Vorlesungsskript

Einführung in die Komplexitätstheorie

Wintersemester 2021/22

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

17. Februar 2022

Inhaltsverzeichnis

Inhaltsverzeichnis

1 Einführung

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

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

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

Erreichbarkeitsproblem in Digraphen (REACH):

Eingabe: Ein gerichteter Graph $G = (V, E)$ mit $V = \{1, \ldots, n\}$ und $E \subseteq V \times V$.
Gefragt: Gibt es in G einen Weg von Knoten 1 zu Knoten n?

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

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

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

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

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

Algorithmus `suche-Weg`(G)

```
1   input: 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 $O(n^3)$ lösbar (Algorithmus von Dinitz).
- MaxFlow ist in Platz $O(n^2)$ lösbar.

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

Perfektes Matching in bipartiten Graphen (Matching):

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

Gefragt: Besitzt G ein perfektes Matching?

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

Komplexitätsbetrachtungen:

- Matching ist in Zeit $O((n+m)\sqrt{n}$ entscheidbar (Algorithmus von Dinitz).
- Matching ist in Platz $O(n^2)$ entscheidbar.

Die bisher betrachteten Probleme können in deterministischer Polynomialzeit gelöst werden und gelten daher als effizient lösbar. Zum Schluss dieses Abschnitts betrachten wir ein Problem, für das vermutlich nur ineffiziente Algorithmen existieren. Wie üblich bezeichnen wir die Gruppe aller Permutationen auf der Menge $\{1, \ldots, n\}$ 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 $\sqcup, \rhd \notin \Sigma$ ($\sqcup$ heißt* ***Blank*** *und $\rhd$ heißt* ***Anfangssymbol****,*
- *Γ das* ***Arbeitsalphabet*** *mit $\Sigma \cup \{\sqcup, \rhd\} \subseteq \Gamma$,*
- *$\delta : Q \times \Gamma^k \to (Q \cup \{q_h, q_{ja}, q_{nein}\}) \times (\Gamma \times \{L, R, N\})^k$ 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 $a_i = \rhd$ die Bedingung $a'_i = \rhd$ und $D_i = R$ erfüllt ist (d.h. das Anfangszeichen $\rhd$ darf nicht durch ein anderes Zeichen überschrieben werden und der Kopf muss nach dem Lesen von $\rhd$ immer nach rechts bewegt werden).

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

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

Definition 3. *Eine Konfiguration $K' = (q', u'_1, v'_1, \ldots, u'_k, v'_k)$ heißt* ***Folgekonfiguration*** *von $K = (q, u_1, a_1 v_1, \ldots, u_k, a_k v_k)$ (kurz: $K \underset{M}{\longrightarrow} K'$), falls eine Anweisung $(q, a_1, \ldots, a_k) \mapsto (q', a'_1, D_1, \ldots, a'_k, D_k)$ in δ und $b_1, \ldots, b_k \in \Gamma$ existieren, so dass für $i = 1, \ldots, k$ jeweils eine der folgenden drei Bedingungen gilt:*

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

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

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

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

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

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*

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

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 $\mathit{range}(f) = \{f(x) \,|\, x \in \Sigma^*\}$ 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.*

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*

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

*die **Rechenzeit** von M bei Eingabe x, wobei $\sup \mathbb{N} = \infty$ 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 zudem immer $D_k \neq L$ und ist M eine DTM, bei der $\delta(q, a_1, \ldots, a_k)$ nicht von a_k abhängt, 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, da von ihm nicht gelesen werden kann (*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

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

*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

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

bzw.

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

zusammen. Ferner sei

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

Die wichtigsten Platzkomplexitätsklassen sind

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

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

3 Grundlegende Beziehungen

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

3.1 Robustheit von Komplexitätsklassen

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

Lemma 15 (Bandreduktion)**.**
*Zu jeder $s(n)$-platzbeschränkten Offline-k-*DTM *M mit $k \geq 3$ ex. eine $s(n)$-platzbeschränkte Offline-2-*DTM *M' mit $L(M') = L(M)$.*

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

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

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

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

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

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

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 $\Gamma' = \Sigma \cup \{\sqcup,\triangleright\} \cup \Gamma^m$ mit $m = \lceil 8/c \rceil$ und simuliert M wie folgt.

Initialisierung: M' kopiert die Eingabe $x = x_1 \ldots x_n$ 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

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

beschränkt ist, da auf Band 2 maximal $time_N(x) + 1$ Felder und auf den Bändern 3 bis $k + 2$ maximal $space_N(x)$ Felder besucht werden.

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. ■

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

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

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

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

Insbesondere erhalten wir somit die Inklusionskette

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

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

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

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

3.3 Der Satz von Savitch

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

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

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

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

Satz 20 (Savitch, 1970)**.**
Für jede echte Komplexitätsfunktion $s(n) \geq \log n$ gilt

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

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

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

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

Diese Beobachtung führt sofort auf folgende Prozedur $\texttt{reach}(K, K', i)$, 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.

Initialisierung: $M(x)$ schreibt das Tripel $(K_x, \hat{K}_x, \lceil s(|n|) \log c \rceil)$ auf das 2. Band, wobei für das Eingabeband nur die Kopfposition, nicht jedoch die Beschriftung notiert wird (also z.B. $K_x = (q_0, 1, \varepsilon, \triangleright)$) und jede akzeptierende Endkonfiguration mit der Konfiguration $\hat{K}_x = (q_{\text{ja}}, 1, \varepsilon, \triangleright)$ identifiziert wird.

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

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

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

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

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

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

Korollar 21.

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

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

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

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

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

3.4 Der Satz von Immerman und Szelepcsényi

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

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

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

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

Gefragt: Ist F eine Tautologie, d. h. 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.

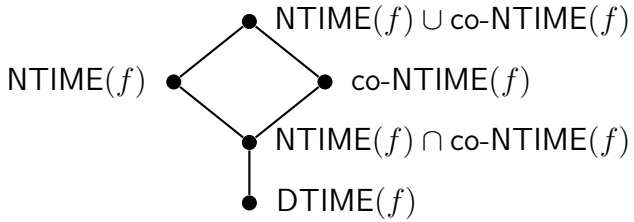
Deterministische Rechnungen lassen sich leicht komplementieren (durch Vertauschen der Zustände q_{ja} und q_{nein}). Daher sind deterministische Komplexitätsklassen unter Komplementbildung abgeschlossen.

Proposition 23.

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

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

Damit ergibt sich folgende Inklusionsstruktur:



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

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

Proposition 25.

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

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

Beweis. Siehe Übungen. ■

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

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

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

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

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

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

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

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

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

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

