Vorlesungsskript

Einführung in die Komplexitätstheorie

Wintersemester 2021/22

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

1. Dezember 2021

Inhaltsverzeichnis

Inhaltsverzeichnis

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)
- Kryptografie (Wieviel Rechenressourcen benötigt ein Gegner, um ein Kryptosystem zu brechen?)

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

Erreichbarkeitsproblem in Digraphen (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:

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

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. Dieser markiert nach und nach alle Knoten, die vom Knoten 1 aus erreichbar sind. Hierzu speichert er jeden markierten Knoten solange in einer Menge S bis er sämtliche Nachbarknoten markiert hat. Genaueres ist folgendem Algorithmus zu entnehmen:

Algorithmus `suche-Weg`(G)

```
input: Digraph 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 (und/oder die Anzahl m der Kanten) als Bezugsgröße dienen. Der Ressourcenverbrauch hängt auch davon ab, wie wir die Eingabe kodieren. So führt die Repräsentation eines Graphen als Adjazenzliste oftmals zu effizienteren Lösungsverfahren.

Komplexitätsbetrachtungen:

- Reach ist in Zeit $O(n^2)$ entscheidbar.

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

- $\forall e \in E : f(e) \le c(e)$ und
- $\forall v \in V - \{1, n\} : \sum_{(v,u)\in E} f(v, u) = \sum_{(u,v)\in E} f(u, v)$,

mit max. Wert $w(f) = \sum_{(1,u)\in E} f(1, u) - \sum_{(u,1)\in E} f(u, 1)$.

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 $O(n^3)$ lösbar (Algorithmus von Dinitz).
- MAXFLOW ist in Platz $O(n^2)$ 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 $G = (U, W, E)$ mit $U \cap W = \emptyset$ und $e \cap U \neq \emptyset \neq e \cap W$ für alle Kanten $e \in E$.

Gefragt: Besitzt G ein perfektes Matching?

Zur Erinnerung: Eine Kantenmenge $M \subseteq E$ heißt ***Matching***, falls für alle Kanten $e, e' \in M$ mit $e \neq e'$ gilt: $e \cap e' = \emptyset$. Gilt zudem $\|M\| = n/2$, so heißt M ***perfekt*** (n ist die Knotenzahl von G).

Komplexitätsbetrachtungen:

- MATCHING ist in Zeit $O((n + m)\sqrt{n})$ entscheidbar (Algorithmus von Dinitz).
- MATCHING ist in Platz $O(n^2)$ 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. Wie üblich bezeichnen wir die Gruppe aller Permutationen auf der Menge $\{1, \dots, n\}$ mit S_n.

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 $O(n!)$ lösbar (Ausprobieren aller Rundreisen).
- TSP ist in Platz $O(n)$ lösbar (mit demselben Algorithmus).
- Durch dynamisches Programmieren* lässt sich TSP in Zeit $O(n^2 2^n)$ lösen, der Platzverbrauch erhöht sich dabei jedoch auf $O(n2^n)$ (siehe Übungen).

*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 5-Tupel $M = (Q, \Sigma, \Gamma, \delta, q_0)$. 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 $v'_i = \begin{cases} \sqcup, & v_i = \varepsilon, \\ v_i, & \text{sonst}, \end{cases}$*

*Eine **Rechnung** von M bei Eingabe x ist eine Folge von Konfigurationen $K_0, K_1, K_2 \ldots$ mit $K_0 = K_x$ und $K_0 \vdash K_1 \underset{M}{\longrightarrow} K_2 \cdots$.*

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 $q \in \{q_h, q_{\text{ja}}, q_{\text{nein}}\}$ heißt **Endkonfiguration**. Im Fall $q = q_{\text{ja}}$ (bzw. $q = q_{\text{nein}}$) heißt K **akzeptierende** (bzw. **verwerfende**) **Endkonfiguration**.*

Definition 6.
*Eine DTM M **hält** bei Eingabe $x \in \Sigma^*$ (kurz: $M(x)$ hält), falls es eine Endkonfiguration $K = (q, u_1, v_1, \dots, u_k, v_k)$ gibt mit*

$$K_x \xrightarrow[M]{}^{*} K.$$

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

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

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 ein $i \geq 0$. 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 ***semi-entscheidbar*** (oder ***rekursiv aufzählbar***). 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:*

$$x \in L \Rightarrow M(x) \text{ hält und akzeptiert}$$
$$x \notin L \Rightarrow M(x) \text{ hält und akzeptiert nicht.}$$

*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 semi-entscheidbar ist, wenn eine berechenbare 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*

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

die Eigenschaft hat, dass für $(q', a'_1, D_1, \dots, a'_k, D_k) \in \delta(q, a_1, \dots, a_k)$ 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 $\delta(q, a_1, \dots, a_k) = (q', a'_1, D_1, \dots, a'_k, D_k)$ durch $(q', a'_1, D_1, \dots, a'_k, D_k) \in \delta(q, a_1, \dots, a_k)$ ersetzen. In beiden Fällen schreiben wir auch $(q, a_1, \dots, a_k) \mapsto (q', a'_1, D_1, \dots, a'_k, D_k)$.

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

Definition 11. *Sei M eine* NTM.

a) *$M(x)$ **hält** (kurz $M(x)\downarrow$), falls $M(x)$ nur endlich lange Rechnungen ausführt. Andernfalls schreiben wir $M(x)\uparrow$.*
b) *Eine Rechnung von $M(x)$ heißt **akzeptierend** (bzw. **verwerfend**), falls sie K_x in eine akzeptierende (bzw. verwerfende) Endkonfiguration überführt.*
c) *$M(x)$ **akzeptiert**, falls $M(x)$ mindestens eine akzeptierende Rechnung ausführt.*
d) *Die von M **akzeptierte Sprache** ist*

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

e) *M **entscheidet** $L(M)$, falls M bei allen Eingaben hält.*

2.3 Zeitkomplexität

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

Definition 12.

a) *Sei M eine TM (d.h. eine DTM oder NTM) und sei $x \in \Sigma^*$ eine Eingabe. Dann ist*

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

*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

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

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 aller polynomiell beschränkten Funktionen bezeichnen wir mit $\mathrm{poly}(n)$. Die wichtigsten Zeitkomplexitätsklassen sind

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

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 $(q, a_1, \dots, a_k) \mapsto (q', a'_1, D_1, \dots, a'_k, D_k)$ die Bedingung*

$$a'_1 = a_1 \wedge [a_1 = \sqcup \Rightarrow D_1 = L]$$

*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 dieses kann nicht als Speicher benutzt 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 Anzahl aller während der Rechnung besuchten Bandfelder.

Definition 14.

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

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

*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 $\sum_{i=2}^{k-1} |u_i v_i|$. Man beachte, dass $space_M(x)$ im Fall $time_M(x) = \infty$ undefiniert ist.*

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

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

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

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

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)**.**
*Zu jeder $s(n)$-platzbeschränkten Offline-k-*DTM *M mit $k \geq 3$ ex. eine $s(n)$-platzbeschränkte Offline-2-*DTM *M' mit $L(M') = L(M)$.*

Beweis. Sei $M = (Q, \Sigma, \Gamma, \delta, q_0)$ eine Offline-k-DTM mit $k \geq 3$. Betrachte die Offline-2-DTM $M' = (Q', \Sigma, \Gamma', \delta', q_0')$ mit $\Gamma' = \Sigma \cup \{\sqcup, \triangleright\} \cup (\Gamma \cup \hat{\Gamma})^{k-1}$, 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

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

mit

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

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

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

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 k-DTM M' mit $L(M') = L$ und $time_{M'}(x) \le 2 + |x| + c \cdot time_M(x)$. M' verwendet das Alphabet $\Gamma' = \Sigma \cup \{\sqcup, \triangleright\} \cup \Gamma^m$ mit $m = \lceil 8/c \rceil$ und simuliert M wie folgt.

