Vorlesungsskript

Komplexitätstheorie

Wintersemester 2010/11

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

7. Februar 2011

Inhaltsverzeichnis

Inhaltsverzeichnis

1 Einführung

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

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

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

Erreichbarkeitsproblem in 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:

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

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

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

Algorithmus `suche-Weg`(G)

```
1  Input: Gerichteter Graph G = (V,E) mit V = {1,...,n}
2    S := {1}
3    markiere Knoten 1
4    repeat
5      waehle einen 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   until S = ∅
12   if n ist markiert then accept else reject
```

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

Komplexitätsbetrachtungen:

- REACH ist in Zeit n^3 entscheidbar.

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

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

Maximaler Fluß (MaxFlow):

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

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

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

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

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

Komplexitätsbetrachtungen:

- MaxFlow ist in Zeit n^5 lösbar.
- MaxFlow ist in Platz n^2 lösbar.

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

Perfektes Matching in bipartiten Graphen (Matching):

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

Gefragt: Besitzt G ein perfektes Matching?

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

Komplexitätsbetrachtungen:

- Matching ist in Zeit n^3 entscheidbar.
- Matching ist in Platz n^2 entscheidbar.

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

Travelling Salesman Problem (TSP):

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

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

Komplexitätsbetrachtungen:

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

[a] Hierzu berechnen wir für alle Teilmengen $S \subseteq \{2, \ldots, n\}$ 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 1 (**Mehrband-Turingmaschine**)**.**

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Dabei ergibt sich y aus $u_k v_k$, indem das erste Symbol $\triangleright$ und sämtliche Blanks am Ende entfernt werden, d. h. $u_k v_k = \triangleright y \sqcup^i$ für ein $i \geq 0$. Für $M(x) = ja$ sagen wir auch „$M(x)$ akzeptiert“ und für $M(x) = nein$ „$M(x)$ verwirft“.

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

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

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

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

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

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

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

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

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

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

2.2 Nichtdeterministische Berechnungen

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

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

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

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

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

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

oder einfach $(q, a_1, \ldots, a_k) \mapsto (q', a_1', D_1, \ldots, a_k', D_k)$.
Wir werden NTMs nur zum Erkennen von Sprachen (d.h. als Akzeptoren) und nicht zum Berechnen von Funktionen benutzen.

Definition 11. *Sei M eine* NTM.

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

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

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

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

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

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

2.3 Zeitkomplexität

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

Definition 12.

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

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

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

b) *Sei $t : \mathbb{N} \to \mathbb{N}$ eine Funktion. Dann ist M $t(n)$-**zeitbeschränkt**, falls für alle $n \geq 0$ und alle $x \in \Sigma^*$ mit $|x| \leq n$ gilt:*

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

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

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

bzw.

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

zusammen. Ferner sei

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

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

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

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

2.4 Platzkomplexität

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

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

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

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

Dies bedeutet, dass eine Offline-TM nicht auf das Eingabeband schreiben darf (*read-only*). Beim Transducer dient das letzte Band als Ausgabeband, auch 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 Anzahl aller während der Rechnung besuchten Bandfelder.

Definition 14.

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

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

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

*b) Sei $s : \mathbb{N} \to \mathbb{N}$. Dann ist M $s(n)$**-platzbeschränkt**, falls für alle $n \geq 0$ und alle $x \in \Sigma^*$ mit $|x| \leq n$ gilt:*

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

D.h., $space_M(x)$ ist undefiniert, falls $time_M(x) = \infty$ undefiniert ist.

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

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

bzw.

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

zusammen. Ferner sei

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

Die wichtigsten Platzkomplexitätsklassen sind

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

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

3 Grundlegende Beziehungen

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

3.1 Robustheit von Komplexitätsklassen

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

Lemma 15 (Bandreduktion)**.**
*Zu jeder $s(n)$-platzbeschränkten Offline-*DTM *M ex. eine $s(n)$-platzbeschränkte Offline-2-*DTM *M' mit $L(M') = L(M)$.*

Beweis. Sei $M = (Q, \Sigma, \Gamma, \delta, q_0)$ eine Offline-k-DTM mit $k \geq 3$. Betrachte die Offline-*2*-DTM $M' = (Q', \Sigma, \Gamma', \delta', q_0')$ mit $\Gamma' = \Gamma \cup (\Gamma \cup \hat{\Gamma})^{k-1}$, wobei $\hat{\Gamma}$ für jedes $a \in \Gamma$ die markierte Variante $\hat{a}$ enthält. M' hat dasselbe Eingabeband wie M, speichert aber die Inhalte von $(k-1)$ übereinander liegenden Feldern der Arbeitsbänder von M auf einem Feld ihres Arbeitsbandes. Zur Speicherung der Kopfpositionen von M werden Markierungen benutzt.

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

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

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

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

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

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

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

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

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

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

mit

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

wobei

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

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

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

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

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

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

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

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

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

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

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

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

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 17.

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

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

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

Beweis. i) Sei $L \in \mathsf{DSPACE}(cs(n))$ für eine Konstante $c \geq 0$. Ist $s(n) < 4$ für unendlich viele n, so folgt $L \in \mathsf{DSPACE}(\mathcal{O}(1)) = \mathsf{DSPACE}(0)$. Gilt dagegen $s(n) \geq 4$ für fast alle n, so existiert für $c' = 1/2c$ eine Offline-k-DTM M, die L für fast alle Eingaben in Platz $2 + c'cs(n) = 2 + s(n)/2 \leq s(n)$ entscheidet. Wegen $s(n) \geq 2$ können wir M leicht so modifizieren, dass sie auch die endlich vielen Ausnahmen in Platz $s(n)$ entscheidet.

ii) Sei $L \in \mathsf{DTIME}(ct(n))$ für ein $c \geq 0$. Nach vorigem Satz existiert für $c' = \varepsilon/(1+\varepsilon)c$ eine DTM M, die L in Zeit $2 + n + c'ct(n) = 2 + n + \varepsilon t(n)/(1+\varepsilon) \leq 2 + (t(n) + \varepsilon t(n))/(1+\varepsilon) = t(n)$ entscheidet.

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

3.2 Deterministische Simulationen von nichtdeterministischen Berechnungen

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

Satz 18.

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

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

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

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

der maximale Verzweigungsgrad von N. Dann ist jede Rechnung

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

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 $L \in \mathsf{NSPACE}(s(n))$ und sei $N = (Q, \Sigma, \Gamma, \delta, q_0)$ eine Offline-*2*-NTM, die L in Platz $s(n)$ entscheidet. Bei einer Eingabe x der Länge n kann N

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

D.h. ausgehend von der Startkonfiguration K_x kann N in Platz $s(n)$ höchstens

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

verschiedene Konfigurationen erreichen, wobei c eine von N abhängige Konstante ist. Wie im Beweis von i) sei d der maximale Verzweigungsgrad von N. Um N zu simulieren, generiert M auf dem Band 3 für $t = 1, 2, \ldots$ der Reihe nach alle Folgen $(d_1, \ldots, d_t) \in \{1, \ldots, d\}^t$ und simuliert die zugehörige Rechnung von $N(x)$ auf den ersten beiden Bändern. 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 die Laufzeit von M durch

$$time_M(x) \;\leq\; t(n)^{\mathcal{O}(1)} \;=\; 2^{O(s(n)+\log n)}$$

beschränkt ist. ■

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

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

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

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

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

Insbesondere erhalten wir somit die Inklusionskette

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

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

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

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

3.3 Der Satz von Savitch

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

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

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

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

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

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

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

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

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

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

Initialisierung: $M(x)$ schreibt das Tripel $(K_x, \hat{K}_x, \lceil s(|n|)\log c\rceil)$ auf das 2. Band, wobei für das Eingabeband nur die Kopfposition, nicht jedoch die Beschriftung notiert wird (also z.B. $K_x = (q_0, 1, \varepsilon, \triangleright)$). Während der Simulation wird auf dem 2. Band ein Keller (*stack*) von Tripeln der Form (K, K', i) implementiert, die jeweils für die Frage stehen, ob $K \xrightarrow[N]{}^{\leq 2^i} K'$ 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, K', i) das am weitesten rechts auf dem 2. Band stehende Tripel (also das oberste Kellerelement).

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

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

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

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

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

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

Korollar 22.

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

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

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

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

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

3.4 Der Satz von Immerman und Szelepcsényi

Definition 23.

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

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

Beispiel 24.

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

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

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

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

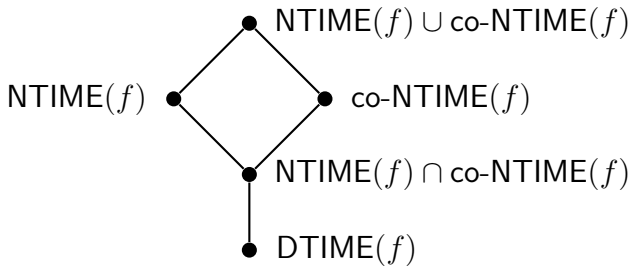
2) *Wir wir gesehen haben, impliziert der Satz von Savitch den Abschluss von* NPSPACE *unter Komplementbildung.*
3) *Dagegen wurde die Frage ob die Klasse* $\mathsf{CSL} = \mathsf{NSPACE}(n)$ *der kontextsensitiven Sprachen unter Komplementbildung abgeschlossen ist, erst in den 80ern gelöst (siehe Satz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

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

Proposition 25.

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

Damit ergibt sich folgende Inklusionsstruktur:



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

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

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

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

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

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

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

$$L_0 = \{(x, r, t, K) | t \geq 1 \text{ und } K_x \xrightarrow[N]{}^{\leq t} K\}.$$

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* $(x, r(x, t-1), t, K)$*,* $t \geq 1$*, strong ist.*

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

Pseudocode für $N_0(x, r, t, K)$

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

Da N_0 genau dann eine akzeptierende Rechnung hat, wenn eine Konfiguration K_i mit $K_x \xrightarrow[N]{}^{\leq t-1} K_i$ und $K_i \xrightarrow[N]{}^{\leq 1} K$ 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 $x_0 = (x, r(x, t-1), t, K)$, $t \geq 1$, strong ist, also $N_0(x_0)$ genau im Fall $x_0 \notin L_0$ eine verwerfende Endkonfiguration erreichen kann.

Um bei Eingabe x_0 eine verwerfende Endkonfiguration zu erreichen, muss N_0 $r = r(x, t-1)$ Konfigurationen K_i finden, für die zwar $K_x \xrightarrow[N]{}^{\leq t-1} K_i$ aber nicht $K_i \xrightarrow[N]{}^{\leq 1} K$ 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.

Pseudocode für $N'(x)$

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

Behauptung 2. *Im t-ten Durchlauf der repeat-Schleife wird r^- in Zeile 5 auf den Wert $r(x, t-1)$ gesetzt. Folglich wird N_0 von N' in Zeile 8 nur mit Eingaben der Form $(x, r(x, t-1), t, K_i)$ aufgerufen.*

Beweis der Behauptung. Wir führen Induktion über t:

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

$t \rightsquigarrow t + 1$**:** Da r^- zu Beginn des $t+1$-ten Durchlaufs auf den Wert von r gesetzt wird, müssen wir zeigen, dass r im t-ten Durchlauf auf $r(x, t)$ hochgezählt wird. Nach Induktionsvoraussetzung wird N_0 im t-ten Durchlauf nur mit Eingaben der Form $(x, r(x, t-1), t, K_i)$ aufgerufen. Da N_0 wegen Beh. 1 auf all

diesen Eingaben strong ist und keine dieser Simulationen im Zustand q_h endet (andernfalls würde N' sofort stoppen), werden alle in $\leq t$ Schritten erreichbaren Konfigurationen K_i als solche erkannt und somit wird r tatsächlich auf den Wert $r(x,t)$ hochgezählt. ■

Behauptung 3. *Bei Beendigung der repeat-Schleife in Zeile 15 gilt* $r = r^- = \|\{K | K_x \xrightarrow[N]{}^* K\}\|$.

