F'_c(x_1, \ldots, x_n, y_1, \ldots, y_m, z)$ die 3-KNF Formel, die aus F_c dadurch entsteht, dass wir zu jeder Klausel mit ≤ 2 Literalen die neue Variable z hinzufügen.

Dann ist die Reduktion $f : c \mapsto F'_c$ in FL berechenbar. Es bleibt also nur noch die Korrektheit von f zu zeigen, d.h.

$$c \in \textsc{CirSat} \Leftrightarrow F'_c \in \textsc{NaeSat}.$$

Ist $c = (g_1, \ldots, g_m) \in$ CirSat, so existiert eine Eingabe $x \in \{0,1\}^n$ mit $c(x) = 1$. Wir betrachten die Belegung $a = xyz \in \{0,1\}^{n+m+1}$ mit $y = y_1 \ldots y_m$, wobei $y_i = g_i(x)$ und $z = 0$. Da $F_c(xy) = 1$ ist, enthält jede Klausel von F_c (und damit auch von F'_c) mindestens ein wahres Literal. Wegen $z = 0$ müssen wir nur noch zeigen, dass nicht alle Literale in den Dreierklauseln von F_c unter a wahr werden. Da a jedoch für jedes oder-Gatter $g_i = (\vee, j, k)$ die drei Klauseln

$$\{\bar{y}_i, y_j, y_k\}, \{\bar{y}_j, y_i\}, \{\bar{y}_k, y_i\}$$

und für jedes und-Gatter $g_i = (\wedge, j, k)$ die drei Klauseln

$$\{y_i, \bar{y}_j, \bar{y}_k\}, \{y_j, \bar{y}_i\}, \{y_k, \bar{y}_j\}$$

erfüllt, kann weder $y_i = 0$ und $y_j = y_k = 1$ noch $y_i = 1$ und $y_j = y_k = 0$ gelten, da im ersten Fall die Klausel $\{\bar{y}_j, y_i\}$ und im zweiten Fall die Klausel $\{y_j, \bar{y}_i\}$ falsch wäre.

Ist umgekehrt $F'_c \in$ NaeSat, so existiert eine Belegung $xyz \in \{0,1\}^{n+m+1}$ unter der in jeder Klausel von F'_c beide Wahrheitswerte vorkommen. Da dies dann auch auf die Belegung $\bar{x}\bar{y}\bar{z}$ zutrifft, können wir $z = 0$ annehmen. Dann erfüllt aber die Belegung xy die Formel F_c. □

Definition 66. *Sei $G = (V, E)$ ein ungerichteter Graph.*

a) Eine Menge $C \subseteq V$ heißt ***Clique*** *in G, falls für alle $u, v \in C$ mit $u \neq v$ gilt: $\{u, v\} \in E$.*

b) $I \subseteq V$ heißt ***unabhängig*** *(oder* ***stabil****), falls für alle $u, v \in I$ gilt: $\{u, v\} \notin E$.*

c) $K \subseteq V$ heißt ***Knotenüberdeckung****, falls für alle $e \in E$ gilt: $e \cap K \neq \emptyset$.*

Für einen gegebenen Graphen G und eine Zahl k betrachten wir die folgenden Fragestellungen:

CLIQUE: Besitzt G eine Clique der Größe k?

INDEPENDENT SET (IS): Besitzt G eine stabile Menge der Größe k?

NODE COVER (NC): Besitzt G eine Knotenüberdeckung der Größe k?

Satz 67. IS *ist* NP*-vollständig.*

Beweis. Wir reduzieren 3-SAT auf IS. Sei

$$F = \{C_1, \dots, C_m\} \text{ mit } C_i = \{l_{i,1}, \dots, l_{i,k_i}\} \text{ und } k_i \leq 3 \text{ für } i = 1, \dots, m$$

eine 3-KNF-Formel über den Variablen $x_1, \dots, x_n$. Betrachte den Graphen $G = (V, E)$ mit

$$\begin{aligned} V &= \{v_{ij} \mid 1 \leq i \leq m, 1 \leq j \leq k_i\} \\ E &= \{\{v_{ij}, v_{ij'}\} \mid 1 \leq i \leq m, 1 \leq j < j' \leq k_i\} \\ &\quad \cup \{\{v_{s,t}, v_{u,v}\} \mid l_{st} \text{ und } l_{uv} \text{ sind komplementär}\}. \end{aligned}$$

Dabei heißen zwei Literale **komplementär**, wenn das eine die Negation des anderen ist. Nun gilt

$$\begin{aligned} F \in \text{3-SAT} &\Leftrightarrow \text{es gibt eine Belegung, die in jeder Klausel } C_i \text{ mindestens ein Literal wahr macht} \\ &\Leftrightarrow \text{es gibt } m \text{ Literale } l_{1,j_1}, \dots, l_{m,j_m}\text{, die paarweise nicht komplementär sind} \\ &\Leftrightarrow \text{es gibt } m \text{ Knoten } v_{1,j_1}, \dots, v_{m,j_m}\text{, die nicht durch Kanten verbunden sind} \\ &\Leftrightarrow G \text{ besitzt eine stabile Knotenmenge der Größe } m. \end{aligned}$$