Initialisierung: M' kopiert die Eingabe $x = x_1 \dots x_n$ in Blockform auf das zweite Band. Hierzu fasst M' je m Zeichen von x zu einem Block $(x_{im+1}, \dots, x_{(i+1)m})$, $i = 0, \dots, l = \lceil n/m \rceil - 1$, zusammen, wobei der letzte Block $(x_{lm+1}, \dots, x_n, \sqcup, \dots, \sqcup)$ 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

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

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 $i_j \in \{1, \dots, m\}$ von M innerhalb der gerade gelesenen Blöcke auf den Bändern $j = 1, \dots, k$. 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 $|x| = n$

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

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.

i) $\mathsf{DSPACE}(O(s(n))) = \mathsf{DSPACE}(s(n))$, *falls* $s(n) \geq 2$.

ii) $\mathsf{DTIME}(O(t(n))) = \mathsf{DTIME}(t(n))$, *falls* $t(n) \geq (1+\varepsilon)n + 2$ *für ein* $\varepsilon > 0$ *ist.*

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

Beweis. *i)* Sei $L \in \mathsf{DSPACE}(cs(n)+c)$ für eine Konstante $c \geq 0$. Ist $s(n) < 6$ für alle n, so folgt $L \in \mathsf{DSPACE}(\mathcal{O}(1)) = \mathsf{DSPACE}(0)$. Gilt dagegen $s(n) \geq 6$ für alle $n \geq n_0$, so existiert für $c' = 1/2c$ eine Offline-k-DTM M, die L für fast alle Eingaben in Platz $2 + c'cs(n) + c'c \leq 3 + s(n)/2 \leq s(n)$ entscheidet. Wegen $s(n) \geq 2$ können wir M leicht so modifizieren, dass sie auch die endlich vielen Ausnahmen in Platz $s(n)$ entscheidet.

ii) Sei $L \in \mathsf{DTIME}(ct(n)+c)$ für ein $c > 0$. Nach vorigem Satz existiert für $c' = \varepsilon/(2+2\varepsilon)c$ eine DTM M, die L in Zeit $time_M(x) \leq 2 + n + c'(ct(n)+c)$ entscheidet. Wegen $t(n) \geq (1+\varepsilon)n$ und da für alle $n \geq n_0 := \lceil (4+2c'c)/\varepsilon \rceil$ die Ungleichung $2 + c'c \leq \varepsilon n/2$ gilt, folgt

$$time_M(x) \leq 2 + n + c'ct(n) + c'c = \underbrace{c'ct(n)}_{=\frac{\varepsilon t(n)}{2+2\varepsilon}} + \underbrace{2 + c'c + n}_{\leq \frac{(\varepsilon+2)n}{2} \leq \frac{(\varepsilon+2)t(n)}{2+2\varepsilon}} \leq t(n)$$

für alle $n \geq n_0$. Zudem können wir M im Beweis des vorigen Satzes so konstruieren, dass $M(x)$ auch alle Eingaben x mit $|x| < n_0$ in Zeit $n + 2 \leq t(n)$ entscheidet.

iii) Klar, da $\mathsf{DTIME}(O(n)) = \mathsf{DTIME}(O((1+\varepsilon)n+2))$ und diese Klasse nach *ii)* für jedes $\varepsilon > 0$ gleich $\mathsf{DTIME}((1+\varepsilon)n+2)$ ist. ∎

3.2 Deterministische Simulationen von nichtdeterministischen Berechnungen

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

Satz 18.

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

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

der Länge t von $N(x)$ eindeutig durch eine Folge $(i_1, \dots, i_t) \in \{1, \dots, d\}^t$ beschreibbar. Betrachte die Offline-*(k+2)*-DTM M, die auf ihrem 2. Band für $t = 1, 2, \dots$ der Reihe nach alle Folgen $(i_1, \dots, i_t) \in \{1, \dots, d\}^t$ generiert. 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

$$space_M(x) \;\leq\; time_N(x) + space_N(x) \;\leq\; (k+1)(time_N(x)+1)$$

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. Da N bei einer Eingabe x der Länge n

- höchstens $\|Q\|$ verschiedene Zustände annehmen,
- die Köpfe des Eingabe- bzw. Arbeitsbandes auf höchstens $n+2$ bzw. $s(n)$ verschiedenen Bandfeldern positionieren,

- und das Arbeitsband mit höchstens $\|\Gamma\|^{s(n)}$ verschiedenen Beschriftungen versehen kann,

kann $N(x)$ ausgehend von der Startkonfiguration K_x höchstens

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

verschiedene Konfigurationen erreichen, wobei c eine von N abhängige Konstante ist. Um N zu simulieren, testet M für $s = 1, 2, \ldots$, ob $N(x)$ eine akzeptierende Endkonfiguration $K = (q_{\text{ja}}, u_1, v_1, u_2, v_2)$ der *Größe* $|u_2v_2| = s$ 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 alle von der Startkonfiguration K_x erreichbaren Konfigurationen der Größe s bestimmen, was in Zeit $(c^{s(n)+\log n})^{O(1)} = 2^{O(s(n)+logn)}$ möglich ist. ∎

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

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

und für jede 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 Inklusionen

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

was wiederum $\mathsf{NSPACE}(s) \subseteq \mathsf{DSPACE}(2^{O(s)})$ impliziert. Wie wir im nächsten Abschnitt sehen werden, lässt sich dies noch erheblich verbessern.

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

- $M(x) = 1^{f(|x|)}$,
- $space_M(x) = O(f(|x|))$ *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 20 (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.

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, K' und jede Zahl i folgende Äquivalenz gilt:

$$K \underset{N}{\to}^{\leq 2^i} K' \Leftrightarrow \exists K_j : K \underset{N}{\to}^{\leq 2^{i-1}} K_j \wedge K_j \underset{N}{\to}^{\leq 2^{i-1}} K'.$$

Diese Beobachtung führt sofort auf folgende Prozedur $\texttt{reach}(K, K', i)$, um die Gültigkeit von $K \underset{N}{\to}^{\leq 2^i} K'$ zu testen.

Prozedur `reach`(K, K', i)

if $i = 0$ **then return**$(K = K'$ **or** $K \xrightarrow[N]{} K')$
for each `Konfiguration` K_j **do**
  if `reach`$(K, K_j, i-1)$ **and** `reach`$(K_j, K', i-1)$ **then**
    return$(true)$
return$(false)$

Nun können wir N durch folgende Offline-3-DTM M simulieren. M benutzt ihr 2. Band als Laufzeitkeller zur Verwaltung der Inkarnationen der rekursiven Aufrufe von `reach`. Hierzu speichert M auf ihrem 2. Band eine Folge von Tripeln der Form (K, K', i). Das 3. Band wird zum Kopieren von Tripeln auf dem 2. Band und zur Berechnung von K_{j+1} aus K_j benutzt.

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 z.B. $K_x = (q_0, 1, \varepsilon, \triangleright))$ und jede akzeptierende Endkonfiguration mit der Konfiguration $\hat{K}_x = (q_{\text{ja}}, 1, \varepsilon, \triangleright)$ identifiziert wird.

Simulation: Sei (K, K', i) das am weitesten rechts auf dem 2. Band stehende Tripel (also das oberste Kellerelement). Im Fall $i = 0$ testet M direkt, ob $K \xrightarrow[N]{}^{\leq 1} K'$ gilt und gibt die Antwort zurück.

Andernfalls fügt M beginnend mit $j = 1$ das Tripel $(K, K_j, i-1)$ hinzu und berechnet (rekursiv) die Antwort für dieses 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, K', i) eine negative Antwort).

Erhält $(K, K_j, i-1)$ eine positive Antwort, so ersetzt M das Tripel $(K, K_j, i-1)$ durch das Tripel $(K_j, K', i-1)$ 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 auch (K, 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 $O(s(|x|)^2)$ Felder. ■

Korollar 21.

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

Wir wir gesehen haben, impliziert der Satz von Savitch den Abschluss von $\mathsf{NPSPACE}$ unter Komplementbildung. Dagegen wurde die Frage ob auch die Klasse $\mathsf{CSL} = \mathsf{NSPACE}(n)$ der kontextsensitiven Sprachen unter Komplementbildung abgeschlossen ist, erst in den 80ern von Neil Immerman und unabhängig davon von Robert Szelepcsényi gelöst.

