Vorlesungsskript

Einführung in die Komplexitätstheorie

Wintersemester 2019/20

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

12. Februar 2020

Inhaltsverzeichnis

1 Einführung

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

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

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

Erreichbarkeitsproblem in Digraphen (Reach):

Gegeben: Ein gerichteter Graph $G = (V, E)$ mit $V = \{1, \ldots, n\}$ 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:

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

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

Komplexitätsbetrachtungen:

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

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

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

Maximaler Fluß (MaxFlow):

Gegeben: Ein gerichteter Graph $G = (V, E)$ mit $V = \{1, \ldots, n\}$, $E \subseteq V \times V$ 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 max. Wert $w(f) = \sum_{(1,u)\in E} f(1, u) - \sum_{(u,1)\in E} f(u, 1)$.

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

Komplexitätsbetrachtungen:

- MaxFlow ist in Zeit $O(n^5)$ lösbar.
- MaxFlow ist in Platz $O(n^2)$ lösbar.

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

Perfektes Matching in bipartiten Graphen (Matching):

Gegeben: Ein bipartiter Graph $G = (U, W, E)$ mit $U \cap W = \emptyset$ und $e \cap U \neq \emptyset \neq e \cap W$ für alle Kanten $e \in E$.

Gefragt: Besitzt G ein perfektes Matching?

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

Komplexitätsbetrachtungen:

- Matching ist in Zeit $O(n^3)$ entscheidbar.
- Matching ist in Platz $O(n^2)$ entscheidbar.

Die bisher betrachteten Probleme können in deterministischer Polynomialzeit gelöst werden und gelten daher als effizient lösbar. Zum Schluss dieses Abschnitts betrachten wir ein Problem, für das vermutlich nur ineffiziente Algorithmen existieren. Wie üblich bezeichnen wir die Gruppe aller Permutationen auf der Menge $\{1, \ldots, n\}$ mit S_n.

Travelling Salesman Problem (TSP):

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

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

Komplexitätsbetrachtungen:

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

*Hierzu berechnen wir für alle Teilmengen $S \subseteq \{2, \ldots, n\}$ 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 5-Tupel $M = (Q, \Sigma, \Gamma, \delta, q_0)$. Dabei ist*

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

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

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

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

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

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

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

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

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

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 ***Ergebnis*** *$M(x)$ der Rechnung von M bei Eingabe x,*

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

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

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

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

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

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

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

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

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

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

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

Aus dem Grundstudium wissen wir, dass eine nichtleere Sprache $L \subseteq \Sigma^*$ genau dann semi-entscheidbar ist, wenn eine berechenbare Funktion $f : \Sigma^* \to \Sigma^*$ existiert, deren Bild $\mathit{range}(f) = \{f(x) \mid x \in \Sigma^*\}$ 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 $(q, a_1, \ldots, a_k) \mapsto (q', a'_1, D_1, \ldots, a'_k, D_k)$.
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) *$M(x)$ **hält** (kurz $M(x)\downarrow$), falls $M(x)$ nur endlich lange Rechnungen ausführt. Andernfalls schreiben wir $M(x)\uparrow$.*

b) *Eine Rechnung von $M(x)$ heißt **akzeptierend** (bzw. **verwerfend**), falls sie K_x in eine akzeptierende (bzw. verwerfende) Endkonfiguration überführt.*

c) *$M(x)$ **akzeptiert**, falls $M(x)$ mindestens eine akzeptierende Rechnung ausführt. Andernfalls **verwirft** $M(x)$.*

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

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

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

2.3 Zeitkomplexität

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

Definition 12.

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

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

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

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

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

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

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

bzw.

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

zusammen. Ferner sei

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

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

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

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

2.4 Platzkomplexität

Zur Definition von Platzkomplexitätsklassen verwenden wir so genannte Offline-Turingmaschinen und Transducer. Diese haben die

Eigenschaft, dass sie das erste Band nur als Eingabeband (also nur zum Lesen) bzw. das k-te Band nur als Ausgabeband (also nur zum Schreiben) benutzen. Der Grund für diese Einschränkungen liegt darin, sinnvolle Definitionen für Komplexitätsklassen mit einem sublinearen Platzverbrauch zu erhalten.

Definition 13. *Eine TM M heißt **Offline-TM**, falls für jede Anweisung $(q, a_1, \ldots, a_k) \mapsto (q', a_1', D_1, \ldots, a_k', D_k)$ 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 dieses kann nicht als Speicher benutzt werden (*write-only*).

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

Definition 14.

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

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

*der **Platzverbrauch** von M bei Eingabe x. Für eine Offline-TM ersetzen wir $\sum_{i=1}^{k} |u_i v_i|$ durch $\sum_{i=2}^{k} |u_i v_i|$ und für einen Transducer durch $\sum_{i=2}^{k-1} |u_i v_i|$. Man beachte, dass $space_M(x)$ im Fall $time_M(x) = \infty$ undefiniert ist.*

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

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

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

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

bzw.

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

zusammen. Ferner sei

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

Die wichtigsten Platzkomplexitätsklassen sind

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

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

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

mit

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

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

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

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

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

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

3.2 Deterministische Simulationen von nichtdeterministischen Berechnungen

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

Satz 18.

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

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

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

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

der maximale Verzweigungsgrad von N. Dann ist jede Rechnung

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

der Länge t von $N(x)$ eindeutig durch eine Folge $(i_1, \ldots, i_t) \in \{1, \ldots, d\}^t$ beschreibbar. Betrachte die Offline-*(k+2)*-DTM M, die auf ihrem 2. Band für $t = 1, 2, \ldots$ der Reihe nach alle Folgen $(i_1, \ldots, i_t) \in \{1, \ldots, d\}^t$ generiert. Für jede solche Folge kopiert M die Eingabe auf Band 3 und simuliert die zugehörige Rechnung von $N(x)$ auf den Bändern 3 bis $k + 2$. M akzeptiert, sobald N bei einer dieser Simulationen in den Zustand q_{ja} gelangt. Wird dagegen ein t erreicht, für das alle d^t Simulationen von N im Zustand q_{nein} oder q_{h} enden, so verwirft M. Nun ist leicht zu sehen, dass $L(M) = L(N)$ und der Platzverbrauch von M durch

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

beschränkt ist.

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

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

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

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

verschiedene Konfigurationen erreichen, wobei c eine von N abhängige Konstante ist. Um N zu simulieren, testet M für $s = 1, 2, \ldots,$ ob $N(x)$ eine akzeptierende Endkonfiguration $K = (q_{\text{ja}}, u_1, v_1, u_2, v_2)$ der *Größe* $|u_2 v_2| = s$ erreichen kann. Ist dies der Fall, akzeptiert M. Erreicht dagegen s einen Wert, so dass $N(x)$ keine Konfiguration der Größe s erreichen kann, verwirft M. Hierzu muss M für $s = 1, 2, \ldots, s(n)$ jeweils alle von der Startkonfiguration K_x erreichbaren Konfigurationen der Größe s bestimmen, was in Zeit $(c^{s(n)+\log n})^{O(1)} = 2^{O(s(n)+logn)}$ möglich ist. ■

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

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

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

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

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

Insbesondere erhalten wir somit die Inklusionskette

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

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

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

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

Prozedur reach(K, K', i)

```
1  if i = 0 then return(K = K' or K ->_N K')
2  for each Konfiguration K_j do
3    if reach(K, K_j, i − 1) and reach(K_j, K', i − 1) then
4      return(true)
5  return(false)
```

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

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. $K_x = (q_0, 1, \varepsilon, \triangleright)$).

Simulation: Sei (K, K', i) das am weitesten rechts auf dem 2. Band stehende Tripel (also das oberste Kellerelement).

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

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

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

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

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

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

Korollar 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

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

Definition 23. *Für eine Sprachklasse* C *bezeichne* $\mathsf{co\text{-}C} = \{\overline{L} | L \in \mathsf{C}\}$ *die zu* C ***komplementäre Sprachklasse****. Dabei bezeichnet* $\overline{L} = \Sigma^* - L$ *das* ***Komplement*** *einer Sprache* $L \subseteq \Sigma^*$.

Die zu NP komplementäre Klasse ist 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. erfüllen alle Belegungen $\vec{a} \in \{0,1\}^n$ die Formel F?

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.

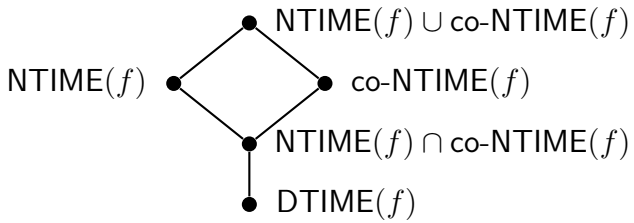
Deterministische Rechnungen lassen sich leicht komplementieren (durch Vertauschen der Zustände q_{ja} und q_{nein}). Daher sind deterministische Komplexitätsklassen unter Komplementbildung abgeschlossen.

Proposition 24.

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

ii) co-DTIME$(t(n))$ = DTIME$(t(n))$.

Damit ergibt sich folgende Inklusionsstruktur:



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

Definition 25. *Eine* NTM N *heißt* ***strong*** *bei Eingabe x, falls $N(x)$ mindestens eine akzeptierende oder mindestens eine verwerfende Rechnung ausführt, aber nicht beides.*

Proposition 26.

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

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

Beweis. Siehe Übungen. ■

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

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

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

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

wobei $K_1, \ldots, K_{c^{s(n)}}$ wie im Beweis des Satzes von Savitch eine Aufzählung aller Konfigurationen von $N(x)$ ist, die Platz höchstens $s(n)$ benötigen.

Behauptung 28. *Es existiert eine Offline-*NTM N_0 *mit $L(N_0) = L_0$, die $O(s(|x|))$ Felder besucht und bei allen Eingaben der Form $\langle x, r(x, t-1), t, K\rangle$ strong ist (d.h. $\langle x, r(x, t-1), t, K\rangle \notin L_0 \Leftrightarrow N_0(\langle x, r(x, t-1), t, K\rangle)$ hat eine verwerfende Rechnung.*

Beweis der Behauptung. $N_0(\langle x, r, t, K\rangle)$ benutzt einen mit dem Wert 0 initialisierten Zähler z und rät der Reihe nach für jede Konfiguration K_i eine Rechnung von $N(x)$ der Länge $\leq t-1$, die in K_i endet. Falls dies gelingt, erhöht N_0 den Zähler z 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 z den Wert r hat, andernfalls im Zustand q_{h}.

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

```
1  if t = 0 then halte im Zustand q_nein
2  z := 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      z := z + 1
7      if K_i →^{≤1}_N K then
8        halte im Zustand q_ja
9  if z = 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 $x_0 = \langle x, r(x, t-1), t, K\rangle$, $t \geq 1$, strong ist, also $N_0(x_0)$ genau im Fall $x_0 \notin L_0$ eine verwerfende Rechnung hat.

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 dieser Information für $x \notin L$ verifiziert, dass keine akzeptierende Konfigurationen erreichbar 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 bei Eingabe ⟨x, r⁻, t, K_i⟩
9      if N_0 akzeptiert then
10       r := r + 1
11       if K_i ist akzeptierende Endkonfiguration then
12         halte im Zustand q_ja
13     if N_0 haelt im Zustand q_h then
14       halte im Zustand q_h
15 until (r = r⁻)
16 halte im Zustand q_nein
```

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

Beweis der Behauptung. Wir führen Induktion über t:

$t = 1$**:** Im ersten Durchlauf der repeat-Schleife erhält r^- in Zeile 5 den Wert $1 = r(x, 0)$.

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

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

Beweis der Behauptung. Wir wissen bereits, dass im t-ten Durchlauf der repeat-Schleife r den Wert $r(x,t)$ und r^- den Wert $r(x,t-1)$ erhält. Wird daher die repeat-Schleife nach t_e Durchläufen verlassen, so gilt $r = r^- = r(x,t_e) = r(x,t_e-1)$.

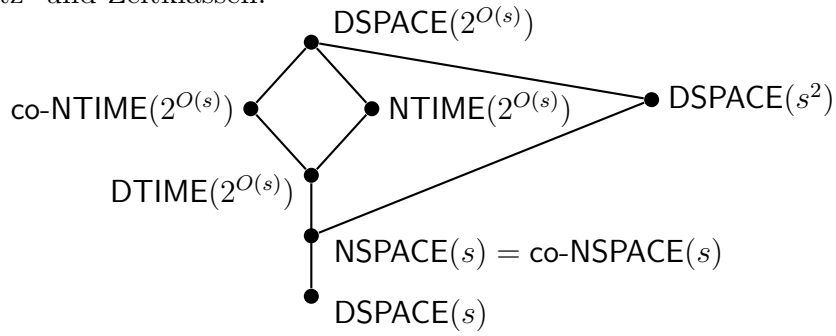
Angenommen $r(x,t_e) < \|\{K | K_x \underset{N}{\rightarrow}^* K\}\|$. Dann gibt es eine Konfiguration K, die für ein $t' > t_e$ in t' Schritten, aber nicht in t_e Schritten erreichbar ist. Betrachte eine Rechnung $K_x = K_0 \underset{N}{\rightarrow} K_1 \underset{N}{\rightarrow} \ldots \underset{N}{\rightarrow} K_{t'} = K$ minimaler Länge, die in K endet. Dann gilt $K_x \underset{N}{\rightarrow}^{t_e} K_{t_e}$, aber nicht $K_x \underset{N}{\rightarrow}^{\leq t_e - 1} K_{t_e}$ und daher folgt $r(x,t_e) > r(x,t_e-1)$. Widerspruch! ■

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

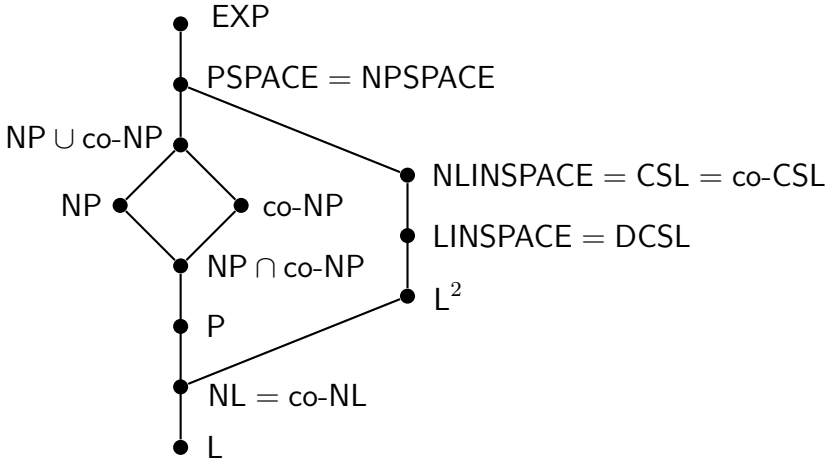
Korollar 31.

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

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



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



Eine zentrale Fragestellung der Komplexitätstheorie ist, welche dieser Inklusionen echt sind. Dies untersuchen wir im nächsten Kapitel.

4 Hierarchiesätze

4.1 Unentscheidbarkeit mittels Diagonalisierung

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

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

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

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

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

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

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

Satz 32. *Die Diagonalsprache*

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

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

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

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

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

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

so führt dies wegen

$$x_d \in D \overset{\text{(Def. von } D)}{\Rightarrow} M_d(x_d) \text{ akzeptiert} \overset{(**)}{\Rightarrow} x_d \notin D \;\lightning$$
$$x_d \notin D \overset{\text{(Def. von } D)}{\Rightarrow} M_d(x_d) \text{ akz. nicht} \overset{(**)}{\Rightarrow} x_d \in D \;\lightning$$

zu einem Widerspruch. ■

Satz 33. *Für jede berechenbare Funktion $g : \mathbb{N} \longrightarrow \mathbb{N}$ existiert eine entscheidbare Sprache $D_g \notin \mathsf{DTIME}(g(n))$.*

Beweis. Betrachte die Diagonalsprache

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

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

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

Dies führt jedoch auf einen Widerspruch:

$$x_d \in D_g \overset{(*)}{\Rightarrow} M_d(x_d) \text{ akzeptiert} \overset{(**)}{\Rightarrow} x_d \notin D_g \;\lightning$$
$$x_d \notin D_g \overset{(*)}{\Rightarrow} M_d(x_d) \text{ verwirft} \overset{(**)}{\Rightarrow} x_d \in D_g \;\lightning$$

■

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

4.2 Das Gap-Theorem

Satz 34 (Gap-Theorem)**.**
Es gibt eine berechenbare Funktion $g : \mathbb{N} \to \mathbb{N}$ mit

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

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

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

Betrachte hierzu das Prädikat

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

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

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

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

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

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

Um zu zeigen, dass jede Sprache $L \in \mathsf{DTIME}(2^{g(n)})$ bereits in $\mathsf{DTIME}(g(n))$ enthalten ist, sei M_k eine beliebige $2^{g(n)}$-zeitbeschränkte DTM mit $L(M_k) = L$. Dann muss M_k alle Eingaben x der Länge $n \geq k$ in Zeit $time_{M_k}(x) \leq g(n)$ entscheiden, da andernfalls $P(n, g(n))$ wegen $time_{M_k}(x) \in [g(n)+1, 2^{g(n)}]$ verletzt wäre. Folglich ist $L \in \mathsf{DTIME}(g(n))$, da die endlich vielen Eingaben x der Länge $n < k$ durch table-lookup in Zeit $n + 2 \leq g(n)$ entscheidbar sind. ■

Es ist leicht zu sehen, dass der Beweis des Gap-Theorems für jede berechenbare Funktion h eine berechenbare Zeitschranke g liefert, so dass $\mathsf{DTIME}(h(g(n))) = \mathsf{DTIME}(g(n))$ ist. Folglich ist die im Beweis von Satz 33 definierte Sprache D_g nicht in Zeit $h(g(n))$ entscheidbar.

4.3 Zeit- und Platzhierarchiesätze

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

Satz 35. *Es gibt eine universelle 3-DTM U, die für jede DTM M und jedes $x \in \{0,1\}^*$ bei Eingabe $\langle M, x\rangle$ eine Simulation von M bei Eingabe x in Zeit $O(|\langle M\rangle|(time_M(x))^2)$ und Platz $O(|\langle M\rangle| space_M(x))$ durchführt und dasselbe Ergebnis liefert:*

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

Beweis. Betrachte folgende Offline-*3*-DTM U:

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

Simulation: U simuliert jeden Rechenschritt von M wie folgt: Zunächst inspiziert U die auf dem 1. Band gespeicherte Kodierung von M, um die durch den Inhalt des 3. Bandes bestimmte Aktion von M zu ermitteln. Diese führt sie sodann auf dem 2. Band aus und aktualisert dabei auf dem 3. Band den Zustand und die gelesenen Zeichen von M. Insgesamt benötigt U für die Simulation eines Rechenschrittes von M Zeit $\mathcal{O}(kbg(n)) = \mathcal{O}(n \cdot g(n)))$.