□

Korollar 68. CLIQUE *ist* NP*-vollständig.*

Beweis. Es ist leicht zu sehen, dass jede Clique in einem Graphen $G = (V, E)$ eine stabile Menge in dem zu G komplementären Graphen $\bar{G} = (V, E')$ mit $E' = \binom{V}{2} \setminus E$ ist und umgekehrt. Daher lässt sich IS mittels der Reduktionsfunktion

$$f : (G, k) \mapsto (\bar{G}, k)$$

auf CLIQUE reduzieren. □

Korollar 69. NC *ist* NP*-vollständig.*

Beweis. Offensichtlich ist eine Menge I genau dann stabil, wenn ihr Komplement $V \setminus I$ eine Knotenüberdeckung ist. Daher lässt sich IS mittels der Reduktionsfunktion

$$f : (G, k) \mapsto (G, n - k)$$

auf NC reduzieren, wobei $n = \|V\|$ die Anzahl der Knoten in G ist. □

5.4 NL-vollständige Probleme

In diesem Abschnitt präsentieren wir einen effizienten Algorithmus für das 2-Sat-Problem.

Satz 70. 2-Sat $\in$ NL.

Beweis. Sei F eine 2-KNF-Formel über den Variablen $x_1, \ldots, x_n$. Betrachte den Graphen $G = (V, E)$ mit

$$V = \{x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n\},$$

der für jede Zweierklausel $\{l_1, l_2\}$ von F die beiden Kanten $(\bar{l}_1, l_2)$ und $(\bar{l}_2, l_1)$ und für jede Einerklausel $\{l\}$ die Kante $(\bar{l}, l)$ enthält. Hierbei sei $\bar{\bar{x}}_i = x_i$. Aufgrund der Konstruktion von G ist klar, dass

($*$) eine Belegung α genau dann F erfüllt, wenn für jede Kante $(l, l') \in E$ mit $\alpha(l) = 1$ auch $\alpha(l') = 1$ ist, und

($**$) l' von l aus genau dann erreichbar ist, wenn $\bar{l}$ von $\bar{l}'$ aus erreichbar ist.

Behauptung 71. *F ist genau dann erfüllbar, wenn für keinen Knoten x_i in G ein Pfad von x_i über $\bar{x}_i$ zurück nach x_i existiert.*

Beweis von Beh. 1. Wenn in G ein Pfad von x_i über $\bar{x}_i$ nach x_i existiert, kann F nicht erfüllbar sein, da wegen ($*$) jede erfüllende Belegung, die x_i (bzw. $\bar{x}_i$) den Wert 1 zuweist, auch $\bar{x}_i$ (bzw. x_i) diesen Wert zuweisen müsste. Existiert dagegen kein derartiger Pfad, so lässt sich für F wie folgt eine erfüllende Belegung α konstruieren:

1) Wähle einen beliebigen Knoten l aus G, für den $\alpha(l)$ noch undefiniert ist. Falls $\bar{l}$ von l aus erreichbar ist, ersetze l durch $\bar{l}$ (dies garantiert, dass $\bar{l}$ von l aus nun nicht mehr erreichbar ist).
2) Weise jedem von l aus erreichbaren Knoten l' den Wert 1 (und $\bar{l}'$ den Wert 0) zu.
3) Falls α noch nicht auf allen Knoten definiert ist, gehe zu 1).

□

Wir müssen noch zeigen, dass bei der Ausführung von 2) keine Konflikte auftreten können. Tatsächlich existiert aufgrund der Symmetriebedingung ($**$) für jeden von l aus erreichbaren Knoten l' ein Pfad von $\bar{l}'$ zu $\bar{l}$.

- Daher kann l' nicht schon in einer früheren Runde den Wert 0 erhalten haben (sonst hätte in dieser Runde $\bar{l}'$ und somit auch $\bar{l}$ den Wert 1 erhalten, was der Wahl von l widerspricht).
- Zudem kann von l aus nicht auch $\bar{l}'$ erreichbar sein (sonst würde ein Pfad von l über $\bar{l}'$ nach $\bar{l}$ existieren, was wir durch die Wahl von l ebenfalls ausgeschlossen haben).

Eine NL-Maschine kann bei Eingabe einer 2-KNF Formel F eine Variable x_i und einen Pfad von x_i über $\bar{x}_i$ zurück nach x_i raten. Dies zeigt, dass das Komplement von 2-Sat in NL ist. Wegen $\mathsf{NL} = \text{co-}\mathsf{NL}$ folgt 2-Sat $\in$ NL. □

In den Übungen werden wir sehen, dass 2-Sat und Reach NL-vollständig sind.