Definition 22. *Für eine Sprachklasse* C *bezeichne* $\mathsf{co}\text{-}\mathsf{C} = \{\overline{L} | L \in \mathsf{C}\}$ *die zu* C ***komplementäre Sprachklasse****. Dabei bezeichnet* $\overline{L} = \Sigma^* - L$ *das* ***Komplement*** *einer Sprache* $L \subseteq \Sigma^*$.

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

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

Gefragt: Ist F eine Tautologie, d. h. erfüllen alle Belegungen $\vec{a} \in \{0,1\}^n$ die Formel F?

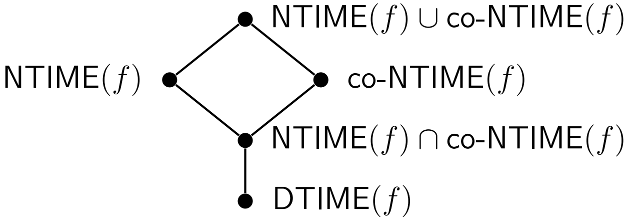
Die Frage ob NP unter Komplementbildung abgeschlossen ist (d.h., ob $\mathsf{NP} = \mathsf{co\text{-}NP}$ gilt), ist ähnlich wie das $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$-Problem ungelöst. Deterministische Rechnungen lassen sich leicht komplementieren (durch Vertauschen der Zustände q_{ja} und q_{nein}). Daher sind deterministische Komplexitätsklassen unter Komplementbildung abgeschlossen.

Proposition 23.

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

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

Damit ergibt sich folgende Inklusionsstruktur:



Dagegen erfordert die Komplementierung von nichtdeterministischen Berechnungen das Vorliegen gewisser Zusatzeigenschaften.

Definition 24. *Eine* NTM N *heißt* ***strong*** *bei Eingabe x, falls $N(x)$ mindestens eine akzeptierende oder mindestens eine verwerfende Rechnung ausführt, aber nicht beides.*

Proposition 25.

i) $\mathsf{NTIME}(t(n)) \cap \mathsf{co\text{-}NTIME}(t(n)) = \{L(M) \mid M$ *ist eine $t(n)$-zeitbeschränkte NTM, die bei allen Eingaben strong ist*$\}$,

ii) $\mathsf{NSPACE}(s(n)) \cap \mathsf{co\text{-}NSPACE}(s(n)) = \{L(M) \mid M$ *ist eine $s(n)$-platzbeschr. Offline-NTM, die bei allen Eingaben strong ist*$\}$.

Beweis. Siehe Übungen. ■

Satz 26 (Immerman und Szelepcsényi, 1987)**.** *Für jede echte Komplexitätsfunktion $s(n) \geq \log n$ gilt*

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

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

$$r(x, t-1) = \|\{K | K_x \underset{N}{\to}^{\leq t-1} K\}\|$$

aller in höchstens $t-1$ Schritten erreichbaren Konfigurationen strong ist. Betrachte die Sprache

$$L_0 = \{\langle x, r, t, K\rangle | 1 \leq r, t \leq c^{s(n)} \text{ und } K_x \underset{N}{\to}^{\leq t} K\}.$$

Behauptung 27. *Es existiert eine Offline-*NTM N_0 *mit $L(N_0) = L_0$, die $O(s(|x|))$ Felder besucht und bei allen Eingaben $w = \langle x, r, t, K\rangle$ mit $r = r(x, t-1)$ strong ist (d.h. $N_0(w)$ hat genau im Fall $w \notin L_0$ eine verwerfende Rechnung).*

Beweis der Behauptung. $N_0(\langle x, r, t, K\rangle)$ benutzt einen mit dem Wert 0 initialisierten Zähler z und rät der Reihe nach für jede Konfiguration K' der Größe $\leq s(n)$ eine Rechnung von $N(x)$ der Länge $\leq t-1$, die in K' endet. Falls dies gelingt, erhöht N_0 den Zähler z um 1 und testet, ob $K' \underset{N}{\to}^{\leq 1} K$ gilt. Falls ja, so hält N_0 im Zustand q_{ja}.

Nachdem N_0 alle Konfigurationen K' der Größe $\leq s(n)$ durchlaufen hat, hält N_0 im Zustand q_{nein}, wenn z den Wert r hat, andernfalls im Zustand q_{h}.

Pseudocode für $N_0(\langle x, r, t, K\rangle)$

if $t = 0$ **then** halte im Zustand $q_{\texttt{nein}}$
$z := 0$
for each Konfiguration K' der Größe $\leq s(n)$ **do**
  rate eine Rechnung α der Laenge $\leq t-1$ von $N(x)$
  if α endet in K' **then**
    $z := z + 1$
    if $K' \underset{N}{\to}^{\leq 1} K$ **then**
      halte im Zustand $q_{\texttt{ja}}$
if $z = r$ **then**
  halte im Zustand $q_{\texttt{nein}}$
else
  halte im Zustand $q_{\texttt{h}}$

Da N_0 genau dann eine akzeptierende Rechnung hat, wenn eine Konfiguration K' mit $K_x \underset{N}{\to}^{\leq t-1} K'$ und $K' \underset{N}{\to}^{\leq 1} K$ 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 allen Eingaben $w = \langle x, r, t, K\rangle$ mit $r = r(x, t-1)$ strong ist, also $N_0(w)$ genau im Fall $w \notin L_0$ eine verwerfende Rechnung hat.

Um bei Eingabe $w = \langle x, r, t, K\rangle$ eine verwerfende Endkonfiguration zu erreichen, muss N_0 $r = r(x, t-1)$ Konfigurationen K' finden, für die zwar $K_x \underset{N}{\to}^{\leq t-1} K'$ aber nicht $K' \underset{N}{\to}^{\leq 1} K$ 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 w tatsächlich nicht zu L_0 gehört. Die Umkehrung folgt analog. $\square$

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). Die Kenntnis der Anzahlen $r(x, t)$ versetzt N' in die Lage, für alle Eingaben $x \notin L$ zu verifizieren, dass $N(x)$ keine akzeptierende Endkonfiguration erreichen kann.

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** K' der Größe $\leq s(n)$ **do**
8   simuliere N_0 bei Eingabe $\langle x, r^-, t, K_i\rangle$
9   **if** N_0 akzeptiert **then**
10       $r := r + 1$
11   **if** K' ist akzeptierende Endkonfiguration **then**
12   halte im Zustand $q_{\texttt{ja}}$
13   **if** N_0 haelt im Zustand $q_{\texttt{h}}$ **then**
14   halte im Zustand $q_{\texttt{h}}$
15 **until** $(r = r^-)$
16 halte im Zustand $q_{\texttt{nein}}$

Behauptung 28. *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 $\langle x, r(x, t-1), t, K_i\rangle$ aufgerufen.*

Beweis der Behauptung. Wir führen Induktion über t:

$t = 1$**:** Im ersten Durchlauf der repeat-Schleife erhält r^- in Zeile 5 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 $\langle x, r(x, t-1), t, K_i\rangle$ 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 $\leq t$ Schritten erreichbaren Konfigurationen K_i als solche erkannt und somit wird r tatsächlich auf den Wert $r(x,t)$ hochgezählt. □

Behauptung 29. *Bei Beendigung der repeat-Schleife in Zeile 15 gilt* $r = r^- = \|\{K | K_x \xrightarrow[N]{}^* K\}\|$.

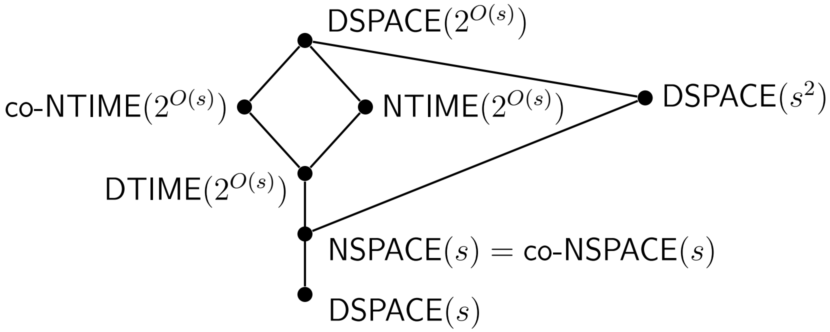
Beweis der Behauptung. Wir wissen bereits, dass im t-ten Durchlauf der repeat-Schleife r den Wert $r(x,t)$ und r^- den Wert $r(x,t-1)$ erhält. Wird daher die repeat-Schleife nach t_e Durchläufen verlassen, so gilt $r = r^- = r(x,t_e) = r(x,t_e-1)$.