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

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

Korollar 36. *(Zeithierarchiesatz)*
Für jede echte Komplexitätsfunktion $g(n) \geq n + 2$ gilt

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

Beweis. Es genügt zu zeigen, dass D_g für jede echte Komplexitätsfunktion $g(n) \geq n + 2$ in Zeit $O(ng^2(n))$ entscheidbar ist. Betrachte folgende *4*-DTM M'. M' überprüft bei einer Eingabe x der Länge n zuerst, ob x die Kodierung $\langle M\rangle$ einer k-DTM M ist. Falls ja, erzeugt M' auf dem 4. Band den String $1^{g(n)}$ in Zeit $\mathcal{O}(g(n))$ und simuliert $M(x)$ wie im Beweis von Theorem 35. Dabei vermindert M' die Anzahl der Einsen auf dem 4. Band nach jedem simulierten Schritt von $M(x)$ um 1. M' bricht die Simulation ab, sobald M stoppt oder der Zähler auf Band 4 den Wert 0 erreicht. M' hält genau dann im Zustand q_{ja}, wenn die Simulation von M im Zustand q_{ja} endet. Nun ist leicht zu sehen, dass M' $\mathcal{O}(n \cdot g(n)^2)$-zeitbeschränkt ist und die Sprache D_g entscheidet. ■

Korollar 37.

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

Beweis.

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

■

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

Satz 38. *Sei $f(n) \geq n+2$ 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 (ohne Beweis).

Satz 39 (Platzhierarchiesatz)**.** *Sind $g(n), f(n) \geq 2$ und ist f eine echte Komplexitätsfunktion mit*

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

dann ist

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

Damit lässt sich im Fall $2 \leq g(n) \leq f(n)$ die Frage, ob die Inklusion von $\mathsf{DSPACE}(g(n))$ in $\mathsf{DSPACE}(f(n))$ echt ist, eindeutig beantworten: Sie ist genau dann echt, wenn $\liminf_{n\to\infty} g(n)/f(n) = 0$ ist, da andernfalls $f(n) = O(g(n))$ ist und somit beide Klassen gleich sind.

Korollar 40.

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

Durch Kombination der Beweistechnik von Satz 39 mit der Technik von Immerman und Szelepcsényi erhalten wir auch für nichtdeterministische Platzklassen eine sehr fein abgestufte Hierarchie.

Satz 41 (Nichtdeterministischer Platzhierarchiesatz)**.** *Sind $g(n), f(n) \geq 2$ und ist f eine echte Komplexitätsfunktion mit*

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

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 42 (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

