Vorlesungsskript

Komplexitätstheorie

Wintersemester 2006/07

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

6. Februar 2007

Inhaltsverzeichnis

1 Einführung

In der Komplexitätstheorie werden algorithmische Probleme daraufhin untersucht, wieviel 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 $V = \{1, \ldots, n\}$ und $E \subseteq V \times V$.
>
> ***Gefragt:*** Gibt es in G einen Weg von Knoten 1 zu Knoten n?

Zur Erinnerung: Eine Folge $(v_1, \ldots, v_k)$ von Knoten heißt **Weg** in G, falls für $j = 1, \ldots, k-1$ 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:

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

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.

Um REACH zu entscheiden, markieren wir nach und nach alle Knoten, die vom Knoten 1 aus erreichbar sind. Dabei speichern wir alle markierten Knoten solange in einer Menge S, bis wir auch deren Nachbarknoten markiert haben. Genaueres ist folgendem Algorithmus zu entnehmen:

1 **Eingabe:** $G = (V, E)$

```
2    S ← {1}
3    markiere 1
4    repeat
5      wähle Knoten u ∈ S
6      S ← S − {u}
7      for all (u, v) ∈ E do
8        if v ist nicht markiert  then
9          markiere v
10         S ← S ∪ {v}
11       end
12     end
13   until S = ∅
14   if n ist markiert then accept  else reject end
```

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 (vgl. Definition 2.12).

Diskussion: (informal)

- 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 $G = (V, E)$, $V = \{1, \ldots, n\}$, $E \subseteq V^2$ mit einer Kapazitätsfunktion $c : E \to \mathbb{N}$.

Gesucht: Ein Fluss $f : E \to \mathbb{N}$ von 1 nach n in G, d.h.

- $\forall e \in E : f(e) \leq c(e)$
- $\forall v \in V - \{1, n\} : \sum\limits_{(v,u)\in E}^{f} (v, u) = \sum\limits_{(u,v)\in E}^{f} (u, v)$

mit maximalem Wert $w(f) := \sum\limits_{(1,v)\in E}^{f} (1, v) = \sum\limits_{(v,n)\in E}^{f} (v, n)$.

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

Diskussion: (informal)

- 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 $G = (U, V, E)$, $U = V = \{1, \ldots, n\}$, $E \subseteq U \times V$.

Gesucht: 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**.

Diskussion: (informal)

- MATCHING ist in Zeit n^3 lösbar.
- MATCHING ist in Platz n^2 lösbar.

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 $\pi(n+1) := \pi(1)$.

Diskussion: (informal)

- 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* lässt sich TSP in Zeit $n^2 \cdot 2^n$ lösen, der Platzverbrauch erhöht sich dabei jedoch auf $n \cdot 2^n$.

* Hierzu berechnen wir für alle Teilmengen $S \subseteq \{2, \ldots, n\}$ 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 2.1 (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 2.2 *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, sowie*
- *sich der Kopf auf Band i auf dem ersten Zeichen von v_i befindet.*

Definition 2.3 *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. $D_i = L$, $u_i = u_i' b_i$ *und* $v_i' = b_i a_i' v_i$,
2. $D_i = R$, $u_i' = u_i a_i'$ *und* $v_i' = \begin{cases} \sqcup, & v_i = \varepsilon, \\ v_i, & \textit{sonst}, \end{cases}$
3. $D_i = N$, $u_i' = u_i$ *und* $v_i' = a_i' v_i$.

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 2.4 *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 2.5 *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 2.6 *Eine DTM M **hält bei Eingabe** $x \in \Sigma^*$ **im Zustand** q (kurz: $M(x)$ hält im Zustand q), falls 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,*

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

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 2.7 *Die von einer* DTM M ***akzeptierte Sprache** ist*

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

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

Definition 2.8 *Sei $L \subseteq \Sigma^*$. Eine* DTM M ***entscheidet*** *L, falls für alle $x \in \Sigma^*$ gilt:*

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

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

Definition 2.9 *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 2.10 *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*

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

eine Funktion mit der Eigenschaft ist, dass im Fall $(q', a'_1, D_1, \ldots, a'_k, D_k) \in \delta(q, a_1, \ldots, a_k)$ und $a_i = \rhd$ immer $a'_i = \rhd$ 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 $(q, a_1, \ldots, a_k) \mapsto (q', a'_1, D_1, \ldots, a'_k, D_k)$).
Wir werden NTMs nur zum Erkennen von Sprachen (d.h. als Akzeptoren) und nicht zum Berechnen von Funktionen benutzen.

Definition 2.11

a) Sei M eine NTM. *$M(x)$* ***akzeptiert****, falls $M(x)$ nur endlich lange Rechnungen ausführt und eine akzeptierende Endkonfiguration K existiert mit $K_x \to^* K$. Akzeptiert $M(x)$ nicht und hat $M(x)$ nur endlich lange Rechnungen, so* ***verwirft*** *$M(x)$. Falls unendlich lange Rechnungen existieren, ist $M(x) = \uparrow$ (undefiniert).*

b) M ***akzeptiert*** *die Sprache $L(M) = \{x \in \Sigma^* \mid M(x)$ akzeptiert$\}$. 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 2.12

$a)$ *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 $\max \mathbb{N} := \infty$.*

$b)$ *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

$$\begin{aligned}\mathrm{DTIME}(t(n)) &= \{L(M) \mid M \text{ ist eine } t(n)\text{-zeitbeschränkte DTM}\} \text{ bzw.}\\ \mathrm{NTIME}(t(n)) &= \{L(M) \mid M \text{ ist eine } t(n)\text{-zeitbeschränkte NTM}\}\end{aligned}$$

zusammen. Ferner sei

$$\mathrm{FTIME}(t(n)) = \{f \mid f \text{ wird von einer } t(n)\text{-zeitbeschränkten DTM berechnet}\}.$$

Die wichtigsten Zeitkomplexitätsklassen sind

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

Die Klassen $\mathrm{NP}, \mathrm{NE}, \mathrm{NEXP}$ und $\mathrm{FP}, \mathrm{FE}, \mathrm{FEXP}$ 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 dürfen. Der Grund für diese Einschränkungen liegt darin, sinnvolle Definitionen für Komplexitätsklassen mit einem sublinearen Platzverbrauch zu erhalten.

Definition 2.13 *Eine **Offline-TM** ist eine TM M mit der Eigenschaft, dass für jede Anweisung $(q, a_1, \ldots, a_k) \mapsto (q', a'_1, D_1, \ldots, a'_k, D_k)$ von M*

$$a'_1 = a_1 \text{ und } a_1 = \sqcup \Rightarrow D_1 = L$$

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

Dies bedeutet, dass eine Offline-Turingmaschine 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 Summe aller während einer Rechnung besuchten Bandfelder.

Definition 2.14

$a)$ *Sei M eine Turingmaschine 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-*Turingmaschine *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

$\mathrm{DSPACE}(s(n)) = \{L(M) \mid M \text{ ist eine } s(n)\text{-platzbeschränkte Offline-DTM}\}$ bzw.
$\mathrm{NSPACE}(s(n)) = \{L(M) \mid M \text{ ist eine } s(n)\text{-platzbeschränkte Offline-NTM}\}$

zusammen. Ferner sei

$\mathrm{FSPACE}(s(n)) := \{f \mid f \text{ wird von einem } s(n)\text{-platzbeschränkten Transducer berechnet}\}$.

Die wichtigsten Platzkomplexitätsklassen sind

$$\begin{aligned}
\mathrm{L} &= \mathrm{DSPACE}(\log n) \\
\mathrm{L}^c &= \mathrm{DSPACE}(\log^c n) \\
\mathrm{LINSPACE} &= \mathrm{DSPACE}(O(n)) \\
\mathrm{PSPACE} &= \mathrm{DSPACE}(n^{O(1)}) \\
\mathrm{ESPACE} &= \mathrm{DSPACE}(2^{O(n)}) \\
\mathrm{EXPSPACE} &= \mathrm{DSPACE}(2^{n^{O(1)}})
\end{aligned}$$

Die Klassen $\mathrm{NL}, \mathrm{NLINSPACE}$ und $\mathrm{NPSPACE}$, sowie $\mathrm{FL}, \mathrm{FLINSPACE}$ und $\mathrm{FPSPACE}$ sind analog definiert, wobei $\mathrm{NPSPACE}$ 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. Zuvor befassen wir uns jedoch mit Robustheitseigenschaften dieser Klassen.

3.1 Robustheit von Komplexitätsklassen

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

Lemma 3.1 (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:

$$K_x = (q_0', \varepsilon, \triangleright x, \varepsilon, \triangleright) \underset{M'}{\longrightarrow} (q_1', \triangleright, x, \triangleright, \sqcup) \underset{M'}{\longrightarrow} (q_2', \varepsilon, \triangleright x, \triangleright, \begin{pmatrix} \hat{\triangleright} \\ \vdots \\ \hat{\triangleright} \end{pmatrix})$$

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.

Offensichtlich gilt nun $L(M') = L(M)$ und $space_{M'}(x) \leq space_M(x)$. ■

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 3.2 (Lineare Platzkompression und Beschleunigung)

Für alle $c > 0$ gilt

$i)$ $\mathrm{DSPACE}(s(n)) \subseteq \mathrm{DSPACE}(2 + c \cdot s(n))$ ***(linear space compression),***

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

Beweis $i)$ Sei $L \in \mathrm{DSPACE}(s(n))$ 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

$$j = \begin{cases} i, & D_2 = N \\ i+1, & D_2 = R, i < m \\ 1, & D_2 = R, i = m \\ m, & D_2 = L, i = 1 \\ i-1, & D_2 = L, i > 1 \end{cases}$$

und

$$D_2' = \begin{cases} N, & D_2 = N \text{ oder } D_2 = R, i < m \text{ oder } D_2 = L, i > 1 \\ L, & D_2 = L, i = 1 \\ R, & D_2 = R, i = m \end{cases}$$

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. Außerdem gilt

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

$ii)$ $L \in \mathrm{DTIME}(t(n))$ und sei $M = (Q, \Sigma, \Gamma, \delta, q_0)$ eine $t(n)$-zeitbeschränkte k-DTM mit $L(M) = L$, wobei wir $k \geq 2$ annehmen. Wir konstruieren eine DTM M' mit $L(M') = L$ und $time_{M'}(x) \leq 2 + |x| + c \cdot time_M(x)$. 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 $x = x_1 \dots x_n$ 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: Sobald M in einen der Zustände q_{ja} bzw. q_{nein} wechselt, tut M' dies ebenfalls.

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

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

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 3.3

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

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

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

Beweis *i)* Sei $L \in \text{DSPACE}(cs(n))$ 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 $L \in \mathrm{DTIME}(ct(n))$ 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 $L \in \mathrm{DTIME}(cn)$ 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 3.4 (Beziehungen zwischen det. und nichtdet. Zeit- und Platzklassen)

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

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

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

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

der maximale Verzweigungsgrad von N. Dann ist jede Rechnung

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

der Länge t von $N(x)$ eindeutig durch eine Folge $(d_1, \ldots, d_t) \in \{1, \ldots, d\}^t$ beschreibbar. Um N zu simulieren, generiert M auf dem Band 2 für $t = 1, 2, \ldots$ der Reihe nach alle Folgen $(d_1, \ldots, d_t) \in \{1, \ldots, d\}^t$. 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

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

beschränkt ist.
ii) Sei $L \in \mathrm{NSPACE}(s(n))$ 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 3.5 $s(n) \geq \log n \Rightarrow \mathrm{NSPACE}(s(n)) \subseteq \mathrm{DTIME}(2^{O(s(n))})$.

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

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

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

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

Insbesondere erhalten wir somit die Inklusionskette

$$\mathrm{L} \subseteq \mathrm{NL} \subseteq \mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} \subseteq \mathrm{NPSPACE} \subseteq \mathrm{EXP} \subseteq \mathrm{NEXP} \subseteq \mathrm{EXPSPACE} \subseteq \ldots$$

Des weiteren impliziert Satz 3.2 die beiden Inklusionen

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

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 3.6 *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 3.7 (Satz von Savitch, 1970)

Für jede echte Komplexitätsfunktion $s(n) \geq \log n$ gilt

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

Beweis Sei $L \in \mathrm{NSPACE}(s)$ und sei N eine Offline-2-NTM , die L in Platz $s(n)$ entscheidet. Wie im Beweis von Satz 3.4 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 die Platz höchstens $s(n)$ benötigen. Dann ist leicht zu sehen, dass für zwei Konfigurationen K, $\hat{K}$ und eine Zahl i folgende Äquivalenz gilt:

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

Nun können wir N durch folgende Offline-3-DTM M simulieren.

Initialisierung: Bei Eingabe x schreibt M 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 $\hat{K}_x = (q_{\text{ja}}, 2, \triangleright, \sqcup \ldots \sqcup)$). 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 $K \underset{N}{\longrightarrow}^{\leq 2^i} \hat{K}$ 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. Falls $i = 0$ ist, testet M, ob $K \underset{N}{\longrightarrow}^{\leq 1} \hat{K}$ gilt und gibt die Antwort zurück. Andernfalls ($i > 0$) fügt M für wachsendes $j = 1, 2, \ldots$ 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 (falls $j < c^{s(n)}$ 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 wird es durch das Tripel $(K_j, \hat{K}, i-1)$ ersetzt und die zugehörige Antwort berechnet. Ist diese negativ, so löscht M das Tripel $(K, K_j, i-1)$ und fährt mit dem Tripel $(K, K_{j+1}, i-1)$ fort. Erfährt dagegen $(K_j, \hat{K}, i-1)$ eine positive Antwort, so löscht M dieses Tripel und beantwortet $(K, \hat{K}, i)$ positiv.

Akzeptanzverhalten: M akzeptiert, falls das Tripel $(K_x, \hat{K}_x, \lceil s(|n|) \log c \rceil)$ positiv beantwortet wird.

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 3.8

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

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

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

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

3.4 Der Satz von Immerman und Szelepcsényi

Definition 3.9

$a)$ *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 3.10

1) Die zu NP komplementäre Klasse ist $\text{co-NP} = \{L | \overline{L} \in \mathrm{NP}\}$. 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 $\mathrm{NP} = \text{co-NP}$ gilt), ist ähnlich wie das $\mathrm{P} \stackrel{?}{=} \mathrm{NP}$-Problem ungelöst.

2) Dagegen wurde die Frage ob die Klasse $\mathrm{CSL} = \mathrm{NSPACE}(n)$ der kontextsensitiven Sprachen unter Komplementbildung abgeschlossen ist, in den 80ern gelöst (siehe Satz von Immerman und Szelepcsényi), d.h. es gilt $\mathrm{CSL} = \text{co-CSL}$.