Angenommen $r(x,t_e) < \|\{K | K_x \xrightarrow[N]{}^* K\}\|$. 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 $K_x = K_0 \xrightarrow[N]{} K_1 \xrightarrow[N]{} \dots \xrightarrow[N]{} K_{t'} = K$ minimaler Länge, die in K endet. Dann gilt $K_x \xrightarrow[N]{}^{t_e} K_{t_e}$, aber nicht $K_x \xrightarrow[N]{}^{\leq t_e - 1} K_{t_e}$ und daher folgt $r(x,t_e) > r(x,t_e-1)$. Widerspruch! □

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 29 hat $N'(x)$ nur dann eine verwerfende Rechnung, wenn im letzten Durchlauf der repeat-Schleife alle erreichbaren Konfigurationen K gefunden wurden und sich darunter keine akzeptierende Endkonfiguration befand. Dies impliziert $x \notin L$. Die Umkehrung, dass $N'(x)$ für alle $x \notin L$ eine verwerfende Rechnung hat, folgt analog. ■

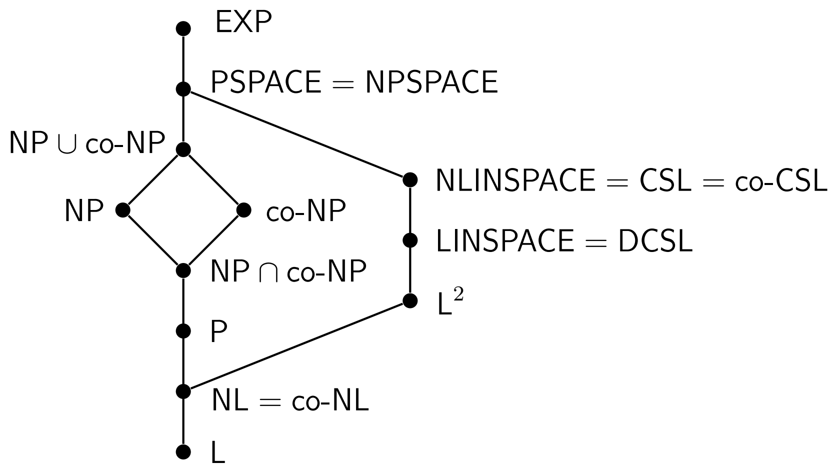
Korollar 30.

1. NL = co-NL,
2. CSL = NLINSPACE = co-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. Dies untersuchen wir im nächsten Kapitel.

4 Hierarchiesätze

4.1 Unentscheidbarkeit mittels Diagonalisierung

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 $\Gamma = \{a_1, \ldots, a_\ell\}$ (also z.B. $a_1 = \sqcup$, $a_2 = \triangleright$, $a_3 = 0$, $a_4 = 1$ etc.). Dann kodieren wir jedes $\alpha \in Q \cup \Gamma \cup \{q_{\mathrm{h}}, q_{\mathrm{ja}}, q_{\mathrm{nein}}, L, R, N\}$ 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, $j = 1, \ldots, l$	$bin_b(m + j)$
$q_{\mathrm{h}}, q_{\mathrm{ja}}, q_{\mathrm{nein}}, L, R, N$	$bin_b(m + l + 1), \ldots, bin_b(m + l + 6)$

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

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

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 DTM's und NTM's mit mehreren Bändern erweitern. Die Kodierung einer TM M bezeichnen wir mit $\langle M \rangle$. Umgekehrt können wir jedem Binärstring w eine TM M_w wie folgt zuordnen:

$$M_w = \begin{cases} M, & \langle M \rangle = w \\ M_0, & \text{sonst,} \end{cases}$$

wobei M_0 eine beliebig aber fest gewählte TM ist (z.B. eine, die nach einem Schritt im Zustand q_{nein} hält). Für M_w schreiben wir auch M_i, wobei i die Zahl mit der Binärdarstellung $1w$ ist. Ein Paar (M, x) bestehend aus einer TM M und einer Eingabe $x \in \{0, 1\}^*$ kodieren wir durch das Wort $\langle M, x \rangle = \langle M \rangle \# x$.

Satz 31. *Die Diagonalsprache*

$$D = \{x_i | M_i \textit{ ist eine DTM und akzeptiert die Eingabe } x_i\}$$

ist semi-entscheidbar, aber nicht entscheidbar. Hierbei ist $x_1 = \varepsilon$, $x_2 = 0$, $x_3 = 1$, $x_4 = 00, \ldots$ die Folge aller Binärstrings in lexikografischer Reihenfolge.

Beweis. Es ist klar, dass D semi-entscheidbar ist, da es eine DTM gibt, die bei Eingabe x_i die Berechnung von M_i bei Eingabe x_i simuliert und genau dann akzeptiert, wenn dies $M_i(x_i)$ tut.

Dass $\bar{D}$ nicht semi-entscheidbar (und damit D nicht entscheidbar) ist, liegt daran, dass die charakteristische Funktion von $\bar{D}$ „komplementär“ zur Diagonalen der Matrix ist, deren Zeilen die charakteristischen Funktionen aller semi-entscheidbaren Sprachen $L(M_i) \subseteq \{0, 1\}^*$ auflisten. Wir zeigen durch einen einfachen Widerspruchsbeweis, dass keine Zeile der Matrix mit dem Komplement ihrer Diagonalen übereinstimmen kann. Wäre $\bar{D}$ semi-entscheidbar, gäbe es also eine DTM M_d, die $\bar{D}$ akzeptiert,

$$L(M_d) = \bar{D} \quad (*),$$

	x_1	x_2	x_3	x_4	$\cdots$
M_1	**1**	0	0	0	$\cdots$
M_2	0	**1**	0	0	$\cdots$
M_3	1	0	**0**	0	$\cdots$
M_4	0	0	0	**1**	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

M_d	**0**	**0**	**1**	**0**	$\cdots$

so führt dies wegen

$$\begin{array}{llllll} x_d \in D & \overset{\text{(Def. von } D)}{\Rightarrow} & M_d(x_d) \text{ akzeptiert} & \overset{(*)}{\Rightarrow} & x_d \notin D & \lightning \\ x_d \notin D & \overset{\text{(Def. von } D)}{\Rightarrow} & M_d(x_d) \text{ akz. nicht} & \overset{(*)}{\Rightarrow} & x_d \in D & \lightning \end{array}$$

zu einem Widerspruch. ■

Satz 32. *Für jede berechenbare Funktion $g : \mathbb{N} \longrightarrow \mathbb{N}$ existiert eine entscheidbare Sprache $D_g \notin \mathsf{DTIME}(g(n))$.*

Beweis. Betrachte die Diagonalsprache

$$D_g = \{x_i \in \{0,1\}^* \mid M_i \text{ ist eine DTM und akzeptiert die Eingabe } x_i \text{ in } \leq g(|x_i|) \text{ Schritten}\} \qquad (*)$$

Offensichtlich ist D_g entscheidbar. Unter der Annahme, dass $D_g \in \mathsf{DTIME}(g(n))$ ist, existiert eine DTM M_d, die das Komplement von D_g in Zeit $g(n)$ entscheidet, d.h.