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

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

- die Sprache

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

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

Betrachte folgende NTM M:

```
1  input: 0^n
2    x := ε
3    s := 0
4    while h(s + |x|) ≤ n do
5      s := h(s + |x|)
6      x := x^+
7    if n < h(s + |x|) − 1 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
- im Fall $n = s(x^+) - 1$ die Frage, ob $0^s\#\langle M_{i(x)}\rangle \in D$ enthalten ist, in Zeit $h(|0^s\#\langle M_{i(x)}\rangle|) \leq h(s + |x|) = n + 1$ entscheidbar ist.

L kann aber nicht in $\mathsf{NTIME}(g(n))$ enthalten sein, da sonst eine Konstante c und eine 2-NTM M_i ex. würden mit $L(M_i) = L$ und $time_{M_i}(0^n) \leq cg(n)$ 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 42 liefert für langsam wachsende Zeitschranken eine feinere Hierarchie als Satz 38. Beispielsweise impliziert Satz 42, 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 38 eine feinere Hierarchie. So impliziert Satz 38 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 42 nur von $\mathsf{NTIME}(h(n)2^{2^{n+1}}) = \mathsf{NTIME}(h(n)2^{2^n}2^{2^n})$ separieren lässt.

5 Reduktionen

5.1 Logspace-Reduktionen

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 43. *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 44. $\mathsf{FL} \subseteq \mathsf{FP}$.

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 45. *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.

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

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

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

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

iii) Jeder Knoten j wird mindestens einmal besucht:

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

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

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

Die Klauseln in a) und b) 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 46.

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 47.

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 48. *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 49. *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

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

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 49 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 50. 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 $l = \|\Gamma\| + \|(Q \cup \{q_{\text{h}}, q_{\text{ja}}, q_{\text{nein}}\}) \times \Gamma\|$ verschiedene Tabelleneinträge besitzt, können wir jeden Eintrag $T_{i,j}$ durch eine Bitfolge $t_{i,j,1} \cdots t_{i,j,m}$ der Länge $m = \lceil \log_2 l \rceil$ kodieren.

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

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

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

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

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

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

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

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

Eine leichte Modifikation des Beweises von Satz 50 liefert folgendes Resultat.

Korollar 51. *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 51 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 52.

*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 Binärsprache*

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

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

Korollar 53 (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 $\mathsf{P} \neq NP$.

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 54. *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 55. $\mathsf{TALLY} \subseteq \mathsf{PSK}$.

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

Korollar 56. $\mathsf{PSK} \not\subseteq \mathsf{RE}$.

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 57. *Für eine Zahl $k \geq 1$ sei $\Gamma_k = \{0, \ldots, k-1\}$ das Alphabet, das die k Ziffern $0, \ldots, k-1$ als Zeichen enthält. Falls k aus dem Kontext ersichtlich ist, schreiben wir für Γ_k auch einfach Γ.*

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

$$\exists B = \{x \in \Sigma^* \,|\, \exists k \geq 1 \; \exists y \in \Gamma_k^* : x\#y \in B\}$$

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

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

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

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

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

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

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

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

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

Satz 59. CirSat *ist* NP*-vollständig.*

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

Im Beweis von Satz 58 haben wir gezeigt, dass für jede NP-Sprache A eine balancierte Sprache $B \subseteq \Sigma^*$ in P mit $A = \exists B$ existiert, d.h. es gibt ein Polynom q mit

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

Sei $m = \lceil \log_2 \|\Sigma\| \rceil$. Da die Binärsprache

$$B' = \{bin_m(x_1) \cdots bin_m(x_n) bin_m(\#) y \,|\, x_1 \cdots x_n \# y \in B\}$$

in P entscheidbar ist, existiert nach Korollar 51 eine FL-Funktion f, die einen Schaltkreis $f(1^n) = c_n$ mit $m(n+1) + q(n)$ Eingängen berechnet, so dass für alle $z \in \{0,1\}^{m(n+1)+q(n)}$ gilt:

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

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, $|x| = n$,

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

■

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

Satz 62. 3-Sat *ist* NP*-vollständig.*

Beweis. Es ist leicht zu sehen, dass 3-Sat $\in$ NP ist. Um 3-Sat als hart für NP nachzuweisen, reicht es aufgrund der Transitivität von $\leq$ 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 wiedergibt. Konkret enthält F_c für jedes Gatter g_i folgende Klauseln:

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

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

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

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

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

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 63. 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 64. *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 65. 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

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

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 66. 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 67. 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 68. 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 69. *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.*

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. Daher folgt aus der Behauptung, dass das Komplement von 2-SAT in NL ist. Wegen NL = co-NL folgt auch 2-SAT $\in$ NL.

Nun zum Beweis der Behauptung. 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).

Wegen $(**)$ treten bei der Ausführung von 2) keine Konflikte auf:

- Hätte l' in einer früheren Runde den Wert 0 erhalten, dann hätte in dieser Runde $\bar{l}'$ und somit auch $\bar{l}$ den Wert 1 erhalten, was der Wahl von l widerspricht.
- Wäre von l aus auch $\bar{l}'$ erreichbar, dann würde ein Pfad von l über $\bar{l}'$ nach $\bar{l}$ existieren, was durch die Wahl von l ebenfalls ausgeschlossen ist.

Zudem erfüllt α die Formel F, da für jede Kante $(l, l') \in E$ mit $\alpha(l) = 1$ auch $\alpha(l') = 1$ ist. ■

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 M, dass sie nur in einem der drei Zustände q_{ja}, q_{nein} oder q_{h} halten (hierfür schreiben wir auch $M(x) = 1$, $M(x) = 0$ bzw. $M(x) = ?$). Die von einer PTM M **akzeptierte Sprache** ist

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

In den Übungen werden wir sehen, dass jede Sprache in $\mathsf{RE} \cup \mathsf{co\text{-}RE}$ von einer PTM akzeptiert wird.

6.1 Die Klassen PP, BPP, RP und ZPP

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

Satz 70. $\mathsf{co\text{-}NP} \subseteq \mathsf{PP}$.

Beweis. Sei $A \in \mathsf{co\text{-}NP}$ und sei N eine polynomiell zeitbeschränkte NTM mit $L(N) = \bar{A}$. Sei N' die PP-TM, die sich aus N ergibt, wenn wir den Zustand q_{ja} durch q_{nein} und die beiden Zustände q_{nein}, q_{h} durch q_{ja} ersetzen. Dann gilt für alle $x \in \Sigma^*$:

$$\begin{array}{lcll} x \in A & \Rightarrow & N(x) \text{ akz. nicht} & \Rightarrow \Pr[N'(x) = 1] = 1 \\ x \notin A & \Rightarrow & N(x) \text{ akz.} & \Rightarrow \Pr[N'(x) = 1] < 1 \end{array}$$

Betrachte folgende PP-TM M, die bei Eingabe x zufällig eine der beiden folgenden Möglichkeiten wählt:

- M verwirft sofort,
- M simuliert N' bei Eingabe x.

Dann gilt für alle $x \in \Sigma^*$,

$$\Pr[M(x) = 1] = \Pr[N'(x) = 1]/2$$

und somit

$$\begin{array}{lcl} x \in A & \Rightarrow & \Pr[M(x) = 1] = 1/2, \\ x \notin A & \Rightarrow & \Pr[M(x) = 1] < 1/2. \end{array}$$

■

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

Sei M eine PTM und sei $L(M) = A$. Die **Fehlerwahrscheinlichkeit** von M bei Eingabe x ist

$$\Pr[M(x) = \bar{A}(x)],$$

wobei $\bar{A}(x)$ die charakteristische Funktion von $\bar{A}$ ist.

Da eine PTM M bei jeder Eingabe $x \in L(M)$ mit Wahrscheinlichkeit $\geq 1/2$ den Wert 1 (also mit Wahrscheinlichkeit $\leq 1/2$ den Wert 0) und bei jeder Eingabe $x \in \overline{L(M)}$ mit Wahrscheinlichkeit $< 1/2$ den Wert 1 liefert, ist die Fehlerwahrscheinlichkeit jeder PTM M für alle $x \in L(M) \leq 1/2$ und für alle $x \in \overline{L(M)}$ sogar $< 1/2$.

Lemma 71. *Für jede Sprache $A \in \mathsf{PP}$ existiert eine* PP-TM M *mit $L(M) = A$, die nie mit Wahrscheinlichkeit $1/2$ akzeptiert und somit bei allen Eingaben eine Fehlerwahrscheinlichkeit $< 1/2$ hat.*

Beweis. Sei $A \in \mathsf{PP}$ und sei N eine PP-TM mit $L(N) = A$. 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 für $\epsilon(x) = k^{-p(|x|)}$,

$$\begin{aligned} x \in A &\Rightarrow \Pr[N(x) = 1] \geq 1/2, \\ x \notin A &\Rightarrow \Pr[N(x) = 1] \leq 1/2 - \epsilon(x). \end{aligned}$$

Sei N' eine PP-TM mit $\Pr[N'(x) = 1] = 1/2 + \epsilon(x)/2$ und betrachte die PP-TM 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

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

■

Eine direkte Folgerung von Lemma 71 ist der Komplementabschluss von PP.

Korollar 72. $\mathsf{NP} \subseteq \mathsf{PP} = \mathsf{co\text{-}PP}$.

Tatsächlich liefert Lemma 71 sogar den Abschluss von PP unter symmetrischer Differenz.

Satz 73. 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 PP-TMs 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 PP-TM 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 Akzeptanzwahrscheinlichkeit 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 $x \in L_1 \triangle L_2$ mit Wahrscheinlichkeit

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

und alle Eingaben $x \in \overline{L_1 \triangle L_2} = (L_1 \cap L_2) \cup (\bar{L}_1 \cap \bar{L}_2)$ 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):

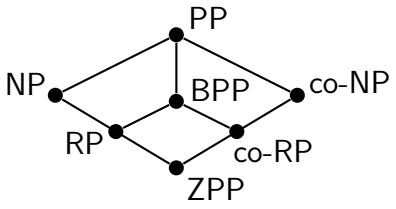
Gegeben: Eine boolsche Formel $F(x_1, \ldots, x_n)$.
Gefragt: Wird F von mindestens 2^{n-1} Belegungen erfüllt?

Definition 74. *Sei M eine* PP-TM *und sei $A = L(M)$. M heißt*

- BPP-TM*, falls für alle x gilt:* $\Pr[M(x) = A(x)] \geq 2/3$,
- RP-TM*, falls für alle $x \notin A$ gilt:* $\Pr[M(x) = 0] = 1$,
- ZPP-TM*, falls für alle x gilt:* $\Pr[M(x) = A(x)] \geq 1/2$ *und* $\Pr[M(x) = \bar{A}(x)] = 0$.

Die Klasse BPP *(bounded error probabilistic polynomial time) enthält alle Sprachen, die von einer* BPP-TM *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 RP-TM oder BPP-TM M o.B.d.A. annehmen können, dass M niemals ? ausgibt (indem wir z.B. ? durch 0 ersetzen). Allerdings ist nicht ausgeschlossen, dass M ein falsches Ergebnis $M(x) = \bar{A}(x)$ liefert. Probabilistische Algorithmen mit dieser Eigenschaft werden auch als **Monte Carlo Algorithmen** bezeichnet. Im Unterschied zu einer BPP-TM, die bei allen Eingaben x „lügen“ kann, ist dies einer RP-TM nur im Fall $x \in A$ erlaubt. Man spricht hier auch von **zwei-** bzw. **einseitigem Fehler**. Eine ZPP-TM darf dagegen überhaupt nicht lügen. Algorithmen von diesem Typ werden als **Las Vegas Algorithmen** bezeichnet. Aus Definition 74 folgt sofort, dass BPP und ZPP unter Komplement abgeschlossen sind. Zudem ist leicht zu sehen, dass $\mathsf{P} \subseteq \mathsf{ZPP} \subseteq \mathsf{RP} \subseteq \mathsf{NP}$ gilt.



Satz 75. $\mathsf{ZPP} = \mathsf{RP} \cap \text{co-RP}$.

Beweis. Die Inklusion von links nach rechts ist klar. Für die umgekehrte Richtung sei A eine Sprache in $\mathsf{RP} \cap \text{co-RP}$. Dann existieren RP-TMs M_A und $M_{\bar{A}}$ für A und $\overline{A}$, wobei wir annehmen, dass M_A und $M_{\bar{A}}$ niemals ? ausgeben. Weiter sei $\overline{M}_{\bar{A}}$ die PP-TM, die aus $M_{\bar{A}}$ durch Vertauschen von q_{ja} und q_{nein} hervorgeht. Dann gilt

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

M_A kann also nur Eingaben $x \in A$ akzeptieren, während $\overline{M}_{\bar{A}}$ nur Eingaben $x \notin A$ verwerfen kann. Daher kann die Kombination $M_A(x) = 1$ und $\overline{M}_{\bar{A}}(x) = 0$ nicht auftreten. Weiter ergeben sich hieraus für die PP-TM M, die bei Eingabe x die beiden PP-TMs $M_A(x)$ und $\overline{M}_{\bar{A}}(x)$ simuliert und sich gemäß der Tabelle

	$M_A(x) = 1$	$M_A(x) = 0$
$\overline{M}_{\bar{A}}(x) = 1$	1	?
$\overline{M}_{\bar{A}}(x) = 0$	–	0

verhält, folgende Äquivalenzen:

$$\begin{aligned} M(x) = 1 &\Leftrightarrow M_A(x) = 1, \\ M(x) = 0 &\Leftrightarrow \overline{M}_{\bar{A}}(x) = 0. \end{aligned}$$

Nun ist leicht zu sehen, dass folgende Implikationen gelten:

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

Dies zeigt, dass M eine ZPP-TM für A ist, da für alle Eingaben x gilt:

$$\Pr[M(x) = A(x)] \geq 1/2 \text{ und } \Pr[M(x) = \bar{A}(x)] = 0. \qquad \blacksquare$$

6.2 Anzahl-Operatoren

Definition 76 (Anzahlklassen)**.** *Zu einer (k,q)-balancierten Sprache $B \subseteq \Sigma^*$ definieren wir die Funktion $\#B : \Sigma^* \to \mathbb{N}$ und für $\mathsf{Op} \in \{\exists^p, \forall^p, \exists^{\geq 1/2}, \oplus\}$ die Sprachen $\mathsf{Op}B \subseteq \Sigma^*$ wie folgt:*

$$\begin{aligned}
\#B(x) &= \|\{y \in \Gamma^{q(|x|)} \,|\, x\#y \in B\}\| \\
\exists^p B &= \{x \in \Sigma^* \,|\, \#B(x) > 0\} \\
\forall^p B &= \{x \in \Sigma^* \,|\, \#B(x) = k^{q(|x|)}\} \\
\exists^{\geq 1/2} B &= \{x \in \Sigma^* \,|\, \#B(x) \geq k^{q(|x|)}/2\} \\
\oplus B &= \{x \in \Sigma^* \,|\, \#B(x) \text{ ist ungerade}\}
\end{aligned}$$

*B heißt **einseitig**, falls $\#B(x)$ für keine Eingabe x im (offenen) Intervall $(1, k^{q(n)}/2)$ liegt, und **zweiseitig**, falls $\#B(x)$ für kein x im Intervall $(k^{q(n)}/3, 2 \cdot k^{q(n)}/3)$ liegt. Für eine Sprachklasse C und $\mathsf{Op} \in \{\#, \exists^p, \forall^p, \exists^{\geq 1/2}, \oplus\}$ sei*

$$\mathsf{Op} \cdot \mathsf{C} = \{\mathsf{Op}B \,|\, B \in \mathsf{C} \text{ ist balanciert}\}$$

Zudem definieren wir die Operatoren R und BP durch

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

und

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

Wie wir bereits wissen, gilt $\exists^p \cdot \mathsf{P} = \mathsf{NP}$, was sofort und $\forall^p \cdot \mathsf{P} = \text{co-}\mathsf{NP}$ impliziert. Als nächstes zeigen wir die Charakterisierungen $\mathsf{R} \cdot \mathsf{P} = \mathsf{RP}$, $\exists^{\geq 1/2} \cdot \mathsf{P} = \mathsf{PP}$ und $\mathsf{BP} \cdot \mathsf{P} = \mathsf{BPP}$. Als Hilfsmittel benutzen wir folgendes Lemma.