3) Andererseits ist $\text{co-CFL} \neq \mathrm{CFL}$. Dies folgt aus der Tatsache, dass kontextfreie Sprachen zwar unter Vereinigung abgeschlossen sind, aber nicht unter Schnitt. ◁

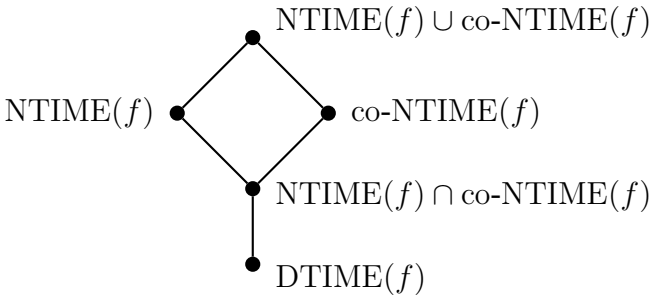
Da sich deterministische Rechnungen leicht komplementieren lassen (durch einfaches Vertauschen der Zustände q_{ja} und q_{nein}), sind deterministische Komplexitätsklassen unter Komplementbildung abgeschlossen.

Proposition 3.11

$i)$ co-DSPACE$(s(n))$ = DSPACE$(s(n))$,

$ii)$ co-DTIME$(t(n))$ = 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 3.12 *Eine* NTM N *heißt* ***strong*** *bei Eingabe* x*, falls es entweder akzeptierende oder verwerfende Rechnungen bei Eingabe* x *gibt (aber nicht beides zugleich).*

Satz 3.13 (Satz von Immerman und Szelepcsényi, 1987)

Für jede echte Komplexitätsfunktion $s(n) \geq \log n$ *gilt*

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

Beweis Sei $L \in \text{NSPACE}(s)$ und sei N eine $s(n)$-platzbeschränkte Offline-NTM mit $L(N) = L$. Wir konstruieren eine $O(s(n))$-platzbeschränkte Offline-NTM N' mit $L(N') = \overline{L}$. 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

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

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

$$L_0 = \{(x, r, t, K) | K_x \underset{N}{\rightarrow}^{\leq t} K\}.$$

Behauptung 1 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 von Beh. 1 $N_0(x, r, t, K)$ 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 $K_i \underset{N}{\rightarrow}^{\leq 1} K$ 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}.

Algorithmus 3.14 $N_0(x, r, t, K)$

```
1  c ← 0
2  for each Konfiguration K_i do
3    rate eine Rechnung α der Länge ≤ t − 1 von N(x)
4    if α endet in K_i then
5      c ← c + 1
6      if K_i ⟶_N^{≤1} K then halte im Zustand q_ja end
7    end
8  end
9  if c = r then halte im Zustand q_nein else halte im Zustand q_h end
```

Da N_0 genau dann eine akzeptierende Rechnung hat, wenn eine Konfiguration K_i mit $K_x \underset{N}{\longrightarrow}^{\leq t-1} K_i$ und $K_i \underset{N}{\longrightarrow}^{\leq 1} K$ 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 Eingaben der Form $x_0 = (x, r(x, t-1), t, K)$ strong ist. N_0 also 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 $K_x \underset{N}{\longrightarrow}^{\leq t-1} K_i$ aber nicht $K_i \underset{N}{\longrightarrow}^{\leq 1} K$ 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 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.

Algorithmus 3.15 $N'(x)$

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

14 halte im Zustand q_{ja}

Behauptung 2 Im t-ten Durchlauf der repeat-Schleife (Zeile 3) wird r^- 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 von Beh. 2 Wir beweisen Beh. 2 durch 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^- vor dem $t+1$-ten Durchlauf der for-Schleife auf den Wert von r gesetzt wird, müssen wir zeigen, dass r im t-ten Durchlauf der for-Schleife auf $r(x, t)$ hochgezählt wird. Nach Induktionsvoraussetzung wird N_0 im t-ten Durchlauf der for-Schleife 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 $\leq t$ Schritten erreichbaren Konfigurationen K_i als solche erkannt und somit wird r tatsächlich auf den Wert $r(x, t)$ hochgezählt. □

Behauptung 3 Beim Verlassen der repeat-Schleife gilt $r = r^- = \|\{K | K_x \underset{N}{\longrightarrow}^* K\}\|$.

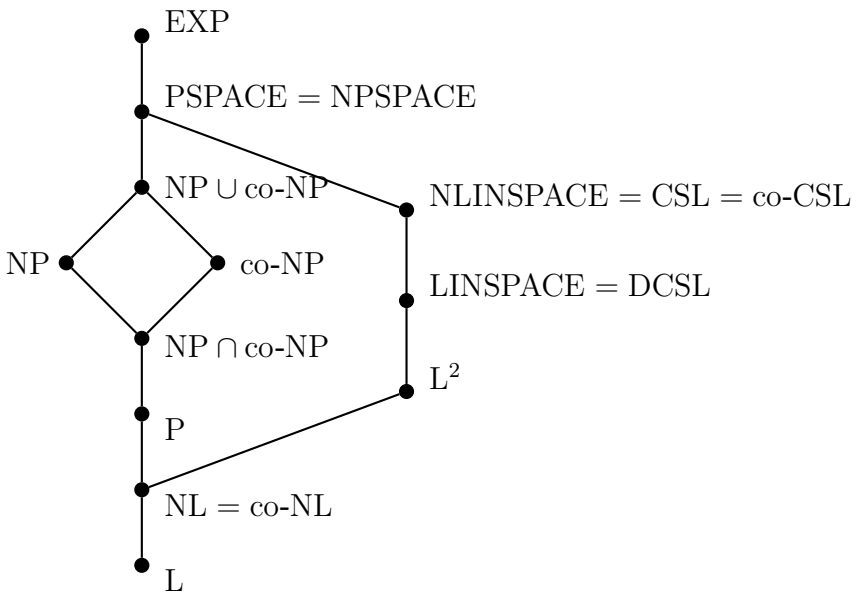
Beweis von Beh. 3 Wir wissen bereits, dass im t-ten Durchlauf der repeat-Schleife r den Wert $r(x, t)$ und r^- den Wert $r(x, t-1)$ erhält. Wird daher die repeat-Schleife nach t_e Durchläufen verlassen, so gilt $r = r^- = r(x, t_e) = r(x, t_e - 1)$.
Angenommen $r(x, t_e) < \|\{K | K_x \underset{N}{\longrightarrow}^* K\}\|$. 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 $K_x = K_0 \underset{N}{\longrightarrow} K_1 \underset{N}{\longrightarrow} \ldots \underset{N}{\longrightarrow} K_{t'} = K$ minimaler Länge, die in K endet. Dann gilt $K_x \underset{N}{\longrightarrow}^{t_e} K_{t_e}$, aber nicht $K_x \underset{N}{\longrightarrow}^{\leq t_e - 1} K_{t_e}$ und daher folgt $r(x, t_e) > r(x, t_e - 1)$. Widerspruch! □

Da N' Platz $O(s(n))$ benötigt, bleibt nur noch die Gleichheit $L(N') = \overline{L}$ zu zeigen. Wegen Behauptung 3 akzeptiert N' eine Eingabe x nur dann, wenn im letzten Durchlauf der repeat-Schleife alle erreichbaren Konfigurationen K als solche erkannt werden und darunter keine akzeptierende Endkonfiguration ist. Dies impliziert $x \notin L$. Die Umkehrung folgt analog. ■

Korollar 3.16

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

Damit ergibt sich folgende Inklusionsstruktur für die wichtigsten bisher betrachteten Komplexitätsklassen:



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 $Q = \{q_0, q_1, \ldots, q_m\}$ und $\Gamma = \{a_1, \ldots, a_l\}$. Dann kodieren wir Zustände und Zeichen α wie folgt durch Binärzahlen $c(\alpha)$ der Länge $b = \lceil \log_2(\|Q\| + \|\Gamma\| + 6)\rceil = \lceil \log_2(m + l + 7)\rceil$:

Zustand bzw. Zeichen α	Binärkodierung $c(\alpha)$
q_i, $i = 0, \ldots, m$	$\mathit{bin}_b(i)$
a_j, $j = 1, \ldots, l$	$\mathit{bin}_b(m + j)$
$q_{\mathrm{h}}, q_{\mathrm{ja}}, q_{\mathrm{nein}}, L, R, N$	$\mathit{bin}_b(m + l + 1), \ldots, \mathit{bin}_b(m + l + 6)$

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. 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 $x \in \Sigma^*$ kodieren wir durch das Wort $\langle M, x\rangle = \langle M\rangle\#c(x)$, wobei $c(x)$ die Binärkodierung $c(x_1)\cdots c(x_n)$ der Eingabe $x = x_1 \cdots x_n$ über dem Eingabealphabet Σ von M ist.

Definition 4.1 *Das **Halteproblem** ist*

$$H := \{\langle M, x\rangle | M(x) \textit{ hält}\}.$$

Satz 4.2 *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

$$D = \{\langle M\rangle | M(\langle M\rangle) \text{ verw.}\} \quad (*)$$

entscheidbar. Dann können wir aber eine Turingmaschine M_d konstruieren, die eine Eingabe $\langle M \rangle$ genau dann akzeptiert, wenn $M(\langle M \rangle)$ verwirft,

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

M_d verhält sich also 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	$\Uparrow$	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 es in der Matrix keine Zeile für die Sprache $L(M_d) = D$ geben:

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

■

Satz 4.3 *Für jede rekursive Funktion $f : \mathbb{N} \longrightarrow \mathbb{N}$ existiert eine rekursive Sprache $D_f \notin \mathrm{DTIME}(f(n))$.*

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 $D_f \in \mathrm{DTIME}(f(n))$ 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:

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

■

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 4.4 (Gap Theorem)

Es gibt eine rekursive Funktion $f : \mathbb{N} \to \mathbb{N}$ mit

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

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

$$\forall x \in \Sigma^* : time_M(x) \leq 2^{f(|x|)} \Rightarrow \text{für fast alle } x \in \Sigma^* : time_M(x) \leq f(|x|).$$

Betrachte hierzu das Prädikat

$$P(k,t) : t \geq k+2 \text{ und für } i = 1, \ldots, k \text{ und } \forall x \in \Sigma_i^k : time_{M_i}(x) \notin [t+1, 2^t].$$

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

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

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

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

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

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

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

Korollar 4.7

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

Beweis

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

■

Aus dem Beweis von Satz 4.5 können wir weiterhin die Existenz einer universellen TM folgern.

Korollar 4.8 *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 4.9 *Mit Hilfe einer aufwendigeren 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

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

Für $g(n) = n^2$ erhalten wir beispielsweise die echten Inklusionen $\mathrm{DTIME}(g(n)) \subsetneq \mathrm{DTIME}(f(n))$ 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

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

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

Satz 4.10 (Platzhierarchiesatz) *Sei f eine echte Komplexitätsfunktion und gelte*

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

Dann ist

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

Damit lässt sich für Zeitschranken $g(n) \leq f(n)$ die Frage, ob die Inklusion von $\mathrm{DSPACE}(g(n))$ in $\mathrm{DSPACE}(f(n))$ 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 4.11

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

Durch Kombination der Beweistechnik von Satz 4.10 mit der Technik von Immerman und Szelepcsényi erhalten wir auch für nichtdeterministische Platzklassen eine sehr fein abgestufte Hierarchie.

Satz 4.12 (nichtdeterministischer Platzhierarchiesatz) *Sei f eine echte Komplexitätsfunktion und gelte*

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

Dann ist

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

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

Satz 4.13 (nichtdeterministischer Zeithierarchiesatz) *Sei f eine echte Komplexitätsfunktion und gelte*

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

Dann ist

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

5 Reduktionen

5.1 Logspace-Reduktionen

Oft können wir die Komplexität zweier Probleme A und B dadurch miteinander vergleichen, dass wir das Lösen der Frage $x \in A$ auf eine Frage der Form $y \in B$ zurückführen. Erfordert die Berechnung von y nur einen relativ geringen Rechenaufwand, so lässt sich jeder Algorithmus für B in einen Algorithmus für A umwandeln, der nur unwesentlich mehr Rechenressourcen benötigt.

Definition 5.1 *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 $f \in \mathrm{FL}$ existiert, so dass für alle $x \in \Sigma^*$ gilt,*

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

Lemma 5.2 $\mathrm{FL} \subseteq \mathrm{FP}$.

Beweis Sei $f \in \mathrm{FL}$. Dann ist die Sprache

$$L_f = \{\langle x, i, b\rangle \mid \text{das } i\text{-te Zeichen von } f(x) \text{ ist } b\}$$

in L und wegen $\mathrm{L} \subseteq \mathrm{P}$ auch in Polynomialzeit entscheidbar. Da auf logarithmischem Platz nur Rechnungen polynomieller Länge ausgeführt werden können, gilt zudem

$$|f(x)| = |x|^{O(1)}.$$

Folglich ist f in FP berechenbar. ■

Beispiel 5.3 Es ist nicht schwer, das Hamiltonkreisproblem

Hamiltonkreisproblem (HAM):

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

auf das Erfüllbarkeitsproblem SAT für boolesche Formeln

Erfüllbarkeitsproblem für boolesche Formeln (SAT):

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

zu reduzieren. Hierzu benötigen wir eine Funktion $f \in \mathrm{FL}$, 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 $n + n\binom{n}{2} + n + n(\binom{n}{2} - \|E\|)$ Klauseln, so ist leicht zu sehen, dass

- G genau dann einen Hamiltonkreis besitzt, wenn F_G erfüllbar ist, und
- die Reduktionsfunktion $f : G \mapsto F_G$ in FL berechenbar ist. ◁

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

Definition 5.4

*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 5.5

1. *Die $\leq_m^{log}$-Reduzierbarkeit ist reflexiv und transitiv.*

2. *Die Klassen* $\mathrm{L}, \mathrm{NL}, \mathrm{NP}, \text{co-NP}, \mathrm{PSPACE}, \mathrm{EXP}$ *und* $\mathrm{EXPSPACE}$ *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 5.6 *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	$g_l(a)$
0	0
1	1
x_i	a_i
$(\neg, j)$	$1 - g_j(a)$
$(\wedge, j, k)$	$g_j(a)g_k(a)$
$(\vee, j, k)$	$g_j(a) + g_k(a) - g_j(a)g_k(a)$

c *berechnet die boolesche Funktion* $c(a) = g_m(a)$*.* c *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 **Fanin** 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 5.7 Für die Reduktion REACH $\leq$ CIRVAL benötigen wir eine Funktion $f \in \mathrm{FL}$ mit der Eigenschaft, dass für alle Graphen G gilt,

$$G \in \text{REACH} \Leftrightarrow f(G) \in \text{CIRVAL}.$$

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

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

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

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

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 5.7 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 5.8 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 $L \in \mathrm{P}$ 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 $L \in \mathrm{P}$ existiert eine 1-DTM $M = (Q, \Sigma, \Gamma, \delta, q_0)$, die L in Zeit n^c entscheidet. Wir beschreiben die Rechnung von $M(x)$, $|x| = n$, durch eine Tabelle $T = (T_{i,j})$, $(i,j) \in \{1, \ldots, n^c\} \times \{1, \ldots, n^c + 2\}$, mit

$$T_{i,j} \quad = \quad \begin{cases} (q_i, a_{i,j}), & \text{nach } i \text{ Schritten liest } M \text{ das } j\text{-te Zeichen auf dem Band,} \\ a_{i,j}, & \text{sonst,} \end{cases}$$

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 $i < n^c$ erreichte Endkonfiguration bis zum Zeitpunkt $i = n^c$ bei.