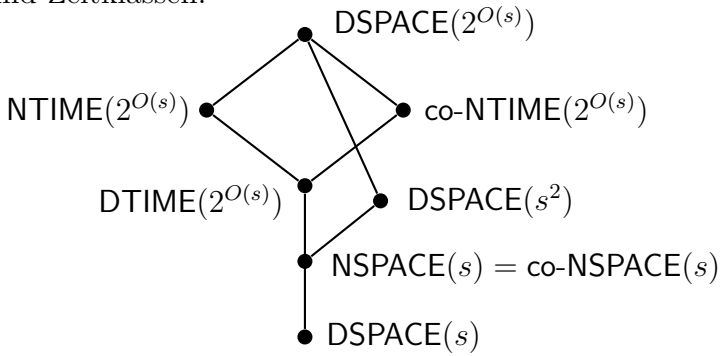
Beweis der Behauptung. Wir wissen bereits, dass im t-ten Durchlauf der repeat-Schleife r den Wert $r(x,t)$ und r^- den Wert $r(x,t-1)$ erhält. Wird daher die repeat-Schleife nach t_e Durchläufen verlassen, so gilt $r = r^- = r(x,t_e) = r(x,t_e-1)$.
Angenommen $r(x,t_e) < \|\{K | K_x \xrightarrow[N]{}^*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

Da N' offenbar die Sprache L in Platz $O(s(n))$ entscheidet, bleibt nur noch zu zeigen, dass N' bei allen Eingaben strong ist. Wegen Behauptung 3 hat $N'(x)$ genau dann eine verwerfende Rechnung, wenn im letzten Durchlauf der repeat-Schleife alle erreichbaren Konfigurationen K als solche erkannt werden und darunter keine akzeptierende Endkonfiguration ist. Dies impliziert $x \notin L$. Umgekehrt ist leicht zu sehen, dass $N'(x)$ im Fall $x \notin L$ eine verwerfende Rechnung hat. ■

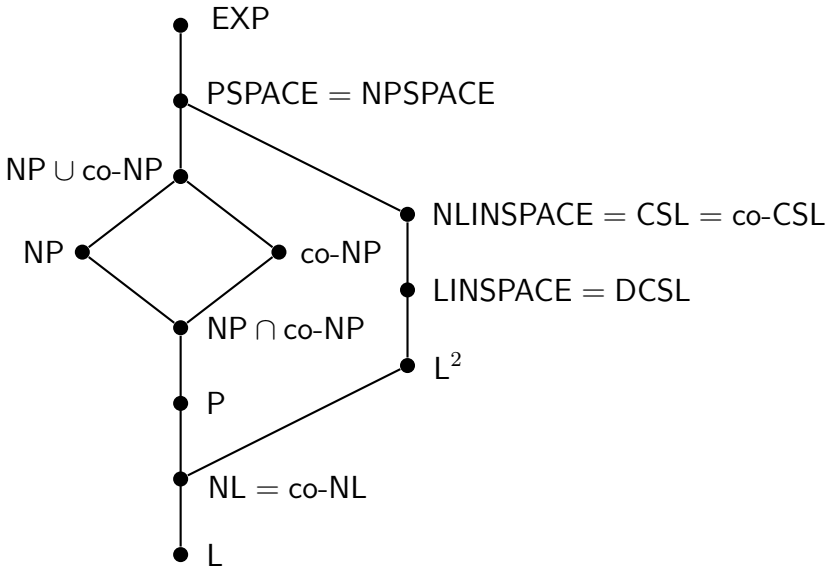
Korollar 28.

1. NL = co-NL,
2. CSL = NLINSPACE = co-CSL.

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



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



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

4 Hierarchiesätze

4.1 Diagonalisierung und die Unentscheidbarkeit des Halteproblems

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

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

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

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

wobei

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

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

Definition 29. *Das **Halteproblem** ist*

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

Satz 30. *H ist rekursiv aufzählbar, aber nicht entscheidbar.*

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

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

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

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

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

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$
M_1	**ja**	↑	nein	nein	$\cdots$
M_2	nein	**↑**	nein	↑	$\cdots$
M_3	ja	↑	**nein**	↑	$\cdots$
M_4	↑	nein	↑	**ja**	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

M_d	**nein**	**nein**	**ja**	**nein**	$\cdots$

Folglich kann keine Zeile dieser Matrix mit M_d übereinstimmen:

$$\langle M_d\rangle \in D \overset{(*)}{\Rightarrow} M_d(\langle M_d\rangle) = \text{nein} \overset{(**)}{\Rightarrow} \langle M_d\rangle \notin D \quad \lightning$$
$$\langle M_d\rangle \notin D \overset{(*)}{\Rightarrow} M_d(\langle M_d\rangle) \neq \text{nein} \overset{(**)}{\Rightarrow} \langle M_d\rangle \in D \quad \lightning$$

■

Satz 31. *Für jede rekursive Funktion $f : \mathbb{N} \longrightarrow \mathbb{N}$ existiert eine rekursive Sprache $D_f \notin \mathsf{DTIME}(f(n))$.*

Beweis. Wir definieren

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

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

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

Dies führt jedoch auf einen Widerspruch:

$$\langle M_d\rangle \in D_f \overset{(*)}{\Rightarrow} M_d(\langle M_d\rangle) \text{ verw.} \overset{(**)}{\Rightarrow} \langle M_d\rangle \notin D_f \quad \lightning$$
$$\langle M_d\rangle \notin D_f \overset{(*,**)}{\Rightarrow} M_d(\langle M_d\rangle) \text{ akz.} \overset{(**)}{\Rightarrow} \langle M_d\rangle \in D_f \quad \lightning$$

■

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

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

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

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

Betrachte hierzu das Prädikat:

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

Hierbei bezeichnet Σ_i das Eingabealphabet von M_i. 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 $f(n)$ wie folgt induktiv definieren:

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

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

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

4.3 Zeit- und Platzhierarchiesätze

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

Satz 33. *Für jede echte Komplexitätsfunktion $f(n) \geq n+2$ gilt*

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

Beweis. Betrachte folgende 4-DTM M':

Initialisierung: M' überprüft bei einer Eingabe x der Länge n 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 $\mathcal{O}(kbn) = \mathcal{O}(n^2)$. Abschließend erzeugt M' auf dem 4. Band den String $1^{f(n)}$ in Zeit $\mathcal{O}(f(n))$.

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 $\mathcal{O}(kbf(n)) = \mathcal{O}(n \cdot f(n)))$.

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' $\mathcal{O}(n \cdot f(n)^2)$-zeitbeschränkt ist und die Sprache D_f entscheidet. ■

Korollar 34. *(Zeithierarchiesatz)*
Für jede echte Komplexitätsfunktion $f(n) \geq n+2$ gilt

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

Korollar 35.

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

Beweis.

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

■

Aus dem Beweis von Satz 33 können wir weiterhin die Existenz einer universellen TM folgern.

Korollar 36. *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.

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

Satz 37. *Sei $f \geq n+2$ eine echte Komplexitätsfunktion und gelte*

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

Dann ist

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

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

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

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

Satz 38 (Platzhierarchiesatz)**.** *Sei f eine echte Komplexitätsfunktion und gelte*

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

Dann ist

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

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

Korollar 39.

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

Durch Kombination der Beweistechnik von Satz 38 mit der Technik von Immerman und Szelepcsényi erhalten wir auch für nichtdeterministische Platzklassen eine sehr fein abgestufte Hierarchie.

Satz 40 (Nichtdeterministischer Platzhierarchiesatz)**.** *Sei f eine echte Komplexitätsfunktion und gelte*

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

Dann ist

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

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

Satz 41 (Nichtdeterministischer Zeithierarchiesatz)**.** *Sei $f(n) \geq n+2$ eine echte Komplexitätsfunktion und gelte*

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

Dann ist

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

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

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

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

$$s(x) = \begin{cases} 0, & x = \varepsilon \\ h(s(x^-) + |x|) + 1, & \text{sonst} \end{cases}$$

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

- die Sprache

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

  ist von einer 2-NTM M_D in Zeit $time_{M_D}(x) \leq h(|x|)$ entscheidbar.
- die Funktion $0^n \to 0^{h(n)}$ ist von einem Transducer T in Zeit $h(n)+1$ berechenbar, d.h. $T(0^n)$ schreibt in jedem Rechenschritt (außer dem ersten) eine weitere Null auf's Ausgabeband.

Betrachte folgende NTM M:

```
1  input 0^n
2    x := ε
3    s := 0
4    while h(s + |x|) + 1 ≤ n do
5      s := h(s + |x|) + 1
6      x := x^+
7    if n < h(s + |x|) then (* s = s(x) ≤ n < s(x^+) − 1 *)
8      akz. falls M_{i(x)}(0^{n+1}) in ≤ f(n)/|⟨M_{i(x)}⟩| Schritten akz.
9    else (* n = s(x^+) − 1 *)
10     akz. falls 0^s#⟨M_{i(x)}⟩ ∈ D ist
```

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

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

L kann aber nicht in $\mathsf{NTIME}(g(n))$ enthalten sein, da sonst eine Konstante c und eine 2-NTM M_i ex. würden mit $L(M_i) = L$ und $time_{M_i}(0^n) \leq cg(n)$ für fast alle n (siehe Übungen; Simulation von NTMs durch 2-NTMs). Wählen wir nun $k \geq 0$ so groß, dass für $x = 0^k\#\langle M_i\rangle$ und alle $n \geq s(x)$ die Ungleichung $|\langle M_i\rangle| time_{M_i}(0^{n+1}) \leq f(n)$ gilt, so folgt für alle $n \in [s(x), s(x^+) - 2]$:

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

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

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

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

Satz 41 liefert für langsam wachsende Zeitschranken eine feinere Hierarchie als Satz 37. Beispielsweise impliziert Satz 41, dass $\mathsf{NTIME}(n^k)$ für jede unbeschränkte monotone Funktion h echt in der Klasse $\mathsf{NTIME}(n^k h(n))$ enthalten ist, da $(n+1)^k = \mathcal{O}(n^k) = o(n^k h(n))$ ist.

Für schnell wachsende Zeitschranken liefert dagegen Satz 37 eine feinere Hierarchie. So impliziert Satz 37 zum Beispiel, dass die Klasse $\mathsf{DTIME}(2^{2^n})$ für jede unbeschränkte monotone Funktion h echt in $\mathsf{DTIME}(h(n)2^n 2^{2^n})$ enthalten ist, während sich $\mathsf{NTIME}(2^{2^n})$ mit Satz 41 nur von $\mathsf{NTIME}(h(n)2^{2^{n+1}}) = \mathsf{NTIME}(h(n)2^n 2^{2^n})$ separieren lässt.

5 Reduktionen

5.1 Logspace-Reduktionen

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

Definition 42. *Seien A und B Sprachen über einem Alphabet Σ. A ist auf B* ***logspace-reduzierbar*** *(in Zeichen: $A \leq_m^{log} B$ oder einfach $A \leq B$), falls eine Funktion $f \in \mathsf{FL}$ existiert, so dass für alle $x \in \Sigma^*$ gilt,*

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

Lemma 43. $\mathsf{FL} \subseteq \mathsf{FP}$.

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

Beispiel 44. *Wir reduzieren das Hamiltonkreisproblem auf das Erfüllbarkeitsproblem* Sat *für aussagenlogische Formeln.*

Hamiltonkreisproblem (Ham):

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

Erfüllbarkeitsproblem für boolesche Formeln (Sat):

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

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

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

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

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

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

c) Jeder Knoten j wird mindestens einmal besucht:

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

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

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

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

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

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

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

Definition 45.

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

b) Eine C*-harte Sprache, die zur Klasse* C *gehört, heißt* C-***vollständig****.*

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

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

Lemma 46.

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

Beweis. Siehe Übungen. ■

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

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

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

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

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

Bemerkung: Die Anzahl der Eingänge eines Gatters g wird als **Fan-in** von g bezeichnet, die Anzahl der Ausgänge (also die Anzahl der Gatter, die g als Eingabe benutzen) als **Fanout**. Boolesche Formeln entsprechen also den booleschen Schaltkreisen mit (maximalem) Fan-out 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 48. *Für die Reduktion* Reach $\leq$ CirVal *benötigen wir eine Funktion $f \in \mathsf{FL}$ mit der Eigenschaft, dass für alle Graphen G gilt:*

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

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

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

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

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

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

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

Dann folgt

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

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 48 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 49. CirVal *ist* P*-vollständig.*

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Eine leichte Modifikation des Beweises von Satz 49 liefert folgendes Resultat.

Korollar 50. *Sei $L \subseteq \{0,1\}^*$ eine beliebige Sprache in P. Dann existiert eine Funktion $f \in \mathsf{FL}$, die bei Eingabe 1^n einen Schaltkreis c_n mit n Eingängen berechnet, so dass für alle $x \in \{0,1\}^n$ gilt:*

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

Korollar 50 besagt insbesondere, dass es für jede Sprache $L \subseteq \{0,1\}^*$ in P eine Schaltkreisfamilie $(c_n)_{n\geq 0}$ polynomieller Größe gibt, so dass

c_n für alle Eingaben $x \in \{0,1\}^n$ die charakteristische Funktion von L berechnet.