Lemma 77. *Für jede* PP-TM M *existiert eine (k,q)-balancierte Sprache $B \in \mathsf{P}$, so dass für alle Eingaben $x \in \Sigma^*$, $|x| = n$, gilt:*

$$\Pr[M(x) = 1] = \#B(x)/k^{q(n)}$$

Beweis. Sei q eine polynomielle Zeitschranke für M und sei $k = \mathrm{kgV}(1, 2, \ldots, c)$, wobei $c \geq 1$ der maximale Verzweigungsgrad von M ist. Wir benutzen Strings der Länge $q(n)$ über dem Alphabet $\Gamma = \{0, \ldots, k-1\}$, um die Rechnungen von M bei Eingabe $x \in \Sigma^n$ zu beschreiben. Hierzu ordnen wir dem String $y = y_1 \cdots y_{q(n)} \in \Gamma^{q(n)}$ diejenige Rechnung von $M(x)$ zu, bei der M im i-ten Rechenschritt in die Konfiguration K_j mit Index $j = y_i \bmod c_i$ übergeht, wobei $K_0, \ldots, K_{c_i - 1}$ die $c_i \geq 1$ möglichen Folgekonfigurationen in diesem Schritt sind. Das Ergebnis der durch y beschriebenen Rechnung von $M(x)$ bezeichnen wir mit $M_y(x)$. Nun ist leicht zu sehen, dass die Sprache

$$B = \{x\#y \mid x \in \Sigma^*, y \in \Gamma^{q(|x|)}, M_y(x) = 1\}$$

sowohl (k,q)-balanciert als auch in P entscheidbar ist und die Gleichung

$$\Pr[M(x) = 1] = \Pr_{y \in_R \Gamma^{q(n)}}[M_y(x) = 1] = \#B(x)/k^{q(n)}$$

erfüllt. ∎

Satz 78. $\exists^{\geq 1/2} \cdot \mathsf{P} = \mathsf{PP}$, $\mathsf{BP} \cdot \mathsf{P} = \mathsf{BPP}$ *und* $\mathsf{R} \cdot \mathsf{P} = \mathsf{RP}$.

Beweis. Die Inklusionen $\exists^{\geq 1/2} \cdot \mathsf{P} \subseteq \mathsf{PP}$, $\mathsf{BP} \cdot \mathsf{P} \subseteq \mathsf{BPP}$ und $\mathsf{R} \cdot \mathsf{P} \subseteq \mathsf{RP}$ sind klar.

Zum Nachweis der umgekehrten Inklusionen sei $A \in \mathsf{PP}$ und sei M eine PP-TM mit $L(M) = A$. Nach Lemma 77 existiert eine (k,q)-balancierte Sprache $B \in \mathsf{P}$, so dass für alle Eingaben x, $|x| = n$, die Gleichheit

$$\Pr[M(x) = 1] = \#B(x)/k^{q(n)}$$

gilt. Wegen

$$x \in A \Leftrightarrow \Pr[M(x) = 1] \geq 1/2 \Leftrightarrow \#B(x) \geq k^{q(n)}/2$$

folgt somit $A = \exists^{\geq 1/2} B \in \exists^{\geq 1/2} \cdot \mathsf{P}$. Ist M eine BPP-TM, so gilt

$$\begin{aligned}
x \in A &\Rightarrow \Pr[M(x) = 1] \geq 2/3 \Rightarrow \#B(x) \geq 2k^{q(n)}/3, \\
x \notin A &\Rightarrow \Pr[M(x) = 1] \leq 1/3 \Rightarrow \#B(x) \leq k^{q(n)}/3,
\end{aligned}$$

also ist B zweiseitig und es folgt $A = \exists^{\geq 1/2} B \in \mathsf{BP} \cdot \mathsf{P}$. Ist M eine RP-TM, so gilt

$$\begin{aligned} x \in A &\Rightarrow \Pr[M(x) = 1] \geq 1/2 \Rightarrow \#B(x) \geq k^{q(n)}/2, \\ x \notin A &\Rightarrow \Pr[M(x) = 1] = 0 \quad \Rightarrow \#B(x) = 0, \end{aligned}$$

also ist B einseitig und es folgt $A = \exists^{\geq 1/2} B \in \mathsf{R} \cdot \mathsf{P}$. ■

6.3 Reduktion der Fehlerwahrscheinlichkeit

In diesem Abschnitt zeigen wir, wie sich für RP-, ZPP- und BPP-Maschinen M die Wahrscheinlichkeit $\Pr[M(x) \neq A(x)]$ für ein inkorrektes (d.h. $M(x) = \bar{A}(x)$) oder unentschiedenes Ergebnis (d.h. $M(x) = ?$) für ein beliebiges Polynom q auf einen Wert $\leq 2^{-q(|x|)}$ verkleinern lässt.

Definition 79. *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 $\langle y_1, \ldots, y_m \rangle$ von Wörtern y_i berechnet mit*

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

*Der Begriff der **konjunktiven Reduktionzierbarkeit** $A \leq_c B$ ist analog definiert. Gilt dagegen*

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

*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 co-NP unter diesen Reduktionen abgeschlossen sind. Als nächstes zeigen wir, wie sich die (einseitige) Fehlerwahrscheinlichkeit von RP-TMs verkleinern lässt.

Satz 80. *Falls C unter disjunktiven Reduktionen abgeschlossen ist, existieren für jede Sprache $A \in \mathsf{R} \cdot \mathsf{C}$ und jedes Polynom r eine (k, p)-balancierte Sprache $C \in \mathsf{C}$, so dass für alle x, $|x| = n$, gilt:*

$$\begin{aligned} x \in A &\Rightarrow \#C(x) \geq (1 - 2^{-r(n)}) k^{p(n)}, \\ x \notin A &\Rightarrow \#C(x) = 0. \end{aligned}$$

Beweis. Sei $A \in \mathsf{R} \cdot \mathsf{C}$ und sei $B \in \mathsf{C}$ eine (k, q)-balancierte Sprache mit

$$\begin{aligned} x \in A &\Rightarrow \#B(x) \geq k^{q(|x|)}/2 \\ x \notin A &\Rightarrow \#B(x) = 0. \end{aligned}$$

Dann ist die Sprache

$$C = \{x \# y_1 \cdots y_{r(n)} \,|\, y_1, \ldots, y_{r(n)} \in \Gamma^{q(|x|)}, \exists i : x \# y_i \in B\}$$

disjunktiv auf B reduzierbar und somit in C. Sei $p(n)$ das Polynom $p(n) = q(n)r(n)$. Dann ist C (k, p)-balanciert und es gilt

$$\begin{aligned} &x \in A \Rightarrow \Pr_{y \in_R \Gamma^{q(n)}}[x \# y \notin B] < 1/2 \Rightarrow \Pr_{z \in_R \Gamma^{p(n)}}[x \# z \notin C] < 2^{-r(n)} \\ &x \notin A \Rightarrow \Pr_{y \in_R \Gamma^{q(n)}}[x \# y \in B] = 0 \Rightarrow \Pr_{z \in_R \Gamma^{p(n)}}[x \# z \in C] = 0. \end{aligned}$$ ■

Korollar 81. *Für jedes Polynom r und jede Sprache $A \in \mathsf{RP}$ existiert eine* RP-TM *M mit*

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

Ganz analog lässt sich die Zuverlässigkeit einer ZPP-TM M verbessern, indem wir sie $r(n)$-mal laufen lassen und nur dann ? ausgeben, wenn jedesmal $M(x) = ?$ war.

Korollar 82. *Für jedes Polynom r und jede Sprache $A \in \mathsf{ZPP}$ existiert eine* ZPP-TM *M mit $L(M) = A$ und $\Pr[M(x) = ?] \leq 2^{-r(|x|)}$ für jede Eingabe x.*

Für die Reduktion der Fehlerwahrscheinlichkeit von BPP-TMs benötigen wir das folgende Lemma.

Lemma 83. *Sei E ein Ereignis, das mit Wahrscheinlichkeit $1/2 - \epsilon$, $\epsilon \geq 0$, auftritt. Dann ist die Wahrscheinlichkeit, dass sich E bei $m = 2t + 1$ unabhängigen Wiederholungen mehr als t-mal ereignet, höchstens $1/2(1 - 4\epsilon^2)^t$.*

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

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

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

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

Wegen

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

erhalten wir somit

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

■

Satz 84. *Sei C unter majority-Reduktionen abgeschlossen. Dann existieren für jede Sprache $A \in \mathsf{BP} \cdot \mathsf{C}$ und jedes Polynom r eine (k, p)-balancierte Sprache $C \in \mathsf{C}$ mit $C \subseteq \{x\#y \mid y \in \Gamma^{p(|x|)}\}$, so dass für alle x, $|x| = n$, gilt:*

$$\Pr_{z \in_R \Gamma^{p(n)}}[A(x) = C(x\#z)] \geq 1 - 2^{-r(n)}$$

Beweis. Seien $A \in \mathsf{BP} \cdot \mathsf{C}$, Γ ein Alphabet der Größe k und $B \in \mathsf{C}$ eine (k, q)-balancierte Sprache mit

$$\Pr_{y \in_R \Gamma^{q(n)}}[A(x) \neq B(x\#y)] \leq 1/3 = 1/2 - 1/6$$

Sei $t(n) = \lceil (r(n) - 1)/\log_2(9/8) \rceil$ und sei $p(n)$ das Polynom $p(n) = q(n)(2t(n) + 1)$. Dann ist die (k, p)-balancierte Sprache

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

auf B majority-reduzierbar und somit in C. Weiter folgt nach Lemma 83 für ein zufällig gewähltes $z = y_1 \cdots y_{2t(n)+1} \in_R \Gamma^{p(n)}$,

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

■

Korollar 85. *Für jede Sprache $A \in \mathsf{BPP}$ und jedes Polynom r ex. eine* BPP-TM *M mit*

$$\Pr[M(x) = A(x)] \geq 1 - 2^{r(|x|)}.$$

Satz 86. $\mathsf{BPP} \subseteq \mathsf{PSK}$.

Beweis. Sei $A \in \mathsf{BPP}$. Dann ist auch die Sprache $B = bin(A)$ in $\mathsf{BPP} = \mathsf{BP} \cdot \mathsf{P}$. Nach Satz 84 existieren für das Polynom $r(n) = n + 1$ eine (k, p)-balancierte Sprache $C \in \mathsf{C}$ mit $C \subseteq \{x\#y \,|\, y \in \Gamma^{p(|x|)}\}$, so dass für alle $x \in \{0,1\}^*$, $|x| = n$ gilt:

$$\Pr_{z \in_R \Gamma^{p(n)}}[B(x) \neq C(x\#z)] \leq 2^{-n-1}$$

Daher folgt für ein zufällig gewähltes $z \in_R \Gamma^{p(n)}$,

$$\Pr[\exists x \in \{0,1\}^n : B(x) \neq C(x\#z)] \leq \sum_{x \in \{0,1\}^n} \Pr[B(x) \neq C(x\#z)] < 1$$

Also muss für jede Eingabelänge n ein String z_n mit $B(x) = C(x\#z_n)$ für alle $x \in \{0,1\}^n$ existieren. Da mit C auch die Binärsprache $C' = \{x bin(\#z) \mid x\#z \in C\} \in \mathsf{P}$ ist, existiert nach Korollar 51 eine Funktion $f \in \mathsf{FL}$ die bei Eingabe 1^n einen Schaltkreis $f(1^n) = c'_n$ ausgibt mit $c'_n(x bin(\#z)) = C'(x bin(\#z))$ für alle $x \in \{0,1\}^n$ und $z \in \Gamma^{p(n)}$. Folglich sind auch die n-stelligen booleschen Funktionen $f_{z_n} \colon x \mapsto c'_n(x bin(\#z_n)) = B(x)$ durch Schaltkreise c_n polynomieller Größe berechenbar, d.h. B und damit auch A sind in PSK. ■

Man beachte, dass zwar die Schaltkreisfamilie $(c'_n)_{n \geq 0}$ logspace uniform ist (mittels $f \in \mathsf{FL}$), aber nicht die Familie $(c_n)_{n \geq 0}$, da das Auffinden von z_n u.U. sehr aufwändig sein kann.

6.4 Abschlusseigenschaften von Anzahl-Klassen

In diesem Abschnitt gehen wir der Frage nach, unter welchen Reduktionen und Operatoren Anzahl-Klassen abgeschlossen sind.

Lemma 87. *Für jede unter $\leq_m^{log}$ abgeschlossene Sprachklasse C gilt*

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

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

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

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

(i) Sei $A \in \mathsf{co}\text{-}\exists^p \cdot \mathcal{C}$. Dann existiert eine (k, q)-balancierte Sprache $B \in \mathsf{C}$ mit $\bar{A} = \exists B$. Betrachte die Sprache

$$\hat{B} = \{x\#y \,|\, x \in \Sigma^*, y \in \Gamma^{q(|x|)}, x\#y \notin B\}$$

Diese ist ebenfalls (k, q)-balanciert und wegen $\hat{B} \leq_m \bar{B}$ in $\mathsf{co}\text{-}\mathsf{C}$. Zudem gilt $\#\hat{B}(x) = k^{q(n)} - \#B(x)$. Daher folgt

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

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

(ii) Sei $A \in \mathsf{co}\text{-}\mathsf{BP} \cdot \mathcal{C}$ und sei $B \in \mathsf{C}$ eine (k, q)-balancierte zweiseitige Sprache mit $\bar{A} = \exists^{\geq 1/2} B$. Dann ist die in (i) definierte (k, q)-balancierte Sprache $\hat{B} \in \mathsf{co}\text{-}\mathsf{C}$ 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 (k, q)-balancierte Sprache mit $A = \oplus B$. Betrachte wieder die in (i) definierte (k, q)-balancierte Sprache $\hat{B} \in \mathsf{co}\text{-}\mathsf{C}$ sowie die $(k, q+1)$-balancierte Sprache

$$B' = \{x\#0y \,|\, x\#y \in B\} \cup \{x\#1^{p(|x|)+1} \,|\, x \in \Sigma^*\} \in \mathsf{C},$$

wobei wir o.B.d.A. $\{0,1\} \subseteq \Gamma$ annehmen. Dann gilt $\#B'(x) = \#B(x) + 1$, d.h. $\#B(x)$ und $\#B'(x)$ haben unterschiedliche Parität. Dies zeigt $A \in \mathsf{co}\text{-}\oplus \cdot \mathcal{C}$. Zudem hat $\#B(x)$ im Fall k gerade die gleiche Parität wie $\#\hat{B}(x)$ und im Fall k ungerade die gleiche Parität wie $\#\hat{B}'(x) = \#\hat{B}(x) + 1$ (Definition der Sprache $\hat{B}' \in \mathsf{co}\text{-}\mathsf{C}$ ist analog zu B'). Dies zeigt $A \in \oplus \cdot \mathsf{co}\text{-}\mathcal{C}$. ■

Lemma 88. *Sei C unter $\leq_m^{log}$ abgeschlossen und sei $B \in \mathsf{C}$ eine (k, q)-balancierte Sprache. Dann existieren für jede Funktion $f \in \mathsf{FL}$ (k, p)-balancierte Sprachen $B', B'' \in \mathsf{C}$ mit*

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

Beweis. Sei p ein Polynom mit $q(|f(x)|) \leq p(|x|)$ für alle x. Betrachte die Sprachen

$$B' = \{x\#y'y'' \,|\, f(x)\#y' \in B, y'' = 0^{p(n)-q(|f(x)|)}\}$$
$$B'' = \{x\#y'y'' \,|\, f(x)\#y' \in B, y'' \in \Gamma^{p(n)-q(|f(x)|)}\}$$

Dann gilt $B', B'' \leq_m^{log} B$, also $B', B'' \in \mathsf{C}$. Da jedes Präfix y' mit $f(x)\#y' \in B$ genau eine Verlängerung y'' mit $x\#y'y'' \in B'$ und genau $k^{p(|x|)-q(|f(x)|)}$ Verlängerungen y'' mit $x\#y'y'' \in B''$ hat, folgt

$$\#B'(x) = \#B(f(x)) \text{ und } \#B''(x) = \#B(f(x))k^{p(|x|)-q(|f(x)|)}. \quad \blacksquare$$

Mit obigem Lemma ist es nun leicht, folgende Abschlusseigenschaften zu zeigen.

Satz 89. *Sei* C *eine unter* $\leq_m^{log}$ *abgeschlossene Sprachklasse und sei* $\mathsf{Op} \in \{\exists^p, \forall^p, \mathsf{R}, \mathsf{BP}, \exists^{\geq 1/2}, \oplus\}$. *Dann ist auch die Klasse* $\mathsf{Op} \cdot \mathsf{C}$ *unter* $\leq_m^{log}$ *abgeschlossen.*

Beweis. Sei $A \in \mathsf{Op} \cdot \mathsf{C}$ mittels einer (k, q)-balancierten Sprache $B \in \mathsf{C}$ und gelte $A' \leq_m^{log} A$ mittels einer FL-Funktion f. Nach obigem Lemma existieren ein Polynom p und (k, p)-balancierte Sprachen $B', B'' \in \mathsf{C}$ mit

$$\#B'(x) = \#B(f(x)) \text{ und } \#B''(x)/k^{p(|x|)} = \#B(f(x))/k^{q(|f(x)|)}$$

Nun folgt für $\mathsf{Op} = \oplus$,

$$x \in A' \Leftrightarrow f(x) \in A \Leftrightarrow \#B(f(x)) \equiv_2 1 \Leftrightarrow \#B'(x) \equiv_2 1,$$

weshalb $A' = \oplus B' \in \oplus \cdot \mathsf{C}$ ist. Entsprechend folgt für $\mathsf{Op} = \exists^p$,

$$x \in A' \Leftrightarrow f(x) \in A \Leftrightarrow \#B(f(x))/k^{q(|f(x)|)} > 0 \Leftrightarrow \#B''(x)/k^{p(|x|)} > 0,$$

weshalb $A' = \exists B' \in \exists^p \cdot \mathsf{C}$ ist. Die Fälle $\mathsf{Op} \in \{\forall^p, \mathsf{R}, \mathsf{BP}, \exists^{\geq 1/2}\}$ folgen analog, wobei für $\mathsf{Op} \in \{\mathsf{R}, \mathsf{BP}\}$ noch zu beachten ist, dass mit B auch B'' ein- bzw. zweiseitig ist. $\blacksquare$

Satz 90. *Sei* C *eine unter* $\leq_m^{log}$ *abgeschlossene Sprachklasse. Dann gilt* $\exists^p \cdot \exists^p \cdot \mathsf{C} = \exists^p \cdot \mathsf{C}$, $\forall^p \cdot \forall^p \cdot \mathsf{C} = \forall^p \cdot \mathsf{C}$ *und* $\oplus \cdot \oplus \cdot \mathsf{C} = \oplus \cdot \mathsf{C}$.

Beweis. Siehe Übungen. $\blacksquare$

Das nächste Lemma zeigt, dass mit C auch die Klassen $\exists^p \cdot \mathsf{C}$, $\forall^p \cdot \mathsf{C}$ und $\mathsf{BP} \cdot \mathsf{C}$ unter majority-Reduktionen abgeschlossen sind.

Satz 91. *Sei* C *eine unter majority-Reduktionen abgeschlossene Sprachklasse und sei* $\mathsf{Op} \in \{\exists^p, \forall^p, \mathsf{BP}\}$. *Dann ist auch* $\mathsf{Op} \cdot \mathsf{C}$ *unter majority-Reduktionen abgeschlossen.*

Beweis. Sei $A \in \mathsf{Op} \cdot \mathsf{C}$ mittels einer (k, q)-balancierten Sprache $B \in \mathsf{C}$ und und gelte $A' \leq_{maj} A$ mittels einer FL-Funktion $f : x \mapsto \langle y_1, \ldots, y_{m(|x|)} \rangle$. Da C unter majority-Reduktionen (und damit auch unter $\leq_m^{log}$) abgeschlossen ist, ist nach Satz 89 auch $\mathsf{Op} \cdot \mathsf{C}$ unter $\leq_m^{log}$ abgeschlossen. Daher können wir annehmen, dass alle Strings y_i in der Liste $f(x)$ die gleiche Länge $r(|x|)$ für ein Polynom r haben. Betrachte die $(k, m(n)q(r(n)))$-balancierte Sprache

$$B' = \left\{ x\#z_1 \ldots z_m \;\middle|\; \begin{array}{l} z_1, \ldots, z_m \in \Gamma^{q(r(n))}, f(x) = y_1\# \ldots \#y_m \\ \text{und } \|\{i \,|\, y_i\#z_i \in B\}\| \geq m/2 \end{array} \right\}$$

Da B' auf B majority-reduzierbar ist, folgt $B' \in \mathsf{C}$. Nun folgt im Fall $\mathsf{Op} \in \{\exists^p, \forall^p\}$ sofort $A' \in \mathsf{Op} \cdot \mathsf{C}$, da $A' = \mathsf{Op}B'$ ist.

Da C unter majority-Reduktionen abgeschlossen ist, können wir im Fall $\mathsf{Op} = \mathsf{BP}$ zusätzlich annehmen, dass

$$\Pr_{z \in_R \Gamma^{q(r(n))}}[B(y_i\#z) \neq A(y_i)] \leq 2^{-m(n)-1}$$

ist. Daher gilt für alle x,

$$\Pr_{z \in_R \Gamma^{m(n)q(r(n))}}[B'(x\#z) \neq A'(x)] \leq m(n)2^{-m(n)-1} < \frac{1}{3},$$

und somit ist B' zweiseitig und $A' = \exists^{\geq 1/2} B' \in \mathsf{BP} \cdot \mathsf{C}$. $\blacksquare$

Nun folgt auch leicht der Abschluss von $\mathsf{BP}\cdot\mathsf{C}$ unter dem BP-Operator, falls C unter majority-Reduktionen abgeschlossen ist.

Satz 92. *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 $A \in \mathsf{BP}\cdot\mathsf{BP}\cdot\mathsf{C}$. Da mit C auch $\mathsf{BP}\cdot\mathsf{C}$ unter majority-Reduktionen abgeschlossen ist, existieren eine (k_1, q_1)-balancierte Sprache $B \in \mathsf{BP}\cdot\mathsf{C}$, so dass für alle x, $|x| = n$, gilt

$$\Pr_{y\in_R \Gamma_{k_1}^{q_1(n)}}[B(x\#y) \neq A(x)] \leq 1/6.$$

Zudem existieren eine (k_2, q_2)-balancierte Sprache $B' \in \mathsf{C}$, so dass