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

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

Da T nicht mehr als $\|\Gamma\| + \|\mathbb{Q} \times \Gamma\|$ verschiedene Tabelleneinträge besitzt, können wir jeden Eintrag $T_{i,j}$ durch eine Bitfolge $t_{i,j,1} \cdots t_{i,j,m}$ der Länge $m = \lceil \log_2 \|\Gamma\| + \|\mathbb{Q} \times \Gamma\| \rceil$ kodieren.
Da sich der Eintrag $T_{i,j}$ im Fall $i \in \{2, \ldots, n^c\}$ und $j \in \{2, \ldots, n^c + 1\}$ 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.

- Für jeden der $n^c + 2 + 2(n^c - 1) = 3n^c$ Randeinträge $T_{i,j}$ mit $i = 1$ oder $j \in \{1, n^c + 2\}$ 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 $(n^c - 1)n^c$ 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 $c_{n^c,2,1}$, 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 $i = 1, \ldots, n^c$ zeigen, dass die von den Schaltkreisen $c_{i,j,k}$, $j = 1, \ldots, n^c$, $k = 1, \ldots, m$ berechneten Werte die Tabelleneinträge $T_{i,j}$, $j = 1, \ldots, n^c$, kodieren. Wegen

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

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

Eine leichte Modifikation des Beweises von Satz 5.8 liefert uns folgendes Resultat.

Korollar 5.9 *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:*

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

Da c_n bei Eingabe 1^n in logarithmischen Platz berechenbar ist, hat c_n nur polynomielle Größe, d.h. polynomiell viele Gatter. Im Gegensatz zum **uniformen** Modell der Turingmaschine, die alle Instanzen eines gegebenen Problems entscheidet, stellen Schaltkreise ein **nichtuniformes** Berechnungsmodell dar, da für jede Eingabegröße n ein anderer Schaltkreis zum Einsatz kommt. Um eine unendliche Sprache zu entscheiden wird also eine ganze Familie von Schaltkreisen benötigt. Probleme, für die Schaltkreisfamilien polynomieller Größe existieren, werden zur Klasse PSK zusammengefasst.

Definition 5.10 *Eine Sprache $L \subseteq \{0,1\}^*$ hat **polynomielle Schaltkreiskomplexität** (kurz: $L \in$ PSK), falls es eine Folge von booleschen Schaltkreisen c_n, $n \geq 0$, mit n Eingängen und $n^{O(1)}$ Gattern gibt, so daß für alle $x \in \{0,1\}^*$ gilt:*

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

*Falls das Eingabealphabet Σ aus mehr als zwei Zeichen besteht, können wir jedes Zeichen $a \in \Sigma$ durch einen Binärstring bin(a) der Länge $m = \log_2 \|\Sigma\|$ kodieren. Eine Sprache $L \subseteq \Sigma^*$ hat **polynomielle Schaltkreiskomplexität**, wenn ihre Binärkodierung bin$(L) = \{$bin$(x_1) \cdots$ bin$(x_n) \mid x_1 \cdots x_n \in L\} \in$ PSK ist.*

Korollar 5.11 (Savage 1972) $\mathrm{P} \subseteq \mathrm{PSK}$.

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. Wir werden später sehen, dass auch die Annahme NP $\subseteq$ PSK

schwerwiegende Konsequenzen hat. Selbst für EXP ist die Inklusion in PSK offen. Dagegen zeigt ein einfaches Diagonalisierungsargument, dass in $\mathrm{EXPSPACE}$ Sprachen mit superpolynomieller Schaltkreiskomplexität existieren.

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

Definition 5.12 *Eine Sprache $T \subseteq \{0,1\}^*$ heißt **tally** (kurz: $T \in \mathrm{Tally}$), falls jedes Wort $x \in T$ die Form $x = 1^n$ hat.*

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

Proposition 5.13 $\mathrm{Tally} \subseteq \mathrm{PSK}$.

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 5.14 $\mathrm{PSK} \not\subseteq \mathrm{RE}$.

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 5.15

*a) Sei p ein Polynom. Eine Sprache 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

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

Jeder String y mit $x\#y \in B$ wird auch als **Zeuge** (engl. *witness, certificate*) für die Zugehörigkeit von x zur Sprache $A = \exists B$ bezeichnet.

Satz 5.16 (Zeugen-Charakterisierung von NP)

Eine Sprache A liegt genau dann in NP, wenn eine p-balancierte Sprache $B \in \mathrm{P}$ für ein Polynom p existiert mit $A = \exists B$, d.h.

$$\mathrm{NP} = \{\exists B \mid B \in \mathrm{P} \textit{ ist polynomiell balanciert}\}.$$

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 $B \in \mathrm{P}$, 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. ■

Theorem 5.17 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 $L \in \mathrm{NP}$ 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 5.16 haben wir gezeigt, dass für jede NP-Sprache $A \subseteq \Sigma^*$ eine p-balancierte Sprache $B \in \mathrm{P}$ 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 5.9 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:

$$x\#y \in B \Leftrightarrow c_n(\mathit{bin}_m(x_1) \cdots \mathit{bin}_m(x_n)\mathit{bin}_m(\#)y) = 1.$$

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 $\mathit{bin}_m(x_1) \cdots \mathit{bin}_m(x_n)\mathit{bin}_m(\#)$ vorgegebenen Werten ersetzt werden. Dann ist auch g in FL berechenbar und es gilt für alle Eingaben x, $|x| = n$,

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

■

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 5.18 *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 $k_i \leq k$ 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 5.19 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. ◁

Theorem 5.20 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 $\leq$ 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.

$$c \in \text{CIRSAT} \Leftrightarrow F_c \in \text{3-SAT}.$$

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

Theorem 5.21 $\mathrm{NAESAT} \in \mathrm{NPC}$.

Beweis $\mathrm{NAESAT} \in \mathrm{NP}$ ist klar. Wir zeigen $\mathrm{CIRSAT} \leq \mathrm{NAESAT}$ 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.

$$c \in \mathrm{CIRSAT} \Leftrightarrow F'_c \in \mathrm{NAESAT}.$$

Ist $c = (g_1, \ldots, g_m) \in \mathrm{CIRSAT}$, 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 $F'_c \in \mathrm{NAESAT}$, 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 5.22 *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?

Theorem 5.23 IS *ist* NP*-vollständig.*

Beweis Wir reduzieren 3-SAT auf IS. Sei

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

eine 3-KNF-Formel über den Variablen $x_1, \ldots, x_n$. 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

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

■

Korollar 5.24 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 5.25 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.

Theorem 5.26 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 1 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. Aufgrund der Symmetriebedingung $(**)$ existiert für jeden solchen Knoten l' ein Pfad von $\bar{l}'$ zu $\bar{l}$. Folglich können keine Konflikte auftreten, da

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

3) Falls α noch nicht auf allen Knoten definiert ist, gehe zu 1). □

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 NL = co-NL folgt 2-SAT $\in$ NL. ■

In den Übungen werden wir sehen, dass 2-SAT und REACH NL-vollständig sind.

6 Probabilistische Berechnungen

Eine **probabilistische Turingmaschine (PTM)** M ist genau so definiert wie eine NTM. Es wird jedoch ein anderes Akzeptanzkriterium benutzt. Wir stellen uns vor, dass M in jedem Rechenschritt zufällig einen Konfigurationsübergang wählt. Dabei wird jeder mögliche Übergang $K \to_M K'$ mit derselben Wahrscheinlichkeit

$$\Pr[K \to_M K'] = \begin{cases} \|\{K'' | K \to_M K''\}\|^{-1}, & K \to_M K' \\ 0, & \text{sonst.} \end{cases}$$

gewählt. Eine Rechnung $\alpha = (K_1, K_2, \ldots, K_m)$ wird also mit der Wahrscheinlichkeit

$$\Pr[\alpha] \;=\; \Pr[K_1 \to_M \cdots \to_M K_m] \;=\; \prod_{i=1}^{m-1} \Pr[K_i \to_M K_{i+1}]$$

ausgeführt. Die **Akzeptanzwahrscheinlichkeit** von $M(x)$ ist

$$\Pr[M(x) = \text{ja}] = \sum_{\alpha} \Pr[\alpha],$$

wobei sich die Summation über alle akzeptierenden Rechnungen α von $M(x)$ erstreckt. Wir vereinbaren für PTMs, dass $M(x)$ neben „ja“ noch die Werte „nein“ (falls sie verwirft) oder „?“ (in allen anderen Fällen) annehmen kann.

Definition 6.1

a) Die von einer PTM M *akzeptierte Sprache ist*

$$L(M) = \{x \in \Sigma^* \mid \Pr[M(x) = \text{ja}] \geq 1/2\}.$$

b) Eine Sprache $L \subseteq \Sigma^*$ gehört zur Klasse PP *(probabilistic polynomial time), falls eine polynomiell zeitbeschränkte* PTM *(*PPTM*)* M *mit* $L(M) = L$ *existiert.*

Theorem 6.2 $\mathrm{NP} \subseteq \mathrm{PP}$.

Beweis Sei $L \in \mathrm{NP}$ und sei N eine polynomiell zeitbeschränkte NTM mit $L(N) = L$. Fassen wir N als PPTM auf, so gilt für alle $x \in \Sigma^*$,

$$\begin{aligned} x \in L &\;\Rightarrow\; \Pr[N(x) = \text{ja}] \geq c^{-p(|x|)}, \\ x \notin L &\;\Rightarrow\; \Pr[N(x) = \text{ja}] = 0, \end{aligned}$$

wobei c der maximale Verzweigungsgrad und p eine polynomielle Zeitschranke für N ist. Betrachte folgende PPTM M, die bei Eingabe x zufällig eine der beiden folgenden Möglichkeiten wählt:

- M simuliert N bei Eingabe x,
- M führt eine probabilistische Rechnung aus, bei der sie mit Wahrscheinlichkeit $1 - c^{-p(|x|)}$ akzeptiert (z.B. indem sie einen Zufallsstring $s = s_1 \cdots s_{p(|x|)} \in \{1, \ldots, c\}^{p(|x|)}$ auf's Band schreibt und nur im Fall $s = 1^{p(|x|)}$ verwirft).

Dann gilt für alle $x \in \Sigma^*$,

$$\text{Pr}[M(x) = \text{ja}] = 1/2(\text{Pr}[N(x) = \text{ja}] + 1 - c^{-p(|x|)})$$

und somit

$$\begin{aligned} x \in L &\Rightarrow \text{Pr}[M(x) = \text{ja}] \geq 1/2(c^{-p(|x|)} + 1 - c^{-p(|x|)}) = 1/2, \\ x \notin L &\Rightarrow \text{Pr}[M(x) = \text{ja}] = 1/2(1 - c^{-p(|x|)}) < 1/2. \end{aligned}$$

■

Als nächstes zeigen wir, dass PP unter Komplementbildung abgeschlossen ist. Das folgende Lemma zeigt, wie sich eine PPTM, die sich bei manchen Eingaben indifferent verhält (also genau mit Wk $1/2$ akzeptiert) in eine äquivalente PPTM verwandeln lässt, die dies nicht tut.

Lemma 6.3 *Für jede Sprache $L \in \text{PP}$ existiert eine* PPTM *M mit $L(M) = L$, die bei keiner Eingabe $x \in \Sigma^*$ mit Wahrscheinlichkeit $1/2$ akzeptiert, d.h. für alle x gilt*

$$\text{Pr}[M(x) \neq L(x)] < 1/2,$$

wobei $L(x)$ für alle $x \in L$ gleich ja *und für alle $x \notin L$ gleich* nein *ist.*

Beweis Sei $L \in \text{PP}$ und sei N eine PPTM mit $L(N) = L$. Weiter sei p eine polynomielle Zeitschranke und $c \geq 2$ der maximale Verzweigungsgrad von N. Da $\text{Pr}[N(x) = \text{ja}]$ nur Werte der Form $i/k^{-p(|x|)}$ für $k = \text{kgV}(2, \ldots, c)$ annehmen kann, folgt

$$\begin{aligned} x \in L &\Rightarrow \text{Pr}[N(x) = \text{ja}] \geq 1/2, \\ x \notin L &\Rightarrow \text{Pr}[N(x) = \text{ja}] \leq 1/2 - k^{-p(|x|)}/2. \end{aligned}$$

Sei N' eine PPTM mit $\text{Pr}[N'(x) = \text{ja}] = 1/2(1 + \epsilon)$, wobei $\epsilon = k^{-p(|x|)}/2$, und betrachte die PPTM M, die bei Eingabe x zufällig wählt, ob sie N oder N' bei Eingabe x simuliert. Dann gilt

$$\text{Pr}[M(x) = \text{ja}] = \frac{\text{Pr}[N(x) = \text{ja}] + \text{Pr}[N'(x) = \text{ja}]}{2}$$

und somit

$$\begin{aligned} x \in L &\Rightarrow \text{Pr}[M(x) = \text{ja}] \geq \frac{1/2 + 1/2(1 + \varepsilon)}{2} = 1/2 + \varepsilon/4 > 1/2 \\ x \notin L &\Rightarrow \text{Pr}[M(x) = \text{ja}] \leq \frac{1/2 - \varepsilon + 1/2(1 + \varepsilon)}{2} = 1/2 - \varepsilon/4 < 1/2. \end{aligned}$$

■

Eine direkte Folgerung von Lemma 6.3 ist der Komplementabschluss von PP.

Korollar 6.4 $\mathrm{PP} = \text{co-PP}$.

Tatsächlich liefert Lemma 6.3 sogar den Abschluss von PP unter symmetrischer Differenz.