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

Definition 51.

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

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

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

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

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

Korollar 52 (Savage 1972)**.**
Es gilt $\mathsf{P} \subseteq \mathsf{PSK}$.

Ob auch alle NP-Sprachen polynomielle Schaltkreiskomplexität haben, ist ein berühmtes offenes Problem. Gelingt es nämlich, für ein

NP-Problem superpolynomielle untere Schranken für die Schaltkreisgröße zu zeigen, so folgt mit dem Resultat von Savage $\mathsf{P} \neq \mathsf{NP}$. Selbst für NEXP ist die Inklusion in PSK offen. Dagegen zeigt ein einfaches Diagonalisierungsargument, dass in EXPSPACE Sprachen mit superpolynomieller Schaltkreiskomplexität existieren. Wir werden später sehen, dass bereits die Annahme $\mathsf{NP} \subseteq \mathsf{PSK}$ schwerwiegende Konsequenzen für uniforme Komplexitätsklassen hat.

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

Definition 53. *Eine Sprache T heißt **tally** (kurz: $T \in \mathsf{TALLY}$), falls jedes Wort $x \in T$ die Form $x = 1^n$ hat.*

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

Proposition 54. $\mathsf{TALLY} \subseteq \mathsf{PSK}$.

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

Korollar 55. $\mathsf{PSK} \not\subseteq \mathsf{RE}$.

5.3 NP-vollständige Probleme

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

Definition 56. *Sei $B \subseteq \Sigma^*$ eine Sprache und sei p ein Polynom.*

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

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

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

c) Für eine Sprachklasse C sei

$$\exists \mathsf{C} = \{\exists B \mid B \in \mathsf{C} \text{ ist polynomiell balanciert}\}.$$

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

Satz 57 (Zeugen-Charakterisierung von NP)**.**
$\mathsf{NP} = \exists \mathsf{P}$.

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

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

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

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

Satz 58. 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 $L \in$ 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 57 haben wir gezeigt, dass für jede NP-Sprache $A \subseteq \Sigma^*$ eine p-balancierte Sprache $B \in$ 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 die Sprache

$$\{bin_m(x_1) \cdots bin_m(x_n) bin_m(\#) y \mid x_1 \cdots x_n \# y \in B\}$$

in P entscheidbar ist, existiert nach Korollar 50 eine FL-Funktion f, die für 1^n einen Schaltkreis c_n mit $m(n+1) + p(n)$ Eingängen berechnet, so dass für alle $x \in \Sigma^*$, $x = x_1 \cdots x_n$, und $y \in \{0,1\}^{p(n)}$ gilt:

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

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

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

■

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

Satz 61. 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.

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

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

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

Not-All-Equal-Satisfiability (NaeSat):

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

Satz 62. NaeSat $\in \mathsf{NPC}$.

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

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

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

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

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

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

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

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

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

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

Definition 63. *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 **Kantenüberdeckung**, 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?

Vertex Cover (VC): Besitzt G eine Kantenüberdeckung der Größe k?

Satz 64. 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 65. 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 66. VC *ist* NP*-vollständig.*

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

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

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

5.4 NL-vollständige Probleme

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

Satz 67. 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 4. *F ist genau dann erfüllbar, wenn für keinen Knoten x_i in G ein Pfad von x_i über $\bar{x}_i$ zurück nach x_i existiert.*

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

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

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

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

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

$$\Pr[\alpha] \;=\; \Pr[K_1 \to_M K_2 \to_M \cdots \to_M K_m] \;=\; \prod_{i=1}^{m-1} \Pr[K_i \to_M K_{i+1}]$$

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

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

wobei sich die Summation über alle akzeptierenden Rechnungen α von $M(x)$ erstreckt. Wir vereinbaren für PTMs, dass $M(x)$ am Ende jeder haltenden Rechnung entweder akzeptiert (hierfür schreiben wir kurz $M(x) = 1$) oder verwirft ($M(x) = 0$) oder „?“ ausgibt ($M(x) = ?$).

Definition 68.

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

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

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.*

Satz 69. $\mathsf{NP} \subseteq \mathsf{PP}$.

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

$$\begin{aligned} x \in L &\;\Rightarrow\; \Pr[N(x) = 1] \geq c^{-p(|x|)}, \\ x \notin L &\;\Rightarrow\; \Pr[N(x) = 1] = 0, \end{aligned}$$

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 $x \in \Sigma^*$,

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

und somit

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

■

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 Wahrscheinlichkeit $1/2$ akzeptiert) in eine äquivalente PPTM verwandeln lässt, die dies nicht tut.

Lemma 70. *Für jede Sprache $L \in \mathsf{PP}$ 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*

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

wobei $L(x)$ für alle $x \in L$ gleich 1 und für alle $x \notin L$ gleich 0 ist.

Beweis. Sei $L \in \mathsf{PP}$ 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 $\Pr[N(x) = 1]$ nur Werte der Form $i/k^{-p(|x|)}$ für $k = \mathrm{kgV}(2, \ldots, c)$ annehmen kann, folgt

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

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

$$\Pr[M(x) = 1] = \frac{\Pr[N(x) = 1] + \Pr[N'(x) = 1]}{2}$$

und somit

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

■

Eine direkte Folgerung von Lemma 70 ist der Komplementabschluss von PP.

Korollar 71. $\mathsf{PP} = \mathsf{co\text{-}PP}$.

Tatsächlich liefert Lemma 70 sogar den Abschluss von PP unter symmetrischer Differenz.

Satz 72. PP *ist unter symmetrischer Differenz abgeschlossen, d.h.*

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

Beweis. Nach obigem Lemma existieren PPTMs M_1 und M_2 mit

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

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

$$\Pr[M_1(x) = 1] \cdot \Pr[M_2(x) = 0] + \Pr[M_1(x) = 0] \cdot \Pr[M_2(x) = 1].$$

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

$$\begin{aligned} \Pr[M(x) = 1] &= (1/2 + \epsilon_1)(1/2 + \epsilon_2) + (1/2 - \epsilon_1)(1/2 - \epsilon_2) \\ &= (1/2 + 2\epsilon_1\epsilon_2) > 1/2 \end{aligned}$$

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

$$\begin{aligned} \Pr[M(x) = 1] &= (1/2 + \epsilon_1)(1/2 - \epsilon_2) + (1/2 - \epsilon_1)(1/2 + \epsilon_2) \\ &= (1/2 - 2\epsilon_1\epsilon_2) < 1/2. \end{aligned}$$

■

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?

6.1 Die Klassen BPP, RP und ZPP

Definition 73. *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:* $M(x) = 0$,
- 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 ? 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.

Satz 74. $\mathsf{ZPP} = \mathsf{RP} \cap \mathsf{co\text{-}RP}$.

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

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

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 $M_1(x) = 1$ und $\overline{M}_2(x) = 0$ 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

	$M_1(x) = 1$	$M_1(x) = 0$
$\overline{M}_2(x) = 1$	1	?
$\overline{M}_2(x) = 0$	–	0

verhält, folgende Äquivalenzen:

$$\begin{aligned} M(x) = 1 &\Leftrightarrow M_1(x) = 1, \\ M(x) = 0 &\Leftrightarrow \overline{M}_2(x) = 0. \end{aligned}$$

Nun ist leicht zu sehen, dass folgende Implikationen gelten:

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

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

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

■

6.2 Anzahl-Operatoren

Definition 75 (Anzahlklassen)**.** *Sei* C *eine Sprachklasse und sei $B \subseteq \Sigma^*$ eine p-balancierte Sprache in* 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):

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

Für $\mathsf{Op} \in \{\#, \exists, \forall, \exists^{\geq 1/2}, \oplus\}$ *sei*

$$\mathsf{Op} \cdot \mathsf{C} = \{\mathsf{Op}B \mid B \in \mathsf{C} \text{ ist polynomiell balanciert}\}$$

die durch Anwendung des Operators Op *auf* C *definierte Funktionen- bzw. Sprachklasse. Für* $\exists^{\geq 1/2} \cdot \mathsf{C}$ *schreiben wir auch* $\mathsf{P} \cdot \mathsf{C}$*. Zudem definieren wir die Operatoren* R *und* BP *durch*

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

und

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

Dabei heißt eine p-balancierte Sprache B ***einseitig****, falls* $\#B(x)$ *für keine Eingabe x einen Wert in* $[1, 2^{p(n)-1} - 1]$ *und* ***zweiseitig****, falls* $\#B(x)$ *für kein x einen Wert in* $(2^{p(n)}/3, 2^{p(n)+1}/3)$ *annimmt.*

Lemma 76. *Sei* C *eine Sprachklasse, die unter* $\leq_m^{log}$ *abgeschlossen ist. Dann gilt*

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

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

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

Beweis. Wir zeigen nur die Inklusionen von links nach rechts. Die umgekehrten Inklusionen folgen analog.

(i) Sei $A \in \mathsf{co}\text{-}\exists \cdot \mathcal{C}$. Dann existieren ein Polynom p und eine p-balancierte Sprache $B \in \mathsf{C}$ 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 $\mathsf{co}\text{-}\mathsf{C}$ 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 $A = \forall \hat{B} \in \forall \cdot \mathsf{co}\text{-}\mathcal{C}$.

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

(iii) Sei $A \in \oplus \cdot \mathcal{C}$ und sei $B \in \mathsf{C}$ eine p-balancierte Sprache mit $A = \oplus B$. Dann hat die in (i) definierte p-balancierte $\mathsf{co}\text{-}\mathsf{C}$-Sprache $\hat{B}$ die gleiche Parität wie $\#B(x)$ und daher gilt $A = \oplus \hat{B} \in \oplus \cdot \mathsf{co}\text{-}\mathcal{C}$. Zudem ist

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

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

Wir wissen bereits, dass $\exists \cdot \mathsf{P} = \mathsf{NP}$ und $\forall \cdot \mathsf{P} = \mathsf{co}\text{-}\mathsf{NP}$ ist. Was passiert, wenn wir diese Operatoren mehrmals anwenden?

Lemma 77. *Sei* $B \in \mathsf{C}$ *eine p-balancierte Sprache und sei* C *unter* $\leq_m^{log}$ *abgeschlossen. Dann existieren für jede Funktion* $f \in \mathsf{FL}$ *ein Polynom q und eine q-balancierte Sprache* $B' \in \mathsf{C}$ *mit*

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

Beweis. Sei q ein Polynom mit $p(|f(x)|) \leq q(|x|)$ für alle x 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 $\exists \cdot \mathsf{C}$, $\forall \cdot \mathsf{C}$, $\mathsf{R} \cdot \mathsf{C}$ und $\mathsf{BP} \cdot \mathsf{C}$ zu zeigen.

Satz 78. *Sei* C *eine unter* $\leq_m^{log}$ *abgeschlossene Sprachklasse. Dann gilt*

(i) *Für* $\mathsf{Op} \in \{\exists, \forall, \mathsf{R}, \mathsf{BP}, \mathsf{P}\}$ *ist* $\mathsf{Op} \cdot \mathsf{C}$ *unter* $\leq_m^{log}$ *abgeschlossen,*

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

Beweis.

(i) Sei $A \in \mathsf{Op} \cdot \mathsf{C}$ mittels einem Polynom p und einer p-balancierten Sprache $B \in \mathsf{C}$. Weiter gelte $L \leq A$ mittels einer FL-Funktion f. Nach obigem Lemma existieren ein Polynom q und eine q-balancierte Sprache $B' \in \mathsf{C}$ mit

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

Nun folgt z.B. für $\mathsf{Op} = \exists$:

$$x \in L \Leftrightarrow f(x) \in A \Leftrightarrow \#B(f(x)) > 0 \Leftrightarrow \#B'(x) > 0.$$

Dies zeigt, dass $L = \exists B' \in \exists \cdot \mathsf{C}$ ist. Die übrigen Fälle folgen analog.

(ii) Siehe Übungen.

■

6.3 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.

Satz 79. *Sei* q *ein beliebiges Polynom. Dann existiert zu jeder Sprache* $L \in \mathsf{RP}$ *eine* RPTM M *mit*

$$\begin{aligned} x \in L &\Rightarrow \Pr[M(x) = 1] \geq 1 - 2^{-q(|x|)}, \\ x \notin L &\Rightarrow \Pr[M(x) = 0] = 1. \end{aligned}$$

Beweis. Sei M eine RPTM für L. Dann gilt

$$\begin{aligned} x \in L &\Rightarrow \Pr[M(x) = 0] \leq 1/2, \\ x \notin L &\Rightarrow \Pr[M(x) = 0] = 1. \end{aligned}$$

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

$$\begin{aligned} x \in L &\Rightarrow \Pr[M(x) = 0] \leq 1/2 \Rightarrow \Pr[M'(x) = 0] \leq 2^{-q(|x|)}, \\ x \notin L &\Rightarrow \Pr[M(x) = 0] = 1 \Rightarrow \Pr[M'(x) = 0] = 1. \end{aligned}$$

■

Ganz analog lässt sich die Zuverlässigkeit einer ZPPTM verbessern.