$$\Pr_{z\in_R \Gamma_{k_2}^{q_2(n+1+q_1(n))}}[B(x\#y) \neq B'(x\#y\#z)] \leq 1/6$$

für alle x und $y \in \Gamma_{k_1}^{q_1(n)}$ gilt. Sei q das Polynom $q(n) = q_1(n) + q_2(q_1(n) + n + 1)$ und $k = \mathrm{kgV}(k_1, k_2)$. Mit B' ist auch die (k, q)-balancierte Sprache

$$B'' = \{x\#yz \,|\, y \in \Gamma_k^{q_1(n)}, z \in \Gamma_k^{q_2(n+1+q_1(n))}, x\#\tilde{y}\#\tilde{z} \in B'\}$$

in C, wobei $\tilde{y} = \tilde{y}_1 \ldots \tilde{y}_{q_1(n)}$ aus $y = y_1 \ldots y_{q_1(n)}$ mittels $\tilde{y}_i = y_i \bmod k_1$ und $\tilde{z}$ aus z mittels $\tilde{z}_i = z_i \bmod k_2$ entsteht. Wegen

$$\Pr_{u\in_R \Gamma_k^{q(n)}}[A(x) \neq B''(x\#u)] \leq 1/6 + 1/6 = 1/3$$

folgt dann $A \in \mathsf{BP}\cdot\mathsf{C}$. ∎

Korollar 93. *Sei* C *eine unter majority-Reduktionen abgeschlossene Sprachklasse. Dann sind die Klassen* $\mathsf{BP}\cdot\exists^p\cdot\mathsf{C}$ *und* $\mathsf{BP}\cdot\forall^p\cdot\mathsf{C}$ *unter dem* BP*-Operator abgeschlossen.*

Insbesondere sind also die Klassen BPP, $\mathsf{BP}\cdot\mathsf{NP}$ und $\mathsf{BP}\cdot\mathsf{co\text{-}NP}$ unter dem BP-Operator abgeschlossen. Analog folgt für jede Sprachklasse C, die unter disjunktiven Reduktionen abgeschlossen ist, die Gleichheit $\mathsf{R}\cdot\mathsf{R}\cdot\mathsf{C} = \mathsf{R}\cdot\mathsf{C}$.

7 Die Polynomialzeithierachie

Die Polynomialzeithierachie extrapoliert den Übergang von P zu den beiden Klassen $\exists^p\cdot\mathsf{P} = \mathsf{NP}$ und $\forall^p\cdot\mathsf{P} = \mathsf{co\text{-}NP}$. Sie besteht aus den Stufen Σ_k^p und Π_k^p, $k \geq 0$, welche induktiv wie folgt definiert sind:

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

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 $\Sigma_k^p = \mathsf{co}\text{-}\Pi_k^p$ 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 kollabiert, schon gar nicht auf eine kleine Stufe.

Satz 94. *Für alle* $k \geq 1$ *gilt:* $\Sigma_k^p = \Sigma_{k+1}^p \Leftrightarrow \Sigma_k^p = \Pi_k^p \Leftrightarrow \mathsf{PH} = \Sigma_k^p$.

Beweis. Wir zeigen die drei Implikationen $\Sigma_k^p = \Sigma_{k+1}^p \Rightarrow \Sigma_k^p = \Pi_k^p \Rightarrow \mathsf{PH} = \Sigma_k^p \Rightarrow \Sigma_k^p = \Sigma_{k+1}^p$. Wegen $\Pi_k^p \subseteq \Sigma_{k+1}^p$ impliziert die Gleichheit $\Sigma_k^p = \Sigma_{k+1}^p$ sofort $\Pi_k^p \subseteq \Sigma_k^p$, was mit $\Sigma_k^p = \Pi_k^p$ gleichbedeutend ist. Für die zweite Implikation sei $\Sigma_k^p = \Pi_k^p$ angenommen. Wir zeigen durch Induktion über l, dass dann $\Sigma_{k+l}^p = \Sigma_k^p$ für alle $l \geq 0$ gilt. Der Induktionsanfang $l = 0$ ist klar. Für den Induktionsschritt setzen wir

die Gleichheit $\Sigma^p_{k+l} = \Sigma^p_k$ (bzw. $\Pi^p_{k+l} = \Pi^p_k$) voraus und folgern

$$\Sigma^p_{k+l+1} = \exists^p \cdot \Pi^p_{k+l} = \exists^p \cdot \Pi^p_k = \exists^p \cdot \Sigma^p_k = \Sigma^p_k.$$

Die Implikation $\mathsf{PH} = \Sigma^p_k \Rightarrow \Sigma^p_k = \Sigma^p_{k+1}$ 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. In den Übungen werden wir sehen, dass unter der Voraussetzung $\mathsf{PH} \neq \Sigma^p_2$ keine NP-vollständige Sprache in PSK enthalten ist. Allgemeiner liefert die Polynomialzeithierarchie eine Folge von stärker werdenden Hypothesen der Form $\mathsf{PH} \neq \Sigma^p_k$ für $k = 0, 1, 2, \ldots$, die mit $\Sigma^p_0 \subsetneq \Sigma^p_1 \subsetneq \cdots \subsetneq \Sigma^p_k \subsetneq \Sigma^p_{k+1}$, also für $k = 0$ mit $\mathsf{P} \neq \mathsf{NP}$ und für $k = 1$ mit $\mathsf{NP} \neq \text{co-NP}$ äquivalent sind.

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

Satz 95 (Lautemann 1983, Sipser 1983)**.** *Für jede Klasse* C*, die unter majority-Reduktionen abgeschlossen ist, gilt*

$$\mathsf{BP} \cdot \mathsf{C} \subseteq \mathsf{R} \cdot \forall^p \cdot \mathsf{C}.$$

Beweis. Sei $A \in \mathsf{BP} \cdot \mathsf{C}$. Dann existiert eine (k, p)-balancierte Sprache $B \in \mathsf{C}$, so dass für alle x, $|x| = n$, gilt:

$$\Pr_{y \in_R \Gamma^{p(n)}}[A(x) \neq B(x\#y)] \leq k^{-n}$$

Setzen wir $B_x = \{y \in \Gamma^{p(n)} \,|\, x\#y \in B\}$, so folgt $\#B(x) = \|B_x\|$ und

$$x \in A \Rightarrow \#B(x) \geq (1 - k^{-n})k^{p(n)}$$
$$x \notin A \Rightarrow \#B(x) \leq k^{p(n)-n}$$

Wir können o.B.d.A. $p(n) \geq 1$ und $k \geq 2$ annehmen. Sei $\oplus$ die Addition modulo k auf Γ, d.h. $i \oplus j = (i + j) \bmod k$, und für zwei Strings $y, z \in \Gamma^*$ derselben Länge $|y| = |z| = l$ sei

$$y_1 \cdots y_l \oplus z_1 \cdots z_l = v_1 \cdots v_l \in \Gamma^l \text{ mit } v_i = y_i \oplus z_i \text{ für } i = 1, \ldots, l$$

Für $z \in \Gamma^{p(n)}$ sei $B_x \oplus z = \{y \oplus z \mid y \in B_x\}$. Betrachte die Sprachen

$B' = \{x\#u_1 \ldots u_{p(n)}\#z \,|\, u_1, \ldots, u_{p(n)} \in \Gamma^{p(n)}, \exists i : u_i \in B_x \oplus z\}$ und

$B'' = \{x\#u_1 \ldots u_{p(n)} \,|\, u_1, \ldots, u_{p(n)} \in \Gamma^{p(n)}, \forall z \in \Gamma^{p(n)} \exists i : u_i \in B_x \oplus z\}$.

Wir zeigen für alle x mit $|x| = n \geq 2$ und $k^n > p(n)$ die Implikationen

$$x \in A \Rightarrow \#B''(x) \geq k^{p(n)^2}/2$$
$$x \notin A \Rightarrow \#B''(x) = 0$$

Dies beweist $A = \exists^{\geq 1/2} B'' \in \mathsf{R} \cdot \forall^p \cdot \mathsf{C}$, da dann die (k, p^2)-balancierte Sprache B'' einseitig und wegen

$$x\#u \in B'' \Leftrightarrow \forall z \in \Gamma^{p(n)} : x\#u\#z \in B'$$

in $\forall^p \cdot \mathsf{C}$ ist (wegen $B' \leq_d B$ folgt $B' \leq_{maj} B$, also $B' \in \mathsf{C}$).

Sei also $x \in A$ mit $|x| = n \geq 2$ und sei $z \in \Gamma^{p(n)}$ beliebig. Da $\#B(x) \geq (1 - k^{-n})k^{p(n)}$ ist, ist $x\#u_1 \ldots u_{p(n)}\#z$ für zufällig gewählte $u_1, \ldots, u_{p(n)} \in_R \Gamma^{p(n)}$ mit Wahrscheinlichkeit

$$\Pr[x\#u_1 \ldots u_{p(n)}\#z \notin B'] = \Pr[\forall i : u_i \notin B_x \oplus z] \leq k^{-np(n)}$$

nicht in B' enthalten. Daher gilt für ein zufällig gewähltes $u \in_R \Gamma^{p(n)^2}$,

$$\begin{aligned}\Pr[\#B'(x\#u) < k^{p(n)}] &= \Pr[\exists z \in \Gamma^{p(n)} : x\#u_1 \ldots u_{p(n)}\#z \notin B'] \\ &\leq k^{p(n)-np(n)} \leq 1/2\end{aligned}$$

und somit $\#B''(x) \geq k^{p(n)^2}/2$.

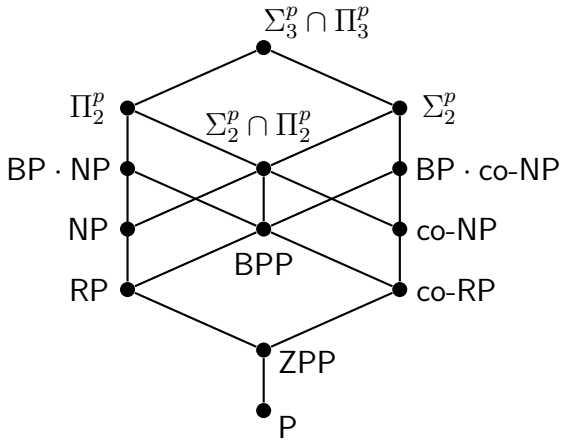
Für den Nachweis der 2. Implikation nehmen wir an, dass $\#B''(x) > 0$ für ein x der Länge n mit $k^n > p(n)$ ist. Dann existiert eine Folge von Wörtern $u_1, \ldots, u_{p(n)} \in \Gamma^{p(n)}$, so dass jedes $z \in \Gamma^{p(n)}$ in mindestens einer der Mengen $u_i \ominus B_x$ enthalten ist (wegen $u_i \in B_x \oplus z \Leftrightarrow z \in u_i \ominus B_x$, wobei $u_i \ominus B_x = \{u_i \ominus y \mid y \in B_x\}$ und der Operator $\ominus$ analog zu $\oplus$ definiert ist). Da die Mengen $u_i \ominus B_x$ die gleiche Größe wie B_x haben, folgt

$$p(n) \cdot \#B(x) \geq k^{p(n)},$$

also $\#B(x) \geq k^{p(n)}/p(n) > k^{p(n)-n}$. Daher muss x zu A gehören. ■

Insbesondere liefert Satz 95 folgende Inklusionen, indem wir $\mathsf{C} = \mathsf{P}$ bzw. $\mathsf{C} = \text{co-NP}$ setzen (beachte, dass $\mathsf{R} \cdot \forall^p \cdot \text{co-NP} = \mathsf{R} \cdot \text{co-NP} \subseteq \mathsf{BP} \cdot \text{co-NP}$ gilt).

Korollar 96. $\mathsf{BPP} \subseteq \Sigma_2^p \cap \Pi_2^p$ *und* $\mathsf{BP} \cdot \text{co-NP} = \mathsf{R} \cdot \text{co-NP} \subseteq \Sigma_2^p$.



Zum Schluss des Kapitels fassen wir bekannte Kollapskonsequenzen unter verschiedenen Annahmen über die Zugehörigkeit eines NP-harten Entscheidungsproblems A zu bestimmten Komplexitätsklassen zusammen.

Annahme	Konsequenz
$A \in \mathsf{P}$	$\mathsf{PH} = \mathsf{P}$ (Satz 94)
$A \in \text{co-NP}$	$\mathsf{PH} = \mathsf{NP}$ (Satz 94)
$A \in \mathsf{BPP}$	$\mathsf{PH} = \Sigma_2^p = \mathsf{BPP}$ (siehe Übungen)
$A \in \mathsf{PSK}$	$\mathsf{PH} = \Sigma_2^p$ (siehe Übungen)

8 Turing-Operatoren

In diesem Kapitel 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.

Definition 97. *Eine **Orakel-Turingmaschine (OTM)** M ist eine TM, die zusätzlich ein **Orakelalphabet** sowie ein spezielles write-only **Orakelfrageband** besitzt. Außerdem hat M drei spezielle Zustände $q_?, q_+, q_-$. Als Orakel kann eine beliebige Sprache A über dem Orakelalphabet 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 OTM 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 zusätzlich 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 98. *Die **Rechenzeit** einer OTM M mit Orakelalphabet Γ bei einer Eingabe $x \in \Sigma^*$ ist*

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

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

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

Wir fassen alle Sprachen, die von einer (nicht-)deterministischen OTM (kurz DOTM bzw. NOTM) M mit Orakel A in Zeit $t(n)$ entscheidbar sind, in den relativierten Komplexitätsklassen

$$\mathsf{DTIME}^A(t(n)) = \{L(M^A) \,|\, M \text{ ist eine } t(n)\text{-zeitbeschränkte DOTM}\}$$

und

$$\mathsf{NTIME}^A(t(n)) = \{L(M^A) \,|\, M \text{ ist eine } t(n)\text{-zeitbeschränkte NOTM}\}$$

zusammen. Die Klassen $\mathsf{DTIME}^A(t(n))$ und $\mathsf{NTIME}^A(t(n))$ werden auch als **Relativierungen** von $\mathsf{DTIME}(t(n))$ und $\mathsf{NTIME}(t(n))$ zum Orakel A bezeichnet. Beispielsweise enthält die relativierte Klasse P^A alle Sprachen, die von einer polynomiell zeitbeschränkten DOTM (kurz P-OTM) M mit Orakel A entschieden werden,

$$\mathsf{P}^A = \{L(M^A) \,|\, M \text{ ist eine P-OTM}\}$$

Entsprechend erhalten wir die Klasse NP^A. Ebenso wie DTMs und NTMs lassen sich auch PTMs (also probabilistische TMs) oder Transducer mit einem Orakelmechanismus ausstatten, wodurch wir POTMs oder Orakeltransducer erhalten. Ist die Rechenzeit dieser Maschinen polynomiell beschränkt, so bezeichnen wir sie als PP-OTMs bzw. FP-OTMs. Entsprechend erhalten wir dann die relativierten Komplexitätsklassen PP^A, FP^A, BPP^A, RP^A, ZPP^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. 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. $\mathsf{P}(A)$). Diese Notationen verwenden wir auch für alle übrigen relativierten Komplexitätsklassen.

Satz 99.

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

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

(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 P^{P}. Dann existiert eine P-OTM M und ein Orakel $A \in \mathsf{P}$ mit $L(M^A) = L$. Sei M' eine P-TM mit $L(M') = A$. Betrachte die DTM M'', die bei Eingabe x die OTM $M(x)$ 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) Wir zeigen zuerst die Inklusion $\mathsf{NP}^{\mathsf{NP} \cap \text{co-NP}} \subseteq \mathsf{NP}$. Sei $L = L(M^A)$ für eine NP-OTM M und sei A ein Orakel in $\mathsf{NP} \cap \text{co-NP}$. Dann existieren NP-TMs M' und M'' mit $L(M') = A$ und $L(M'') = \overline{A}$. Betrachte folgende NP-TM 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.

Nun gilt $L(M^*) = L(M^A) = L$ und daher ist $L \in \mathsf{NP}$. Dies zeigt $\mathsf{NP}^{\mathsf{NP}\cap\text{co-}\mathsf{NP}} \subseteq \mathsf{NP}$. Da $\mathsf{P}^{\mathsf{NP}\cap\text{co-}\mathsf{NP}}$ unter Komplementbildung abgeschlossen ist, folgt auch sofort $\mathsf{P}^{\mathsf{NP}\cap\text{co-}\mathsf{NP}} \subseteq \mathsf{NP} \cap \text{co-}\mathsf{NP}$. Die umgekehrten Inklusionen sind trivial.

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

$$x \in L \Leftrightarrow \exists y \in \Gamma^{p(|x|)} : x\#y \in A$$

Dann ist $\bar{A} \in \mathsf{NP}$ und L wird von der NP-OTM M relativ zum Orakel $\bar{A}$ akzeptiert, die bei Eingabe x ein Wort $y \in \Gamma^{p(|x|)}$ rät und bei negativer Antwort auf die Orakelfrage $x\#y$ akzeptiert. Mit demselben Argument folgt auch Σ_k^p in $\mathsf{NP}^{\Sigma_{k-1}^p}$. Der einzige Unterschied ist, dass nun $A \in \Pi_{k-1}^p$ bzw. $\bar{A} \in \Sigma_{k-1}^p$ und folglich $L = L(M^{\bar{A}}) \in \mathsf{NP}^{\Sigma_{k-1}^p}$ ist.

Für die umgekehrte Richtung sei $L = L(M^A)$ für ein NP-Orakel A und eine NP-OTM M. Sei p eine polynomielle Zeitschranke für M und sei $k = \mathrm{kgV}(1,2,\ldots,c)$, wobei $c \geq 1$ der maximale Verzweigungsgrad von M ist. Zudem existieren ein Polynom q und eine (k,q)-balancierte Sprache $B \in \mathsf{P}$ mit

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

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

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

wobei C alle Strings $x\#rz_1 \ldots z_{p(n)}$ enthält, so dass

- $r \in \Gamma^{p(n)}$ eine akzeptierende Rechnung α von $M(x)$ kodiert, während der m Orakelfragen $y_1, \ldots, y_m$ gestellt und mit $a_1, \ldots, a_m \in \{0,1\}$ beantwortet werden, sowie
- $z_1, \ldots, z_{p(n)} \in \Gamma^{q(p(n))}$ sind 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 $\text{co-}\mathsf{NP}$ liegt, zeigt diese Charakterisierung, dass L in $\exists^p \cdot \text{co-}\mathsf{NP}$ enthalten ist.

Mit diesem Argument lässt sich auch die Inklusion $\mathsf{NP}^{\Sigma_k^p} \subseteq \Sigma_{k+1}^p$ zeigen. Der einzige Unterschied ist, dass nun A in Σ_k^p (bzw. $\bar{A}$ in Π_k^p) und B in Π_{k-1}^p liegen. Daher ist leicht zu sehen, dass C nun in Π_k^p liegt und somit $L = L(M^A) = \exists C$ in $\exists^p \cdot \Pi_k^p = \Sigma_{k+1}^p$ enthalten ist. ■

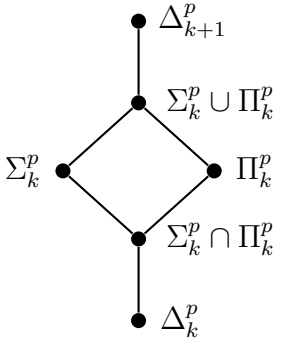
Der vorige Satz liefert folgende Charakterisierung für die k-te Stufe der Polynomialzeithierachie ($k \geq 1$):

$$\Sigma_k^p = \mathsf{NP}^{\Sigma_{k-1}^p} = \mathsf{NP}^{\mathsf{NP}^{\cdot^{\cdot^{\cdot^{\mathsf{NP}}}}}}$$

$$\Pi_k^p = \text{co-}\mathsf{NP}^{\Sigma_{k-1}^p} = \text{co-}\mathsf{NP}^{\mathsf{NP}^{\cdot^{\cdot^{\cdot^{\mathsf{NP}}}}}}$$

wobei die Türme die Höhe k haben. Zudem erhalten wir noch die neuen Stufen

$$\Delta_k^p := \mathsf{P}^{\Sigma_{k-1}^p} = \mathsf{P}^{\mathsf{NP}^{\cdot^{\cdot^{\cdot^{\mathsf{NP}}}}}}$$



9 Das relativierte P/NP-Problem

Satz 100 (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

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

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

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

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

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

$$n_i = \min\{n \in \mathbb{N} \,|\, n \geq (n_{i-1})^{i-1} + i, n^i + i < 2^n\},\ i \geq 1.$$

Die Bedingung $n_i \geq (n_{i-1})^{i-1} + i$ stellt sicher, dass $M_{i-1}(0^{n_{i-1}})$ für kein $j \geq i$ ihr Orakel über ein Wort y der Länge n_j befragen kann, und die Bedingung $(n_i)^i + i < 2^{n_i}$ garantiert, dass M_i bei Eingabe 0^{n_i} nicht alle Wörter der Länge n_i als Orakelfrage stellen kann.

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 in Stufe i so definiert, dass 0^{n_i} in $L(M_i^{B_{i-1}}) \triangle L(B_i)$ enthalten ist. Zudem führt $M_i^{B_i}(0^{n_i})$ die gleiche Rechnung wie $M_i^{B_{i-1}}(0^{n_i})$ aus, da $M_i^{B_{i-1}}(0^{n_i})$ keine Orakelfrage in $B_i \backslash B_{i-1}$ stellt. Da zudem $B \backslash B_i$ nur Wörter der Länge $\geq n_{i+1} > (n_i)^i + i$ enthält, folgt $0^{n_i} \in L(M_i^B) \triangle L(B)$ und somit $L(B) \notin \mathsf{P}^B$. ■

Es gibt sogar relativierte Welten, in denen alle Stufen der Polynomialzeithierarchie verschieden sind.

Satz 101. *Es existiert ein Orakel C, so dass $\Sigma_k^P(C) \neq \Sigma_{k+1}^P(C)$ für alle $k \geq 0$ (und somit $\mathsf{PH}^C \neq \mathsf{PSPACE}^C$) gilt.*

Da die Antwort auf die Frage, ob $\mathsf{P}^A \neq \mathsf{NP}^A$ (oder allgemeiner, ob PH^A echt) ist, von der Wahl des Orakels A abhängt, lassen sich diese Fragen nicht mit relativierbaren Beweismethoden beantworten. Andererseits wurden alle bisher bekannten Separierungen zwischen Komplexitätsklassen mit relativierbaren Beweistechniken erzielt. Beispiele hierfür sind die Zeit- und Platzhierarchiesätze

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

falls $g(n) \cdot \log g(n) = o(f(n))$, und

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

falls $g(n) = o(f(n))$. Auch die Inklusionen