Theorem 6.5 PP *ist unter symmetrischer Differenz abgeschlossen, d.h.*

$$L_1, L_2 \in \mathrm{PP} \Rightarrow L_1 \triangle L_2 = (L_1 \setminus L_2) \cup (L_2 \setminus L_1) \in \mathrm{PP}.$$

Beweis Nach obigem Lemma existieren PPTMs M_1 und M_2 mit

$$\begin{aligned} x \in L_i &\Rightarrow \Pr[M_i(x) = \text{ja}] = 1/2 + \epsilon_i, \\ x \notin L_i &\Rightarrow \Pr[M_i(x) = \text{ja}] = 1/2 - \epsilon_i. \end{aligned}$$

wobei $\epsilon_1, \epsilon_2 > 0$ sind und von x abhängen dürfen. Dann hat die PPTM M, die bei Eingabe x zunächst $M_1(x)$ und dann $M_2(x)$ simuliert und nur dann akzeptiert, wenn dies genau eine der beiden Maschinen tut, eine Akzeptanzwk von

$$\Pr[M_1(x) = \text{ja}] \cdot \Pr[M_2(x) = \text{nein}] + \Pr[M_1(x) = \text{nein}] \cdot \Pr[M_2(x) = \text{ja}].$$

Folglich akzeptiert M alle Eingaben $x \in (L_1 \setminus L_2) \cup (L_2 \setminus L_1)$ mit Wk

$$\Pr[M(x) = \text{ja}] = (1/2 + \epsilon_1)(1/2 + \epsilon_2) + (1/2 - \epsilon_1)(1/2 - \epsilon_2) = (1/2 + 2\epsilon_1\epsilon_2) > 1/2$$

und alle Eingaben $x \in (L_1 \cap L_2) \cup \overline{L_1 \cup L_2}$ mit Wk

$$\Pr[M(x) = \text{ja}] = (1/2 + \epsilon_1)(1/2 - \epsilon_2) + (1/2 - \epsilon_1)(1/2 + \epsilon_2) = (1/2 - 2\epsilon_1\epsilon_2) < 1/2.$$ ■

Anfang der 90er Jahre konnte auch der Abschluss von PP unter Schnitt und Vereinigung bewiesen werden. In den Übungen werden wir sehen, dass folgendes Problem PP-vollständig ist.

MajoritySat (MAJSAT):

Gegeben: Eine boolsche Formel $F(x_1, \ldots, x_n)$.

Gefragt: Wird F von mindestens der Hälfte aller 2^n Belegungen erfüllt?

Definition 6.6 *Sei M eine* PPTM *und sei $L = L(M)$. M heißt*

- BPPTM, *falls für alle x gilt:* $\Pr[M(x) \neq L(x)] \leq 1/3$,
- RPTM, *falls für alle $x \notin L$ gilt:* $M(x) = \text{nein}$,
- ZPPTM, *falls für alle x gilt:* $\Pr[M(x) = \bar{L}(x)] = 0$ *und* $\Pr[M(x) = ?] \leq 1/2$.

Die Klasse BPP *(bounded error probabilistic polynomial time) enthält alle Sprachen, die von einer* BPPTM *akzeptiert werden. Entsprechend sind die Klassen* RP *(random polynomial time) und* ZPP *(zero error probabilistic polynomial time) definiert.*

Man beachte, dass wir im Falle einer RPTM oder BPPTM M o.B.d.A. davon ausgehen können, dass M niemals ein ? ausgibt. Allerdings ist nicht ausgeschlossen, dass M ein falsches Ergebnis $M(x) = \bar{L}(x)$ liefert. Probabilistische Algorithmen mit dieser Eigenschaft werden auch als **Monte Carlo Algorithmen** bezeichnet. Im Unterschied zu einer BPPTM, die bei allen Eingaben x „lügen“ kann, ist dies einer RPTM nur im Fall $x \in L$ erlaubt. Man spricht hier auch von **ein-** bzw. **zweiseitigem Fehler**. Eine ZPPTM M darf dagegen überhaupt keine Fehler machen. Algorithmen von diesem Typ werden als **Las Vegas Algorithmen** bezeichnet.

Theorem 6.7 $\mathrm{ZPP} = \mathrm{RP} \cap \text{co-RP}$.

Beweis Die Inklusion von links nach rechts ist klar. Für die umgekehrte Richtung sei L eine Sprache in $\mathrm{RP} \cap \text{co-RP}$. Dann existieren RPTMs M_1 und M_2 für L und $\overline{L}$, wobei wir annehmen, dass M_1 und M_2 niemals ein ? ausgeben. Weiter sei $\overline{M}_2$ die PPTM, die aus M_2 durch Vertauschen von q_{ja} und q_{nein} hervorgeht. Dann gilt

$$\begin{aligned} x \in L &\Rightarrow \Pr[M_1(x) = \text{ja}] \geq 1/2 \wedge \Pr[\overline{M}_2(x) = \text{ja}] = 1, \\ x \notin L &\Rightarrow \Pr[M_1(x) = \text{nein}] = 1 \wedge \Pr[\overline{M}_2(x) = \text{nein}] \geq 1/2. \end{aligned}$$

M_1 kann also nur Eingaben $x \in L$ akzeptieren, während $\overline{M}_2$ nur Eingaben $x \notin L$ verwerfen kann. Daher kann die Kombination $M_1(x) = \text{ja}$ und $\overline{M}_2(x) = \text{nein}$ nicht auftreten. Weiter ergeben sich hieraus für die PPTM M, die bei Eingabe x die beiden PPTMs $M_1(x)$ und $\overline{M}_2(x)$ simuliert und sich gemäß der Tabelle

	$M_1(x) = \text{ja}$	$M_1(x) = \text{nein}$
$\overline{M}_2(x) = \text{ja}$	ja	?
$\overline{M}_2(x) = \text{nein}$	–	nein

verhält, folgende Äquivalenzen:

$$\begin{aligned} M(x) = \text{ja} &\Leftrightarrow M_1(x) = \text{ja}, \\ M(x) = \text{nein} &\Leftrightarrow \overline{M}_2(x) = \text{nein}. \end{aligned}$$

Nun ist leicht zu sehen, dass folgende Implikationen gelten:

$$\begin{aligned} x \in L &\Rightarrow \Pr[M(x) = \text{ja}] = \Pr[M_1(x) = \text{ja}] \geq 1/2 \text{ und} \\ &\quad\ \Pr[M(x) = \text{nein}] = \Pr[\overline{M}_2(x) = \text{nein}] = 0 \\ x \notin L &\Rightarrow \Pr[M(x) = \text{nein}] = \Pr[\overline{M}_2(x) = \text{nein}] \geq 1/2 \text{ und} \\ &\quad\ \Pr[M(x) = \text{ja}] = \Pr[M_1(x) = \text{ja}] = 0. \end{aligned}$$

Dies zeigt, dass M eine ZPPTM für L ist, da für alle Eingaben x gilt:

$$\Pr[M(x) = \bar{L}(x)] = 0 \text{ und } \Pr[M(x) = ?] \leq 1/2.$$

■

6.1 Reduktion der Fehlerwahrscheinlichkeit

In diesem Abschnitt zeigen wir, wie sich für RP-, ZPP- und BPP-Maschinen M die Wahrscheinlichkeit $\Pr[M(x) \neq L(x)]$ für ein inkorrektes oder indifferentes Ergebnis auf einen exponentiell kleinen Wert $2^{-q(|x|)}$ reduzieren lässt. Wir betrachten zunächst den Fall einer RPTM.

Theorem 6.8 *Sei q ein beliebiges Polynom. Dann existiert zu jeder Sprache $L \in \mathrm{RP}$ eine* RPTM M *mit*

$$\begin{aligned} x \in L &\Rightarrow \Pr[M(x) = \text{ja}] \geq 1 - 2^{-q(|x|)}, \\ x \notin L &\Rightarrow \Pr[M(x) = \text{nein}] = 1. \end{aligned}$$

Beweis Sei M eine RPTM für L. Dann gilt

$$\begin{aligned} x \in L &\Rightarrow \Pr[M(x) = \text{nein}] \leq 1/2, \\ x \notin L &\Rightarrow \Pr[M(x) = \text{nein}] = 1. \end{aligned}$$

Betrachte die PPTM M', die $q(|x|)$ Simulationen von M ausführt und nur dann ihre Eingabe x verwirft, wenn M sie bei allen $q(|x|)$ Simulationen verwirft, und andernfalls akzeptiert (M' gibt also niemals ? aus). Dann gilt

$$\begin{aligned} x \in L &\Rightarrow \Pr[M(x) = \text{nein}] \leq 1/2 \Rightarrow \Pr[M'(x) = \text{nein}] \leq 2^{-q(|x|)}, \\ x \notin L &\Rightarrow \Pr[M(x) = \text{nein}] = 1 \Rightarrow \Pr[M'(x) = \text{nein}] = 1. \end{aligned}$$

■

Ganz analog lässt sich die Zuverlässigkeit einer ZPPTM verbessern.

Theorem 6.9 *Für jedes Polynom q und jede Sprache $L \in \mathrm{ZPP}$ existiert eine* ZPPTM M *mit* $\Pr[M(x) = ?] \leq 2^{-q(|x|)}$ *bei allen Eingaben x.*

Für die Reduktion der Fehlerwahrscheinlichkeit von BPPTMs benötigen wir das folgende Lemma.

Lemma 6.10 *Sei E ein Ereignis, das mit Wahrscheinlichkeit $1/2 - \epsilon$, $\epsilon > 0$, auftritt. Dann ist die Wahrscheinlichkeit, dass sich E bei $m = 2t + 1$ unabhängigen Wiederholungen mehr als t-mal ereignet, höchstens $1/2(1 - 4\epsilon^2)^t$.*

Beweis Für $i = 1, \ldots, m$ sei X_i die Indikatorvariable

$$X_i = \begin{cases} 1, & \text{Ereignis } E \text{ tritt beim } i\text{-ten Versuch ein,} \\ 0, & \text{sonst} \end{cases}$$

und X sei die Zufallsvariable $X = \sum_{i=1}^{m} X_i$. Dann ist X binomial verteilt mit Parametern m und $p = 1/2 - \epsilon$. Folglich gilt für $i > m/2$,

$$\begin{aligned} \Pr[X = i] &= \binom{m}{i} (1/2 - \epsilon)^i (1/2 + \epsilon)^{m-i} \\ &= \binom{m}{i} (1/2 - \epsilon)^{m/2} (1/2 + \epsilon)^{m/2} \left(\frac{1/2 - \epsilon}{1/2 + \epsilon}\right)^{i - m/2} \\ &\leq \binom{m}{i} \underbrace{(1/2 - \epsilon)^{m/2} (1/2 + \epsilon)^{m/2}}_{(1/4 - \epsilon^2)^{m/2}}. \end{aligned}$$

Wegen

$$\sum_{i=t+1}^{m} \binom{m}{i} \leq 2^{m-1} = \frac{4^{m/2}}{2}$$

erhalten wir somit

$$\sum_{i=t+1}^{m} \Pr[X = i] \;\leq\; \left(1/4 - \epsilon^2\right)^{m/2} \sum_{i=t+1}^{m} \binom{m}{i} \;\leq\; \frac{(1-4\epsilon^2)^{m/2}}{2} \;\leq\; \frac{(1-4\epsilon^2)^{t}}{2}.$$

■

Theorem 6.11 *Für jedes Polynom q und jede Sprache $L \in$ BPP existiert eine* BPPTM M *mit* $\Pr[M(x) \neq L(x)] \leq 2^{-q(|x|)}$ *bei allen Eingaben x.*

Beweis Sei M eine BPPTM für L. Dann gilt

$$\Pr[M(x) \neq L(x)] \leq 1/3 = 1/2 - 1/6.$$

Betrachte die PPTM M', die bei Eingabe x, $|x| = n$, $2t(n)+1$ Simulationen von $M(x)$ ausführt und x akzeptiert, falls M bei mehr als $t(n)$ dieser Simulationen akzeptiert, wobei $t(n) = (q(n) - 1)/\log_2(9/8)$ ist. Dann folgt nach obigem Lemma

$$\begin{aligned}
\Pr[M'(x) \neq L(x)] &= \Pr[M \text{ lügt bei mehr als } t(n) \text{ Simulationen}] \\
&\leq 1/2\,(1 - 4/36)^{t(n)} \\
&\leq 1/2\,(8/9)^{t(n)} \\
&= 2^{-q(|x|)}.
\end{aligned}$$

■

Theorem 6.12 $\mathrm{BPP} \subseteq \mathrm{PSK}$.

Beweis Sei $L \in \mathrm{BPP}$ und sei M eine BPPTM für L. Nach vorigem Satz können wir annehmen, dass für alle Eingaben x, $|x| = n$,

$$\Pr[M(x) \neq L(x)] < 2^{-n}$$

gilt. Sei p eine polynomielle Zeitschranke und $c \in \mathbb{N}$ der maximale Verzweigungsgrad von M. Setzen wir $k = \mathrm{kgV}(2, 3, \ldots, c)$, dann können wir für eine gegebene Eingabelänge n jeder Folge $r = r_1 \cdots r_{p(n)}$ aus der Menge $R_n = \{1, \ldots, k\}^{p(n)}$ eindeutig eine Rechnung von $M(x)$ zuordnen, indem wir im i-ten Rechenschritt aus den c_i zur Auswahl stehenden Folgekonfigurationen $K_0, \ldots, K_{c_i-1}$ die $(r_i \bmod c_i)$-te wählen. Bezeichnen wir das Ergebnis der so beschriebenen Rechnung mit $M_r(x)$, so gilt

$$\Pr[M(x) \neq L(x)] = \Pr_{r \in_R R_n}[M_r(x) \neq L(x)] < 2^{-n}.$$

Daher folgt

$$\Pr_{r \in_R R_n}[\exists x \in \{0,1\}^n : M_r(x) \neq L(x)] \;\leq \sum_{x \in \{0,1\}^n} \Pr_{r \in_R R_n}[M_r(x) \neq L(x)] \;<\; 1.$$

Also muss für jede Eingabelänge n eine Folge r_n existieren, so dass M_{r_n} alle Eingaben der Länge n korrekt entscheidet. Wie wir in Korollar 5.9 gesehen haben, können die polynomiell zeitbeschränkten Rechnungen von M_{r_n} bei allen Eingaben der Länge n durch einen Schaltkreis polynomieller Größe simuliert werden. ■

7 Die Polynomialzeithierachie

7.1 Anzahl-Operatoren

Definition 7.1 (Anzahlklassen) *Sei $\mathcal{C}$ eine Sprachklasse und sei $B \subseteq \Sigma^*$ eine p-balancierte Sprache in $\mathcal{C}$. Durch B werden die Funktion $\#B : \Sigma^* \to \mathbb{N}$ mit*

$$\#B(x) = \|\{y \in \{0,1\}^* \mid x\#y \in B\}\|$$

und folgende Sprachen definiert (n bezeichnet die Länge von x):

$$\begin{aligned}
\exists B &= \{x \in \Sigma^* \mid \#B(x) > 0\},\\
\forall B &= \{x \in \Sigma^* \mid \#B(x) = 2^{p(n)}\},\\
\exists^{\geq 1/2} B &= \{x \in \Sigma^* \mid \#B(x) \geq 2^{p(n)-1}\}\\
\oplus B &= \{x \in \Sigma^* \mid \#B(x) \text{ ist ungerade}\}.
\end{aligned}$$

*B heißt **einseitig**, falls $\#B(x)$ nur Werte in $\{0\} \cup [2^{p(n)-1}, 2^{p(n)}]$ annimmt, und **zweiseitig**, falls $\#B(x)$ keine Werte in $[2^{p(n)}/3, 2^{p(n)+1}/3]$ annimmt. Für $\mathrm{O} \in \{\#, \exists, \forall, \oplus\}$ sei*

$$\mathrm{O} \cdot \mathcal{C} = \{\mathrm{O}B \mid B \in \mathcal{C} \text{ ist polynomiell balanciert}\}$$

die durch Anwendung des Operators O auf $\mathcal{C}$ definierte Funktionen- bzw. Sprachklasse. Zudem definieren wir die Operatoren R und BP durch

$$\mathrm{R} \cdot \mathcal{C} = \{\exists^{\geq 1/2} B \mid B \in \mathcal{C} \text{ ist einseitig}\}$$

und

$$\mathrm{BP} \cdot \mathcal{C} = \{\exists^{\geq 1/2} B \mid B \in \mathcal{C} \text{ ist zweiseitig}\}.$$

Lemma 7.2 *Sei $\mathcal{C}$ eine Sprachklasse, die unter $\leq_m^{log}$ abgeschlossen ist. Dann gilt*

(i) $\text{co-}\exists \cdot \mathcal{C} = \forall \cdot \text{co-}\mathcal{C}$,

(ii) $\text{co-BP} \cdot \mathcal{C} = \mathrm{BP} \cdot \text{co-}\mathcal{C}$,

(iii) $\oplus \cdot \mathcal{C} = \oplus \cdot \text{co-}\mathcal{C} = \text{co-}\oplus \cdot \mathcal{C}$.