Satz 80. *Für jedes Polynom* q *und jede Sprache* $L \in \mathsf{ZPP}$ *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 81. *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 $i > m/2$,

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

Wegen

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

erhalten wir somit

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

■

Satz 82. *Für jede Sprache $L \in \mathsf{BPP}$ und jedes Polynom q ex. eine BPTM M' mit*

$$\Pr[M'(x) = L(x)] \geq 1 - 2^{q(n)}.$$

Beweis. Sei M eine BPTM mit

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

Sei $t(n) = \lfloor (q(n) - 1)/\log_2(9/8) \rfloor$. Betrachte die PPTM M', die bei Eingabe x $2t(|x|) + 1$ Simulationen von $M(x)$ ausführt und ihre Eingabe genau dann akzeptiert, wenn M bei mindestens $t(|x|) + 1$ Simulationen akzeptiert. Dann folgt mit Lemma 81,

$$\Pr[M'(x) \neq L(x)] \leq 1/2 \underbrace{(1 - 4/36)}_{8/9}^{t(n)} \leq 2^{-q(|x|)}.$$

■

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

Lemma 83. *Sei M eine* PPTM. *Dann existieren eine Sprache $B \in \mathsf{P}$ und ein Polynom q, so dass für alle Eingaben x, $|x| = n$, gilt:*

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

Beweis. Sei p eine polynomielle Zeitschranke für M und sei $k = \mathrm{kgV}(2, 3, \ldots, c)$, wobei $c \in \mathbb{N}$ der maximale Verzweigungsgrad von M ist. Sei x eine Eingabe der Länge n. Wir ordnen jeder Folge $r = r_1 \cdots r_{p(n)}$ aus der Menge $R_n = \{0, \ldots, k - 1\}^{p(n)}$ eindeutig eine Rechnung von $M(x)$ zu, 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

$$\Pr[M(x) = 1] = \Pr_{r \in_R R_n}[M_r(x) = 1].$$

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 wir auch die Binärstrings $y \in D_n$ zur Kodierung der Folgen $r \in R_n$ verwenden und für $M_r(x)$ auch $M_y(x)$ schreiben. Betrachte nun die Sprache

$$B = \{x\#y \mid y \in D_n, M_y(x) = 1\}.$$

Definieren wir $\alpha(n) = \|D_n\|/2^{q(n)}$, so ist $\alpha(n) \in (1/2, 1]$, da D_n mehr als die Hälfte aller Strings der Länge $q(n)$ enthält. Zudem gilt

$$\begin{aligned}\Pr_{y \in_R \{0,1\}^{q(n)}}[x\#y \in B] &= \Pr[y \in D_n] \cdot \Pr[M_y(x) = 1 \mid y \in D_n] \\ &= \alpha(n)\Pr[M(x) = 1],\end{aligned}$$

womit das Lemma bewiesen ist. ■

Korollar 84. $\mathsf{R} \cdot \mathsf{P} = \mathsf{RP}$.

Für den Nachweis von $\mathsf{BP} \cdot \mathsf{P} = \mathsf{BPP}$ und $\mathsf{P} \cdot \mathsf{P} = \mathsf{PP}$ genügt eine einfache Modifikation des obigen Lemmas.

Lemma 85. *Sei M eine* PPTM *und sei $\beta = \Pr[M(x) = 1] - 1/2$. Dann existieren eine Sprache $B' \in \mathsf{P}$ und ein Polynom q, so dass für alle Eingaben x, $|x| = n$, gilt:*

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

Beweis. Der Beweis verläuft vollkommen analog wie der Beweis von Lemma 83, nur dass wir jetzt anstelle von B folgende Sprache betrachten:

$$B' = B \cup \{x\#y \mid y \in \{0,1\}^{q(|x|)} - D_n \text{ endet mit } 0\}.$$

Da D_n eine gerade Anzahl von Strings enthält, enthält auch die Menge $\{0,1\}^{q(n)} - D_n$ eine gerade Anzahl, wovon genau die Hälfte in $B' - B$ enthalten ist. Daher gilt für ein zufällig aus $\{0,1\}^{q(n)}$ gewähltes y:

$$\begin{aligned}\Pr[x\#y \in B'] &= \Pr[x\#y \in B] + \Pr[y \in \{0,1\}^{q(n)} - D_n] \cdot 1/2 \\ &= \alpha(n)\Pr[M(x) = 1] + (1 - \alpha(n))\,1/2 \\ &= \alpha(n)\,\underbrace{(\Pr[M(x) = 1] - 1/2)}_{\beta} + 1/2,\end{aligned}$$

womit das Lemma bewiesen ist. ■

Korollar 86. $\mathsf{BP} \cdot \mathsf{P} = \mathsf{BPP}$ *und* $\mathsf{P} \cdot \mathsf{P} = \mathsf{PP}$.

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

Definition 87. *A ist auf B* ***disjunktiv reduzierbar*** *(in Zeichen: $A \leq_d B$), falls eine Funktion $f \in \mathsf{FL}$ existiert, die für jedes Wort x eine Liste $y_1\# \cdots \#y_m$ von Wörtern y_i berechnet 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 $A \leq_{maj} B$ schreiben.*

Es ist leicht zu sehen, dass die Klassen P, NP und $\mathsf{co\text{-}NP}$ unter beiden Typen von Reduktionen abgeschlossen sind. Falls C unter disjunktiven Reduktionen abgeschlossen ist, lässt sich Satz 79 leicht von RP auf $\mathsf{R} \cdot \mathsf{C}$ verallgemeinern.

Satz 88. *Falls C unter disjunktiven Reduktionen abgeschlossen ist, existieren für jede Sprache $L \in \mathsf{R} \cdot \mathsf{C}$ und jedes Polynom q eine Sprache*

$B \in \mathsf{C}$ und ein Polynom r, so dass für alle x, $|x| = n$, und für ein zufällig gewähltes $z \in_R \{0,1\}^{r(n)}$ gilt

$$x \in L \Rightarrow \Pr[B(x\#z) = 1] \geq 1 - 2^{-q(|x|)},$$
$$x \notin L \Rightarrow \Pr[B(x\#z) = 0] = 1.$$

Beweis. Sei $L \in \mathsf{R} \cdot \mathsf{C}$ und sei $A \in \mathsf{C}$ eine p-balancierte Sprache mit

$$x \in L \Rightarrow \Pr_{y \in_R \{0,1\}^{p(n)}}[A(x\#y) = 1] \geq 1/2,$$
$$x \notin L \Rightarrow \Pr_{y \in_R \{0,1\}^{p(n)}}[A(x\#y) = 0] = 1.$$

Dann ist die Sprache

$$B = \{x\#y_1 \cdots y_{q(n)} \mid y_1, \ldots, y_{q(n)} \in \{0,1\}^{p(n)}, \exists i : x\#y_i \in A\}$$

disjunktiv auf A reduzierbar und somit in C. Außerdem ist leicht zu sehen, dass B für $r(n) = p(n)q(n)$ die im Satz genannten Eigenschaften besitzt. ■

Satz 89. *Falls C unter majority-Reduktionen abgeschlossen ist, existieren für jede Sprache $L \in \mathsf{BP} \cdot \mathsf{C}$ und jedes Polynom q eine Sprache $B \in \mathsf{C}$ und ein Polynom r, so dass für alle x, $|x| = n$, und für ein zufällig gewähltes $z \in_R \{0,1\}^{r(n)}$ gilt*

$$\Pr[L(x) = B(x\#z)] \geq 1 - 2^{-q(n)}.$$

Beweis. Sei $L \in \mathsf{BP} \cdot \mathsf{C}$ und sei $A \in \mathsf{C}$ eine p-balancierte Sprache mit

$$\Pr_{y \in_R \{0,1\}^{p(n)}}[L(x) \neq A(x\#y)] \leq 1/3 = 1/2 - 1/6.$$

Dann ist die Sprache

$$B = \left\{ x\#y_1 \cdots y_{2t(n)+1} \;\middle|\; \begin{array}{l} y_1, \ldots, y_{2t(n)+1} \in \{0,1\}^{p(n)}, \\ \|\{i : x\#y_i \in A\}\| > t(n) \end{array} \right\},$$

wobei $t(n) = (q(n) - 1)/\log_2(9/8)$ ist, auf A majority-reduzierbar und somit in C. Weiter folgt nach Lemma 81 für $r(n) = p(n)(2t(n) + 1)$ und für ein zufällig gewähltes $z = y_1 \cdots y_{2t(n)+1} \in_R \{0,1\}^{r(n)}$

$$\begin{aligned} \Pr[L(x) \neq B(x\#z)] &= \Pr[\|\{i : L(x) \neq A(x\#y_i)\}\| > t(n)] \\ &\leq 1/2(\underbrace{1 - 4/36}_{8/9})^{t(n)} \leq 2^{-q(|x|)}. \end{aligned}$$

■

Satz 90. $\mathsf{BPP} \subseteq \mathsf{PSK}$.

Beweis. Sei $L \in \mathsf{BPP} = \mathsf{BP} \cdot \mathsf{P}$. Nach vorigem Satz ex. ein Polynom r und eine r-balancierte Sprache $B \in \mathsf{C}$, so dass für alle x, $|x| = n$, und für ein zufällig gewähltes $z \in_R \{0,1\}^{r(n)}$

$$\Pr[L(x) \neq B(x\#z)] < 2^{-q(n)}$$

gilt. Daher folgt

$$\begin{aligned} &\Pr_{z \in_R \{0,1\}^{r(n)}}[\exists x \in \{0,1\}^n : B(x\#z) \neq L(x)] \\ &\leq \sum_{x \in \{0,1\}^n} \Pr_{z \in_R \{0,1\}^{r(n)}}[B(x\#z) \neq L(x)] \;<\; 1. \end{aligned}$$

Also muss für jede Eingabelänge n eine Folge z_n existieren, so dass für alle Eingaben der Länge n $L(x) = B(x\#z_n)$ ist. Da $B \in \mathsf{P}$ ist, kann die Funktion $x \mapsto B(x\#z_n)$ nach Korollar 50 durch einen Schaltkreis polynomieller Größe berechnet werden. ■

7 Die Polynomialzeithierachie

7.1 Die Polynomialzeithierachie

Definition 91. *Die **Polynomialzeithierachie** besteht aus den Stufen Σ_k^p und Π_k^p, $k \geq 0$, welche induktiv wie folgt definiert sind:*

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

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

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

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.

Satz 92. *Für alle $k \geq 0$ gilt: $\Sigma_k^p = \Sigma_{k+1}^p \Leftrightarrow \Sigma_k^p = \Pi_k^p \Leftrightarrow \mathsf{PH} = \Sigma_k^p$.*

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 $\mathsf{PH} = \Sigma_k^p \Rightarrow \Sigma_k^p = \Sigma_{k+1}^p$ 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.

Satz 93 (Lautemann 1983, Sipser 1983)**.**
$\mathsf{BPP} \subseteq \Sigma_2^p$.

Beweis. Sei $A \in \mathsf{BPP}$. Dann existieren ein Polynom p und eine p-balancierte Sprache $B \in \mathsf{P}$, so dass für alle x der Länge $n \geq 0$ gilt:

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

Setzen wir $B_x = \{y \in \{0,1\}^{p(n)} \mid x\#y \in B\}$, so enthält B_x für $x \in A$ mehr als $(1 - 2^{-n})2^{p(n)}$ Wörter und für $x \notin A$ weniger als $2^{p(n)-n}$ Wörter. Sei $\oplus$ die bitweise XOR-Operation auf $\{0,1\}^n$, d.h.

$$x_1 \cdots x_n \oplus y_1 \cdots y_n = (x_1 \oplus y_1) \cdots (x_n \oplus z_n).$$

Für $u \in \{0,1\}^{p(n)}$ und $U, V \subseteq \{0,1\}^{p(n)}$ sei

$$V \oplus u = \{v \oplus u \mid v \in V\} \text{ und } U \oplus V = \{u \oplus v \mid u \in U, v \in V\}.$$

Wir zeigen für alle Eingabelängen $n \geq 1$ mit $2^n \geq p(n)$ die Äquivalenz

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

Dies beweist $A \in \Sigma_2^p$, da die Sprache

$$B' = \{x\#u_1 \cdots u_{p(n)}\#v \mid v \in B_x \oplus \{u_1, \ldots, u_{p(n)}\}\}$$

in P entscheidbar ist. Sei also $x \in A$ und sei v ein beliebiger String der Länge $p(n)$. Da B_x mehr als $(1-2^{-n})2^{p(n)}$ Wörter enthält, ist v für ein zufällig gewähltes u mit Wahrscheinlichkeit

$$\Pr_{u \in_R \{0,1\}^{p(n)}}[v \notin B_x \oplus u] = \Pr_{u \in_R \{0,1\}^{p(n)}}[v \oplus u \notin B_x] < 2^{-n}$$

nicht in $B_x \oplus u$ und daher mit Wahrscheinlichkeit

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

nicht in $B_x \oplus \{u_1, \ldots, u_{p(n)}\}$ enthalten. Daher ist

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

d.h. für alle $x \in A$ mit $|x| \geq 1$ müssen Strings $u_1, \ldots, u_{p(n)}$ mit $B_x \oplus \{u_1, \ldots, u_{p(n)}\} = \{0,1\}^{p(n)}$ existieren. Dies zeigt die Implikation von links nach rechts in (7.1).