$$\mathsf{DTIME}^A(f) \subseteq \mathsf{NTIME}^A(f) \subseteq \mathsf{DSPACE}^A(f) \subseteq \mathsf{NSPACE}^A(f),$$

gelten relativ zu einem beliebigen Orakel. Dagegen sind die Inklusion

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

und der Satz von Savitch

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

sowie der Satz von Immerman/Szelepczényi

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

nicht relativierbar (siehe Übungen).

Im Jahr 1981 zeigten Bennet und Gill, dass bei zufälliger Wahl des Orakels A (d.h. A enthält jedes Wort $x \in \{0,1\}^*$ mit Wahrscheinlichkeit $1/2$) die Separierungen

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

sogar mit Wahrscheinlichkeit 1 gelten, 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. Andererseits gilt

$$\Pr[\mathsf{P}^A = \mathsf{BPP}^A] = 1.$$

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 $\mathsf{IP} = \mathsf{PSPACE}$, obwohl $\mathsf{IP}^A \neq \mathsf{PSPACE}^A$ 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 102. *Für eine NTM M bezeichne $\#M(x)$ die Anzahl der akzeptierenden Rechnungen von $M(x)$.*

a) $\#\mathsf{P} = \{\#M \mid M \text{ ist eine NP-TM }\}$.

b) $L \subseteq \Sigma^*$ *gehört zu* $\oplus\mathsf{P}$, *falls eine NP-TM M existiert mit*

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

c) $L \subseteq \Sigma^*$ *gehört zu* UP, *falls die charakteristische Funktion $L(x)$ von L in $\#\mathsf{P}$ ist, d.h. es gibt eine NP-TM M mit*

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

Unter Verwendung von NP-OTMs erhalten wir die relativierten Klassen $\#\mathsf{P}^A$, $\oplus\mathsf{P}^A$ 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?

Folgende Proposition wird ebenfalls in den Übungen bewiesen.

Proposition 103.

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

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

10.1 Satz von Valiant und Vazirani

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

Definition 104. *Ein **Promise-Problem** ist ein Paar (A, B) von Sprachen $A, B \subseteq \Sigma^*$, wobei A das **Promise-Prädikat** genannt wird. Eine Sprache L heißt **Lösung** für (A, B), falls für alle Eingaben $x \in A$ gilt:*

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

Eine Lösung für (A, B) muss also zumindest alle Eingaben in $A \cap B$ und kann darüber hinaus noch beliebige Eingaben in $\bar{A}$ enthalten, d.h. L ist genau dann eine Lösung, wenn $A \cap B \subseteq L \subseteq (A \cap B) \cup \bar{A}$ gilt. Im Fall $A = \Sigma^*$ gibt es also nur eine Lösung $L = B$.

Beispiel 105. *Sei* USAT *die Menge aller booleschen Formeln, die genau eine erfüllende Belegung haben, und 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* $F \in 1\text{SAT}$ *richtig zu entscheiden, ob F erfüllbar ist oder nicht. Somit ist jede Sprache L mit* $\text{USAT} \subseteq L \subseteq \text{SAT}$ *eine Lösung für* $(1\text{SAT}, \text{SAT})$. *Neben* USAT *und* SAT *ist z.B. auch* $\oplus$SAT *eine Lösung.* ◁

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

Definition 106. *Eine Sprache A heißt **randomisiert reduzierbar** auf eine Sprache B, falls es eine Funktion $f \in \mathsf{FL}$ und Polynome p, q gibt, so dass für alle Eingaben x gilt:*

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

Für die randomisierte Reduktion von SAT auf das Promise-Problem $(1\text{SAT}, \text{SAT})$ benutzen wir lineare Hashfunktionen.

Definition 107. *$Lin(n, k)$ bezeichne die Menge aller linearen Funktionen von $\mathbb{F}_2^n$ nach $\mathbb{F}_2^k$, wobei $\mathbb{F}_2 = (\{0, 1\}, \oplus, \cdot, 0, 1)$ der zweielementige Körper ist.*

Bemerkung 108. *Jede Funktion $h \in Lin(n, k)$ lässt sich eindeutig durch eine Matrix $A_h = (a_{ij}) \in \{0, 1\}^{k \times n}$ beschreiben, d.h. es gilt*

$$h(x_1 \cdots x_n) = y_1 \cdots y_k \Leftrightarrow \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{k1} & \cdots & a_{kn} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ y_k \end{pmatrix}.$$

Bezeichnen wir also die Abbildung $x_1 \cdots x_n \mapsto a_{i1}x_1 \oplus \cdots \oplus a_{in}x_n$ mit h_i, so gilt $y_i = h_i(x_1 \cdots x_n)$ für $i = 1, \ldots, k$.

Lemma 109. *Für $x, x' \in \{0, 1\}^n \setminus \{0^n\}$ und $y, y' \in \{0, 1\}^k$ gilt im Fall $x \neq x'$ für eine zufällig unter Gleichverteilung gewählte Funktion $h \in_R Lin(n, k)$,*

$$\Pr[h(x) = y] = 2^{-k} \text{ und } \Pr[h(x) = y, h(x') = y'] = 2^{-2k}.$$

Beweis. Wir zeigen zuerst, dass $h(x)$ im Fall $x \neq 0^n$ auf dem Wertebereich $\{0, 1\}^k$ gleichverteilt ist, falls h zufällig aus $Lin(n, k)$ gewählt wird. In diesem Fall existiert nämlich ein Index j mit $x_j = 1$. Da sich der Wert von $h_i(x)$ ändert, falls wir das Bit a_{ij} in A_h flippen, haben die beiden Mengen $H_0 = \{h \,|\, h_i(x) = 0\}$ und $H_1 = \{h \,|\, h_i(x) = 1\}$ die gleiche Mächtigkeit, was

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

impliziert. Da die einzelnen Zeilen von A_h unabhängig gewählt werden, sind auch die Bitwerte $h_1(x), \ldots, h_k(x)$ unabhängig, und es folgt für

jedes $y = y_1 \cdots y_k \in \{0,1\}^k$,

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

Als nächstes zeigen wir, dass im Fall $x \neq x'$ die beiden Werte $h(x)$ und $h(x')$ stochastisch unabhängig sind. Sei j eine Position mit $x'_j = 0$ und $x_j = 1$ (falls nötig, vertauschen wir x und x'). Dann ändert sich durch Flippen von a_{ij} zwar der Wert von $h_i(x)$, aber $h_i(x')$ bleibt unverändert. Folglich sind die beiden Mengen $H'_0 = \{h \,|\, h_i(x') = y_i \wedge h_i(x) = 0\}$ und $H'_1 = \{h \,|\, h_i(x') = y_i \wedge h_i(x) = 1\}$ gleich groß, woraus

$$\Pr[h_i(x) = 0 \,|\, h_i(x') = y_i] = 1/2$$

folgt. Daher ist

$$\Pr[h_i(x) = y_i \wedge h_i(x') = y'_i] = \underbrace{\Pr[h_i(x') = y'_i]}_{1/2} \underbrace{\Pr[h_i(x) = y_i | h_i(x') = y'_i]}_{1/2},$$

was für beliebige Werte $y, y' \in \{0,1\}^k$ die Gleichheit

$$\Pr[h(x) = y \wedge h(x') = y'] = \prod_{i=1}^{k} \underbrace{\Pr[h_i(x) = y_i \wedge h_i(x') = y'_i]}_{1/4} = 2^{-2k}$$

impliziert. ■

Lemma 110. *Für $x \in \{0,1\}^n$ und für eine zufällig unter Gleichverteilung gewählte Funktion $h \in_R Lin(n,k)$ sei Z_x die ZV*

$$Z_x = \begin{cases} 1, & h(x) = 1^k, \\ 0, & \text{sonst} \end{cases}$$

und für $B \subseteq \{0,1\}^n$ sei S_B die ZV $S_B = \sum_{x \in B} Z_x$. Dann gilt im Fall $\emptyset \neq B \subseteq \{0,1\}^n - \{0^n\}$ und $k = \lfloor \log_2(3\|B\|) \rfloor$ die Abschätzung $\Pr[S_B = 1] \geq 2/9$.

Beweis. Nach Lemma 109 sind die ZVen Z_x, $x \in \{0,1\}^n$, paarweise unabhängig und wegen $0^n \notin B$ gilt $\Pr[Z_x = 1] = 2^{-k}$ für alle $x \in B$. Setzen wir $b = \|B\|$, so folgt

$$\Pr[S_B \geq 2] \leq \sum_{\{x,x'\} \in \binom{B}{2}} \underbrace{\Pr[h(x) = h(x') = 1^k]}_{2^{-2k}} = \binom{b}{2} \cdot 2^{-2k}$$

und

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

Somit gilt

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

Da $k = \lfloor \log_2(3b) \rfloor$ im Intervall $(\log_2(3b) - 1, \log_2(3b)]$ liegt, muss $2^{-k} b$ im Intervall $[1/3, \frac{2}{3})$ liegen. Da aber die Funktion $f(x) = x(1-x)$ auf diesem Intervall nach unten durch $2/9$ beschränkt ist, folgt $\Pr[S_B = 1] \geq 2/9$. ■

Satz 111 (Valiant, Vazirani 1986)**.** SAT *ist auf jede Lösung 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 eine Zahl $k \in \{1, \cdots, n+1\}$ und eine lineare Hashfunktion $h \in Lin(n, n+1)$ mit Matrizendarstellung $A_h = (a_{ij}) \in \{0,1\}^{(n+1) \times n}$ sei $F_{k,h}$ die boolesche Formel

$$F_{k,h}(x_1, \ldots, x_n) = F(x_1, \ldots, x_n) \wedge \bigwedge_{i=1}^{k} \bigoplus_{a_{ij}=1} x_j,$$

die unter einer Belegung $a \in \{0,1\}^n$ genau dann wahr wird, wenn $F(a) = 1$ ist und die ersten k Bits von $h(a)$ den Wert 1 haben. Betrachte die Reduktionsfunktion

$$f : F\#y \mapsto F_{k_y,h_y},$$

wobei $y \in \{0,1\}^{m+n(n+1)}$ für $m = \lceil \log_2(n+1) \rceil$ die Länge $m+n(n+1)$ hat und die ersten m Bits von $y = y_1 \dots y_m$ eine Zahl $k_y = 1 + \Sigma_{i=1}^{m} y_i 2^{i-1} \in \{1, \dots, 2^m\}$ und die restlichen $n(n+1)$ Bits von y eine Matrix $A_{h_y} \in \{0,1\}^{(n+1)\times n}$ repräsentieren.

Sei nun L eine Lösung von $(1\text{SAT}, \text{SAT})$ und sei b die Anzahl der erfüllenden Belegungen von F. Im Fall $F \in \text{SAT}$ ist dann $b \in \{1, \cdots, 2^n - 1\}$ und somit $k = \lfloor \log_2(3b) \rfloor \in \{1, \cdots, n+1\}$. Wegen $\text{USAT} \subseteq L$ gilt daher für $y \in_R \{0,1\}^{m+n(n+1)}$,

$$\begin{aligned} \Pr[f(F\#y) \in L] &\geq \Pr[f(F\#y) \in \text{USAT}] \\ &\geq \underbrace{\Pr[k_y = k]}_{= 2^{-m} > 1/2(n+1)} \underbrace{\Pr[F_{k_y,h_y} \in \text{USAT} \,|\, k_y = k]}_{\geq 2/9 \text{ (nach Lemma 109)}} \\ &> 1/9(n+1) \end{aligned}$$

Andererseits ist die Formel $f(F\#y)$ im Fall $F \notin \text{SAT}$ für kein y erfüllbar. Daher folgt wegen $L \subseteq \text{SAT}$

$$\Pr[f(F\#y) \in L] \leq \Pr[f(F\#y) \in \text{SAT}] = 0 \qquad \blacksquare$$

Korollar 112.

(i) SAT *ist auf* USAT *und* $\oplus$SAT *randomisiert reduzierbar.*

(ii) Falls das Promise-Problem $(1\text{SAT}, \text{SAT})$ *eine Lösung in* BPP *hat, folgt* NP $\subseteq$ BPP *(was wiederum* PH $=$ RP *impliziert; siehe Übungen).*

Beweis. (i) Klar, da USAT und $\oplus$SAT Lösungen des Promise-Problems $(1\text{SAT}, \text{SAT})$ sind.

(ii) Sei $A \in$ BPP eine Lösung von $(1\text{SAT}, \text{SAT})$ und $f \in$ FL eine randomisierte Reduktion von SAT auf A. Da BPP unter disjunktiven Reduktionen abgeschlossen ist, können wir annehmen, dass

$$\Pr_{z \in_R \{0,1\}^{p(n)}}[f(x\#z) \in A] \geq 5/6$$

für ein Polynom p und alle $x \in \text{SAT}$ gilt. Weiter sei M eine BPP-TM mit

$$\Pr[M(y) \neq A(y)] \leq 1/6.$$

Dann gilt $\Pr[M'(x) \neq \text{SAT}(x)] \leq 1/3$, wobei die BPP-TM $M'(x)$ zufällig ein $z \in_R \{0,1\}^{p(n)}$ wählt und $M(f(x\#z))$ simuliert. $\blacksquare$

Um aus der randomisierten Reduktion von SAT auf $\oplus$SAT weitere Konsequenzen ziehen zu können, benötigen wir noch bestimmte Abschlusseigenschaften von $\oplus$P.

Satz 113. *Für jede Klasse* C*, die unter konjunktiven Reduktionen abgeschlossen ist, gilt*

$$\oplus \mathsf{P}^{\oplus \cdot \mathsf{C}} = \oplus \cdot \mathsf{C}.$$

Beweis. Sei $L \in \oplus \mathsf{P}^{\oplus \cdot \mathsf{C}}$ via einer NP-OTM M und einem Orakel $A \in \oplus \cdot \mathsf{C}$. Da mit C auch $\oplus \cdot \mathsf{C}$ unter $\leq_m^{log}$ abgeschlossen ist, können wir annehmen, dass $M(x)$ für ein Polynom q auf jedem Pfad genau $q(n)$ Orakelfragen der Länge $q(n)$ stellt. Weiter sei $B \subseteq \Sigma^*$ eine (k,p)-balancierte Sprache in C mit $A = \oplus B$ und seien B_0 und B_1 $(p+1)$-balancierte Sprachen mit $\#B_1(y) \equiv_2 A(y)$ und $\#B_0(y) \equiv_2 \bar{A}(y)$, also z.B.

$$B_1 = \{y\#1z \,|\, y\#z \in B\} \text{ und } B_0 = B_1 \cup \{y\#0^{p(|y|)+1} \,|\, y \in \Sigma^*\}.$$

Betrachte nun die balancierte Sprache

$$B' = \left\{ x\#\alpha z_1 \dots z_{q(n)} \;\middle|\; \begin{array}{l} \alpha \text{ kodiert eine akz. Rechnung von } M(x) \\ \text{mit Orakelfragen } y_1, \dots, y_{q(n)} \text{ und für} \\ i = 1, \dots, q(n) \text{ gilt } y_i\#z_i \in B_{a_i}, \text{ wobei } a_i \\ \text{die Antwort auf } y_i \text{ ist } (1 \mathrel{\widehat{=}} \text{ja}, 0 \mathrel{\widehat{=}} \text{nein}) \end{array} \right\}.$$

Dann ist leicht zu sehen, dass B' konjunktiv auf B reduzierbar und somit in C ist. Zudem gilt $L = \oplus B'$, da

$$\#B'(x) = \sum_{\alpha \text{ akz. Rechnung von } M(x)} \|\{z_1 \dots z_{q(n)} \,|\, x\#\alpha z_1 \dots z_{q(n)} \in B'\}\|$$

ist und für jede akzeptierende Rechnung α von $M(x)$ mit Orakelfragen $y_1, \dots, y_{q(n)}$ und zugehörigen Antworten $a_1, \dots, a_{q(n)}$ die Anzahl

$$\|\{z_1 \dots z_{q(n)} \,|\, x\#\alpha z_1 \dots z_{q(n)} \in B'\}\| = \prod_{i=1}^{q(n)} \#B_{a_i}(y_i)$$

genau dann ungerade ist, wenn alle Antworten korrekt sind. ■

Korollar 114. *Für jede Klasse* C*, die unter konjunktiven Reduktionen abgeschlossen ist, ist* $\oplus \cdot \mathsf{C}$ *unter majority-Reduktionen abgeschlossen und es gilt* $\exists^p \cdot \mathsf{C} \subseteq \mathsf{R} \cdot \oplus \cdot \mathsf{C} \subseteq \mathsf{BP} \cdot \oplus \cdot \mathsf{C}$.

Beweis. Der Abschluss von $\oplus \cdot \mathsf{C}$ unter majority-Reduktionen folgt direkt aus Satz 113. Für die Inklusionen $\exists^p \cdot \mathsf{C} \subseteq \mathsf{R} \cdot \oplus \cdot \mathsf{C} \subseteq \mathsf{BP} \cdot \oplus \cdot \mathsf{C}$ reicht es daher zu zeigen, dass es zu jeder balancierten Sprache B eine balancierte Sprache $B' \leq_m^{log} B$ gibt, so dass $\exists B$ auf $\oplus B'$ randomisiert reduzierbar ist. Wir können O.B.d.A. annehmen, dass B $(2, q)$-balanciert für ein Polynom q ist. Sei B' die Sprache

$$\{x\#z\#y \,|\, z = bin_m(k)A_h \in \{0,1\}^{m+q(n)(q(n)+1)}, 0 \leq k \leq q(n),$$
$$1^{k+1} \text{ ist Präfix von } h(y) \text{ und } x\#y \in B\},$$

wobei $m = \lceil \log_2(q(n)+1) \rceil$ und $h \in Lin(q(n)+1, q(n))$ ist. Dann gilt $B' \leq_m^{log} B$ und für alle $x \in \exists B$ ist

$$\Pr_{z \in_R \{0,1\}^{m+q(n)(q(n)+1)}}[\#B'(x\#z) = 1] > 1/9(q(n)+1),$$

d.h. $\exists B$ ist mittels $f : x\#z \mapsto x\#z$ randomisiert auf jede Lösung L des Promise-Problems $(\{x\#z \,|\, \#B'(x\#z) \leq 1\}, \{x\#z \,|\, \#B'(x\#z) \geq 1\})$ (also insbesondere auf $L = \oplus B'$) reduzierbar. ■

10.2 Satz von Toda

In diesem Abschnitt beweisen wir den Satz von Toda. Er besagt, dass $\mathsf{PH} \subseteq \mathsf{BP} \cdot \oplus\mathsf{P} \subseteq \mathsf{PP}^{\oplus\mathsf{P}} \subseteq \mathsf{P}^{\mathsf{PP}}$ gilt. Für die erste Inklusion benötigen wir noch folgendes Lemma.

Lemma 115. *Für* $\mathsf{Op} \in \{\exists^p, \forall^p, \exists^{\geq 1/2}, \oplus\}$ *und jede unter majority-Reduktionen abgeschlossene Klasse* C *gilt* $\mathsf{Op} \cdot \mathsf{BP} \cdot \mathsf{C} \subseteq \mathsf{BP} \cdot \mathsf{Op} \cdot \mathsf{C}$.

Beweis. Zu $L \in \mathsf{Op} \cdot \mathsf{BP} \cdot \mathsf{C}$ existiert eine (k, q)-balancierte Sprache $A \in \mathsf{BP} \cdot \mathsf{C}$ mit $L = \mathsf{Op}A$. Zu A existiert eine (k', p)-balancierte Sprache $B \in \mathsf{C}$ mit

$$\Pr_{z \in_R \Gamma_{k'}^{p(n)}}[A(x\#y) \neq B(x\#y\#z)] \leq 1/3k^{q(n)}$$

Da auch die Sprache $B' = \{x\#z\#y \,|\, x\#y\#z \in B\}$ in C ist, folgt wegen

$$\begin{aligned}\Pr_z[L(x) \neq \mathsf{Op}B'(x\#z)] &\leq \Pr_z[\#A(x) \neq \#B'(x\#z)] \\ &\leq \Pr_z[\exists y \in \Gamma_k^{q(n)} : A(x\#y) \neq B'(x\#z\#y)] \\ &\leq k^{q(n)}/3k^{q(n)} = 1/3,\end{aligned}$$

dass $\mathsf{Op}B'$ zweiseitig und somit $L = \exists^{\geq 1/2}\mathsf{Op}B' \in \mathsf{BP} \cdot \mathsf{Op} \cdot \mathsf{C}$ ist. ■

Satz 116 (Toda, 1992)**.** $\mathsf{PH} \subseteq \mathsf{BP} \cdot \oplus\mathsf{P}$.

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

$k = 0$**:** klar.

$k \rightsquigarrow k+1$**:** Mit $\exists^p \cdot \mathsf{BP} \cdot \oplus\mathsf{P} \subseteq \mathsf{BP} \cdot \exists^p \cdot \oplus\mathsf{P}$ (Lemma 115), $\exists^p \cdot \oplus\mathsf{P} \subseteq \mathsf{BP} \cdot \oplus\mathsf{P}$ (Kor. 112 und 114) und $\mathsf{BP} \cdot \mathsf{BP} \cdot \oplus\mathsf{P} = \mathsf{BP} \cdot \oplus\mathsf{P}$ (Korollar 92) folgt

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

■

Zum Beweis der Inklusion $\mathsf{PP}^{\oplus\mathsf{P}} \subseteq \mathsf{P}^{\mathsf{PP}}$ benötigen wir eine Reihe von grundlegenden Abschlusseigenschaften der Funktionenklasse $\#\mathsf{P}$.

Lemma 117. *Seien $f, g \in \#\mathsf{P}$, $t \in \mathsf{FP}$, $B \in \mathsf{P}$ eine (k,q)-balancierte Sprache und p ein Polynom. Dann sind folgende Funktionen in $\#\mathsf{P}$:*

(i) $f + g$
(ii) $f \cdot g$
(iii) $x \mapsto \sum_{y, x\#y \in B} f(x\#y)$
(iv) $x \mapsto f(x)^{p(n)}$
(v) $x \mapsto f(t(x))$