$$M_d \text{ ist } g(n)\text{-zeitbeschränkt } (**) \text{ und } L(M_d) = \bar{D}_g \ (***)$$

Dies führt jedoch auf einen Widerspruch:

$$\begin{array}{llllll} x_d \in D_g & \overset{(*)}{\Rightarrow} & M_d(x_d) \text{ akzeptiert} & \overset{(***)}{\Rightarrow} & x_d \notin D_g & \lightning \\ x_d \notin D_g & \overset{(*,**)}{\Rightarrow} & M_d(x_d) \text{ verwirft} & \overset{(***)}{\Rightarrow} & x_d \in D_g & \lightning \end{array}$$

■

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

4.2 Das Gap-Theorem

Satz 33 (Gap-Theorem)**.**
Es gibt eine berechenbare Funktion $g : \mathbb{N} \to \mathbb{N}$ mit

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

Beweis. Wir definieren $g(n) \geq n+2$ so, dass für jede $2^{g(n)}$-zeitb. DTM M gilt:

$$time_M(x) \leq g(|x|) \text{ für fast alle Eingaben } x.$$

Betrachte hierzu das Prädikat

$$\begin{array}{ll} P(n,t): & t \geq n+2 \text{ und für } k = 1, \ldots, n \text{ und alle } x \in \Sigma^n \text{ gilt:} \\ & time_{M_k}(x) \notin [t+1, 2^t], \end{array}$$

wobei Σ das Eingabealphabet von M_k ist. Da P entscheidbar ist und alle Paare (n,t) mit

$$t \geq \max\{time_{M_k}(x) | 1 \leq k \leq n, x \in \Sigma^n, M_k(x) \text{ hält}\}$$

das Prädikat $P(n,t)$ erfüllen, ist die induktiv definierte Funktion

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

berechenbar und erfüllt $P(n, g(n))$ für alle n.

Um zu zeigen, dass jede Sprache $L \in \mathsf{DTIME}(2^{g(n)})$ bereits in $\mathsf{DTIME}(g(n))$ enthalten ist, sei M_k eine beliebige $2^{g(n)}$-zeitbeschränkte DTM mit $L(M_k) = L$. Dann muss M_k alle Eingaben x der Länge $n \geq k$ in Zeit $time_{M_k}(x) \leq g(n)$ entscheiden, da andernfalls

$P(n, g(n))$ wegen $time_{M_k}(x) \in [g(n)+1, 2^{g(n)}]$ verletzt wäre. Folglich ist $L \in \mathsf{DTIME}(g(n))$, da die endlich vielen Eingaben x der Länge $n < k$ durch table-lookup in Zeit $n+2 \le g(n)$ entscheidbar sind. ■

Es ist leicht zu sehen, dass der Beweis des Gap-Theorems für jede berechenbare Funktion h eine berechenbare Zeitschranke g liefert, so dass $\mathsf{DTIME}(h(g(n))) = \mathsf{DTIME}(g(n))$ ist. Folglich ist die im Beweis von Satz 32 definierte Sprache D_g nicht in Zeit $h(g(n))$ entscheidbar.

4.3 Zeit- und Platzhierarchiesätze

Um D_g zu entscheiden, müssen wir einerseits die Zeitschranke $g(|x_i|)$ berechnen und andererseits $M_i(x_i)$ simulieren. Wenn wir voraussetzen, dass g eine echte Komplexitätsfunktion ist, lässt sich $g(|x_i|)$ effizient berechnen. Für die zweite Aufgabe benötigen wir eine möglichst effiziente universelle TM.

Satz 34. *Es gibt eine universelle 3-DTM U, die für jede DTM M und jedes $x \in \{0,1\}^*$ bei Eingabe $\langle M, x\rangle$ eine Simulation von M bei Eingabe x in Zeit $O(|\langle M\rangle|(time_M(x))^2)$ und Platz $O(|\langle M\rangle| space_M(x))$ durchführt und dasselbe Ergebnis wie $M(x)$ liefert:*

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

Beweis. Wir nehmen an, dass M eine Sprache entscheidet und niemals im Zustand q_h hält. Betrachte folgende Offline-*3*-DTM U:

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

Simulation: U simuliert jeden Rechenschritt von M wie folgt: Zunächst inspiziert U 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. Insgesamt benötigt U für die Simulation eines Rechenschrittes von M Zeit $\mathcal{O}(kb \cdot time_M(x)) = \mathcal{O}(|\langle M\rangle| \cdot time_M(x)))$.

Akzeptanzverhalten: Sobald die Simulation von M zu einem Ende kommt, hält U im gleichen Zustand wie M.

Nun ist leicht zu sehen, dass $U(\langle M, x\rangle)$ genau dann akzeptiert, wenn dies $M(x)$ tut, und $O(|\langle M\rangle|(time_M(x))^2)$ Rechenschritte macht sowie auf den Arbeitsbändern $O(|\langle M\rangle| space_M(x))$ Felder besucht. ■

Korollar 35. *(Zeithierarchiesatz)*
Für jede echte Komplexitätsfunktion $g(n) \ge n+2$ gilt

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

Beweis. Es genügt zu zeigen, dass D_g für jede echte Komplexitätsfunktion $g(n) \ge n+2$ in Zeit $O(ng^2(n))$ entscheidbar ist. Betrachte folgende *4*-DTM M'. M' überprüft bei einer Eingabe x der Länge n zuerst, ob x die Kodierung $\langle M\rangle$ einer k-DTM M ist. Falls ja, erzeugt M' auf dem 4. Band den String $1^{g(n)}$ in Zeit $\mathcal{O}(g(n))$ und simuliert $M(x)$ wie im Beweis von Theorem 34. Dabei vermindert M' die Anzahl der Einsen auf dem 4. Band nach jedem simulierten Schritt von $M(x)$ um 1. 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_{ja} endet. Nun ist leicht zu sehen, dass M' $\mathcal{O}(n \cdot g(n)^2)$-zeitbeschränkt ist und die Sprache D_g entscheidet. ■

Korollar 36.

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

Beweis.

$$\begin{aligned}\mathsf{P} = \bigcup_{c>0} \mathsf{DTIME}(n^c + c) &\subseteq \mathsf{DTIME}(2^{n+1}) \\ &\subsetneq \mathsf{DTIME}(n2^{2n+2}) \subseteq \mathsf{E} = \bigcup_{c>0} \mathsf{DTIME}(2^{cn+c}) \subseteq \mathsf{DTIME}(2^{n^2+2}) \\ &\subsetneq \mathsf{DTIME}(n2^{2n^2+4}) \subseteq \bigcup_{c>0} \mathsf{DTIME}(2^{n^c+c}) = \mathsf{EXP}\end{aligned}$$

■

Für Platzklassen erhalten wir eine noch feinere Hierarchie.

Satz 37 (Platzhierarchiesatz)**.** *Ist $f(n) \geq 2$ eine echte Komplexitätsfunktion, so gilt für jede Funktion g mit* $\liminf_{n\to\infty} g(n)/f(n) = 0$,

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

Beweis. Sei $M_1, M_2, \ldots$ eine Aufzählung aller Offline-2-DTMs. Für $x \in \{0, 1, \#\}^*$ sei

$$i(x) = \begin{cases} i, & x = 0^k\#\langle M_i\rangle \text{ kür ein } k \geq 0 \\ 1, & \text{sonst} \end{cases}$$

Betrachte folgende Offline-DTM M:

```
input: x ∈ {0,1,#}*
markiere auf dem 2. Band f(|x|) Felder zur Benutzung
simuliere M_i(x)(x) auf dem 2. Band und akzeptiere
   genau dann, wenn M_i(x)(x) auf dem markierten
   Platz nicht akzeptiert
```

Per Konstruktion von M ist $L = L(M) \in \mathsf{DSPACE}(f(n))$.

Angenommen, es ex. eine DTM M_i mit $L(M_i) = L$ und $space_{M_i}(x) \leq g(|x|)$. Wählen wir nun $k \geq 0$ so, dass für $x = 0^k\#\langle M_i\rangle$ die Ungleichung $|\langle M_i\rangle| space_{M_i}(x) \leq f(|x|)$ gilt (dies ist möglich, da $\liminf_{n\to\infty} g(n)/f(n) = 0$ ist), so hat $M(x)$ genügend Platz, um $M_i(x)$ zu simulieren, d.h. $x \in L(M) \Leftrightarrow x \notin L(M_i)$. Widerspruch. ■

Damit lässt sich im Fall $2 \leq g(n) \leq f(n)$ 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 38.