Beweis *(i)* Sei $A \in \text{co-}\exists \cdot \mathcal{C}$. Dann existieren ein Polynom p und eine p-balancierte Sprache $B \in \mathcal{C}$ mit $\bar{A} = \exists B$. Definiere die Sprache

$$\hat{B} = \{x\#y \mid x\#y \notin B \text{ und } |y| = p(|x|)\}.$$

Dann ist $\hat{B}$ eine ebenfalls p-balancierte Sprache in $\text{co-}\mathcal{C}$ mit $\#\hat{B}(x) = 2^{p(n)} - \#B(x)$. Daher folgt

$$x \in A \Leftrightarrow \#B(x) = 0 \Leftrightarrow \#\hat{B}(x) = 2^{p(n)},$$

also $A = \forall \hat{B} \in \forall \cdot \text{co-}\mathcal{C}$.

(ii) Sei $A \in \text{co-BP}\cdot\mathcal{C}$ und sei $B \in \mathcal{C}$ eine p-balancierte zweiseitige Sprache mit $\bar{A} = \exists^{\geq 1/2} B$. Dann ist die in (i) definierte p-balancierte co-$\mathcal{C}$-Sprache $\hat{B}$ ebenfalls zweiseitig und es gilt $A = \exists^{\geq 1/2}\hat{B} \in \text{BP} \cdot \text{co-}\mathcal{C}$.

(iii) Sei $A \in \oplus \cdot \mathcal{C}$ und sei $B \in \mathcal{C}$ eine p-balancierte Sprache mit $A = \oplus B$. Dann hat $\#\hat{B}(x) = 2^{p(n)} - \#B(x)$ die gleiche Parität wie $\#B(x)$ und daher gilt $A = \oplus\hat{B}$, d.h. A ist in $\oplus \cdot$ co-$\mathcal{C}$. Weiter ist

$$B' = \{x\#0y \mid x\#y \in B\} \cup \{x\#1^{p(|x|)+1}\}$$

eine $(p+1)$-balancierte Sprache in $\mathcal{C}$ mit $\bar{A} = \oplus B'$, d.h. $A \in \text{co-}\oplus \cdot \mathcal{C}$. ■

Wir wissen bereits, dass $\exists \cdot \text{P} = \text{NP}$ ist. Was passiert, wenn wir diese Quantoren wiederholt anwenden?

Lemma 7.3 *Sei $B \in \mathcal{C}$ eine p-balancierte Sprache und sei $\mathcal{C}$ unter $\leq_m^{log}$ abgeschlossen. Dann existiert für jede Funktion $f \in \text{FL}$ eine q-balancierte Sprache $B' \in \mathcal{C}$ mit*

$$\#B'(x)/2^{q(|x|)} = \#B(f(x))/2^{p(|f(x)|)}.$$

Beweis Sei q ein Polynom mit $p(|f(x)|) \leq q(n)$ für alle x der Länge n und sei B' die Sprache

$$B' = \{x\#y \mid |y| = q(n) \text{ und } f(x)\#y' \in B\},$$

wobei y' das Präfix der Länge $p(|f(x)|)$ von y bezeichnet. Dann gilt $B' \leq_m^{log} B$ da jedes Präfix y' mit $f(x)\#y' \in B$ genau $2^{q(|x|)-p(|f(x)|)}$ Verlängerungen y mit $x\#y \in B'$ hat, folgt

$$\#B'(x) = \#B(f(x))2^{q(|x|)-p(|f(x)|)}.$$

■

Mit obigem Lemma ist es nun leicht, folgende Abschlusseigenschaften der Anzahlklassen $\exists \cdot \mathcal{C}$, $\forall \cdot \mathcal{C}$, $\text{R} \cdot \mathcal{C}$ und $\text{BP} \cdot \mathcal{C}$ zu zeigen.

Lemma 7.4 *Sei $\mathcal{C}$ eine unter $\leq_m^{log}$ abgeschlossene Sprachklasse. Dann gilt*

(i) Für $\text{O} \in \{\exists, \forall, \text{R}, \text{BP}\}$ ist $\text{O} \cdot \mathcal{C}$ unter $\leq_m^{log}$ abgeschlossen,

(ii) $\exists \cdot \exists \cdot \mathcal{C} = \exists \cdot \mathcal{C}$ and $\forall \cdot \forall \cdot \mathcal{C} = \forall \cdot \mathcal{C}$.

Beweis Siehe Übungen. ■

Als nächstes wollen wir zeigen, dass $\text{R} \cdot \text{P} = \text{RP}$ ist. Hierfür benötigen wir folgendes Lemma.

Lemma 7.5 *Sei M eine* PPTM *und sei $\alpha = \Pr[M(x) = \text{ja}]$. Dann existieren eine Sprache $B \in \text{P}$ und ein Polynom q, so dass für alle Eingaben x, $|x| = n$, gilt:*

$$\alpha/2 < \Pr_{y \in_R \{0,1\}^{q(n)}}[x\#y \in B] \leq \alpha.$$

Beweis Sei p eine polynomielle Zeitschranke für M und sei $k = \text{kgV}(2, 3, \ldots, c)$, wobei $c \in \mathbb{N}$ der maximale Verzweigungsgrad von M ist. Wie im Beweis der Inklusion von BPP in PSK (siehe Satz 6.12) können wir die Rechnungen von M bei Eingaben der Länge n durch Folgen $r \in \{1, \ldots, k\}^{p(n)}$ beschreiben, so dass

$$\text{Pr}[M(x) = \text{ja}] = \text{Pr}_{r \in_R R_n}[M_r(x) = \text{ja}]$$

ist, wobei $M_r(x)$ das Ergebnis von $M(x)$ bei der durch r beschriebenen Rechnung bezeichnet. Sei nun $q(n) = \lceil \log_2(k^{p(n)}) \rceil$ und sei $D_n \subseteq \{0,1\}^{q(n)}$ die Menge der ersten $k^{p(n)}$ Binärstrings der Länge $q(n)$. Dann können die Folgen $r \in R_n$ durch Binärstrings $y \in D_n$ kodiert werden. Betrachte nun folgende PPTM M':

> $M'(x)$ rät zufällig einen Binärstring $y \in \{0,1\}^{q(n)}$. Ist y Kodierung einer Folge $r \in R_n$, so verhält sich M' wie $M_r(x)$, andernfalls verwirft M'.

Definieren wir $\beta(n) = \|D_n\|/2^{q(n)}$, so ist $\beta(n) \in (1/2, 1]$, da D_n mehr als die Hälfte aller Strings der Länge $q(n)$ enthält. Zudem akzeptiert $M'(x)$ mit Wk

$$\text{Pr}[M'(x) = \text{ja}] = \text{Pr}[y \in D_n] \cdot \text{Pr}[M'(x) = \text{ja} \mid y \in D_n] = \beta(n)\text{Pr}[M(x) = \text{ja}].$$

Es genügt also, $B = \{x\#y \mid y \in \{0,1\}^{q(n)}, M'_y(x) = \text{ja}\}$ zu wählen. ■

Korollar 7.6 $\mathrm{R} \cdot \mathrm{P} = \mathrm{RP}$.

Für die Gleichheit $\mathrm{BP} \cdot \mathrm{P} = \mathrm{BPP}$ genügt eine einfache Modifikation des obigen Lemmas.

Lemma 7.7 *Sei M eine* PPTM *und sei $\beta = \text{Pr}[M(x) = \text{ja}] - 1/2$. Dann existieren eine Sprache $B' \in \mathrm{P}$ und ein Polynom q', so dass für alle Eingaben x, $|x| = n$, gilt:*

$$\beta/2 < \text{Pr}_{z \in_R \{0,1\}^{q'(n)}}[x\#z \in B'] - 1/2 \leq \beta.$$

Beweis Der Beweis verläuft vollkommen analog, nur dass wir jetzt anstelle von M' folgende PPTM M'' betrachten.

> $M''(x)$ rät zufällig einen Binärstring $y \in \{0,1\}^{q(n)}$ und ein Bit $b \in \{0,1\}$. Ist y Kodierung einer Folge $r \in R_n$, so verhält sich M'' wie $M_r(x)$, andernfalls akzeptiert M'' genau dann, wenn $b = 0$ ist.

Dann akzeptiert $M''(x)$ mit Wk

$$\underbrace{\text{Pr}[y \in D_n]}_{\beta(n)} \cdot \underbrace{\text{Pr}[M''(x) = \text{ja} \mid y \in D_n]}_{\text{Pr}[M(x)=\text{ja}]} + \underbrace{\text{Pr}[y \notin D_n]}_{1-\beta(n)} \cdot \underbrace{\text{Pr}[M''(x) = \text{ja} \mid y \notin D_n]}_{1/2}$$
$$= \beta(n)(\text{Pr}[M(x) = \text{ja}] - 1/2) + 1/2.$$

Wir können also $q'(n) = q(n) + 1$ und $B' = \{x\#z \mid z \in \{0,1\}^{q'(n)}, M''_z(x) = \text{ja}\}$ wählen, wobei $M''_z(x)$ das Ergebnis der Rechnung von $M''(x)$ bei Verwendung des Zufallsstrings $z = yb$ beschreibt. ■

Korollar 7.8 $\mathrm{BP} \cdot \mathrm{P} = \mathrm{BPP}$.

Wir sehen also, dass die Anwendung der Operatoren $\exists$, $\forall$, R und BP auf die Klasse P die uns bereits bekannten Klassen NP, $\mathrm{co\text{-}NP}$, RP und BPP liefert.

7.2 Die Polynomialzeithierachie

Definition 7.9 *Die **Polynomialzeithierachie** besteht aus den Stufen Σ_k^p und Π_k^p, $k \geq 0$, welche induktiv wie folgt definiert sind:*

$$\Sigma_0^p = \mathrm{P}, \qquad \Pi_0^p = \mathrm{P},$$
$$\Sigma_{k+1}^p = \exists \cdot \Pi_k^p, \qquad \Pi_{k+1}^p = \forall \cdot \Sigma_k^p, \quad k \geq 0.$$

Die Vereinigung aller Stufen der Polynomialzeithierachie bezeichnen wir mit PH*,*

$$\mathrm{PH} = \bigcup_{k \geq 0} \Sigma_k^p = \bigcup_{k \geq 0} \Pi_k^p.$$

Es ist leicht zu sehen, dass $\Sigma_k^p = \text{co-}\Pi_k^p$ ist. Es ist nicht bekannt, ob die Polynomialzeithierachie echt ist, also $\Sigma_k^p \neq \Sigma_{k+1}^p$ für alle $k \geq 0$ gilt. Die Annahme $\Sigma_k^p = \Sigma_{k+1}^p$ ist mit einem Kollaps von PH auf die k-te Stufe äquivalent. Es gilt allerdings als unwahrscheinlich, dass die Polynomialzeithierachie auf eine kleine Stufe kollabiert.

Theorem 7.10 *Für alle $k \geq 0$ gilt: $\Sigma_k^p = \Sigma_{k+1}^p \Leftrightarrow \Sigma_k^p = \Pi_k^p \Leftrightarrow \mathrm{PH} = \Sigma_k^p$.*

Beweis Wegen $\Pi_k^p \subseteq \Sigma_{k+1}^p$ impliziert die Inklusion $\Sigma_k^p = \Sigma_{k+1}^p$ sofort $\Pi_k^p \subseteq \Sigma_k^p$, was mit $\Sigma_k^p = \Pi_k^p$ gleichbedeutend ist. Für die zweite Implikation zeigen wir durch Induktion über l, dass unter der Voraussetzung $\Sigma_k^p = \Pi_k^p$ alle Stufen Σ_l^p, $l \geq k$, in Σ_k^p enthalten sind. Der Induktionsanfang $l = k$ ist klar. Für den Induktionsschritt setzen wir die Gleichheit $\Sigma_l^p = \Sigma_k^p$ (bzw. $\Pi_l^p = \Pi_k^p$) voraus und folgern

$$\Sigma_{l+1}^p = \exists \cdot \Pi_l^p = \exists \cdot \Pi_k^p = \exists \cdot \Sigma_k^p = \Sigma_k^p.$$

Die Implikation $\mathrm{PH} = \Sigma_k^p \Rightarrow \Sigma_k^p = \Sigma_{k+1}^p$ ist klar. ■

Als Folgerung hieraus ergibt sich, dass eine NP-vollständige Sprache nicht in P (bzw. co-NP) enthalten ist, außer wenn PH auf P bzw. NP kollabiert.

Als nächstes zeigen wir, dass BPP in der zweiten Stufe der Polynomialzeithierarchie enthalten ist.

Theorem 7.11 $\mathrm{BPP} \subseteq \Sigma_2^p \cap \Pi_2^p$.

Beweis Sei $A \in \mathrm{BPP}$. Dann existiert eine Sprache $B \in \mathrm{P}$ und ein Polynom p mit

$$\|\{y \in \{0,1\}^{p(n)} \mid A(x) = B(x\#y)\}\| \geq (1 - 2^{-n})2^{p(n)}.$$

Sei $\oplus$ die bitweise XOR-Operation auf $\{0,1\}^n$, d.h.

$$x_1 \cdots x_n \oplus y_1 \cdots y_n = z_1 \cdots z_n, \text{ wobei } z_i = x_i \oplus y_i$$

und sei

$$B_x = \{y \in \{0,1\}^{p(n)} \mid x\#y \in B\}.$$

Wie wir gleich sehen werden, können wir dann für alle x mit $2^{|x|} > p(|x|)$ die Zugehörigkeit von x zu A durch

$$x \in A \Leftrightarrow \exists u_1, \ldots, u_{p(n)} \in \{0,1\}^{p(n)} \forall v \in \{0,1\}^{p(n)} \exists i : v \oplus u_i \in B_x \tag{7.1}$$

charakterisieren. Dies beweist, dass A zu Σ_2^p gehört, da die Sprache

$$B' = \{x\#u_1 \cdots u_{p(n)}\#v \mid |u_1| = \cdots = |u_{p(n)}| = |v| = p(n), \exists i : v \oplus u_i \in B_x\}$$

in P entscheidbar ist. Wir zeigen zuerst die Richtung von rechts nach links der Äquivalenz (7.1). Hierzu nehmen wir an, dass Wörter $u_1, \ldots, u_{p(n)} \in \{0,1\}^{p(n)}$ existieren, so dass für jedes $v \in \{0,1\}^{p(n)}$ zumindest ein Wort der Form $v \oplus u_i$ in B_x enthalten ist. Dann müssen die $p(n)$ Mengen

$$B_x \oplus u_i = \{v \oplus u_i \mid v \in B_x\} = \{v \mid v \oplus u_i \in B_x\},$$

die alle gleichmächtig zu B_x sind, ganz $\{0,1\}^{p(n)}$ abdecken, also $\|B_x\| \geq 2^{p(n)}/p(n)$ sein. Dann muss aber x zu A gehören, da B_x im Fall $x \in A$ höchstens $2^{-n}2^{p(n)}$ Wörter enthält.