Für den Nachweis der umgekehrten Implikation nehmen wir an, dass für ein x der Länge n Wörter $u_1, \ldots, u_{p(n)}$ existieren mit $B_x \oplus \{u_1, \ldots, u_{p(n)}\} = \{0,1\}^{p(n)}$. Da die Mengen $B_x \oplus u_i$ gleich groß wie B_x sind, muss dann im Fall $2^n \geq p(n)$

$$\|B_x\| \geq 2^{p(n)}/p(n) \geq 2^{p(n)-n}$$

sein und somit x zu A gehören. ■

Da BPP unter Komplement abgeschlossen ist, folgt auch $\mathsf{BPP} \subseteq \Pi_2^p$. Starten wir in obigem Beweis mit einer Sprache A in $\mathsf{BP} \cdot \mathsf{C}$ für eine Klasse C, die unter majority-Reduktionen abgeschlossen ist, so folgt $A \in \exists \cdot \forall \cdot \mathsf{C}$, da wegen $B \in \mathsf{C}$ auch die Sprache

$$B' = \{x\#u_1\#\cdots\#u_{p(n)}\#v \mid v \in B_x \oplus \{u_1, \ldots, u_{p(n)}\}\}$$

in C entscheidbar ist. Insbesondere erhalten wir für $\mathsf{C} = \mathsf{co\text{-}NP}$ die Inklusion $\mathsf{BP} \cdot \mathsf{co\text{-}NP} \subseteq \exists \cdot \forall \cdot \mathsf{co\text{-}NP} = \Sigma_2^p$.

Korollar 94. $\mathsf{BP} \cdot \mathsf{co\text{-}NP} \subseteq \Sigma_2^p$ *bzw.* $\mathsf{BP} \cdot \mathsf{NP} \subseteq \Pi_2^p$.

Der Beweis von Satz 93 lässt sich leicht zu $\exists \cdot \forall \cdot \mathsf{BPP} = \Sigma_2^p$ verschärfen. Folglich ist NP nicht in BPP enthalten, außer wenn PH auf die zweite Stufe kollabiert.

Zum Abschluss dieses Kapitels zeigen wir, dass NP nicht einmal in $\mathsf{BP} \cdot \mathsf{co\text{-}NP}$ enthalten ist, außer wenn $\mathsf{PH} = \mathsf{BP} \cdot \mathsf{NP} = \Sigma_2^p$ ist. Hierfür benötigen wir noch gewisse Abschlusseigenschaften der Klasse $\mathsf{BP} \cdot \mathsf{NP}$.

Wir zeigen zuerst, dass $\mathsf{BP} \cdot \mathsf{C}$ unter dem BP-Operator abgeschlossen ist, falls C unter majority-Reduktionen abgeschlossen ist.

Lemma 95. *Für jede Klasse* C*, die unter majority-Reduktionen abgeschlossen ist, gilt*

$$\mathsf{BP} \cdot \mathsf{BP} \cdot \mathsf{C} = \mathsf{BP} \cdot \mathsf{C}.$$

Beweis. Sei $L \in \mathsf{BP} \cdot \mathsf{BP} \cdot \mathsf{C}$. Da mit C auch $\mathsf{BP} \cdot \mathsf{C}$ unter majority-Reduktionen abgeschlossen ist, existieren eine Sprache $A \in \mathsf{BP} \cdot \mathsf{C}$ 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 $B \in \mathsf{C}$ 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 $B' = \{x\#yz \mid |y| = p(|x|), |z| = q(|x| + p(|x|) + 1), x\#y\#z \in B\} \in \mathsf{C}$,

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

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

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

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

$$\mathsf{R} \cdot \mathsf{R} \cdot \mathsf{C} = \mathsf{R} \cdot \mathsf{C}.$$

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

Lemma 96. *Sei* C *eine unter* $\leq_{maj}$ *abgeschlossene Sprachklasse. Dann gilt*

$$\exists \cdot \mathsf{BP} \cdot \mathsf{C} \subseteq \mathsf{BP} \cdot \exists \cdot \mathsf{C}.$$

Beweis. Sei $L \in \exists \cdot \mathsf{BP} \cdot \mathsf{C}$. Dann existieren eine Sprache $A \in \mathsf{BP} \cdot \mathsf{C}$ 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 $B \in \mathsf{C}$ mit

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

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

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

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

Satz 97. $\mathsf{NP} \subseteq \mathsf{BP} \cdot \mathsf{co\text{-}NP} \Rightarrow \mathsf{PH} = \mathsf{BP} \cdot \mathsf{NP}$.

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

$$\begin{aligned}
\Sigma_{k+1}^p &= \exists \cdot \Pi_k^p \\
&\subseteq \exists \cdot \mathsf{BP} \cdot \mathsf{NP} \\
&\subseteq \mathsf{BP} \cdot \exists \cdot \mathsf{NP} \\
&= \mathsf{BP} \cdot \mathsf{NP} \\
&\subseteq \mathsf{BP} \cdot \mathsf{BP} \cdot \mathsf{co\text{-}NP} \\
&= \mathsf{BP} \cdot \mathsf{co\text{-}NP}.
\end{aligned}$$

■

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 98. *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. GI $\in$ P) klassifiziert werden. Auch die Zugehörigkeit von GI zu NP $\cap$ co-NP ist offen.

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

Definition 99. *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

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

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

Lemma 100. *Für jeden Graphen $G = (V, E)$ mit $V = \{1, \ldots, n\}$ gilt*

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

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

Beweis.

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

$$\varphi(G) = \pi(G) \Leftrightarrow \varphi^{-1} \circ \pi \in Aut(G) \Leftrightarrow \pi \in \varphi \circ Aut(G).$$

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

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

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

(ii) Aus *(i)* folgt sofort

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

■

8.2 GI liegt in co-BPNP

Als nächstes wollen wir zeigen, dass GI fast in co-NP liegt (genauer: $GI \in BP \cdot co\text{-}NP$ bzw. $\overline{GI} \in BP \cdot NP$). Nach Satz 97 ist GI daher nicht NP-vollständig, außer wenn $PH = BP \cdot NP$ ist. Wir betrachten zunächst folgende Verallgemeinerung des BP-Operators.

Definition 101. *Sei $\mathcal{C}$ eine Sprachklasse. Eine Sprache $L \subseteq \Sigma^*$ gehört zu $\widetilde{BP} \cdot \mathcal{C}$, falls Funktionen $g > 0$ in FP 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 $\widetilde{BP}$-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 FP verwendet werden. Es ist leicht zu sehen, dass NP in der Klasse $\widetilde{BP} \cdot P$ enthalten ist (hierzu genügt es, für g die konstante Funktion $g(x) = 1/2$ zu wählen). Daher ist BPP vermutlich echt in $\widetilde{BP} \cdot P$ enthalten.

Satz 102. $\overline{GI} \in \widetilde{BP} \cdot NP$.

Beweis. Seien zwei Graphen $G_i = (V, E_i)$, $i = 1, 2$, mit $V = \{1, \ldots, n\}$ gegeben. Nach Lemma 100 haben die Mengen

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

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 FP berechenbar ist, folgt $\overline{GI} \in \widetilde{BP} \cdot NP$. ■

8.3 Lineare Hashfunktionen

Definition 103. *$Lin(m,k)$ bezeichne die Menge aller linearen Funktionen von $\{0,1\}^m$ nach $\{0,1\}^k$.*

Bemerkung 104. *Jede Funktion $h \in Lin(m,k)$ lässt sich eindeutig 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 105. *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 Wahrscheinlichkeit 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

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

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 wegen

$$E(S_y) = \Pr[S_y = 1] = \Pr[h(y) = 0^k] = 2^{-k},$$

dass $E(S) = \sum_{y \in B} E(S_y) = 2^{-k}\|B\|$ ist. Weiter folgt wegen $S_y^2 = S_y$,

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

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 die Varianz auf paarweise stochastisch unabhängigen Zufallsvariablen additiv ist (siehe Übungen), folgt

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

■

Satz 106. $\mathsf{BP} \cdot \mathsf{NP} = \widetilde{\mathsf{BP}} \cdot \mathsf{NP}$.

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 $\widetilde{\mathsf{BP}} \cdot \mathsf{NP}$. Dann existieren Funktionen $g(x) > 0$ in FP und $f(x)$ in $\#\mathsf{NP}$ 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 ein Polynom p und eine p-balancierte NP-Sprache A 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

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

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 Wahrscheinlichkeit $2/3$ und im Fall $x \notin L$ höchstens mit Wahrscheinlichkeit $1/3$ eintritt. Nach Lemma 105 gilt

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

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

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

■

Korollar 107. GI *ist nicht* NP*-vollständig, außer wenn* $\mathsf{PH} = \mathsf{BP} \cdot \mathsf{NP}$ *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 zu jeder Komplexitätsklasse C eine relativierte Version C^A, in der alle Probleme enthalten sind, die relativ zum Orakel A innerhalb der durch C vorgegebenen Ressourcen lösbar sind.

9.1 Orakel-Turingmaschinen

Definition 108. *Eine **deterministische Orakelmaschine (DOM)** M ist eine DTM, die mit einem speziellen write-only **Orakelfrageband** 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 geschieht innerhalb eines einzigen Rechenschrittes. Die unter einem Orakel A arbeitende DOM wird mit M^A bezeichnet und die von M^A **akzeptierte Sprache** ist $L(M^A)$.*

*Anstelle eines Sprachorakels A benutzen wir zuweilen auch ein funktionales Orakel f. In diesem Fall ist M mit einem speziellen **Orakelantwortband** ausgestattet, auf dem jede Orakelfrage y innerhalb eines Rechenschrittes mit $f(y)$ beantwortet wird.*

*Wir sagen, M ist **nichtadaptiv**, falls die Fragen von M nicht von den Antworten auf zuvor gestellte Fragen abhängen.*

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

Definition 109. *Die **Rechenzeit** einer DOM 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 DOM M ist analog definiert, wobei das write-only Orakelband unberücksichtigt bleibt. Eine polynomiell zeitbeschränkte DOM M bezeichnen wir kurz als PDOM. Alle unter einem Orakel A in Polynomialzeit akzeptierten Sprachen fassen wir in der Klasse

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

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

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

Für P^{C} (bzw. P^A) schreiben wir auch $\mathsf{P}(\mathsf{C})$ (bzw. $\mathsf{P}(A)$). Ebenso wie DTMs lassen sich auch NTMs und PTMs oder Transducer mit einem Orakelmechanismus ausstatten, wodurch wir NOMs und POMs oder Orakeltransducer erhalten. Ist die Rechenzeit dieser Maschinen polynomiell beschränkt, so bezeichnen wir sie als NPOMs bzw. PPOMs. Entsprechend erhalten wir dann die relativierten Komplexitätsklassen NP^A, PP^A, FP^A, BPP^A, RP^A, ZPP^A, L^A, NL^A, FL^A, PSPACE^A usw. Lassen wir nur nichtadaptive Orakelmaschinen zu, so notieren wir dies durch den Index $\|$ und schreiben $\mathsf{P}_\|^A$, $\mathsf{FP}_\|^A$ usw. Falls wir dagegen die Anzahl der Fragen durch eine Funktion $g(n)$ begrenzen, wobei n die Eingabelänge bezeichnet, so schreiben wir $\mathsf{P}^{A[g(n)]}$ usw.

Satz 110.