$$\mathsf{L} \subsetneq \mathsf{L}^2 \subsetneq \mathsf{DCSL} \subseteq \mathsf{CSL} \subsetneq \mathsf{PSPACE} \subsetneq \mathsf{ESPACE} \subsetneq \mathsf{EXPSPACE}.$$

Durch Kombination der Beweistechnik von Satz 37 mit der Technik von Immerman und Szelepcsényi erhalten wir auch für nichtdeterministische Platzklassen eine sehr fein abgestufte Hierarchie (ohne Beweis).

Satz 39 (Nichtdeterministischer Platzhierarchiesatz)**.** *Ist $f(n) \geq 2$ eine echte Komplexitätsfunktion, so gilt für jede Funktion g mit* $\liminf_{n\to\infty} g(n)/f(n) = 0$,

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

Wir bemerken, dass sich mit Hilfe einer aufwändigeren Simulationstechnik von $g(n)$-zeitbeschränkten k-DTMs durch eine *2*-DTM in Zeit $\mathcal{O}(g(n) \cdot \log g(n))$ folgende schärfere Form des Zeithierarchiesatzes erhalten lässt (ohne Beweis).

Satz 40. *Ist $f(n) \geq n + 2$ eine echte Komplexitätsfunktion, so gilt für jede Funktion g mit* $\liminf_{n\to\infty}(g(n)\log g(n))/f(n) = 0$,

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

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 $n^2 \log n \log\log n$.

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

Satz 41 (Nichtdeterministischer Zeithierarchiesatz). *Ist $f(n) \geq n+2$ eine echte Komplexitätsfunktion, so gilt für jede Funktion g mit $g(n+1) = o(f(n))$,*

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

Beweis. Sei $M_1, M_2, \ldots$ eine Aufzählung aller 2-NTMs. Für $x \in \{0,1,\#\}^*$ sei

$$i(x) = \begin{cases} i, & x = 0^k\#\langle M_i\rangle \\ 1, & \text{sonst} \end{cases}$$

und x^+ (x^-) sei der lexikografische Nachfolger (bzw. Vorgänger) von x in $\{0,1,\#\}^*$. Wir ordnen jedem $x \in \{0,1,\#\}^*$ ein Intervall $I_x = [s(x), s(x^+) - 1] \subseteq \mathbb{N}_0$ zu, wobei die Funktion s induktiv durch

$$s(x) = \begin{cases} 0, & x = \varepsilon \\ h(s(x^-) + |x^-|), & \text{sonst} \end{cases}$$

definiert ist. Hierbei ist $h(n) \geq 2^n$ eine monotone Funktion mit folgenden Eigenschaften:

- die Sprache

  $$D = \{0^s\#\langle M_i\rangle \mid M_i(0^s) \text{ akz. nicht in } \leq f(s) \text{ Schritten}\}$$

  ist von einer NTM in Zeit $h(n)$ entscheidbar.
- die Funktion $0^n \to 0^{h(n)}$ ist von einem Transducer T in Zeit $h(n)+1$ berechenbar, d.h. $T(0^n)$ schreibt in jedem Rechenschritt (außer dem ersten) eine weitere Null auf's Ausgabeband.

Betrachte folgende NTM M:

input: 0^n
  $x := \varepsilon$; $s := 0$
  while $h(s+|x|) \leq n$ **do**
    $s := h(s+|x|)$
    $x := x^+$
  if $n < h(s+|x|) - 1$ **then** $(*\ s = s(x) \leq n < s(x^+) - 1\ *)$
    `akz. falls` $M_{i(x)}(0^{n+1})$ `in` $\leq \frac{f(n)}{|\langle M_{i(x)}\rangle|}$ `Schritten akz.`
  else $(*\ s = s(x) \leq n = s(x^+) - 1\ *)$
    `akz. falls` $0^s\#\langle M_{i(x)}\rangle \in D$ `ist`

Es ist leicht zu sehen, dass M $\mathcal{O}(f(n))$-zeitb. und somit $L = L(M) \in \mathsf{NTIME}(f(n))$ enthalten ist. Dies liegt daran, dass

- die Berechnung von x und $s = s(x)$ mit $n \in I_x$ in der while-Schleife wegen $h(n) \geq 2^n$ und der Eigenschaften von T in Zeit $\mathcal{O}(n)$ ausführbar, sowie
- die Frage, ob $M_{i(x)}(0^{n+1})$ in $\leq \frac{f(n)}{|\langle M_{i(x)}\rangle|}$ Schritten akz., in Zeit $\mathcal{O}(f(n))$ und
- im Fall $n = s(x^+) - 1$ die Frage, ob $0^s\#\langle M_{i(x)}\rangle \in D$ enthalten ist, in Zeit $h(|0^s\#\langle M_{i(x)}\rangle|) \leq h(s+|x|) = n+1$ entscheidbar ist.

L kann aber nicht in $\mathsf{NTIME}(g(n))$ enthalten sein, da sonst eine Konstante c und eine 2-NTM M_i ex. würden mit $L(M_i) = L$ und $time_{M_i}(0^n) \leq cg(n)$ (siehe Übungen; Simulation von NTMs durch 2-NTMs). Wählen wir nun $k \geq 0$ so groß, dass für $x = 0^k\#\langle M_i\rangle$ und alle $n \geq s(x)$

$$|\langle M_i\rangle| time_{M_i}(0^{n+1}) \leq |\langle M_i\rangle| cg(0^{n+1}) \leq f(n)$$

gilt, so folgt für alle $n \in [s(x), s(x^+) - 2]$:

$$0^n \in L(M) \Leftrightarrow 0^{n+1} \in L(M_i),$$

was $0^{s(x)} \in L \Leftrightarrow 0^{s(x^+)-1} \in L$ impliziert. Zudem gilt wegen $time_{M_i}(0^{s(x)}) \leq f(s(x))$

$$0^{s(x^+)-1} \in L(M) \Leftrightarrow 0^{s(x)}\#\langle M_i\rangle \in D \Leftrightarrow 0^{s(x)} \notin L(M_i),$$

was wegen $L(M) = L = L(M_i)$ ein Widerspruch ist. ■

Satz 41 liefert für langsam wachsende Zeitschranken eine feinere Hierarchie als Satz 40. Beispielsweise impliziert Satz 41, dass $\mathsf{NTIME}(n^k)$ für jede unbeschränkte monotone Funktion h echt in der Klasse $\mathsf{NTIME}(n^k h(n))$ enthalten ist, da $(n+1)^k = \mathcal{O}(n^k) = o(n^k h(n))$ ist.

Für schnell wachsende Zeitschranken liefert dagegen Satz 40 eine feinere Hierarchie. So impliziert Satz 40 zum Beispiel, dass die Klasse $\mathsf{DTIME}(2^{2^n})$ für jede unbeschränkte monotone Funktion h echt in $\mathsf{DTIME}(h(n)2^n 2^{2^n})$ enthalten ist, während sich $\mathsf{NTIME}(2^{2^n})$ mit Satz 41 nur von $\mathsf{NTIME}(h(n)2^{2^{n+1}}) = \mathsf{NTIME}(h(n)2^{2^n}2^{2^n})$ separieren lässt.

5 Reduktionen

5.1 Logspace-Reduktionen

Um zwei Probleme A und B bzgl. ihrer Komplexität zu vergleichen, können wir versuchen, die Frage, ob $x \in A$ ist, auf eine Frage der Form $y \in B$ zurückzuführen. Lässt sich hierbei y leicht aus x berechnen, so können wir jeden Algorithmus für B in einen Algorithmus für A umwandeln, der vergleichbare Komplexität hat.

Definition 42. *Seien A und B Sprachen mit $A \subseteq \Sigma^*$. 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,*

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

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

Definition 43.

a) *Sei C eine Sprachklasse. Eine Sprache L heißt C-**hart** (bzgl. $\leq$), falls für alle Sprachen $A \in \mathsf{C}$ gilt, $A \leq L$.*
b) *Eine C-harte Sprache L, die zu C gehört, heißt C-**vollständig**.*
c) *C heißt **abgeschlossen unter** $\leq$, falls für alle Sprachen A, B gilt:*
$$A \leq B \wedge B \in \mathsf{C} \Rightarrow A \in \mathsf{C}.$$

Lemma 44.