Zum Nachweis der Implikation von links nach rechts sei $x \in A$ angenommen. Da B_x mindestens $(1-2^{-n})2^{p(n)}$ Wörter enthält, wird ein fester String $v \in \{0,1\}^{p(n)}$ mit Wk

$$\mathrm{Pr}_{u \in_R \{0,1\}^{p(n)}}[v \oplus u \notin B_x] \leq 2^{-n}$$

nicht durch eine einzelne Menge $B_x \oplus u$ abgedeckt. Folglich wird v mit Wk

$$\mathrm{Pr}_{u_1,\ldots,u_{p(n)} \in_R \{0,1\}^{p(n)}}[\forall i : v \oplus u_i \notin B_x] \leq (2^{-n})^{p(n)} = 2^{-np(n)}.$$

von keiner der Mengen $B_x \oplus u_i$, $i = 1, \ldots, p(n)$, abgedeckt. Somit existiert ein solches v mit Wk

$$\mathrm{Pr}_{u_1,\ldots,u_{p(n)} \in_R \{0,1\}^{p(n)}}[\exists v \in \{0,1\}^{p(n)} \forall i : v \oplus u_i \notin B_x] \leq 2^{p(n)}2^{-np(n)}.$$

Da diese Wahrscheinlichkeit für $n > 1$ kleiner als 1 ist, muss es für jedes x mit $|x| > 1$ Wörter $u_1, \ldots, u_{p(n)} \in \{0,1\}^{p(n)}$ geben, so dass die Mengen $B_x \oplus u_i$, $i = 1, \ldots, p(n)$, alle Strings v abdecken. Die Zugehörigkeit von BPP zu Π_2^p ergibt sich unmittelbar aus dem Komplementabschluss von BPP. ■

Starten wir in obigem Beweis mit einer Sprache A in $\mathrm{BP} \cdot \text{co-NP}$ (d.h. die Menge B gehört zu co-NP), dann folgt ebenfalls $A \in \Sigma_2^p$, da die Sprache

$$B' = \{x\#u_1\# \cdots \#u_{p(n)}\#v \mid \exists i : v \oplus u_i \in B_x\}$$

in co-NP entscheidbar und $\exists \cdot \forall \cdot \text{co-NP} = \Sigma_2^p$ ist.

Korollar 7.12 $\mathrm{BP} \cdot \text{co-NP} \subseteq \Sigma_2^p$ *bzw.* $\mathrm{BP} \cdot \mathrm{NP} \subseteq \Pi_2^p$.

Zum Abschluss dieses Kapitels zeigen wir, dass NP-vollständige Sprachen nicht in $\mathrm{BP} \cdot \text{co-NP} = \text{co-BP} \cdot \mathrm{NP}$ enthalten sind, außer wenn $\mathrm{PH} = \mathrm{BP} \cdot \mathrm{NP}$ ist. Hierfür benötigen wir noch gewisse Abschlusseigenschaften der Klasse $\mathrm{BP} \cdot \mathrm{NP}$.

Definition 7.13 *A ist auf B **disjunktiv reduzierbar** (in Zeichen: $A \leq^{log}_{disj} B$), falls eine Funktion $f \in \mathrm{FL}$ existiert, die für jedes Wort x eine Liste $y_1\# \cdots \#y_m$ von Wörtern y_i liefert mit*

$$x \in A \Leftrightarrow \exists i \in \{1, \ldots, m\} : y_i \in B.$$

Gilt dagegen

$$x \in A \Leftrightarrow \|\{i \in \{1, \ldots, m\} \mid y_i \in B\} \geq {}^{m}\!/_{2},$$

*so heißt A **majority-reduzierbar** auf B, wofür wir auch kurz $A \leq^{log}_{maj} B$ schreiben.*

Es ist leicht zu sehen, dass die Klassen P, NP und $\mathrm{co\text{-}NP}$ unter beiden Typen von Reduktionen abgeschlossen sind. Das folgende Lemma lässt sich vollkommen analog zu Satz 6.11 beweisen, in dem wir BPPTMs mit exponentiell kleiner Fehlerwk konstruiert haben.

Lemma 7.14 *Falls $\mathcal{C}$ unter majority-Reduktionen abgeschlossen ist, existieren für jede Sprache $A \in \mathrm{BP} \cdot \mathcal{C}$ und jedes Polynom p eine Sprache $B \in \mathcal{C}$ und ein Polynom q, so dass für alle x, $|x| = n$, gilt*

$$\|\{y \in \{0,1\}^{q(n)} \mid A(x) = B(x\#y)\}\| \geq (1 - 2^{-p(n)})2^{q(n)}.$$

Nun können wir den Abschluss von $\mathrm{BP}\cdot\mathcal{C}$ unter dem BP-Operator zeigen, falls $\mathcal{C}$ unter majority-Reduktionen abgeschlossen ist.

Theorem 7.15 *Für jede Klasse $\mathcal{C}$, die unter majority-Reduktionen abgeschlossen ist, gilt*

$$\mathrm{BP} \cdot \mathrm{BP} \cdot \mathcal{C} = \mathrm{BP} \cdot \mathcal{C}.$$

Beweis Sei $L \in \mathrm{BP} \cdot \mathrm{BP} \cdot \mathcal{C}$. Da mit $\mathcal{C}$ auch $\mathrm{BP} \cdot \mathcal{C}$ unter majority-Reduktionen abgeschlossen ist, existieren eine Sprache $A \in \mathrm{BP} \cdot \mathcal{C}$ und ein Polynom p, so dass für alle x, $|x| = n$, gilt

$$\|\{y \in \{0,1\}^{p(n)} \mid L(x) = A(x\#y)\}\| \geq ({}^{5}\!/_{6})2^{p(n)}.$$

Zudem existieren eine Sprache $B \in \mathcal{C}$ und ein Polynom q, so dass für alle x und $y \in \{0,1\}^{p(n)}$ gilt

$$\|\{z \in \{0,1\}^{q(m)} \mid A(x\#y) = B(x\#y\#z)\}\| \geq ({}^{5}\!/_{6})2^{q(m)},$$

wobei $m = n + p(n) + 1$ ist. Daher folgt für $p'(n) = p(n) + q(n + p(n) + 1)$ und $B' = \{x\#yz \mid |y| = p(|x|), |z| = q(|x| + p(|x|) + 1), x\#y\#z \in B\} \in \mathcal{C}$,

$$x \in L \Rightarrow \|\{u \in \{0,1\}^{p'(n)} \mid x\#u \in B'\}\| \geq ({}^{5}\!/_{6})^2\, 2^{p'(n)} > ({}^{2}\!/_{3})2^{p'(n)},$$
$$x \notin L \Rightarrow \|\{u \in \{0,1\}^{p'(n)} \mid x\#u \in B'\}\| \leq ({}^{1}\!/_{6} + {}^{5}\!/_{6} \cdot {}^{1}\!/_{6})2^{p'(n)} < ({}^{1}\!/_{3})2^{p'(n)}$$

und somit $L \in \mathrm{BP} \cdot \mathcal{C}$. ■

Folglich sind die Klassen BPP und $\mathrm{BP} \cdot \mathrm{NP}$ unter dem BP-Operator abgeschlossen. Analog folgt für jede Sprachklasse $\mathcal{C}$, die unter disjunktiven Reduktionen abgeschlossen ist,

$$\mathrm{R} \cdot \mathrm{R} \cdot \mathcal{C} = \mathrm{R} \cdot \mathcal{C}.$$

Das folgende Lemma zeigt, dass $\mathrm{BP} \cdot \mathrm{NP}$ auch unter dem $\exists$-Operator abgeschlossen ist.

Lemma 7.16 *Sei $\mathcal{C}$ eine unter $\leq^{log}_{maj}$ abgeschlossene Sprachklasse. Dann gilt*

$$\exists \cdot \mathrm{BP} \cdot \mathcal{C} \subseteq \mathrm{BP} \cdot \exists \cdot \mathcal{C}.$$

Beweis Sei $L \in \exists \cdot \mathrm{BP} \cdot \mathcal{C}$. Dann existieren eine Sprache $A \in \mathrm{BP} \cdot \mathcal{C}$ und ein Polynom p mit

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

wobei $n = |x|$ ist. Zu p und A existieren ein Polynom q und eine Sprache $B \in \mathcal{C}$ mit

$$\begin{aligned} x\#y \in A \Rightarrow &\|\{z \in \{0,1\}^{q(n)} \mid x\#y\#z \in B\}\| \geq (1 - 2^{-p(n)-2})2^{q(n)}, \\ x\#y \notin A \Rightarrow &\|\{z \in \{0,1\}^{q(n)} \mid x\#y\#z \in B\}\| \leq (2^{-p(n)-2})2^{q(n)}. \end{aligned}$$

Nun folgt für die Sprache $A' = \{x\#z \mid |z| = q(n), \exists y \in \{0,1\}^{p(n)} : x\#y\#z \in B\}$

$$\begin{aligned} x \in L &\Rightarrow \|\{z \in \{0,1\}^{q(n)} \mid x\#z \in A'\}\| \geq (1 - 2^{-p(n)-2})2^{q(n)} > (2/3)2^{q(n)}, \\ x \notin L &\Rightarrow \|\{z \in \{0,1\}^{q(n)} \mid x\#z \in A'\}\| \leq 2^{p(n)}(2^{-p(n)-2})2^{q(n)} < (1/3)2^{q(n)} \end{aligned}$$

und somit $L \in \mathrm{BP} \cdot \exists \cdot \mathcal{C}$. ■

Theorem 7.17 $\mathrm{NP} \subseteq \mathrm{BP} \cdot \text{co-NP} \Rightarrow \mathrm{PH} = \mathrm{BP} \cdot \mathrm{NP}$.

Beweis Gelte $\mathrm{NP} \subseteq \mathrm{BP} \cdot \text{co-NP}$. Wir zeigen durch Induktion über k, dass dann auch die Klassen Σ^p_k, $k \geq 0$, in $\mathrm{BP} \cdot \text{co-NP}$ enthalten sind. Der Induktionsanfang $k = 0$ ist klar. Für den Induktionsschritt setzen wir die Inklusionen $\mathrm{NP} \subseteq \mathrm{BP} \cdot \text{co-NP}$ und $\Sigma^p_k \subseteq \mathrm{BP} \cdot \text{co-NP}$ (was mit $\Pi^p_k \subseteq \mathrm{BP} \cdot \mathrm{NP}$ gleichbedeutend ist) voraus und folgern

$$\begin{aligned} \Sigma^p_{k+1} &= \exists \cdot \Pi^p_k \\ &\subseteq \exists \cdot \mathrm{BP} \cdot \mathrm{NP} \\ &\subseteq \mathrm{BP} \cdot \exists \cdot \mathrm{NP} \\ &= \mathrm{BP} \cdot \mathrm{NP} \\ &\subseteq \mathrm{BP} \cdot \mathrm{BP} \cdot \text{co-NP} \\ &= \mathrm{BP} \cdot \text{co-NP}. \end{aligned}$$

■

8 Das Graphisomorphieproblem

8.1 Iso- und Automorphismen

In diesem Kapitel wollen wir die Komplexität des Graphisomorphieproblems untersuchen. Hierbei bedeutet es keine Einschränkung, wenn wir voraussetzen, dass beide Graphen dieselbe Knotenmenge $V = \{1, \ldots, n\}$ besitzen.

Definition 8.1 *Seien $G_i = (V, E_i)$, $i = 1, 2$, ungerichtete Graphen mit $V = \{1, \ldots, n\}$ und $E_i \subseteq \binom{V}{2}$. Eine Permutation $\varphi \in S_n$ heißt* **Isomorphismus** *zwischen G_1 und G_2, falls gilt*

$$\forall u, v \in V : \{u, v\} \in E_1 \Leftrightarrow \{\varphi(u), \varphi(v)\} \in E_2.$$

In diesem Fall heißen G_1 und G_2 **isomorph** *(in Zeichen $G_1 \cong G_2$).*

Setzen wir also $\varphi(E_1) = \{\{\varphi(u), \varphi(v)\} \mid \{u, v\} \in E_1\}$ und $\varphi(G_1) = (V, \varphi(E_1))$, so ist φ genau dann ein Isomorphismus zwischen G_1 und G_2, wenn $G_2 = \varphi(G_1)$ ist.

Graphisomorphieproblem (GI):

> ***Gegeben:*** Zwei ungerichtete Graphen G_1 und G_2.
>
> ***Gefragt:*** Sind G_1 und G_2 isomorph?

Es ist leicht zu sehen, dass GI in NP liegt. GI konnte bisher jedoch im Unterschied zu fast allen anderen Problemen in NP weder als NP-vollständig, noch als effizient lösbar (d.h. $\mathrm{GI} \in \mathrm{P}$) klassifiziert werden. Auch die Zugehörigkeit von GI zu $\mathrm{NP} \cap \mathrm{co\text{-}NP}$ ist offen.

Eng verwandt mit GI ist das Problem, für einen gegebenen Graphen die Existenz eines nichttrivialen Automorphismus' zu entscheiden.

Definition 8.2 *Sei $G = (V, E)$ ein Graph. Eine Permutation $\varphi \in S_n$ heißt* **Automorphismus** *von G (kurz: $\varphi \in Aut(G)$), falls $\varphi(G) = G$ ist.*

Es ist leicht zu sehen, dass $Aut(G)$ eine Untergruppe von S_n bildet. Insbesondere besitzt jeder Graph zumindest einen Automorphismus, nämlich die Identität id, die auch als trivialer Automorphismus bezeichnet wird.

Graphautomorphieproblem (GA):

> ***Gegeben:*** Ein ungerichteter Graph G.
>
> ***Gefragt:*** Besitzt G einen nichttrivialen Automorphismus?

Für einen Graphen $G = (V, E)$ mit $V = \{1, \ldots, n\}$ bezeichne

$$\mathit{Iso}(G) = \{\varphi(G) \mid \varphi \in S_n\}$$

die Menge aller Graphen mit Knotenmenge V, die isomorph zu G sind.