Beweis. Zu f, g existieren (k,q)-balancierte Sprachen $A, B \in \mathsf{P}$ mit $f(x) = \#A(x)$ und $g(x) = \#B(x)$. Dann ist $f(x) + g(x) = \#D(x)$ für die $(q+1)$-balancierte Sprache

$$D = \{x\#0y \,|\, x\#y \in A\} \cup \{x\#1y \,|\, x\#y \in B\}$$

Weiter ist $f(x)g(x) = \#E(x)$ für die $(k, 2q)$-balancierte Sprache

$$E = \{x\#yz \,|\, x\#y \in A \wedge x\#z \in B\}$$

Um zu zeigen, dass $h(x) = \sum_{y, x\#y \in B} f(x\#y)$ in $\#\mathsf{P}$ ist, betrachten wir die $(p+q)$-balancierte Sprache

$$F = \{x\#yz \,|\, x\#y \in B, x\#y\#z \in A\}$$

Die Zugehörigkeit von $h'(x) = f(x)^{p(n)}$ zu $\#\mathsf{P}$ folgt mittels der (k, pq)-balancierten Sprache

$$L = \{x\#y_1 \cdots y_{p(n)} \,|\, x\#y_1, \ldots, x\#y_{p(n)} \in A\} \in \mathsf{P}$$

Schließlich folgt $f'(x) = f(t(x))$ in $\#\mathsf{P}$ mittels der $s(n)$-balancierten Sprache

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

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

Lemma 118. *Für jede Funktion $g \in \#\mathsf{P}$ ist die Sprache $B = \{x\#bin(l) \,|\, g(x) \geq l\}$ in PP.*

Beweis. Sei $B \in \mathsf{P}$ eine (k,q)-balancierte Sprache mit $g(x) = \#B(x)$ und betrachte die $(k, q+1)$-balancierte Sprache

$$B' = \{x\#bin(l)\#ay \,|\, num(ay) < k^{q(n)+1}/2 - l \vee a = k-1 \wedge x\#y \in B\}$$

Dann ist $B' \in \mathsf{P}$ und wegen $\#B'(x\#bin(l)) = k^{q(n)+1}/2 - l + g(x)$ ist $B = \exists^{\geq 1/2} B' \in \mathsf{PP}$. ∎

Weiterhin benötigen wir das Konzept der Modul-verstärkenden Polynome.

Lemma 119. *Für alle $n \geq 0$ und $b \in \{0,1\}$ gilt*

$$n \equiv_2 b \Rightarrow ((n+1)^d + 1)^d \equiv_{2^d} b$$

Beweis. Es gilt

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

und

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

∎

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

Lemma 120. *Für jede Sprache $A \in \oplus\mathsf{P}$ und jedes Polynom p ex. eine Funktion $f \in \#\mathsf{P}$ mit $A(x) \equiv_{2^{p(n)}} f(x)$.*

Beweis. Sei h eine #P-Funktion mit $h(x) \equiv_2 A(x)$. Eine viermalige Anwendung von Lemma 117 zeigt, dass dann auch die Funktion

$$f : x \mapsto ((h(x)+1)^{p(n)}+1)^{p(n)}$$

in #P ist. Mit Lemma 119 folgt $A(x) \equiv_{2^{p(n)}} f(x)$. ■

Satz 121 (Toda, 1992)**.** $\mathsf{PP}^{\oplus\mathsf{P}} \subseteq \mathsf{P}^{\mathsf{PP}}$

Beweis. Sei $L \in \mathsf{PP}^{\oplus\mathsf{P}} = \exists^{\geq 1/2} \cdot \mathsf{P}^{\oplus\mathsf{P}} = \exists^{\geq 1/2} \cdot \oplus\mathsf{P}$ und sei A eine (k,q)-balancierte Sprache in $\oplus\mathsf{P}$ mit

$$x \in L \Leftrightarrow \|\{y \in \Gamma^{q(n)} \,|\, x\#y \in A\}\| \geq k^{q(n)}/2$$

Sei p ein Polynom mit $2^{p(n)} > k^{q(n)}$. Nach Lemma 120 ex. eine Funktion $f \in \#\mathsf{P}$ mit

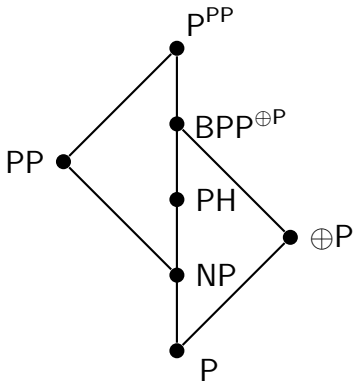
$$f(x\#y) \equiv_{2^{p(n)}} A(x\#y)$$

Betrachte nun die Funktion $g(x) = \sum_{y\in\Gamma^{q(n)}} f(x\#y)$, die nach Lemma 117 in #P ist. Dann gilt

$$x \in L \Leftrightarrow g(x) \bmod 2^{p(n)} \geq k^{q(n)}/2$$

Da eine P^{PP}-Maschine den Wert von $g(x)$ durch eine Binärsuche mit Fragen an das PP-Orakel $B = \{x\#bin(l) \,|\, g(x) \geq l\}$ (siehe Lemma 118) berechnen und die Bedingung $g(x) \bmod 2^{p(n)} \geq k^{q(n)}/2$ überprüfen kann, folgt $L \in \mathsf{P}^{\mathsf{PP}}$. ■

Korollar 122.
$\mathsf{PH} \subseteq \mathsf{BP}\cdot\oplus\mathsf{P} = \mathsf{BPP}^{\oplus\mathsf{P}} \subseteq \mathsf{PP}^{\oplus\mathsf{P}} \subseteq \mathsf{P}^{\mathsf{PP}}$.



11 Interaktive Beweissysteme

In diesem Abschnitt gehen wir folgender Frage nach: Was ist effizient beweisbar? Oder besser: Was ist effizient verifizierbar? Der Aufwand für das Finden eines Beweises wird hierbei bewusst außer Acht gelassen, d.h. wir interessieren uns nur für den Aufwand für das Überprüfen eines Beweises.

Was die Art der Aussagen betrifft, die wir betrachten wollen, so können wir uns o.B.d.A. auf Aussagen der Form „$x \in A$“ beschränken. Denn unabhängig davon, welchen Wahrheitsbegriff wir zu Grunde legen, die Menge aller wahren Aussagen kann immer durch eine Sprache der Form

$$A = \{x \,|\, x \text{ kodiert eine wahre Aussage}\}$$

beschrieben werden.

Beweistheoretische Sicht auf NP

Im klassischen Modell der effizienten Verifizierbarkeit hat ein mächtiger *Prover* P die Aufgabe, zu einer gegebenen Eingabe x einen Beweis y zu finden, dessen Korrektheit ein *Verifier* V in deterministischer Polynomialzeit überprüfen kann. Wie wir bereits wissen, sind genau die Sprachen in der Klasse 8NP auf diese Weise effizient verifizierbar.

Es gibt verschiedene Möglichkeiten, dieses Modell zu verallgemeinern, ohne die Effizienz des Verifiers aufzugeben.

Frage. *Ermöglicht eine Interaktion zwischen P und V die Verifikation weiterer Sprachen?*

Nein, denn P kann bei Eingabe x gleich zu Beginn alle Fragen von V berechnen und die zugehörigen Antworten an V senden.

Frage. *Sind mehr Sprachen entscheidbar, wenn V sowohl Fragen stellen als auch Zufallszahlen benutzen darf?*

Dann sind genau die Sprachen in PSPACE entscheidbar.

Frage. *Sind mehr Sprachen entscheidbar, wenn V mehrere Prover unabhängig voneinander befragen darf?*

Dann sind genau die Sprachen in NEXP entscheidbar.

Definition 123.

- *Ein Multi-Prover Interactive Proof System (MIPS) besteht aus einer PTM V (Verifier) und PTMs $P_1, \ldots, P_k$ (k Prover), die alle Zugriff auf ein read-only Eingabeband haben. Zudem hat jeder Prover P_i ein privates Kommunikationsband mit V. Dabei ist durch ein Protokoll festgelegt, welche Berechnungen von welcher Partei auszuführen sind, und jeder Wechsel zwischen den Parteien kennzeichnet den Beginn einer neuen Runde. Die letzte Runde muss von V ausgeführt werden, wird aber nur als Runde mitgezählt, falls V darin Zufallsbits benutzt. Während V insgesamt nur polynomiell viele Rechenschritte ausführen kann, ist die Laufzeit der Prover unbegrenzt (aber endlich).*
- *Ein MIPS $(V, P_1, \ldots, P_k)$, entscheidet eine Sprache $A \subseteq \Sigma^*$, falls für alle $x \in \Sigma^*$ gilt:*

  $$x \in A \Rightarrow \Pr[(V, P_1, \ldots, P_k)(x) = 1] \geq 2/3 \qquad \textit{(Vollständigkeit)}$$
  $$x \notin A \Rightarrow \forall P_1', \ldots, P_k' : \Pr[(V, P_1', \ldots, P_k')(x) = 1] \leq 1/3 \qquad \textit{(Korrektheit)}$$

- *Ein MIPS mit nur einem Prover heißt IPS.* MIP (IP) *ist die Klasse aller Sprachen, die von einem MIPS (IPS) entschieden werden. Wird die Rundenzahl durch $r(|x|)$ beschränkt, so bezeichnen wir die resultierenden Teilklassen mit* MIP$[r(n)]$ (IP$[r(n)]$).
- *Ein IPS, bei dem V alle benutzten Zufallszahlen dem Prover mitteilt, heißt Arthur-Merlin Protokoll, wobei der Verifier als Arthur und der Prover als Merlin bezeichnet werden.* AM$[r(n)]$ (MA$[r(n)]$) *ist die Klasse aller Sprachen, die von einem Arthur-Merlin Protokoll in höchstens $r(|x|)$ Runden entschieden werden können, wobei Arthur (Merlin) mit der ersten Runde beginnt. Für* AM$[2]$ *bzw.* MA$[3]$ *etc. wird auch einfach* AM *bzw.* MAM *etc. geschrieben.*

Die folgenden Beziehungen zwischen diesen Klassen folgen direkt aus den Definitionen.

Proposition 124.

- $\mathsf{AM}[r] \cup \mathsf{MA}[r] \subseteq \mathsf{IP}[r] \subseteq \mathsf{MIP}[r]$
- $\mathsf{MIP}[0] = \mathsf{IP}[0] = \mathsf{AM}[0] = \mathsf{P}$
- $\mathsf{AM}[1] = \mathsf{BPP}$, $\mathsf{MA}[1] = \mathsf{NP}$, $\mathsf{MIP}[1] = \mathsf{IP}[1] = \mathsf{NP} \cup \mathsf{BPP}$
- $\exists^p \cdot \mathsf{BPP} \subseteq \mathsf{MA}$, $\mathsf{AM} = \mathsf{BP} \cdot \mathsf{NP}$

Dagegen erfordern folgende Beziehungen teilweise umfangreiche Beweise.

Satz 125.

- $\mathsf{MA} \subseteq \mathsf{AM}$
- $\mathsf{IP}[r] \subseteq \mathsf{AM}[r+2]$
- $\mathsf{IP}[\mathcal{O}(1)] = \mathsf{IP}[2] = \mathsf{AM}$
- $\mathsf{IP} = \mathsf{PSPACE}$, $\mathsf{MIP} = \mathsf{NEXP}$

11.1 Iso- und Automorphismen

In diesem und den folgenden Abschnitten untersuchen wir die Komplexität des Graphisomorphieproblems. Hierbei bedeutet es keine Einschränkung, wenn wir voraussetzen, dass beide Graphen dieselbe

Knotenmenge besitzen. Daher betrachten wir nur Graphen mit einer Knotenmenge der Form $V = [n] := \{1, \ldots, n\}$ (n bezeichnet also in diesem Kontext immer die Knotenzahl). Wir bezeichnen die Menge aller Graphen mit Knotenmenge $[n]$ mit $\mathcal{G}_n$ und die Menge aller Permutationen φ auf der Menge $[n]$ mit S_n. Für $\varphi(u)$ schreiben wir auch u^φ.

Definition 126.

- *Für einen Graphen $G = (V, E) \in \mathcal{G}_n$ und eine Permutation $\varphi \in S_n$ sei $G^\varphi = (V, E^\varphi)$ der Graph mit der Kantenmenge $E^\varphi = \{\{u^\varphi, v^\varphi\} \,|\, \{u, v\} \in E\}$.*
- *Eine Permutation $\varphi \in S_n$ heißt **Isomorphismus** zwischen zwei Graphen G_1 und G_2 in $\mathcal{G}_n$, falls $G_1^\varphi = G_2$ ist. In diesem Fall heißen G_1 und G_2 **isomorph** (in Zeichen $G_1 \cong G_2$).*

Die Menge $\{\varphi \in S_n \,|\, G_1^\varphi = G_2\}$ aller Isomorphismen zwischen G_1 und G_2 bezeichnen wir mit $Iso(G_1, G_2)$.

Graphisomorphieproblem (GI):

Gegeben: Zwei 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. Vor kurzem gelang Babai der Nachweis, dass GI in quasipolynomieller Zeit $2^{(\log n)^{O(1)}}$ entscheidbar ist.

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

Definition 127. *Eine Permutation $\varphi \in S_n$ heißt **Automorphismus** eines Graphen G (kurz: $\varphi \in Aut(G)$), falls $G^\varphi = G$ ist.*

Da $Aut(G)$ unter Komposition abgeschlossen ist, bildet $Aut(G)$ eine Untergruppe von S_n. Jeder Graph besitzt die Identität $id \in S_n$ als Automorphismus, welcher als trivialer Automorphismus bezeichnet wird.

Graphautomorphieproblem (GA):

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

Lemma 128. *Für jeden Graphen G gilt*

(i) $\|\{H \in \mathcal{G}_n \,|\, H \cong G\}\| = \frac{n!}{\|Aut(G)\|}$,

(ii) $\|\{(H, \pi) \in \mathcal{G}_n \times S_n \,|\, H \cong G, \pi \in Aut(H)\}\| = n!$.

Beweis.

(i) Wir nennen zwei Permutationen φ und π äquivalent, falls sie G auf denselben Graphen $H = G^\varphi = G^\pi$ abbilden. Wegen

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

sind φ und π genau dann äquivalent, wenn sie in der gleichen (Rechts-)Nebenklasse $Aut(G)\varphi = Aut(G)\pi$ von $Aut(G)$ liegen. Die Anzahl der Nebenklassen entspricht somit der Anzahl der zu G isomorphen Graphen. Zudem wissen wir aus der Gruppentheorie, dass die Nebenklassen einer Untergruppe U die gleiche Größe wie U haben und eine Partition der Gesamtgruppe bilden. Also gibt es genau $\frac{n!}{\|Aut(G)\|}$ Nebenklassen.

(ii) Ist φ ein Isomorphismus zwischen G und H, so gilt $Aut(G) = \varphi Aut(H)\varphi^{-1}$ (d.h. $Aut(G)$ und $Aut(H)$ sind *zueinander konjugierte* Untergruppen von S_n), was $\|Aut(G)\| = \|Aut(H)\|$ impliziert. Daher folgt mit (i),

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

■

11.2 GI liegt in co-IP[2]

Betrachte folgendes 2-Runden IPS für $\overline{\text{GI}}$.

IP[2]-Protokoll für $\overline{\text{GI}}$

input: zwei Graphen $G_i = (V, E_i) \in \mathcal{G}_n$, $i \in \{1, 2\}$
$\boldsymbol{V}$**:** **guess randomly** $(i, \pi) \in_R \{1, 2\} \times S_n$
  $H := G_i^\pi$
$\boldsymbol{V} \to \boldsymbol{P}$**:** H
$\boldsymbol{P}$**:** **if** $H \cong G_1$ **then** $j := 1$ **else** $j := 2$
$\boldsymbol{P} \to \boldsymbol{V}$**:** j
$\boldsymbol{V}$**:** **if** $i = j$ **then** accept **else** reject

Behauptung 129.

i) $G_1 \ncong G_2 \Rightarrow \Pr[(V, P)(G_1, G_2) = 1] = 1$

ii) $G_1 \cong G_2 \Rightarrow \forall P' : \Pr[(V, P')(G_1, G_2) = 1] \leq \frac{1}{2}$

Beweis. *i)* klar.

ii) Sei P' ein beliebiger Prover und seien X, Y und Z die Zufallsvariablen, die die Wahl der Zahlen i und j bzw. des Graphen H bei Ausführung des Protokolls $(V, P')(G_1, G_2)$ beschreiben. Dann ist X gleichverteilt auf der Menge $\{1, 2\}$ und wegen $G_1 \cong G_2$ hat Z den Wertebereich $W(Z) = \{H \in \mathcal{G}_n \mid H \cong G_1\} = \{H \in \mathcal{G}_n \mid H \cong G_2\}$. Zudem gilt für jeden Graphen $H \in W(Z)$,

$$\Pr[Z = H] = \sum_{i=1,2} \underbrace{\Pr[X = i]}_{1/2} \underbrace{\Pr[Z = H | X = i]}_{=:p_i} = \frac{p_1 + p_2}{2}.$$

Wegen

$$p_i = \|Iso(G_i, H)\|/n! = \|Aut(G_i)\|/n!$$

und $\|Aut(G_1)\| = \|Aut(G_2)\|$ folgt $p_1 = p_2 = (p_1 + p_2)/2$ und somit auch $\Pr[Z = H] = \Pr[Z = H | X = i]$ für $i = 1, 2$. Daher sind X und Z stochastisch unabhängig. Da Y nur von Z (und den Eingabegraphen) abhängt, ist mit Z auch Y von X stochastisch unabhängig. Folglich gilt

$$\begin{aligned}\Pr[(V, P')(G_1, G_2) = 1] &= \Pr[X = Y] \\ &= \underbrace{\Pr[X = Y = 1]}_{\Pr[X=1]\Pr[Y=1]} + \underbrace{\Pr[X = Y = 2]}_{\Pr[X=2]\Pr[Y=2]} \\ &= \Pr[Y \in \{1, 2\}]/2 \;\leq\; 1/2.\end{aligned}$$

■

Die Fehlerwahrscheinlichkeit von V im Fall $G_1 \cong G_2$ lässt sich von $1/2$ auf $1/4$ reduzieren, indem das Protokoll zweimal parallel ausgeführt wird (wodurch sich die Anzahl der Runden nicht erhöht).

Die Korrektheit des Protokolls hängt wesentlich von der Geheimhaltung der Zufallszahlen des Verifiers gegenüber dem Prover ab. Wir werden später noch ein IP[2]-Protokoll mit öffentlichen Zufallszahlen (also ein AM-Protokoll) für GI angeben. Interessant ist auch, dass die Laufzeit des Provers polynomiell beschränkt ist, falls er ein GI-Orakel befragen kann. Auch für $\overline{\text{GA}}$ gibt es ein 2-Runden IPS, bei dem der Prover relativ zu einem GA-Orakel polynomielle Laufzeit hat.

IP[2]-Protokoll für $\overline{\text{GA}}$

input: ein Graph $G = (V, E) \in \mathcal{G}_n$
$\boldsymbol{V}$**:** **guess randomly** $\pi \in_R S_n$
  $H := G^\pi$
$\boldsymbol{V} \to \boldsymbol{P}$**:** H
$\boldsymbol{P}$**:** compute $\varphi \in Iso(G, H)$
$\boldsymbol{P} \to \boldsymbol{V}$**:** φ
$\boldsymbol{V}$**:** **if** $\pi = \varphi$ **then** accept **else** reject

Behauptung 130.

i) $G \notin \text{GA} \Rightarrow \Pr[(V, P)(G) = 1] = 1$

ii) $G \in \text{GA} \Rightarrow \forall P' : \Pr[(V, P')(G) = 1] \leq \frac{1}{2}$

Der Beweis ist ähnlich wie bei der vorigen Behauptung. Die Fehlerwahrscheinlichkeit von V lässt sich wieder von $1/2$ auf $1/4$ reduzieren, indem das Protokoll zweimal parallel ausgeführt wird. Auch hier hängt die Korrektheit des Protokolls wesentlich von der Geheimhaltung der Zufallszahlen des Verifiers gegenüber dem Prover ab.

11.3 GI liegt in co-AM

Als nächstes wollen wir zeigen, dass GI fast in co-NP liegt (genauer: $\mathrm{GI} \in \mathsf{BP} \cdot \text{co-NP}$ bzw. $\overline{\mathrm{GI}} \in \mathsf{BP} \cdot \mathsf{NP} = \mathsf{AM}$). Als Konsequenz hiervon ist GI nicht NP-vollständig, außer wenn $\mathsf{PH} = \mathsf{BP} \cdot \mathsf{NP}$ ist. Wir betrachten zunächst folgendes Protokoll für $\overline{\mathrm{GI}}$ mit öffentlichen Zufallszahlen.

---

1 $\boldsymbol{V}$**:** **guess randomly** $(H,\pi) \in_R \mathcal{G}_n \times S_n$
2 $\boldsymbol{V} \to \boldsymbol{P}$**:** (H,π)
3 $\boldsymbol{P}$**:** $I := \mathit{Iso}(G_1,H) \cup \mathit{Iso}(G_2,H)$
4 **if** $I \neq \emptyset$ **then** `compute` $\varphi \in I$ **else** $\varphi := id$
5 $\boldsymbol{P} \to \boldsymbol{V}$**:** φ
6 $\boldsymbol{V}$**:** **if** $H \in \{G_1^\varphi, G_2^\varphi\} \wedge H^\pi = H$ **then** `accept` **else** `reject`

Dieses Protokoll hat folgende Eigenschaften.

Behauptung 131.

i) $G_1 \ncong G_2 \Rightarrow \Pr[(V,P)(G_1,G_2)=1] = 2/2^{\binom{n}{2}}$

ii) $G_1 \cong G_2 \Rightarrow \forall P' : \Pr[(V,P')(G_1,G_2)=1] \leq 1/2^{\binom{n}{2}}$

Beweis. Nach Lemma 128 haben die Mengen

$$X(G_i) = \{(H,\pi) \in \mathcal{G}_n \times S_n \,|\, H \cong G_i, \pi \in Aut(H)\}$$

die Mächtigkeit $\|X(G_i)\| = n!$ und somit folgt für jeden Prover P',

$$\begin{aligned}
G_1 \ncong G_2 \Rightarrow\ & X(G_1) \cap X(G_2) = \emptyset \\
\Rightarrow\ & \|X(G_1) \cup X(G_2)\| = 2n! \\
\Rightarrow\ & \Pr[(V,P)(G_1,G_2)=1] = 2n!/2^{\binom{n}{2}} n! = 2/2^{\binom{n}{2}} \\
G_1 \cong G_2 \Rightarrow\ & X(G_1) = X(G_2) \\
\Rightarrow\ & \|X(G_1) \cup X(G_2)\| = n! \\
\Rightarrow\ & \Pr[(V,P')(G_1,G_2)=1] \leq n!/2^{\binom{n}{2}} n! = 1/2^{\binom{n}{2}}