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

Beweis. Siehe Übungen. ∎

Lemma 45. $\mathsf{FL} \subseteq \mathsf{FP}$.

Beweis. Sei $f \in \mathsf{FL}$ und sei M ein logarithmisch platzbeschränkter Transducer (kurz: FL-Transducer), der f berechnet. Da M bei einer Eingabe x der Länge n nur $2^{O(\log n)}$ verschiedene Konfigurationen einnehmen kann, ist M dann auch polynomiell zeitbeschränkt. ∎

Im nächsten Beispiel reduzieren wir das Hamiltonkreisproblem für Digraphen auf das Erfüllbarkeitsproblem für boolesche Formeln.

Hamiltonkreisproblem für Digraphen (DiHam):

Gegeben: Ein Digraph $G = (V, E)$.
Gefragt: Hat G einen Hamiltonkreis?

Erfüllbarkeitsproblem für boolesche Formeln (Sat):

Gegeben: Eine boolesche Formel F über n Variablen.
Gefragt: Ist F erfüllbar?

Beispiel 46. *Um DiHam auf Sat zu reduzieren, benötigen wir eine Funktion $f \in \mathsf{FL}$, die einen Digraphen $G = (V, E)$ 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 $x_{1,1}, x_{1,2}, \ldots, x_{n,n}$, wobei $x_{i,j}$ für die Aussage steht, dass $j \in V = \{1, \ldots, n\}$ der i-te Knoten auf dem Hamiltonkreis ist. Die folgenden Klauseln stellen dann sicher, dass die Relation $\pi = \{(i, j) \mid x_{i,j} = 1\}$ eine Permutation in S_n ist, die eine Rundreise in G beschreibt:

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

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

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

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

iii) Jeder Knoten j wird mindestens einmal besucht:

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

iv) 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) verifizieren, dass π 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.* ◁

5.2 Polynomielle Schaltkreiskomplexität

Definition 47. *Ein* ***boolescher Schaltkreis c mit n Eingängen $x_1, \ldots, x_n$*** *ist eine Folge $c = (g_1, \ldots, g_m)$ von* ***Gattern***

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

mit $1 \le j, k < \ell$. Der ***am Gatter g_ℓ berechnete Wert $g_\ell(a)$ bei Eingabe $a = a_1 \cdots a_n \in \{0,1\}^n$*** *ist induktiv wie folgt definiert:*

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

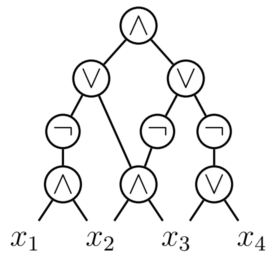
Der Schaltkreis c berechnet die boolesche Funktion $\boldsymbol{c : a \mapsto g_m(a)}$. *Er heißt* ***erfüllbar****, wenn es eine Eingabe $a \in \{0,1\}^n$ mit $c(a) = 1$ gibt.*

Ein Schaltkreis $c = (g_1, \ldots, g_m)$ lässt sich graphisch durch den Digraphen $G_c = (V, E)$ mit $V = \{g_1, \ldots, g_m\}$ und

$$E = \{(g_j, g_\ell) \mid \exists k : g_\ell \in \{(\neg, j), (\wedge, j, k), (\vee, j, k)\}\}$$

darstellen. Im Fall $(g_j, g_\ell) \in E$ wird g_j als auch ***Eingang des Gatters g_ℓ*** bezeichnet.

Beispiel 48.



Aus dieser Darstellung lassen sich die Kantenrichtungen und die Reihenfolge der Gatter rekonstruieren, sofern wir g_j im Fall $(g_j, g_\ell) \in E$ unterhalb von g_ℓ platzieren und die Gatter von unten nach oben und von links nach rechts durchnummerieren. Der obige Graph repräsentiert also den Schaltkreis $c = (x_1, x_2, x_3, x_4, (\wedge, 1, 2), (\wedge, 2, 3), (\vee, 3, 4), (\neg, 5), (\neg, 6), (\neg, 7), (\vee, 6, 8), (\vee, 9, 10), (\wedge, 11, 12))$. ◁

Bemerkung: Die ***Größe size(c)*** eines Schaltkreises c ist die Anzahl m seiner Gatter. 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 Eingang benutzen) als ***Fanout***. Boolesche Formeln entsprechen also den booleschen Schaltkreisen mit (maximalem) Fan-out 1 und umgekehrt. Die ***Tiefe*** eines Schaltkreises c ist die maximale Länge eines Pfades in G_c.

Mit Schaltkreisen lassen sich nicht nur boolesche Funktionen berechnen, sondern auch Sprachen entscheiden.

Definition 49.

a) *Eine Sprache $L \subseteq \{0,1\}^*$ hat **polynomielle Schaltkreiskomplexität** (kurz: $L \in \mathsf{PSK}$), falls es ein $c \geq 1$ und eine Folge von booleschen Schaltkreisen c_n, $n \geq 0$, mit n Eingängen der Größe size$(c_n) \leq n^c + c$ 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 $\Sigma = \{a_0, \ldots, a_{k-1}\}$ hat **polynomielle Schaltkreiskomplexität** (kurz: $L \in \mathsf{PSK}$), falls die Binärsprache*

$$bin(L) = \{bin(x) \mid x \in L\} \in \mathsf{PSK}$$

ist. Hierbei kodieren wir ein Wort $x = x_1 \cdots x_n \in \Sigma^n$ durch den Binärstring $bin(x) = bin(x_1) \cdots bin(x_n)$, wobei wir die Zeichen $a_i \in \Sigma$ für $m = \max\{1, \lceil \log_2 k \rceil\}$ durch die m-stellige Binärdarstellung $bin(i) \in \{0,1\}^m$ der Zahl i kodieren.

Die Turingmaschine ist ein ***uniformes*** Rechenmodell, da alle Instanzen eines Problems von einer einzigen 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 mit Schaltkreisen eine unendliche Sprache entscheiden zu können, wird also eine unendliche Folge c_n, $n \geq 0$, von Schaltkreisen benötigt.

5.3 P-vollständige Probleme

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. In den Übungen werden wir sehen, dass Reach NL-vollständig ist.

Beispiel 50. *Für die Reduktion* Reach $\leq$ CirVal *benötigen wir eine Funktion $f \in \mathsf{FL}$ mit der Eigenschaft, dass für alle Digraphen $G = (V, E)$ mit $V = \{1, \ldots, n\}$ gilt:*

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

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

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

Dabei soll gelten:

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

Die Gatter $g_{i,j,0}$ sind also die booleschen Konstanten

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

und für $k = 1, 2, \ldots, n$ gilt

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

Wählen wir nun $g_{1,n,n}$ als Ausgabegatter, so hat der resultierende Schaltkreis $c = f(G)$ mit 0 Eingängen genau dann den Wert 1, wenn es in G einen Weg von Knoten 1 zu Knoten n gibt. Zudem ist leicht zu sehen, dass c bei Eingabe G in FL berechenbar ist. ◁

Der in Beispiel 50 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.

Satz 51. CIRVAL *ist* P*-vollständig.*

Beweis. Es ist leicht zu sehen, dass CIRVAL $\in \mathsf{P}$ ist. Um zu zeigen, dass CIRVAL hart für P ist, müssen wir für jede Sprache $L \in \mathsf{P}$ eine Funktion $f \in \mathsf{FL}$ 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 $L \in \mathsf{P}$ 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 $l = \|\Gamma\| + \|(Q \cup \{q_{\text{h}}, q_{\text{ja}}, q_{\text{nein}}\}) \times \Gamma\|$ verschiedene Tabelleneinträge besitzt, können wir jeden Eintrag $T_{i,j}$ durch eine Bitfolge $t_{i,j,1} \cdots t_{i,j,m}$ der Länge $m = \lceil \log_2 l \rceil$ 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 $c_{i,j,k} = t_{i,j,k}$, $k = 1, \ldots, m$, die diese Einträge kodieren.