Lemma 8.3 *Für jeden Graphen $G = (V, E)$ mit $V = \{1, \ldots, n\}$ gilt*

(i) $\|\mathit{Iso}(G)\| = \frac{n!}{\|\mathit{Aut}(G)\|}$,

(ii) $\|\{(H, \pi) \mid H \in \mathit{Iso}(G), \pi \in \mathit{Aut}(H)\}\| = n!$.

Beweis

(i) Wir nennen zwei Permutationen φ und π äquivalent, falls $\varphi(G) = \pi(G)$ ist. Da $\mathit{Aut}(G)$ eine Untergruppe von S_n ist, folgt

$$\begin{aligned}\varphi(G) = \pi(G) &\Leftrightarrow \varphi^{-1}(\pi(G)) = G\\ &\Leftrightarrow \varphi^{-1} \circ \pi \in \mathit{Aut}(G)\\ &\Leftrightarrow \varphi^{-1} \circ \pi \circ \mathit{Aut}(G) = \mathit{Aut}(G)\\ &\Leftrightarrow \varphi \circ \mathit{Aut}(G) = \pi \circ \mathit{Aut}(G).\end{aligned}$$

Zwei Permutationen sind also genau dann äquivalent, wenn sie in der gleichen Nebenklasse von $\mathit{Aut}(G)$ liegen, d.h.

$$\|\{\varphi(G) \mid \varphi \in S_n\}\| = \|\{\varphi \circ \mathit{Aut}(G) \mid \varphi \in S_n\}\|.$$

Aus der Gruppentheorie wissen wir jedoch, dass die Nebenklassen von $\mathit{Aut}(G)$ die Gruppe S_n in gleichmächtige Teilmengen partitionieren und daher genau $\frac{n!}{\|\mathit{Aut}(G)\|}$ verschiedene Nebenklassen existieren.

(ii) Aus (i) folgt sofort

$$\|\{(H, \pi) \mid H \in \mathit{Iso}(G), \pi \in \mathit{Aut}(H)\}\| = \sum_{H \in \mathit{Iso}(G)} \underbrace{\|\mathit{Aut}(H)\|}_{=\|\mathit{Aut}(G)\|} = n!.$$

■

8.2 GI liegt in co-BP·NP

Als nächstes wollen wir zeigen, dass GI fast in co-NP liegt (genauer: $\mathrm{GI} \in \mathrm{BP} \cdot \text{co-NP}$ bzw. $\overline{\mathrm{GI}} \in \mathrm{BP} \cdot \mathrm{NP}$). Nach Satz 7.17 ist GI daher nicht NP-vollständig, außer wenn $\mathrm{PH} = \mathrm{BP} \cdot \mathrm{NP}$ ist. Wir betrachten zunächst folgende Verallgemeinerung des BP-Operators.

Definition 8.4 *Sei $\mathcal{C}$ eine Sprachklasse. Eine Sprache $L \subseteq \Sigma^*$ gehört zu $\widetilde{\mathrm{BP}} \cdot \mathcal{C}$, falls Funktionen $g > 0$ in FL und $f \in \# \cdot \mathcal{C}$ existieren, so dass für alle x gilt,*

$$\begin{aligned} x \in L &\Rightarrow f(x) \geq 2g(x), \\ x \notin L &\Rightarrow f(x) \leq g(x). \end{aligned}$$

Bei Anwendung des $\widetilde{\mathrm{BP}}$-Operators auf eine Sprachklasse $\mathcal{C}$ darf also anstelle der fest vorgegebenen Funktion $g(x) = 2^{p(|x|)}/3$ eine beliebige Funktion $g > 0$ aus der Klasse FL verwendet werden. Es ist leicht zu sehen, dass NP in der Klasse $\widetilde{\mathrm{BP}} \cdot \mathrm{P}$ enthalten ist (hierzu genügt es, für g die konstante Funktion $g(x) = 1/2$ zu wählen). Daher ist BPP vermutlich echt in $\widetilde{\mathrm{BP}} \cdot \mathrm{P}$ enthalten.

Theorem 8.5 $\overline{\mathrm{GI}} \in \widetilde{\mathrm{BP}} \cdot \mathrm{NP}$.

Beweis Seien zwei Graphen $G_i = (V, E_i)$, $i = 1, 2$, mit $V = \{1, \ldots, n\}$ gegeben. Nach Lemma 8.3 haben die Mengen

$$X(G_i) = \{(H, \pi) \mid H \in \mathit{Iso}(G_i), \pi \in \mathit{Aut}(H)\}$$

die Mächtigkeit $\|X(G_i)\| = n!$ und somit folgt

$$\begin{aligned} G_1 \cong G_2 &\Rightarrow X(G_1) = X(G_2) \Rightarrow \|X(G_1) \cup X(G_2)\| = n! \\ G_1 \ncong G_2 &\Rightarrow X(G_1) \cap X(G_2) = \emptyset \Rightarrow \|X(G_1) \cup X(G_2)\| = 2n!. \end{aligned}$$

Da die Sprache

$$B = \{\langle G_1, G_2 \rangle \# \langle H, \pi \rangle \mid (H, \pi) \in X(G_1) \cup X(G_2)\}$$

in NP entscheidbar und die Funktion $g(G_1, G_2) = n!$ in FL berechenbar ist, folgt $\overline{\mathrm{GI}} \in \widetilde{\mathrm{BP}} \cdot \mathrm{NP}$. ■

8.3 Lineare Hashfunktionen

Definition 8.6 *$Lin(m, k)$ bezeichne die Menge aller linearen Funktionen von $\{0,1\}^m$ nach $\{0,1\}^k$.*

Bemerkung 8.7 *Jede Funktion $h \in Lin(m, k)$ lässt sich eineindeutig durch eine Matrix $A_h = (a_{ij}) \in \{0,1\}^{k \times m}$ beschreiben, d.h. es gilt*

$$h(y_1 \cdots y_m) = z_1 \cdots z_k \Leftrightarrow \begin{pmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & & \vdots \\ a_{k1} & \cdots & a_{km} \end{pmatrix} \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix} = \begin{pmatrix} z_1 \\ \vdots \\ z_k \end{pmatrix}.$$

Bezeichnen wir also die Abbildung $y_1 \cdots y_m \mapsto a_{i1}y_1 \oplus \cdots \oplus a_{im}y_m$ mit h_i, so gilt $z_i = h_i(y_1 \cdots y_m)$ für $i = 1, \ldots, k$.

Lemma 8.8 *Sei $\emptyset \neq B \subseteq \{0,1\}^m - \{0^m\}$ und für eine zufällig unter Gleichverteilung gewählte Funktion $h \in_R Lin(m,k)$ sei S die ZV*

$$S = \|\{y \in B \mid h(y) = 0^k\}\|.$$

Dann gilt

$$E(S) = 2^{-k}\|B\|,$$
$$Var(S) = 2^{-k}(1-2^{-k})\|B\|.$$

Beweis Sei y ein beliebiger String in B. Wir zeigen zuerst, dass $h(y)$ jeden Wert $z \in \{0,1\}^k$ mit derselben Wk 2^{-k} annimmt. Wegen $y \neq 0^m$ existiert ein Index j mit $y_j = 1$. Da sich der Wert von $h_i(y)$ ändert, falls wir das Bit a_{ij} in A_h flippen, haben die beiden Mengen $H_0 = \{h \mid h_i(y) = 0\}$ und $H_1 = \{h \mid h_i(y) = 1\}$ die gleiche Mächtigkeit, was $\Pr[h_i(y) = 0] = \Pr[h_i(y) = 0] = 1/2$ impliziert. Da die einzelnen Zeilen von A_h unabhängig gewählt werden, sind auch die Werte $h_1(y), \ldots, h_k(y)$ unabhängig, und es folgt

$$\Pr[h(y) = 0^k] = \Pr[h_1(y) = \cdots = h_k(y) = 0] = \prod_{i=1}^{k} \underbrace{\Pr[h_i(y) = 0]}_{1/2} = 2^{-k}.$$

Wegen $S = \sum_{y \in B} S_y$, wobei

$$S_y := \begin{cases} 1, & h(y) = 0^k \\ 0, & \text{sonst.} \end{cases}$$

die Indikatorvariable für das Ereignis $h(y) = 0^k$ ist, folgt nun

$$E(S) = \sum_{y \in B} E(S_y) = \sum_{y \in B} \Pr[S_y = 1] = \sum_{y \in B} \Pr[h(y) = 0^k] = 2^{-k}\|B\|.$$

Weiter folgt wegen $S_y^2 = S_y$, dass

$$Var(S_y) = E(S_y^2) - E(S_y)^2 = 2^{-k} - 2^{-2k} = 2^{-k}(1-2^{-k}) < E(S)$$

ist. Als nächstes zeigen wir, dass die Zufallsvariablen S_y, $y \in B$, paarweise stochastisch unabhängig sind. Seien y, y' zwei Strings in B und sei j eine Position mit $y_j = 0$ und $y'_j = 1$ (falls nötig, vertauschen wir y und y'). Dann ändert sich durch Flippen von a_{ij} zwar der Wert von $h_i(y')$, aber $h_i(y)$ bleibt unverändert. Folglich sind die beiden Mengen $H'_0 = \{h \mid h_i(y) = 0 \wedge h_i(y') = 0\}$ und $H'_1 = \{h \mid h_i(y) = 0 \wedge h_i(y') = 1\}$ gleich groß, woraus

$$\Pr[h_i(y') = 0 \mid h_i(y) = 0] = 1/2.$$

folgt. Dies wiederum impliziert

$$\Pr[h_i(y) = h_i(y') = 0] = \underbrace{\Pr[h_i(y) = 0]}_{1/2} \cdot \underbrace{\Pr[h_i(y') = 0 \mid h_i(y) = 0]}_{1/2} = 1/4,$$

was

$$\Pr[S_y = S_{y'} = 1] = \prod_{i=1}^{k} \Pr[h_i(y) = h_i(y') = 0] \;=\; 4^{-k} = \Pr[S_y = 1] \cdot \Pr[S_{y'} = 1].$$

nach sich zieht. Da also S_y und $S_{y'}$ für $y \neq y'$ stochastisch unabhängig sind, folgt

$$\mathit{Var}(S) = \sum_{y \in B} \mathit{Var}(S_y) = 2^{-k}(1 - 2^{-k})\|B\|.$$

■

Theorem 8.9 $\mathrm{BP} \cdot \mathrm{NP} = \widetilde{\mathrm{BP}} \cdot \mathrm{NP}$.

Beweis Für die Inklusion von links nach rechts genügt es, für g die Funktion $g(x) = 2^{p(|x|)}/3$ zu wählen. Für die umgekehrte Inklusion sei L eine Sprache in $\widetilde{\mathrm{BP}} \cdot \mathrm{NP}$. Dann existieren Funktionen $g(x) > 0$ in FL und $f(x)$ in $\#\mathrm{NP}$ mit

$$\begin{aligned} x \in L &\;\Rightarrow\; f(x) \geq 2g(x), \\ x \notin L &\;\Rightarrow\; f(x) \leq g(x). \end{aligned}$$

Zu f existieren eine p-balancierte NP-Sprache A für ein Polynom p mit

$$f(x) = \#A(x) = \|\{y \in \{0,1\}^{p(|x|)} \mid x\#y \in A\}\|,$$

wobei wir o.B.d.A. annehmen, dass A keine Wörter der Form $x\#0^m$ enthält. Für eine Eingabe x sei

$$B_x = \{y_1 \ldots y_5 \mid \forall i = 1, \ldots, 5 : |y_i| = p(|x|) \text{ und } x\#y_i \in A\}.$$

Wegen $\|B_x\| = f(x)^5$ enthält B_x für alle $x \in L$ mindestens $2^5 g(x)^5$ und für alle $x \notin L$ höchstens $g(x)^5$ Strings der Länge $m(x) = 5p(|x|)$. Setze nun $k(x) = \lceil \log_2(2^2 g(x)^5) \rceil$ und betrachte die NP-Sprache

$$B' = \{x\#h \mid h \in \mathit{Lin}(m(x), k(x)) \text{ und } \exists y \in B_x : h(y) = 0^{k(x)}\}.$$

Dann ist für eine zufällig aus $\mathit{Lin}(m(x), k(x))$ gewählte Funktion h das Ereignis $x\#h \in B'$ gleichbedeutend mit dem Ereignis $S_x \geq 1$ für die Zufallsvariable

$$S_x = \|\{y \in B_x \mid h(y) = 0^{k(x)}\}\|.$$

Daher reicht es zu zeigen, dass das Ereignis $S_x \geq 1$ im Fall $x \in L$ mindestens mit Wk $2/3$ und im Fall $x \notin L$ höchstens mit Wk $1/3$ eintritt. Nach Lemma 8.8 gilt

$$\mathit{Var}(S_x) < \mathit{E}(S_x) = 2^{-k(x)}\|B_x\|.$$

Für $x \in L$ ist $\|B_x\| \geq 2^5 g(x)^5$, also $\mathit{E}(S_x) \geq 2^5 2^{-k(x)} g(x)^5 \geq 4$. Daher folgt mit Tschebyscheff

$$\Pr[S_x = 0] \leq \Pr[|S_x - E(S_x)| \geq E(S_x)] \leq \frac{\mathit{Var}(S_x)}{E(S_x)^2} \leq \frac{1}{E(S_x)} \leq \frac{1}{4},$$

was $\Pr[S_x \geq 1] \geq 3/4$ impliziert. Und da B_x im Fall $x \notin L$ höchstens $g(x)^5$ Strings enthält, folgt

$$\Pr[S_x \geq 1] = \sum_{i=1}^{\infty} \Pr[S_x = i] \leq \sum_{i=0}^{\infty} i \cdot \Pr[S_x = i] = E(S_x) \leq 2^{-k(x)} g(x)^5 \leq \frac{1}{4}.$$

■

Korollar 8.10 GI *ist nicht* NP*-vollständig, außer wenn* $\mathrm{PH} = \mathrm{BP} \cdot \mathrm{NP}$ *ist.*

9 Turing-Operatoren

In diesem Abschnitt betrachten wir Berechnungen, die Zugriff auf eine Orakelsprache A haben, d.h., die Information, ob bestimmte Wörter in A enthalten sind oder nicht, kann durch eine Orakelanfrage, deren Beantwortung nur einen Rechenschritt kostet, abgerufen werden. Auf diese Weise erhalten wir relativierte Versionen $\mathcal{C}^A$ von Komplexitätsklassen $\mathcal{C}$, die alle Probleme enthalten, die relativ zum Orakel A innerhalb der durch $\mathcal{C}$ vorgegebenen Ressourcen lösbar sind.