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

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

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

Beweis. *(i)* Die Inklusion $\mathsf{P} \subseteq \mathsf{P}(\mathsf{P})$ ist klar. Für die umgekehrte Richtung sei L eine Sprache in $\mathsf{P}(\mathsf{P})$. Dann existiert eine PDOM M und ein Orakel $A \in \mathsf{P}$ mit $L(M^A) = L$. Sei M' eine PDTM mit $L(M') = A$. Betrachte die DOM $M''(x)$, 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, $\mathsf{P}^{\mathsf{NP}\cap\text{co-}\mathsf{NP}} \subseteq \mathsf{NP} \cap \text{co-}\mathsf{NP}$ zu zeigen. Sei $L = L(M^A)$ für eine POM M und sei A ein Orakel in $\mathsf{NP} \cap \text{co-}\mathsf{NP}$. 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 $L \in \mathsf{NP}$. Da $\mathsf{P}^{\mathsf{NP}\cap\text{co-}\mathsf{NP}}$ unter Komplementbildung abgeschlossen ist, ist $\mathsf{P}^{\mathsf{NP}\cap\text{co-}\mathsf{NP}}$ auch in co-NP enthalten. Die Inklusion von $\mathsf{NP}^{\mathsf{NP}\cap\text{co-}\mathsf{NP}}$ in NP folgt vollkommen analog.

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

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

Dann ist $\bar{A} \in \mathsf{NP}$ und L wird von der NPOM 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 ein NP-Orakel A und eine NPOM M, deren Rechenzeit durch ein Polynom p und deren Verzweigungsgrad durch 2 beschränkt ist. Dann existieren ein Polynom q und eine Sprache $B \in \mathsf{P}$ 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 $(a_i = 1 \wedge y_i\#(z_i)_{\leq q(|y_i|)} \in B) \vee (a_i = 0 \wedge y_i \notin A)$,

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

Für die umgekehrte Richtung sei $L = L(M^A)$ für ein Orakel $A \in \Sigma_k^p = \exists \cdot \Pi_{k-1}^p$ und eine NPOM M, deren Rechenzeit durch ein Polynom p und deren Verzweigungsgrad durch 2 beschränkt ist. Dann existieren ein Polynom q und eine Sprache $B \in \Pi_{k-1}^p$ 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 einen Binärstring $r = r_1 \cdots r_{p(|x|)} \in \{0,1\}^{p(|x|)}$ kodieren, wobei r_i im Fall, dass α im i-ten Rechenschritt nichtdeterministisch verzweigt, die Richtung (links oder rechts), und im Fall, dass α im i-ten Rechenschritt eine Orakelfrage stellt, die Antwort (ja oder nein) angibt. Dann gilt

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

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 $(a_i = 1 \wedge y_i\#(z_i)_{\leq q(|y_i|)} \in B) \vee (a_i = 0 \wedge y_i \notin A)$,

wobei $(z)_{\leq k}$ das Präfix der Länge k von z bezeichnet. Da C in Π_k^p liegt, zeigt diese Charakterisierung, dass L in $\exists \cdot \Pi_k^p = \Sigma_{k+1}^p$ enthalten ist.

■

9.2 Das relativierte P/NP-Problem

Satz 111 (Baker, Gill und Solovay, 1975)**.** *Es gibt Orakel A und B mit*

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

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

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

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

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

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

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

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

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: $n_0 = 0$ und

$$n_{i+1} = \min\{n > (n_i)^i + 1 \mid n^{i+1} < 2^n\},\ i \geq 0.$$

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 $L(B) \notin \mathsf{P}(B)$. ■

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

$$\begin{aligned}\mathsf{DTIME}^A(f) \subseteq \mathsf{NTIME}^A(f) \subseteq \mathsf{DSPACE}^A(f) \subseteq \mathsf{NSPACE}^A(f) \\ \subseteq \mathsf{DTIME}^A(2^{O(f)}),\end{aligned}$$

die Zeit- und Platzhierarchiesätze wie

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

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

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

und der Satz von Immerman/Szelepczényi,

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

falls $s(n) \geq \log n$. Wie wir gerade gesehen haben, hängt dagegen die Antwort auf die Frage, ob $\mathsf{P}(A) = \mathsf{NP}(A)$ 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.

Satz 112. *Es existiert ein Orakel B, so dass $\Sigma_k^P(B) \neq \Sigma_{k+1}^P(B)$ für alle $k \geq 0$ (und somit $\mathsf{PH}(B) \neq \mathsf{PSPACE}(B)$) gilt.*

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

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

sogar mit Wahrscheinlichkeit 1 erhält, d.h. die Klassen P, NP und $\mathsf{co\text{-}NP}$ 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 $\mathsf{P}(A) \neq \mathsf{NP}(A)$ genau dann relativ zu einem Zufallsorakel mit Wahrscheinlichkeit 1 gilt, wenn sie unrelativiert gilt. Diese Hypothese wurde mehrfach widerlegt. Bekanntestes Beispiel ist die Gleichheit $\mathrm{IP} = \mathsf{PSPACE}$, obwohl $\mathrm{IP}^A \neq \mathsf{PSPACE}^A$ mit Wahrscheinlichkeit 1 gilt.

10 PP und die Polynomialzeithierarchie

In diesem Kapitel zeigen wir, dass PH in der Klasse $\mathsf{BP} \cdot \oplus\mathsf{P}$ enthalten ist. Da diese Klasse im Turing-Abschluss $\mathsf{P}(\mathsf{PP})$ von PP enthalten ist, folgt $\mathsf{PH} \subseteq \mathsf{P}(\mathsf{PP})$.

Definition 113. *Für eine NTM M bezeichne $\#M(x)$ die Anzahl der akzeptierenden Rechnungen von $M(x)$.*

a) $L \subseteq \Sigma^*$ gehört zu $\oplus\mathsf{P}$, falls eine NPTM M existiert mit

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

b) $L \subseteq \Sigma^*$ gehört zu UP, falls eine NPTM M existiert mit $\#M(x) = \chi_L(x)$, d.h.

$$\begin{aligned} x \in L &\Rightarrow \#M(x) = 1, \\ x \notin L &\Rightarrow \#M(x) = 0. \end{aligned}$$

Unter Verwendung von NPOMs erhalten wir die relativierten Klassen $\oplus\mathsf{P}^A$ und UP^A. Es ist nicht schwer zu sehen, dass folgendes Entscheidungsproblem $\oplus$SAT $\oplus\mathsf{P}$-vollständig ist (siehe Übungen).

Gegeben: Eine boolsche Formel $F(x_1, \ldots, x_n)$.
Gefragt: Ist die Anzahl der erfüllenden Belegungen von F ungerade?

Das nachfolgende Lemma wird ebenfalls in den Übungen bewiesen.

Lemma 114.

i) $\oplus\mathsf{P} = \oplus \cdot \mathsf{P}$,

ii) $\mathsf{P}^{\oplus\mathsf{P}} = \oplus\mathsf{P}^{\oplus\mathsf{P}} = \oplus \cdot \oplus\mathsf{P} = \oplus\mathsf{P}$.

10.1 Satz von Valiant und Vazirani

Bei manchen Anwendungen ist der Bereich der tatsächlich vorkommenden Probleminstanzen eingeschränkt. Daher spielt es keine Rolle, wenn der Algorithmus außerhalb dieses Bereichs inkorrekt arbeitet.

Definition 115. *Ein **Promise-Problem** ist ein Paar (A, B) von Sprachen, wobei A das **Promise-Prädikat** genannt wird. Eine Sprache L heißt **Lösung** für (A, B), falls für alle Eingaben $x \in A$ gilt:*

$$x \in L \Leftrightarrow x \in B.$$

L ist also genau dann eine Lösung für (A, B), wenn $A \cap B \subseteq L \subseteq (A \cap B) \cup \bar{A}$ gilt.

Beispiel 116. *Sei* 1SAT *die Menge aller booleschen Formeln, die höchstens eine erfüllende Belegung haben. Um das Promise-Problem* $(1\text{SAT}, \text{SAT})$ *zu lösen, genügt es, für alle Formeln $F \in$ 1SAT herauszufinden, ob sie erfüllbar sind oder nicht. Somit ist jede Sprache L mit* $\text{USAT} \subseteq L \subseteq \text{SAT}$ *eine Lösung für* $(1\text{SAT}, \text{SAT})$*, wobei die Sprache* $\text{USAT} = 1\text{SAT} \cap \text{SAT}$ *alle Formeln enthält, die genau eine erfüllende Belegung haben. Neben* USAT *und* SAT *ist beispielsweise auch* $\oplus$SAT *eine Lösung für* $(1\text{SAT}, \text{SAT})$. ◁

Als nächstes wollen wir zeigen, dass SAT (und damit jedes NP Problem) auf eine beliebige Lösung von $(1\text{SAT}, \text{SAT})$ randomisiert reduzierbar ist.

Definition 117. *Eine Sprache A heißt **randomisiert reduzierbar** auf eine Sprache B, falls es eine Funktion $f \in \mathsf{FP}$ und Polynome p, q gibt, so dass für alle Eingaben x gilt:*

$$\begin{aligned} x \in A &\Rightarrow \Pr_{y \in_R \{0,1\}^{p(n)}}[f(x\#y) \in B] \geq 1/q(n), \\ x \notin A &\Rightarrow \Pr_{y \in_R \{0,1\}^{p(n)}}[f(x\#y) \in B] = 0. \end{aligned}$$

Lemma 118. *Sei $\emptyset \neq B \subseteq \{0,1\}^m - \{0^m\}$ und sei $k = \lfloor \log_2(3\|B\|) \rfloor$. Weiter sei $S = \|\{y \in B \mid h(y) = 0^k\}\|$ die Anzahl der Strings in B, die durch eine zufällig gewählte Funktion $h \in_R Lin(m,k)$ auf 0^k abgebildet werden. Dann ist*

$$\Pr[S = 1] \geq 2/9.$$

Beweis. Setzen wir $b = \|B\|$, so gilt

$$\Pr[S \geq 2] \leq \sum_{\{y,y'\} \in \binom{B}{2}} \underbrace{\Pr[h(y) = h(y') = 0^k]}_{2^{-2k}} = \binom{b}{2} \cdot 2^{-2k}$$

und

$$\begin{aligned} \Pr[S \geq 1] &\geq \sum_{y \in B} \underbrace{\Pr[h(y) = 0^k]}_{2^{-k}} - \sum_{\{y,y'\} \in \binom{B}{2}} \Pr[h(y) = h(y') = 0^k] \\ &= 2^{-k} b - \binom{b}{2} 2^{-2k}. \end{aligned}$$

Somit folgt

$$\begin{aligned} \Pr[S = 1] = \Pr[S \geq 1] - \Pr[S \geq 2] &\geq 2^{-k} b - 2\binom{b}{2} 2^{-2k} \\ &= 2^{-k} b (1 - 2^{-k}(b-1)) > 2^{-k} b (1 - 2^{-k} b). \end{aligned}$$

Da $k = \lfloor \log_2(3b) \rfloor$ im Intervall $(\log_2(3b) - 1, \log_2(3b)]$ liegt, nimmt $x = 2^{-k} b$ nur Werte im Intervall $[1/3, \frac{2}{3})$ an. Da aber die Funktion $f(x) = x(1-x)$ auf $[1/3, \frac{2}{3}]$ nach unten durch $2/9$ beschränkt ist, folgt $\Pr[S = 1] \geq 2/9$. ∎

Satz 119 (Valiant, Vazirani 1986)**.** SAT *ist auf jede Lösung L von* (1SAT, SAT) *randomisiert reduzierbar.*

Beweis. Sei F eine boolesche Formel über n Variablen $x_1, \ldots, x_n$, wobei wir o.B.d.A. annehmen, dass $F(0^n) = 0$ ist. Für lineare Hashfunktionen $h_j = h_{j1} \cdots h_{jn} \in Lin(n,1) = \{0,1\}^n$, $0 \leq j \leq n$, und eine Zahl $i \in \{0, \ldots, n\}$ sei $F_{i,h_0,\ldots,h_n}(x_1, \ldots, x_n)$ die boolesche Formel

$$F \wedge \bigwedge_{j=0}^{i} (h_j(x_1 \cdots x_n) = 0) = F \wedge \bigwedge_{j=0}^{i} \left(\neg \bigoplus_{k=1}^{n} (h_{jk} \wedge x_k) \right),$$

die unter einer Belegung x genau dann wahr wird, wenn $F(x) = 1$ ist und die $i+1$ Hashfunktionen $h_0, \cdots, h_i$ für x den Wert 0 liefern. Betrachte die Reduktionsfunktion

$$f : F \# h_0 \cdots h_n u \mapsto F' = F_{(u)_2, h_0, \ldots, h_n},$$

wobei u ein Binärstring der Länge $l(n) = \lceil \log_2(n+1) \rceil$ ist und $(u)_2$ den Wert von u (aufgefasst als Binärzahl) bezeichnet. Insgesamt hat der Binärstring $h_0 \cdots h_n u$ also die Länge $l'(n) = n(n+1) + l(n)$.