- Für jeden der $(n^c + c - 1)(n^c + c)$ übrigen Einträge $T_{i,j}$ enthält c_x für $k = 1, \ldots, m$ 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 $i = 1, \ldots, n^c + c$ zeigen, dass die von den Schaltkreisen $c_{i,j,k}$, $j = 1, \ldots, n^c + c$, $k = 1, \ldots, m$ berechneten Werte die Tabelleneinträge $T_{i,j}$, $j = 1, \ldots, n^c + c$, 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 $c_1, \ldots, c_k$ erzeugt und diese Kopien richtig verdrahtet. ■

Eine leichte Modifikation des Beweises von Satz 51 liefert folgendes Resultat.

Korollar 52. *Sei $L \subseteq \{0,1\}^*$ eine beliebige Sprache in* P*. Dann existiert eine Funktion $f \in$* FL*, die bei Eingabe 1^n einen Schaltkreis c_n mit n Eingängen berechnet, so dass für alle $x \in \{0,1\}^n$ gilt:*

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

Korollar 52 besagt insbesondere, dass es für jede Sprache $L \subseteq \{0,1\}^*$ in P eine Schaltkreisfamilie $(c_n)_{n\geq 0}$ polynomieller Größe gibt, so dass c_n für alle Eingaben $x \in \{0,1\}^n$ die charakteristische Funktion von L berechnet.

Korollar 53 (Savage 1972)**.**
Es gilt P $\subseteq$ 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 P $\neq NP$.

Selbst für NEXP ist die Inklusion in PSK offen. Dagegen zeigt ein einfaches Diagonalisierungsargument, dass in EXPSPACE Sprachen mit superpolynomieller Schaltkreiskomplexität existieren. Wir werden später sehen, dass bereits die Annahme NP $\subseteq$ PSK schwerwiegende Konsequenzen für uniforme Komplexitätsklassen hat.

Es ist nicht schwer zu sehen, dass die Inklusion P $\subsetneq$ PSK echt ist. Hierzu betrachten wir Sprachen über einem einelementigen Alphabet.

Definition 54. *Eine Sprache T 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 55. TALLY $\subseteq$ PSK.

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

Korollar 56. PSK $\not\subseteq$ RE.

5.4 NP-vollständige Probleme

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

Definition 57. *Für $k \geq 1$ sei $\Gamma_k = \{0, \ldots, k-1\}$ das Alphabet, das die Ziffern $0, \ldots, k-1$ als Zeichen enthält. Zudem sei Σ ein Alphabet, das für ein $k \geq 1$ das Alphabet Γ_k und nicht das Zeichen k enthält.*

a) *Für $B \subseteq \Sigma^*$ ist die Sprache $\exists B$ definiert durch*

$$\exists B = \{x \in \Sigma^* \,|\, \exists y \in \Gamma_k^* : x\#y \in B\}$$

*Jedes $y \in \Gamma_k^*$ mit $x\#y \in B$ wird auch als **Zeuge** (engl. witness, certificate) für die Zugehörigkeit von x zu $\exists B$ bezeichnet.*

b) *Sei q ein Polynom. B heißt **(k,q)-balanciert** (oder einfach **q-balanciert** bzw. **balanciert**), falls B nur Strings der Form $x\#y$ mit $y \in \Gamma_k^{q(|x|)}$ enthält. Falls B balanciert ist, schreiben wir für $\exists B$ auch $\exists^p B$.*

c) *Für eine Sprachklasse C seien $\exists \cdot \mathsf{C}$ und $\exists^p \cdot \mathsf{C}$ definiert durch*

$$\exists \cdot \mathsf{C} = \{\exists B \,|\, B \in \mathsf{C}\} \text{ und } \exists^p \cdot \mathsf{C} = \{\exists^p B \,|\, B \in \mathsf{C} \text{ ist balanciert}\}$$

Satz 58. $\mathsf{NP} = \exists^p \cdot \mathsf{P}$.

Beweis. Zu jeder NP-Sprache $A \subseteq \Sigma^*$ existiert eine NTM M, die A in Zeit $q(n)$ für ein Polynom q entscheidet. Sei $k \geq 1$ der maximale Verzweigungsgrad von N. Dann können wir jeder Eingabe $x \in \Sigma^*$ der Länge n und jedem String $y = y_1 \cdots y_{q(n)} \in \Gamma_k^{q(n)}$ eindeutig eine Rechnung $M_y(x)$ von $M(x)$ zuordnen, indem wir im i-ten Rechenschritt aus den $c_i \geq 1$ zur Auswahl stehenden Folgekonfigurationen $K_0, \ldots, K_{c_i-1}$ diejenige mit dem Index y_i wählen (in den Fällen $y_i \geq c_i \geq 1$ und $y_i > c_i = 0$ sei $M_y(x) = \text{nein}$). Nun ist leicht zu sehen, dass

$$B = \{x\#y \,|\, x \in \Sigma^*, y \in \Gamma_k^{q(|x|)} \text{ und } M_y(x) = \text{ja}\}$$

eine (k, q)-balancierte Sprache in P mit $L = \exists^p B$ ist.

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

Satz 59. 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 $f \in \mathsf{FL}$ 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 58 haben wir gezeigt, dass für jede NP-Sprache A eine (k, q)-balancierte Sprache $B \subseteq \Sigma^*$ in P mit

$$x \in A \Leftrightarrow \exists y \in \Gamma_k^{q(|x|)} : x\#y \in B.$$

Sei $m = \max\{1, \lceil \log_2 \|\Sigma\| \rceil\}$. Da die Binärsprache $bin(B)$ in P entscheidbar ist, existiert nach Korollar 52 eine FL-Funktion f, die einen Schaltkreis $f(1^n) = c_n$ mit $m(n + 1 + q(n))$ Eingängen berechnet, so dass für alle $z \in \{0,1\}^{m(n+1+q(n))}$ gilt:

$$z \in bin(B) \Leftrightarrow c_n(z) = 1.$$

Betrachte nun die Funktion g, die bei Eingabe x den Schaltkreis c_x mit $mq(n)$ Eingängen 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, $|x| = n$,

$$\begin{aligned} x \in A \quad &\Leftrightarrow \quad \exists y \in \Gamma_k^{q(n)} : c_n(bin(x\#y)) = 1 \\ &\Leftrightarrow \quad \exists y \in \Gamma_k^{q(n)} : c_x(bin(y)) = 1 \\ &\Leftrightarrow \quad \exists y \in \{0,1\}^{mq(n)} : c_x(y) = 1 \\ &\Leftrightarrow \quad c_x \in \text{CIRSAT} \end{aligned}$$

■

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 60. *Eine boolesche Formel F über den Variablen $x_1, \dots, x_n$ 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** $l_{ij} \in \{x_1, \dots, x_n, \bar{x}_1, \dots, \bar{x}_n\}$ ist. Hierbei verwenden wir $\bar{x}$ als abkürzende Schreibweise für $\neg x$. Gilt $k_i \le k$ für $i = 1, \dots, m$, 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 $C = \{l_1, \dots, l_k\}$ der zugehörigen Literale und KNF-Formeln als Menge $F = \{C_1, \dots, C_m\}$ ihrer Klauseln dargestellt.

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

Gegeben: Eine boolesche Formel in k-KNF.
Gefragt: Ist F erfüllbar?

Beispiel 61. *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 62. 3-Sat *ist* NP*-vollständig.*

Beweis. Es ist leicht zu sehen, dass 3-Sat $\in$ 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 $c = \{g_1, \dots, g_m\}$ mit n Eingängen in eine 3-KNF-Formel F_c mit $n + m$ Variablen $x_1, \dots, x_n, y_1, \dots, y_m$, wobei y_i den Wert des Gatters g_i wiedergibt. 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.

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

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 P $=$ 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-Satisfiability (NaeSat):

Gegeben: Eine Formel F in 3-KNF.
Gefragt: Existiert eine Belegung für F, unter der in jeder Klausel beide Wahrheitswerte angenommen werden?