9.1 Orakel-Turingmaschinen

Definition 9.1 *Eine **deterministische Orakel-Turingmaschine (DOTM)** M ist eine DTM, die mit einem speziellen write-only **Orakelband** ausgerüstet ist. Außerdem besitzt M drei spezielle Zustände $q_?, q_+, q_-$. Als Orakel kann eine beliebige Sprache $A \subseteq \Sigma^*$ verwendet werden. Geht M in den **Fragezustand** $q_?$, so hängt der Folgezustand q' davon ab, ob das aktuell auf dem Orakelband stehende Wort y zu A gehört (in diesem Fall ist $q' = q_+$) oder nicht ($q' = q_-$). In beiden Fällen wird das Orakelband gelöscht und der Kopf an den Anfang zurückgesetzt. All dies passiert innerhalb eines einzigen Rechenschrittes. Die unter einem Orakel A arbeitende DOTM wird mit M^A bezeichnet und die von M^A **akzeptierte Sprache** ist $L(M^A)$.*

Wir verlangen von einer Orakel-Turingmaschine, dass sie vorgegebene Ressourcenschranken unabhängig vom benutzten Orakel einhält.

Definition 9.2 *Die **Rechenzeit** einer DOTM M bei Eingabe $x \in \Sigma^*$ ist*

$$time_M(x) = \sup_{A \subseteq \Sigma^*} time_{M^A}(x).$$

*M ist $t(n)$-**zeitbeschränkt**, falls für alle Eingaben x gilt:*

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

Der **Platzverbrauch** einer DOTM M ist analog definiert, wobei das write-only Orakelband unberücksichtigt bleibt. Eine polynomiell zeitbeschränkte DOTM M bezeichnen wir kurz als PDOTM. Alle unter einem Orakel A in Polynomialzeit akzeptierten Sprachen fassen wir in der Klasse

$$\mathrm{P}^A = \{L(M^A) \mid M \text{ ist eine PDOTM}\}$$

zusammen. P^A wird auch als die **Relativierung** der Klasse P zum Orakel A bezeichnet. Für eine Sprachklasse $\mathcal{C}$ sei

$$\mathrm{P}^{\mathcal{C}} = \bigcup_{A \in \mathcal{C}} \mathrm{P}^A.$$

Für $\mathrm{P}^{\mathcal{C}}$ (bzw. P^{A}) schreiben wir auch $\mathrm{P}(\mathcal{C})$ (bzw. $\mathrm{P}(A)$). Ebenso wie DTMs lassen sich auch NTMs und PTMs mit einem Orakelbefragungsmechanismus ausstatten, wodurch wir NOTMs und POTMs erhalten. Ist die Rechenzeit dieser Maschinen polynomiell beschränkt, so bezeichnen wir sie als NPOTMs bzw. PPOTMs. Entsprechend erhalten wir dann die relativierten Komplexitätsklassen NP^{A}, PP^{A}, BPP^{A}, RP^{A}, ZPP^{A}, L^{A}, NL^{A}, PSPACE^{A} usw.

Theorem 9.3

(i) $\mathrm{P}^{\mathrm{P}} = \mathrm{P}$ *und* $\mathrm{NP}^{\mathrm{P}} = \mathrm{NP}$,

(ii) $\mathrm{P}^{\mathrm{NP}\cap\text{co-NP}} = \mathrm{NP} \cap \text{co-NP}$ *und* $\mathrm{NP}^{\mathrm{NP}\cap\text{co-NP}} = \mathrm{NP}$,

(iii) $\mathrm{NP}^{\mathrm{NP}} = \Sigma_2^p$ *und* $\mathrm{NP}^{\Sigma_k^p} = \Sigma_{k+1}^p$ *für* $k \geq 0$.

Beweis *(i)* Die Inklusion $\mathrm{P} \subseteq \mathrm{P}(\mathrm{P})$ ist klar. Für die umgekehrte Richtung sei L eine Sprache in $\mathrm{P}(\mathrm{P})$. Dann existiert eine PDOTM M und ein Orakel $A \in \mathrm{P}$ mit $L(M^A) = L$. Sei M' eine PDTM mit $L(M') = A$. Betrachte die DOTM $M''(x)$, die $M(x)$ simuliert und jedesmal, wenn M eine Orakelfrage y stellt, $M'(y)$ simuliert, um die Zugehörigkeit von y zu A zu entscheiden. Dann gilt

$$L(M'') = L(M^A) = L$$

und da die Beantwortung einer Orakelfrage höchstens Zeit

$$\max_{y,|y|\leq time_M(x)} time_{M'}(y) = |x|^{O(1)}$$

erfordert, ist M'' polynomiell zeitbeschränkt. Die Gleichheit von NP^{P} und NP lässt sich vollkommen analog zeigen.

(ii) Es reicht, $\mathrm{P}^{\mathrm{NP}\cap\text{co-NP}} \subseteq \mathrm{NP} \cap \text{co-NP}$ zu zeigen. Sei $L = L(M^A)$ für eine POTM M und sei A ein Orakel in $\mathrm{NP} \cap \text{co-NP}$. Dann existieren NPTMs M' und M'' mit $L(M') = A$ und $L(M'') = \overline{A}$. Betrachte folgende NPTM M^*:

> $M^*(x)$ simuliert $M(x)$ und jedesmal wenn M eine Orakelfrage y stellt, entscheidet sich M^* nichtdeterministisch dafür, entweder $M'(y)$ oder $M''(y)$ zu simulieren. Falls $M'(y)$ (bzw. $M''(y)$) akzeptiert, führt M^* die Simulation von M im Zustand q_+ (bzw. q_-) fort. Anderfalls bricht M^* die Simulation von M ab und verwirft x.

Nun gilt $L(M^*) = L(M^A) = L$ und daher ist $L \in \mathrm{NP}$. Da $\mathrm{P}^{\mathrm{NP}\cap\text{co-NP}}$ unter Komplementbildung abgeschlossen ist, ist $\mathrm{P}^{\mathrm{NP}\cap\text{co-NP}}$ auch in co-NP enthalten. Die Inklusion von $\mathrm{NP}^{\mathrm{NP}\cap\text{co-NP}}$ in NP folgt vollkommen analog.

(iii) Wir zeigen zuerst die Inklusion von Σ_2^p in $\mathrm{NP}^{\mathrm{NP}}$. Zu jeder Sprache $L \in \Sigma_2^p = \exists \cdot \text{co-NP}$ existieren eine Sprache $A \in \text{co-NP}$ und ein Polynom p mit

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

Dann ist $\bar{A} \in \mathrm{NP}$ und L wird von der NPOTM M relativ zum Orakel $\bar{A}$ akzeptiert, die bei Eingabe x ein Wort $y \in \{0,1\}^{p(|x|)}$ rät und bei negativer Antwort auf die Orakelfrage $x\#y$ akzeptiert.

Für die umgekehrte Richtung sei $L = L(M^A)$ für eine NPOTM M, deren Rechenzeit durch ein Polynom p und deren Verzweigungsgrad durch 2 beschränkt ist, und für ein NP-Orakel A. Dann existieren ein Polynom q und eine Sprache $B \in \mathrm{P}$ mit

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

Nun können wir jede Rechnung α von $M(x)$ durch ein Wort $r = r_1 \cdots r_{p(|x|)} \in \{0,1\}^{p(|x|)}$ kodieren, wobei r_i im Fall, dass α im i-ten Rechenschritt nichtdeterministisch verzweigt, die Richtung, und im Fall, dass α im i-ten Rechenschritt eine Orakelfrage stellt, die Antwort angibt. Nun gilt

$$\begin{aligned} x \in L \quad \Leftrightarrow \quad & \exists r \in \{0,1\}^{p(|x|)} \exists z_1, \ldots, z_{p(|x|)} \in \{0,1\}^{q(p(|x|))} : \\ & x\#rz_1, \ldots, z_{p(|x|)} \in C, \end{aligned}$$

wobei C alle Strings $x\#rz_1, \ldots, z_{p(|x|)}$ enthält mit

- r kodiert eine akzeptierende Rechnung von $M(x)$, während der m Orakelfragen $y_1, \ldots, y_m$ gestellt und mit $a_1, \ldots, a_m$ beantwortet werden, und
- für $i = 1, \ldots, m$ gilt $(a_i = \text{ja} \wedge y_i\#(z_i)_{\leq q(|y_i|)} \in B) \vee (b_i = \text{nein} \wedge y_i \notin A)$,

wobei $(z)_{\leq k}$ das Präfix der Länge k von z bezeichnet. Da C in co-NP liegt, zeigt diese Charakterisierung, dass L in $\exists \cdot \text{co-NP}$ enthalten ist. ■

9.2 Das relativierte P/NP-Problem

Theorem 9.4 (Baker, Gill und Solovay, 1975) *Es gibt Orakel A und B mit*

$$\mathrm{P}(A) = \mathrm{NP}(A) \text{ und } \mathrm{P}(B) \neq \mathrm{NP}(B).$$

Beweis Wählen wir für A eine PSPACE-vollständige Sprache (etwa QBF), so gilt

$$\mathrm{NP}(A) \subseteq \mathrm{NPSPACE} = \mathrm{PSPACE} \subseteq \mathrm{P}(A).$$

Für die Konstruktion eines Orakels $B \subseteq \{0,1\}^*$ mit $\mathrm{P}(B) \neq \mathrm{NP}(B)$ betrachten wir die Testsprache

$$L(B) = \{0^n \mid B \cap \{0,1\}^n \neq \emptyset\}.$$

Da $L(B)$ für jedes Orakel B zu $\mathrm{NP}(B)$ gehört, genügt es, B mittels Diagonalisierung so zu konstruieren, dass $L(B)$ nicht in $\mathrm{P}(B)$ enthalten ist.

Sei $M_1, M_2, \ldots$ eine Aufzählung von PODTMs mit $\mathrm{P}^C = \{L(M_i^C) \mid i \geq 1\}$, wobei wir annehmen, dass die Laufzeit von M_i durch das Polynom $n^i + 1$ beschränkt ist,

$$time_{M_i}(x) \subseteq (|x|)^i + 1.$$

Wir konstruieren B stufenweise als Vereinigung von Sprachen B_i, wobei B_i aus B_{i-1} durch Hinzufügen maximal eines Wortes y der Länge n_i entsteht und die Zahlenfolge n_i, $i \geq 0$, induktiv wie folgt definiert ist:

$$n_i = \begin{cases} 0, & i = 0, \\ \min\{n > (n_{i-1})^{i-1} + 1 \mid n^i < 2^n\}, & i > 0. \end{cases}$$

Die Bedingung $n_{i+1} > (n_i)^i + 1$ stellt sicher, dass $M_i^B(0^{n_i})$ das Orakel über kein Wort y der Länge $|y| \geq n_{i+1}$ befragen kann, und die Bedingung $(n_i)^i < 2^{n_i}$ garantiert, dass $M_i^B(0^{n_i})$ nicht alle Wörter der Länge n_i als Orakelfrage stellt.

Stufenkonstruktion von $B = \bigcup_{i \geq 1} B_i$:

Stufe 0**:** $B_0 = \emptyset$.

Stufe $i \geq 1$**:** Falls $M_i^{B_{i-1}}(0^{n_i})$ akzeptiert, setze $B_i = B_{i-1}$. Andernfalls setze $B_i = B_{i-1} \cup \{y\}$, wobei y das lexikografisch kleinste Wort der Länge n_i ist, das von $M_i^{B_{i-1}}(0^{n_i})$ nicht als Orakelfrage gestellt wird.

B_i wird also in Stufe i so definiert, dass 0^{n_i} in einer der beiden Mengen $L(M_i^{B_{i-1}}(0^{n_i}))$ und $L(B_i)$ enthalten ist, aber nicht in der anderen. Da die Wahl von y in Stufe i zudem garantiert, dass sich $M_i(0^{n_i})$ relativ zu B_i und zu B_{i-1} gleich verhält, und da auch das Hinzufügen weiterer Strings y mit $|y| \geq n_{i+1}$ zu B in den Stufen $j > i$ keinen Einfluss auf dieses Verhalten hat, folgt $0^{n_i} \in L(M_i^B) \triangle L(B)$ und somit $L(B) \notin \mathrm{P}(B)$. ■

Fast alle bisher in der Komplexitätstheorie erzielten Resultate wurden mit relativierbaren Beweistechniken erzielt und gelten daher relativ zu einem beliebigem Orakel. Beispiele hierfür sind alle in dieser Vorlesung gezeigten Inklusionen und Separierungen von Komplexitätsklassen wie

$$\mathrm{DTIME}^A(f) \subseteq \mathrm{NTIME}^A(f) \subseteq \mathrm{DSPACE}^A(f) \subseteq \mathrm{NSPACE}^A(f) \subseteq \mathrm{DTIME}^A(2^{O(f)}),$$

die Zeit- und Platzhierarchiesätze wie

$$\mathrm{DTIME}^A(g(n)) \subsetneq \mathrm{DTIME}^A(f(n)),$$

falls $g(n) \cdot \log g(n) = o(f(n))$, oder der Satz von Savitch,

$$\mathrm{NSPACE}^A(s(n)) \subseteq \mathrm{DSPACE}^A(s^2(n))$$

und der Satz von Immerman/Szelepczényi,

$$\mathrm{NSPACE}^A(s(n)) = \text{co-}\mathrm{NSPACE}^A(s(n)),$$

falls $s(n) \geq \log n$. Wie wir gerade gesehen haben, hängt dagegen die Antwort auf die Frage, ob $\mathrm{P}(A) = \mathrm{NP}(A)$ ist, von der Wahl des Orakels A ab. Daher lässt sich das P/NP-Problem nicht mit relativierbaren Beweismethoden bewältigen. Es gibt sogar relativierte Welten, in denen alle Stufen der Polynomialzeithierarchie verschieden sind.

Theorem 9.5 *Es existiert ein Orakel B, so dass $\Sigma_k^P(B) \neq \Sigma_{k+1}^P(B)$ für alle $k \geq 0$ (und somit $\mathrm{PH}(B) \neq \mathrm{PSPACE}(B)$) gilt.*

Im Jahr 1981 zeigten Bennet und Gill, dass man bei zufälliger Wahl des Orakels A

$$\mathrm{P}(A) \neq \mathrm{NP}(A) \neq \text{co-NP}(A)$$

sogar mit Wahrscheinlichkeit 1 erhält, d.h. die Klassen P, NP und $\mathrm{co\text{-}NP}$ sind unter fast allen Orakeln verschieden. Die Frage, ob PH auch relativ zu einem Zufallsorakel echt ist, ist dagegen noch offen. Die in den 80ern aufgestellte **Zufallsorakelhypothese** besagt, dass eine relativierte Aussage wie $\mathrm{P}(A) \neq \mathrm{NP}(A)$ genau dann relativ zu einem Zufallsorakel mit Wk 1 gilt, wenn sie unrelativiert gilt. Diese Hypothese wurde mehrfach widerlegt. Bekanntestes Beispiel ist die Gleichheit $\mathrm{IP} = \mathrm{PSPACE}$ (siehe Kapitel 11), obwohl $\mathrm{IP}^A \neq \mathrm{PSPACE}^A$ mit Wk 1 gilt.