Sei nun $b = \|\{x \in \{0,1\}^n \mid F(x) = 1\}\|$ die Anzahl der erfüllenden Belegungen von F. Im Fall $x \in$ SAT ist dann $b \in \{1, \cdots, 2^n - 1\}$, d.h. $b' := \lfloor \log_2(3b) \rfloor$ ist in $\{1, \cdots, n+1\}$, und für einen zufällig aus $\{0,1\}^{l'(n)}$ gewählten String $h_0 \cdots h_n u$ gilt

$$\begin{aligned} \Pr[F' \in L] &\geq \underbrace{\Pr[(u)_2 + 1 = b']}_{= 2^{-l(n)} \geq 1/2(n+1)} \underbrace{\Pr[F' \in L \mid (u)_2 + 1 = b']}_{\geq 2/9} \\ &\geq 1/9(n+1). \end{aligned}$$

Dagegen ist diese Wk im Fall $x \notin$ SAT gleich 0. ∎

Korollar 120.

(i) Falls das Promise-Problem (1SAT, SAT) *eine Lösung in* BPP *hat, folgt* NP = RP *und* PH = BPP = Σ_2^p *(siehe Übungen).*

(ii) NP $\subseteq$ R $\cdot \oplus$P $\subseteq$ BP $\cdot \oplus$P.

(iii) Für jede Sprachklasse C*, die unter disjunktiven Reduktionen abgeschlossen ist, gilt* $\exists \cdot$ C $\subseteq$ R $\cdot \oplus \cdot$ C $\subseteq$ BP $\cdot \oplus \cdot$ C.

10.2 Satz von Toda

In diesem Abschnitt beweisen wir den Satz von Toda, der besagt, dass PH in P^{PP} enthalten ist. Hierzu zeigen wir die Inklusionen $\mathsf{PH} \subseteq \mathsf{BP} \cdot \oplus\mathsf{P}$ und $\mathsf{P} \cdot \oplus\mathsf{P} \subseteq \mathsf{P}^{\mathsf{PP}}$. Um die Inklusion $\mathsf{NP} \subseteq \mathsf{BP} \cdot \oplus\mathsf{P}$ auf $\mathsf{PH} \subseteq \mathsf{BP} \cdot \oplus\mathsf{P}$ hochziehen zu können, benötigen wir den Abschluss der Klasse $\mathsf{BP} \cdot \oplus\mathsf{P}$ unter dem BP- (siehe Lemma 95) und dem $\oplus$-Operator.

Lemma 121. *Es gilt*

$$\oplus \cdot \mathsf{BP} \cdot \oplus\mathsf{P} = \mathsf{BP} \cdot \oplus\mathsf{P}.$$

Beweis. Sei $L \in \oplus \cdot \mathsf{BP} \cdot \oplus\mathsf{P} \subseteq \oplus\mathsf{P}^{\mathsf{BP} \cdot \oplus\mathsf{P}}$. Dann existiert ein Orakel $A \in \mathsf{BP} \cdot \oplus\mathsf{P}$ und eine durch ein Polynom p zeitbeschränkte NPTOM M mit

$$x \in L \Leftrightarrow \#M^A(x) \text{ ist ungerade.}$$

O.B.d.A. sei $A \subseteq \{0,1\}^*$. Wegen $A \in \mathsf{BP} \cdot \oplus\mathsf{P}$ existieren ein Polynom q und eine q-bal. Sprache $B \in \oplus\mathsf{P}$, so dass für alle y gilt:

$$\Pr_{z \in_R \{0,1\}^{q(|y|)}}[y \in A \Leftrightarrow y\#z \in B] \geq 1 - 2^{-2|y|-3}.$$

Betrachte die $\oplus\mathsf{P}$-Sprache

$$B' = \{y\#z \mid |z| \geq q(|y|), y\#(z)_{\leq q(|y|)} \in B\},$$

wobei $(z)_{\leq m}$ das Präfix der Länge m von z bezeichnet. Dann gilt für alle $m \geq 1$ und für alle $k \geq q(m)$

$$\begin{aligned}
&\Pr_{z \in_R \{0,1\}^k}[\forall y, |y| \leq m : y \in A \Leftrightarrow y\#z \in B'] \\
&= 1 - \Pr_{z \in_R \{0,1\}^k}[\exists y, |y| \leq m : y \notin A \Leftrightarrow y\#z \in B'] \\
&\geq 1 - (\sum_{i=0}^{m} \sum_{y, |y|=i} 2^{-2|y|-3}) \\
&\geq 1 - (\sum_{i=0}^{m} 2^{i-2i-3}) = 1 - (2^{-3} \cdot \sum_{i=0}^{m} 2^{-i}) > 1 - 2^{-2} = 3/4.
\end{aligned}$$

Sei $\hat{M}$ eine NPTOM, die sich bei Eingabe $x\#z$ genau so verhält wie $M(x)$, nur dass sie jede Orakelfrage y von M durch die Orakelfrage $y\#z$ ersetzt. Betrachte die Sprache

$$C = \{x\#z \mid |z| = q(p(|x|)),\ \#\hat{M}^{B'}(x\#z) \text{ ist ungerade}\}.$$

Dann ist $C \in \oplus\mathsf{P}^{\oplus\mathsf{P}} = \oplus\mathsf{P}$ und es folgt

$$\Pr_{z \in_R \{0,1\}^{q(p(|x|))}}[x \in L \Leftrightarrow x\#z \in C] > \frac{3}{4}.$$

Dies zeigt, dass $L \in \mathsf{BP} \cdot \oplus\mathsf{P}$ ist. ∎

Satz 122. $\mathsf{PH} \subseteq \mathsf{BP} \cdot \oplus\mathsf{P}$

Beweis. Wir zeigen induktiv über k, dass $\Sigma_k^p \cup \Pi_k^p \subseteq \mathsf{BP} \cdot \oplus\mathsf{P}$ gilt.

$k = 1$**:** klar, da $\mathsf{NP} \subseteq \mathsf{BP} \cdot \oplus\mathsf{P}$ gilt und $\mathsf{BP} \cdot \oplus\mathsf{P}$ unter Komplement abgeschlossen ist.

$k \rightsquigarrow k+1$**:** Es gilt

$$\begin{aligned}
\Sigma_{k+1}^p &= \exists \cdot \Pi_k^p \overset{\text{(IV)}}{\subseteq} \exists \cdot \mathsf{BP} \cdot \oplus\mathsf{P} \subseteq \mathsf{BP} \cdot \oplus \cdot \mathsf{BP} \cdot \oplus\mathsf{P} \\
&\subseteq \mathsf{BP} \cdot \mathsf{BP} \cdot \oplus\mathsf{P} \subseteq \mathsf{BP} \cdot \oplus\mathsf{P}.
\end{aligned}$$

∎

Zum Beweis der Inklusion $\mathsf{P} \cdot \oplus\mathsf{P} \subseteq \mathsf{P}^{\mathsf{PP}}$ benötigen wir das Konzept der Modul-verstärkenden Polynome.

Lemma 123. *Sei $Q_d(n) = ((n+1)^d + 1)^d$. Dann gilt für alle n,*

$$\begin{aligned}
n \equiv_2 0 &\Rightarrow Q_d(n) \equiv_{2^d} 0, \\
n \equiv_2 1 &\Rightarrow Q_d(n) \equiv_{2^d} 1.
\end{aligned}$$

Beweis. Für gerades n gilt

$$\begin{aligned} n+1 \equiv_2 1 &\Rightarrow (n+1)^d \equiv_2 1 \\ &\Rightarrow (n+1)^d + 1 \equiv_2 0 \\ &\Rightarrow ((n+1)^d+1)^d \equiv_{2^d} 0 \end{aligned}$$

and für ungerades n ist

$$\begin{aligned} n+1 \equiv_2 0 &\Rightarrow (n+1)^d \equiv_{2^d} 0 \\ &\Rightarrow (n+1)^d + 1 \equiv_{2^d} 1 \\ &\Rightarrow ((n+1)^d+1)^d \equiv_{2^d} 1. \end{aligned}$$

■

Weiterhin benötigen wir eine Reihe von grundlegenden Abschlusseigenschaften der Funktionenklasse #P.

Lemma 124. *Seien $f, g \in \#\mathsf{P}$, $t \in \mathsf{FP}$ und seien p ein Polynom und C eine p-balancierte Sprache in P. Dann sind folgende Funktionen in #P:*

(i) $f + g$,
(ii) $f \cdot g$,
(iii) $h : x \mapsto \sum_{y, x\#y \in C} f(x\#y)$,
(iv) $h' : x10^d \mapsto f(x)^d$,
(v) $f' : x \mapsto f(t(x))$.

Beweis. Zu f, g existieren Polynome q, r und q- bzw. r-balancierte Sprachen A und B mit $f(x) = \#A(x)$ und $g(x) = \#B(x)$. Dann ist $f(x) + g(x) = \#D(x)$ für die $(r+1)$-balancierte Sprache

$$D = \{x\#0y \mid x\#y \in A\} \cup \{x\#1^{q(|x|)-r(|x|)+1}y \mid x\#y \in B\},$$

wobei wir o.B.d.A. $q \geq r$ annehmen. Weiter ist $f(x)g(x) = \#E(x)$ für die $(q+r)$-balancierte Sprache

$$E = \{x\#yz \mid x\#y \in A \wedge x\#z \in B\}.$$

Um zu zeigen, dass $h \in \#\mathsf{P}$ enthalten ist, betrachten wir die $(p+q)$-balancierte Sprache

$$F = \{x\#yz \mid x\#y \in C \wedge x\#y\#z \in A\}.$$

Die Zugehörigkeit von h' zu #P folgt mittels der $nq(n)$-balancierten Sprache

$$L = \{x10^d\#0^{nq(n)-dq(|x|)}y_1 \cdots y_d \mid x\#y_1, \ldots, x\#y_d \in A\} \in \mathsf{P},$$

wobei $n = |x10^d|$ ist, da $\#L(x10^d) = f(x)^d$ gilt. Schließlich folgt $f' \in \#\mathsf{P}$ mittels der $s(n)$-balancierten Sprache

$$A' = \{x\#z0^{s(|x|)-q(|t(x)|)} \mid t(x)\#z \in A\},$$

wobei s ein Polynom mit $q(|t(x)|) \leq s(|x|)$ ist. ■

Nach Definition von $\oplus\mathsf{P}$ existiert für jede Sprache $A \in \oplus\mathsf{P}$ eine Funktion $h \in \#\mathsf{P}$ mit $h(x) \equiv_2 \chi_A(x)$. Das nächste Lemma zeigt, dass sich der Modul von 2 auf 2^d verstärken lässt.

Lemma 125. *Für jede Sprache $A \in \oplus\mathsf{P}$ ex. eine Funktion $f \in \#\mathsf{P}$ mit*

$$\chi_A(x) \equiv_{2^d} f(x10^d).$$

Beweis. Sei h eine #P-Funktion mit $h(x) \equiv_2 \chi_A(x)$. Nach Lemma 124 ist dann auch die Funktion

$$g : x10^d \mapsto (h(x)+1)^d$$

in #P. Nochmalige Anwendung von Lemma 124 zeigt, dass auch die Funktion

$$g' : x10^d10^d \mapsto (g(x10^d)+1)^d = ((h(x)+1)^d+1)^d = Q_d(h(x))$$

in #P ist. Nun definieren wir $f(x\#0^d) = g''(x10^d10^d) = g''(t(x10^d))$ für die FP-Funktion $t : x10^d \mapsto x10^d10^d$. Dann ist $f \in \#\mathsf{P}$ nach Lemma 125 und mit Lemma 123 folgt

$$\chi_A(x) \equiv_{2^d} f(x10^d).$$

■

Satz 126. $\mathsf{PP}^{\oplus\mathsf{P}} \subseteq \mathsf{P}^{\mathsf{PP}}$.