\end{aligned}$$

■

Um hieraus ein AM-Protokoll für $\overline{\mathrm{GI}}$ zu erhalten, müssen wir die Wahrscheinlichkeit im Fall $G_1 \ncong G_2$ von $2/2^{\binom{n}{2}}$ auf mindestens $2/3$ vergrößern, ohne dass sie im Fall $G_1 \cong G_2$ den Wert $1/3$ überschreitet. Hierzu betrachten wir folgende Verallgemeinerung des BP-Operators.

Definition 132. *Sei $\mathcal{C}$ eine Sprachklasse. Eine Sprache $A \subseteq \Sigma^*$ gehört zu $\widetilde{\mathsf{BP}} \cdot \mathcal{C}$, falls Funktionen $g : \Sigma^* \to \mathbb{N}^+$ in FP und $f \in \# \cdot \mathcal{C}$ existieren, so dass für alle x gilt,*

$$\begin{aligned}
x \in A &\Rightarrow f(x) \geq g(x) \\
x \notin A &\Rightarrow f(x) \leq g(x)/2
\end{aligned}$$

Es ist klar, dass $\mathsf{BP} \cdot \mathcal{C}$ in $\widetilde{\mathsf{BP}} \cdot \mathcal{C}$ enthalten ist: Gilt $A \in \mathsf{BP} \cdot \mathcal{C}$ mittels einer zweiseitigen (k,q)-balancierten Sprache $B \in \mathcal{C}$, so reicht es, für g die Funktion $g(x) = \lceil 2k^{q(|x|)}/3 \rceil$ zu wählen. Zudem gilt $\mathsf{NP} \subseteq \widetilde{\mathsf{BP}} \cdot \mathsf{P}$ (wähle $g(x) = 1$). Daher ist BPP vermutlich echt in $\widetilde{\mathsf{BP}} \cdot \mathsf{P}$ enthalten (wogegen $\widetilde{\mathsf{BP}} \cdot \mathsf{NP} = \mathsf{BP} \cdot \mathsf{NP} = \mathsf{AM}$ ist, siehe Satz 134).

Satz 133. $\overline{\mathrm{GI}} \in \widetilde{\mathsf{BP}} \cdot \mathsf{NP}$.

Beweis. Betrachte die NP-Sprache

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

wobei die Mengen $X(G_i)$ wie im Beweis von Behauptung 131 definiert sind. Dann ist $\#B(\langle G_1,G_2\rangle) = \|X(G_1) \cup X(G_2)\|$ und da die Funktion $g(\langle G_1,G_2\rangle) = 2n!$ in FP berechenbar ist, folgt $\overline{\mathrm{GI}} \in \widetilde{\mathsf{BP}} \cdot \mathsf{NP}$. ■

Satz 134. $\widetilde{\mathsf{BP}} \cdot \mathsf{NP} \subseteq \mathsf{BP} \cdot \mathsf{NP}$.

Beweis. Sei $L \subseteq \Sigma^*$ eine Sprache in $\widetilde{\mathsf{BP}} \cdot \mathsf{NP}$. Dann existieren Funktionen $g : \Sigma^* \to \mathbb{N}^+$ in FP und $f(x)$ in $\#\mathsf{NP}$ mit

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

Zu f existiert eine (k,q)-balancierte NP-Sprache A mit

$$f(x) = \#A(x) = \|\{y \in \{0,\ldots,k-1\}^{q(|x|)} \,|\, x\#y \in A\}\|,$$

wobei wir o.B.d.A. annehmen können, dass $k = 2$ ist und A keine Wörter der Form $x\#0^{q(|x|)}$ enthält. Weiter sei $m(x) = 5q(|x|)$ und

$$B = \{x\#y_1 \ldots y_5 \in \{0,1\}^{m(x)} \,|\, \text{für } i = 1,\ldots,5 \text{ gilt } x\#y_i \in A\}$$

Dann enthalten die Mengen $B_x = \{y \in \{0,1\}^{m(x)} \,|\, x\#y \in B\}$ wegen $\|B_x\| = \#B(x) = f(x)^5$ für alle $x \in L$ mindestens $g(x)^5$ und für alle $x \notin L$ höchstens $g(x)^5/2^5$ Strings.

Setze nun $k(x) = \max\{0, \lfloor \log_2 g(x)^5 \rfloor - 2\}$ und betrachte die NP-Sprache

$$B' = \{x\#z \,|\, z \in \{0,1\}^{r(|x|)}, \exists y \in B_x : h_z(y) = 0^{k(x)}\},$$

wobei $r(n)$ ein Polynom mit $r(n) \geq k(x)m(x)$ für alle $x \in \Sigma^n$ ist und (die Matrix A_{h_z} von) $h_z \in Lin(m(x), k(x))$ durch die ersten $k(x)m(x)$ Bits von z repräsentiert wird.

Dann ist für einen zufällig gewählten String $z \in_R \{0,1\}^{r(|x|)}$ das Ereignis $x\#z \in B'$ gleichbedeutend mit dem Ereignis $S_x \geq 1$ für die Zufallsvariable

$$S_x = \sum_{y \in B_x} Z_y \text{ mit } Z_y = \begin{cases} 1, & h_z(y) = 0^{k(x)} \\ 0, & \text{sonst} \end{cases}$$

Daher reicht es zu zeigen, dass der Wert von $\Pr[S_x \geq 1]$ für alle $x \in L$ mindestens $2/3$ und für alle $x \notin L$ höchstens $1/3$ ist.

Da nach Lemma 109 die Indikatorvariablen Z_y paarweise stochastisch unabhängig sind, folgt $Var(S_x) = \sum_{y \in B_x} Var(Z_y)$ (s. Übungen), was

$$Var(S_x) = \#B(x)2^{-k(x)}(1 - 2^{-k(x)}) \leq 2^{-k(x)}\#B(x) = E(S_x)$$

impliziert.

Wir betrachten zuerst den Fall $k(x) = 0$ (d.h. $\lfloor \log_2 g(x)^5 \rfloor \leq 2$ bzw. $g(x)^5 < 8$). In diesem Fall hat $h_z(y)$ für alle $z \in \{0,1\}^{r(|x|)}$ und alle $y \in \{0,1\}^{m(x)}$ den Wert $h(y) = 0^{k(x)} = \varepsilon$ und es folgt $Var(S_x) = 0$ und $E(S_x) = S_x = \#B(x)$. Wegen $g(x)^5 \geq 1$ und $g(x)^5/2^5 < 1$ ist also $\Pr[S_x \geq 1] = 1$, falls $x \in L$, und $\Pr[S_x \geq 1] = 0$, falls $x \notin L$ ist.

Im Fall $k(x) \geq 1$ gilt $g(x)^5/8 < 2^{k(x)} \leq g(x)^5/4$. Da für alle $x \in L$ $\#B(x) \geq g(x)^5$ und somit $E(S) \geq 2^{-k(x)}g(x)^5 \geq 4$ ist, folgt mit Tschebyscheff

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

was $\Pr[S_x \geq 1] \geq 2/3$ impliziert. Andererseits enthält B_x für alle $x \notin L$ höchstens $g(x)^5/2^5$ Strings und es folgt in diesem Fall

$$\Pr[S_x \geq 1] \leq \sum_{y \in B_x} \Pr[Z_y = 1] \leq 2^{-k(x)}g(x)^5 \leq 1/4 < 1/3 \qquad \blacksquare$$

Korollar 135. GI *ist nicht* NP*-vollständig, außer wenn* PH *auf ihre zweite Stufe* Σ_2^p *(bzw. sogar auf* $\mathsf{BP} \cdot \mathsf{NP}$*) kollabiert.*

Beweis. Wegen Lemma 91 und Satz 95 gilt für jede Klasse C, die unter majority-Reduktionen abgeschlossen ist,

$$\exists^p \cdot \forall^p \cdot \mathsf{BP} \cdot \mathsf{C} \subseteq \exists^p \cdot \mathsf{BP} \cdot \forall^p \cdot \mathsf{C} \subseteq \exists^p \cdot \mathsf{R} \cdot \forall^p \cdot \forall^p \cdot \mathsf{C} = \exists^p \cdot \forall^p \cdot \mathsf{C}.$$

Insbesondere gilt also $\exists^p \cdot \forall^p \cdot \mathsf{BP} \cdot \mathsf{co\text{-}NP} = \Sigma_2^p$ und somit ist NP nicht in $\mathsf{BP} \cdot \mathsf{co\text{-}NP}$ enthalten, außer wenn $\Sigma_3^p = \exists^p \cdot \forall^p \cdot \mathsf{NP} \subseteq \exists^p \cdot \forall^p \cdot \mathsf{BP} \cdot \mathsf{co\text{-}NP} = \Sigma_2^p$ ist, was wiederum $\mathsf{PH} = \exists^p \cdot \mathsf{co\text{-}NP} \subseteq \exists^p \cdot \mathsf{BP} \cdot \mathsf{NP} \subseteq \mathsf{BP} \cdot \exists^p \cdot \mathsf{NP} = \mathsf{BP} \cdot \mathsf{NP}$ impliziert. $\blacksquare$

Wir geben abschließend noch das aus den Beweisen der Sätze 133 und 134 resultierende AM-Protokoll für $\overline{\text{GI}}$ an:

AM-Protokoll für $\overline{\text{GI}}$

input: zwei Graphen $G_i = (V, E_i) \in \mathcal{G}_n$, $i \in \{1, 2\}$
V: $k := \max\{0, \lfloor \log_2(8(n!)^5) \rfloor\}$
  $m := 5|\langle G_1, \pi \rangle|$ (mit $\pi \in S_n$)
  guess randomly $h \in_R Lin(m, k)$
$V \to P$: h
P: compute $H_1, \ldots, H_5, \pi_1, \ldots, \pi_5, \varphi_1, \ldots, \varphi_5$ mit
  $(*)$ $\begin{cases} \varphi_i \in \mathit{Iso}(G_1, H_i) \cup \mathit{Iso}(G_2, H_i), \pi_i \in \mathit{Aut}(H_i) \text{ für} \\ i = 1, \ldots, 5 \text{ und } h(\langle H_1, \pi_1 \rangle \cdots \langle H_5, \pi_5 \rangle) = 0^k \end{cases}$
$P \to V$: $H_1, \ldots, H_5, \pi_1, \ldots, \pi_5, \varphi_1, \ldots, \varphi_5$
V: **if** $(*)$ **then accept else reject**

11.4 Zero-Knowledge Protokoll für GI

Interaktive Beweissysteme haben den großen Vorteil, dass sie im Gegensatz zu NP-Beweisen die Möglichkeit der Nichtreproduzierbarkeit bieten. Genauer gesagt, kann der Prover den Verifier von der Zugehörigkeit der Eingabe zu einer Sprache überzeugen, ohne den Verifier in die Lage zu versetzen, dies seinerseits einem Dritten gegenüber zu wiederholen.

Zum Beispiel besteht ein NP-Beweis für die Isomorphie zweier Graphen in der Angabe eines Isomorphismus und ist daher problemlos reproduzierbar, d.h. der Verifier kann damit jeden von der Isomorphie der beiden Graphen überzeugen. Insbesondere in der Kryptografie sind jedoch so genannte Zero-Knowledge Beweise von Interesse, aus denen der Verifier kein Zusatzwissen (also außer der Tatsache, dass die Eingabe zur Sprache gehört) erlangen kann.

Formal lässt sich diese Eigenschaft so charakterisieren, dass der Verifier sämtliche Informationen, die er vom Prover erhält, auch ohne dessen Mitwirkung generieren können muss. Während also die Korrektheit und Vollständigkeit eines interaktiven Beweissystems Eigenschaften des Verifiers sind (also nur von diesem abhängen), handelt es sich bei Zero-Knowledge um eine Eigenschaft des Provers, d.h. kein effizienter Verifier darf durch die Interaktion mit dem Prover ein Zusatzwissen in irgendeiner Form erhalten.

Definition 136.

- *Das Transkript $View_{V,P_1,\ldots,P_k}(x)$ eines MIPS $(V, P_1, \ldots, P_k)$ bei Eingabe x ist die ZV, die sämtliche während der Berechnung von $(V, P_1, \ldots, P_k)(x)$ ausgetauschten Nachrichten angibt.*
- *Ein MIPS $(V, P_1, \ldots, P_k)$ für A hat die perfekte Zero-Knowledge Eigenschaft (kurz PZK), wenn es für jede PPTM V' einen probabilistischen Simulator S mit im Erwartunswert polynomiell beschränkter Laufzeit gibt, so dass für alle Eingaben $x \in A$ die ZVen $S(x)$ und $View_{V',P_1,\ldots,P_k}(x)$ identisch verteilt sind.*

Betrachte folgendes 3-Runden IPS für GI:

IP[3]-Protokoll für GI mit PZK-Eigenschaft

input: zwei Graphen $G_i = (V, E_i) \in \mathcal{G}_n$, $i \in \{1, 2\}$
P: **guess randomly** $i \in_R \{1, 2\}$
  guess randomly $\pi \in_R S_n$
  $H := G_i^\pi$
$P \to V$: H
V: **guess randomly** $j \in_R \{1, 2\}$
$V \to P$: j
P: **if** $j = i$ **then** $\varphi := \pi$
  if $j = 3 - i$ **then**
    compute $\tau \in \mathit{Iso}(G_j, G_i)$; $\varphi := \tau\pi$
  if $j \notin \{1, 2\}$ **then** $\varphi := \mathrm{id}$
$P \to V$: φ
V: **if** $G_j^\varphi = H$ **then accept else reject**

Man beachte, dass der Prover P effizient ist, falls er einen Isomorphismus zwischen G_1 und G_2 als Zusatzeingabe erhält.

Behauptung 137.

i) $G_1 \cong G_2 \Rightarrow \Pr[(V,P)(G_1,G_2)=1]=1$

ii) $G_1 \ncong G_2 \Rightarrow \forall P' : \Pr[(V,P')(G_1,G_2)=1] \leq \frac{1}{2}$

Um zu zeigen, dass das Protokoll die perfekte Zero-Knowledge Eigenschaft besitzt, müssen wir für jeden probabilistischen Verifier V' mit polynomieller Laufzeit einen probabilistischen Simulator S mit erwarteter polynomieller Laufzeit angeben, der die zwischen P und V' ausgetauschten Nachrichten mit exakt den gleichen Wahrscheinlichkeiten erzeugt wie sie bei der Interaktion zwischen P und V' auftreten.

Simulator für obiges PZK-Protokoll

input: Graphen G_1, G_2
  repeat
    guess randomly $i \in_R \{1,2\}$
    guess randomly $\pi \in_R S_n$
    $H := G_i^\pi$
    simuliere V' bei Eingabe (G_1,G_2) und erster Prover-Nachricht H
  until V' antwortet mit einer Nachricht $j \neq 3-i$
  if $j \neq i$ **then** $\pi :=$ id
output (H, j, π)

Da die Verteilung (der zufälligen Wahl) von i im Fall $G_1 \cong G_2$ unabhängig von der Verteilung von H ist (siehe Beweis von Beh. 129), gilt $\Pr[j = 3-i] \leq 1/2$. Damit ist die erwartete Zahl der Durchläufe der repeat-Schleife ≤ 2. Die Fehlerwahrscheinlichkeit von V im Fall $G_1 \cong G_2$ lässt sich von $1/2$ auf $1/4$ reduzieren, indem das Protokoll zweimal hintereinander ausgeführt wird. Man beachte, dass bei einer parallelen Ausführung die PZK-Eigenschaft verloren gehen kann.

12 Komplexität von Anzahlproblemen

In diesem Abschnitt untersuchen wir die Komplexität einer Reihe von konkreten Anzahlproblemen. Jedes NP-Problem hat die Form $A = \exists^p \cdot B$, wobei sich die Sprache B meist 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.

Oft ist das Anzahlproblem sogar deutlich schwieriger zu lösen als das zugrunde liegende NP-Problem. So können wir jede Sprache in PH in Polynomialzeit durch Orakelfragen an #SAT entscheiden, was mit einem SAT-Orakel nicht möglich ist, außer wenn PH auf P(NP) kollabiert. Ein weiteres Beispiel hierfür ist die Bestimmung der Anzahl der erfüllenden Belegungen von Hornformeln. Dieses Problem ist wie #SAT #P-vollständig (unter $\leq^{\mathsf{FP}[1]}$ Reduktionen, siehe unten), obwohl die Erfüllbarkeit von Hornformeln in Polynomialzeit entscheidbar ist.

Es gibt sogar Beispiele, bei denen das zugrunde liegende NP-Problem $A = \exists^p B$ 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, #P-vollständig, obwohl die Frage, ob G ein perfektes Matching besitzt, in Polynomialzeit entscheidbar ist.

In manchen Fällen ist das Anzahlproblem aber auch nicht schwerer als das zugrunde liegende NP-Problem. 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.

12.1 Aussagenlogische Anzahlprobleme

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 138. *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 $\#\textsc{Sat}(F) = 2^n - \#\textsc{Sat}(\neg F)$ ist, folgt $\#\textsc{Sat} \in \mathsf{FP}^{\#\textsc{DnfSat}[1]}$, wofür wir auch $\#\textsc{Sat} \leq^{\mathsf{FP}[1]} \#\textsc{DnfSat}$ schreiben. Da $\#\textsc{Sat}$ $\#\mathsf{P}$-vollständig unter $\leq_{par}$-Reduktionen ist, folgt $\#\mathsf{P} \subseteq \mathsf{FP}^{\#\textsc{DnfSat}[1]}$, d.h. $\#\textsc{DnfSat}$ ist $\#\mathsf{P}$-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 139. *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. $\#\textsc{MonSat}$ ist dagegen $\#\mathsf{P}$-vollständig unter $\leq_{\|}^{\mathsf{FP}[2]}$-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

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

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

Es ist bekannt, dass sogar $\#\textsc{MonSat}$ für 2-KNF-Formeln mit genau 2 (positiven) Literalen pro Klausel $\#\mathsf{P}$-vollständig unter $\leq^{\mathsf{FP}[1]}$-Reduktionen ist.

12.2 Anzahl von Iso- und Automorphismen

Im Gegensatz zu den im vorigen Abschnitt betrachteten aussagenlogischen Anzahlproblemen, die sich als $\#\mathsf{P}$-vollständig unter effizient berechenbaren Reduktionen herausgestellt haben, ist die Komplexität von #GI und von #GA vergleichsweise niedrig. Dabei bestimmt $\#\mathrm{GI}(G_1, G_2) = \|Iso(G_1, G_2)\|$ die Anzahl der Isomorphismen zwischen G_1 und G_2 und $\#\mathrm{GA}(G) = \|Aut(G)\|$ die Anzahl der Automorphismen von G. Wir wissen bereits, dass für isomorphe Graphen $\#\mathrm{GI}(G_1, G_2) = \#\mathrm{GA}(G_1) = \#\mathrm{GA}(G_2)$ 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 $\#\mathrm{GA}(G) = \#\mathrm{GI}(G, G)$ folgt $\#\mathrm{GA} \leq_{par} \#\mathrm{GI}$. 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

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

und somit $\#\mathrm{GI} \in \mathsf{FP}_{\|}^{\#\mathrm{GA}[3]}$. Wir können die beiden Fragen G_1 und G_2 an das #GA-Orakel sogar durch eine Frage H ersetzen, d.h. $\#\mathrm{GI} \in \mathsf{FP}_{\|}^{\#\mathrm{GA}[2]}$. Hierbei entsteht $H = \langle G_1, G_2 \rangle$ aus $K_2 + G_1 + K_1 + G_2$, indem wir einen der beiden Knoten des K_2 mit allen Knoten von G_1 und den Knoten des K_1 mit allen Knoten von G_1 und G_2 verbinden. Dann gilt $\#\mathrm{GA}(H) = \#\mathrm{GA}(G_1) \cdot \#\mathrm{GA}(G_2)$, unabhängig davon, ob G_1 und G_2 isomorph sind oder nicht.

Wir reduzieren nun #GA auf GI. Da das Graphenisomorphieproblem für gefärbte Graphen ColGI und GI logspace-äquivalent sind, d.h. es gilt $\mathrm{ColGI} \equiv_m^{log} \mathrm{GI}$ (siehe Übungen), reicht es, #GA auf ColGI zu reduzieren. Für eine Folge $(i_1, \ldots, i_k)$ von paarweise verschiedenen Knoten $i_j \in \{1, \ldots, n\}$ bezeichne $G_{[i_1,\ldots,i_k]}$ den Graphen, bei dem Knoten i_j mit der Farbe j $(j = 1, \ldots, k)$ 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

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

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

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

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

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

Folglich lässt sich $\#\mathrm{GA}(G)$ durch $\sum_{i=1}^{n-1}(n-i-1) = \sum_{i=1}^{n-2} i = (n-1)(n-2)/2$ nichtadaptive Fragen an GI berechnen. Mit einer Frage mehr lässt sich auch $\#\mathrm{GI} \in \mathsf{FP}_{\|}^{\mathrm{GI}}$ berechnen. Außerdem können wir sogar eine Menge von Generatoren für die Automorphismengruppe $Aut(G)$ in $\mathsf{FP}^{\mathrm{GI}}$ berechnen, indem wir für jedes $j \geq i+1$, für das es ein $\pi \in Aut(G_{[1,\ldots,i-1]})$ mit $\pi(i) = j$ gibt, einen solchen Automorphismus berechnen.