Beweis. Sei $L \in \mathsf{PP}^{\oplus\mathsf{P}} = \mathsf{P} \cdot \mathsf{P}^{\oplus\mathsf{P}} = \mathsf{P} \cdot \oplus\mathsf{P}$ und seien p ein Polynom und A eine p-balancierte Sprache in $\oplus\mathsf{P}$ mit

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

Nach Lemma 125 ex. zu $A \in \oplus\mathsf{P}$ eine Funktion $f \in \#\mathsf{P}$ mit

$$\chi_A(x\#y) \equiv_{2^d} f(x\#y10^d).$$

Betrachte nun die Funktion

$$g : x \mapsto \sum_{y \in \{0,1\}^{p(|x|)}} f(x\#y10^{p(|x|)+1}),$$

die nach Lemma 125 in $\#\mathsf{P}$ ist. Dann gilt

$$x \in L \Leftrightarrow g(x) \bmod 2^{p(|x|)+1} \geq 2^{p(|x|)-1}.$$

Da eine P^{PP}-Maschine den Wert von $g(x)$ durch eine Binärsuche mit Fragen an das PP-Orakel

$$B = \{x\#bin(k) \mid g(x) \geq k\}$$

berechnen und die Bedingung $g(x) \bmod 2^{p(|x|)+1} \geq 2^{p(|x|)-1}$ überprüfen kann, folgt $L \in \mathsf{P}^{\mathsf{PP}}$. ■

Korollar 127 (Toda, 1992)**.** $\mathsf{PH} \subseteq \mathsf{P}(\mathsf{PP})$

Beweis. $\mathsf{PH} \subseteq \mathsf{BP} \cdot \oplus\mathsf{P} \subseteq \mathsf{BPP}^{\oplus\mathsf{P}} \subseteq \mathsf{PP}^{\oplus\mathsf{P}} \subseteq \mathsf{P}^{\mathsf{PP}}$. ■

11 Komplexität von Anzahlproblemen

In diesem Abschnitt untersuchen wir die Komplexität einer Reihe von konkreten Anzahlproblemen. Alle bisher betrachteten NP-Probleme haben die Form $A = \exists \cdot B$, wobei sich die Sprache B in natürlicher Weise aus der Definition von A ergibt. Somit können wir jedem NP-Problem ein Anzahlproblem $\#B$ zuordnen. Offenbar ist das Anzahlproblem $\#B$ mindestens so schwierig wie das zugehörige NP-Problem A.

In manchen Fällen gilt hiervon auch die Umgkehrung. So werden wir beispielsweise sehen, dass sich die Anzahl $\#\mathrm{GI}(G, H)$ der Isomorphismen zwischen zwei Graphen G und H in Polynomialzeit durch Orakelfragen an GI berechnen lässt.

Meist ist das Anzahlproblem jedoch deutlich schwieriger zu lösen als das zugrunde liegende NP-Problem. So können wir jede Sprache in PH in Polynomialzeit durch Orakelfragen an $\#\textsc{Sat}$ entscheiden, was mit einem SAT-Orakel nicht möglich ist, außer wenn PH auf $\mathsf{P}(\mathsf{NP})$ kollabiert.

Es gibt sogar Beispiele, bei denen das zugrunde liegende NP-Problem $A = \exists \cdot B$ effizient entscheidbar ist und das Anzahlproblem $\#B$ dennoch schwierig ist. So ist etwa das Problem $\#\mathrm{BPM}(G)$, die Anzahl der perfekten Matchings in einem bipartiten Graphen G zu bestimmen, $\#\mathsf{P}$-vollständig, obwohl die Frage, ob G ein perfektes Matching besitzt, in Polynomialzeit entscheidbar ist. Als erstes Beispiel, bei dem das Anzahl- und das Entscheidungsproblem unterschiedliche Komplexitäten haben (unter der Voraussetzung $\mathsf{PP} \neq \mathsf{P}$), betrachten wir die Bestimmung der Anzahl aller erfüllenden Belegungen einer DNF-Formel.

Definition 128. *Eine boolesche Formel F über den Variablen $x_1, \ldots, x_n$ ist in **disjunktiver Normalform** (kurz* **DNF***), falls F eine Disjunktion $F = \bigvee_{i=1}^{m} D_i$ von Konjunktionen $D_i = \bigwedge_{j=1}^{k_i} l_{ij}$ von **Literalen** ist.*

#DnfSat

Gegeben: Eine boolsche Formel $F(x_1, \ldots, x_n)$ in DNF.
Gefragt: Wieviele Belegungen $a \in \{0,1\}^n$ erfüllen F?

Es ist leicht zu sehen, dass das zugehörige Entscheidungsproblem DnfSat in P lösbar ist.

DnfSat

Gegeben: Eine boolsche Formel $F(x_1, \ldots, x_n)$ in DNF.
Gefragt: Ist F erfüllbar?

Da aber die Negation einer KNF-Formel F leicht in eine logisch äquivalente DNF-Formel transformiert werden kann und $\#\text{Sat}(F) = 2^n - \#\text{Sat}(\neg F)$ ist, folgt $\#\text{Sat} \in \mathsf{FP}^{\#\text{DnfSat}[1]}$, wofür wir auch $\#\text{Sat} \leq^{\mathsf{FP}[1]} \#\text{DnfSat}$ schreiben. Da #Sat #P-vollständig unter $\leq_{par}$-Reduktionen ist, folgt $\#\mathsf{P} \subseteq \mathsf{FP}^{\#\text{DnfSat}[1]}$, d.h. #DnfSat ist #P-vollständig unter $\leq^{\mathsf{FP}[1]}$-Reduktionen.

Als nächstes Beispiel betrachten wir das Problem, die Anzahl der erfüllenden Belegungen einer monotonen Formel zu bestimmen.

Definition 129. *Eine boolesche Formel F heißt **monoton**, falls sie nur mittels $\vee$ und $\wedge$ aus Variablen und Konstanten aufgebaut ist.*

#MonSat

Gegeben: Eine monotone Formel $F(x_1, \ldots, x_n)$.
Gefragt: Wieviele Belegungen $a \in \{0,1\}^n$ erfüllen F?

Auch hier ist leicht zu sehen, dass das zugehörige Entscheidungsproblem MonSat in P lösbar ist. #MonSat ist dagegen #P-vollständig unter $\leq_{\parallel}^{\mathsf{FP}[2]}$-Reduktionen, da wir zu jeder KNF Formel $F(x_1, \ldots, x_n)$ zwei monotone Formeln $G(x_1, \ldots, x_n, y_1, \ldots, y_n)$ und $H(x_1, \ldots, x_n, y_1, \ldots, y_n)$ finden können mit

$$\#\text{Sat}(F) = \#\text{Sat}(G \wedge \neg H) = \#\text{MonSat}(G) - \#\text{MonSat}(H).$$

So können wir für G etwa die Formel $F'(x_1, \ldots, x_n, y_1, \ldots, y_n) \wedge \bigwedge_{i=1}^{n}(x_i \vee y_i)$ und für H die Formel $G \wedge \bigvee_{i=1}^{n}(x_i \wedge y_i)$ wählen, wobei F' aus F dadurch entsteht, dass wir jedes negative Literal $\bar{x}_i$ durch die Variable y_i ersetzen.

Als nächstes untersuchen wir die Komplexität von #GI und #GA, wobei $\#\text{GI}(G_1, G_2) = \|Iso(G_1, G_2)\|$ die Anzahl der Isomorphismen zwischen G_1 und G_2 und $\#\text{GA}(G) = \|Aut(G)\|$ die Anzahl der Automorphismen von G bestimmt. Wir wissen bereits, dass für isomorphe Graphen $\#\text{GI}(G_1, G_2) = \#\text{GA}(G_1) = \#\text{GA}(G_2)$ ist.

Das Komplement eines Graphen $G = (V, E)$ ist $\overline{G} = (V, \binom{V}{2} - E)$. Es ist leicht zu sehen, dass G und $\overline{G}$ die gleichen Automorphismen haben und $Iso(G, H) = Iso(\overline{G}, \overline{H})$ ist. Man beachte, dass $\overline{G}$ zusammenhängend ist, falls G dies nicht ist. Aus diesem Grund können wir o.B.d.A. annehmen, dass die Eingabegraphen für die Probleme GA, GI, #GA und #GI zusammenhängend sind.

Wegen $\#\text{GA}(G) = \#\text{GI}(G, G)$ folgt $\#\text{GA} \leq_{par} \#\text{GI}$. Es ist auch leicht, #GI auf #GA zu reduzieren. Hierzu betrachten wir folgende Graphoperation. Für zwei Graphen $G_1 = (V_1, E_1)$ und $G_2 = (V_2, E_2)$ sei $G_1 + G_2$ der Graph $(V_1 \cup V_2, E_1 \cup E_2)$, wobei wir die Knoten nötigenfalls so umbennen, dass $V_1 \cap V_2 = \emptyset$ ist. Falls G_1 und G_2 zusammenhängend sind, setzt sich jeder Automorphismus von $G_1 + G_2$, der die Knoten von G_1 und G_2 austauscht, aus einem Isomorphismus zwischen G_1 und G_2 und einem Isomorphismus zwischen G_2 und G_1 zusammen. Folglich gilt für zusammenhängende Graphen

$$\#\text{GA}(G_1 \cup G_2) = \begin{cases} \#\text{GA}(G_1) \cdot \#\text{GA}(G_2), & \text{if } G_1 \ncong G_2 \\ 2 \cdot \#\text{GA}(G_1) \cdot \#\text{GA}(G_2), & \text{if } G_1 \cong G_2 \end{cases}$$

und somit $\#\text{GI} \in \mathsf{FP}_{\parallel}^{\#\text{GA}[3]}$.

Als nächsten reduzieren wir #GA auf GI. Da das Graphenisomorphieproblem für gefärbte Graphen ColGI und GI logspace-äquivalent sind, d.h. es gilt ColGI $\equiv_m^{log}$ GI (siehe Übungen), reicht es, #GA auf ColGI zu reduzieren. Bezeichne $G_{[1,\ldots,i]}$ den Graphen, bei dem Knoten j mit der Farbe j $(j = 1, \ldots, i)$ und alle übrigen Knoten mit 0 gefärbt sind.

Es ist klar, dass $Aut(G_{[1,\ldots,i]})$ eine Untergruppe von $Aut(G_{[1,\ldots,i-1]})$ ist (in Zeichen $Aut(G_{[1,\ldots,i]}) \leq Aut(G_{[1,\ldots,i-1]})$), d.h. es gilt

$$\{id\} = Aut(G_{[1,\ldots,n]}) \leq \cdots \leq Aut(G_{[1]}) \leq Aut(G) \leq S_n,$$

wobei S_n die Gruppe aller Permutationen auf der Menge $\{1, \ldots, n\}$ bezeichnet. Aus der Gruppentheorie wissen wir, dass die Nebenklassen $Aut(G_{[1,\ldots,i]})\rho_j$ $(j = 1, \ldots, k_i)$ der Untergruppe $Aut(G_{[1,\ldots,i]})$ von $Aut(G_{[1,\ldots,i-1]})$ die Gruppe $Aut(G_{[1,\ldots,i-1]})$ in gleichmächtige Teilmengen partitionieren, d.h. $\#\mathrm{GA}(G_{[1,\ldots,i-1]}) = k_i \cdot \#\mathrm{GA}(G_{[1,\ldots,i]})$, und somit

$$\#\mathrm{GA}(G) = \prod_{i=1}^{n} k_i.$$

Da zwei Permutationen $\pi, \varphi \in Aut(G_{[1,\ldots,i]})$ genau dann in derselben Nebenklasse liegen, wenn $\pi\varphi^{-1} \in Aut(G_{[1,\ldots,i]})$ (d.h. $\pi(i) = \varphi(i)$) ist, ist die Anzahl der Nebenklassen gleich

$$\begin{aligned} k_i &= \|\{\pi(i) \mid \pi \in Aut(G_{[1,\ldots,i-1]})\}\| \\ &= \|\{j \geq i+1 \mid \exists \pi \in Aut(G_{[1,\ldots,i-1]}) : \pi(i) = j\}\| + 1. \end{aligned}$$

Anders ausgedrückt, $k_i = \sum_{j=i+1}^{n} k_{ij}$, wobei

$$k_{ij} = \begin{cases} 1, & G_{[1,\ldots,i]} \cong G_{[1,\ldots,i-1,j]}, \\ 0, & \text{sonst} \end{cases}$$

und $G_{[1,\ldots,i-1,j]}$ der Graph ist, bei dem Knoten k mit der Farbe k $(k = 1, \ldots, i-1)$, Knoten j mit der Farbe i und alle übrigen Knoten mit 0 gefärbt sind. Folglich lässt sich $\#\mathrm{GA}(G)$ durch $\sum_{i=1}^{n-1}(n-i-1) = \sum_{i=1}^{n-1} i = n(n-1)/2$ nichtadaptive Fragen an GI berechnen.