```
1   if t = 0 then halte im Zustand q_nein
2   z := 0
3   for each Konfiguration K' der Größe ≤ s(n) do
4     rate eine Rechnung α der Laenge ≤ t − 1 von N(x)
5     if α endet in K' then
6       z := z + 1
7       if K' →_N^{≤1} 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' mit $K_x \xrightarrow[N]{}^{\leq t-1} K'$ und $K' \xrightarrow[N]{}^{\leq 1} K$ 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 allen Eingaben $w = \langle x, r, t, K\rangle$ mit $r = r(x, t-1)$ strong ist, also $N_0(w)$ genau im Fall $w \notin L_0$ eine verwerfende Rechnung hat.

Um bei Eingabe $w = \langle x, r, t, K\rangle$ eine verwerfende Endkonfiguration zu erreichen, muss N_0 $r = r(x, t-1)$ Konfigurationen K' finden, für die zwar $K_x \xrightarrow[N]{}^{\leq t-1} K'$ aber nicht $K' \xrightarrow[N]{}^{\leq 1} K$ 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 w 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). Die Kenntnis der Anzahlen $r(x, t)$ versetzt N' in die Lage, für alle Eingaben $x \notin L$ zu verifizieren, dass $N(x)$ keine akzeptierende Endkonfiguration erreichen kann.

Pseudocode für $N'(x)$

```
1   t := 0
2   r := 1
3   repeat
4     t := t + 1
5     r⁻ := r
6     r := 0
7     for each K' der Größe ≤ s(n) 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' 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 28. *Im t-ten Durchlauf der repeat-Schleife wird r^- in Zeile 5 auf den Wert $r(x, t-1)$ gesetzt. Folglich wird N_0 von N' in Zeile 8 nur mit Eingaben der Form $\langle x, r(x, t-1), t, K_i\rangle$ aufgerufen.*

Beweis der Behauptung. Wir führen Induktion über t:

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

$t \rightsquigarrow t + 1$: Da r^- zu Beginn des $(t+1)$-ten Durchlaufs auf den Wert von r gesetzt wird, müssen wir zeigen, dass r im t-ten Durchlauf auf $r(x, t)$ hochgezählt wird. Nach Induktionsvorausset-

zung wird N_0 im t-ten Durchlauf nur mit Eingaben der Form $\langle x, r(x,t-1), t, K_i\rangle$ aufgerufen. Da N_0 wegen Beh. 1 auf all diesen Eingaben strong ist und keine dieser Simulationen im Zustand q_h endet (andernfalls würde N' sofort stoppen), werden alle in $\leq t$ Schritten erreichbaren Konfigurationen K_i als solche erkannt und somit wird r tatsächlich auf den Wert $r(x,t)$ hochgezählt. □

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

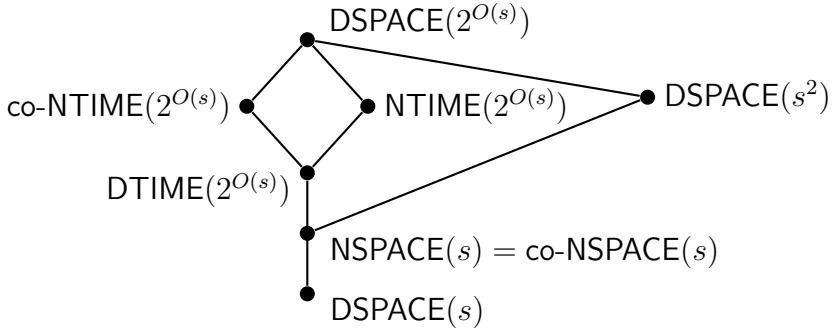
Beweis der Behauptung. Wir wissen bereits, dass im t-ten Durchlauf der repeat-Schleife r den Wert $r(x,t)$ und r^- den Wert $r(x,t-1)$ erhält. Wird daher die repeat-Schleife nach t_e Durchläufen verlassen, so gilt $r = r^- = r(x,t_e) = r(x,t_e-1)$.
Angenommen $r(x,t_e) < |\{K | K_x \underset{N}{\rightarrow}^* K\}|$. 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 29 hat $N'(x)$ nur dann eine verwerfende Rechnung, wenn im letzten Durchlauf der repeat-Schleife alle erreichbaren Konfigurationen K gefunden wurden und sich darunter keine akzeptierende Endkonfiguration befand. Dies impliziert $x \notin L$. Die Umgekehrung, dass $N'(x)$ für alle $x \notin L$ eine verwerfende Rechnung hat, folgt analog. ■

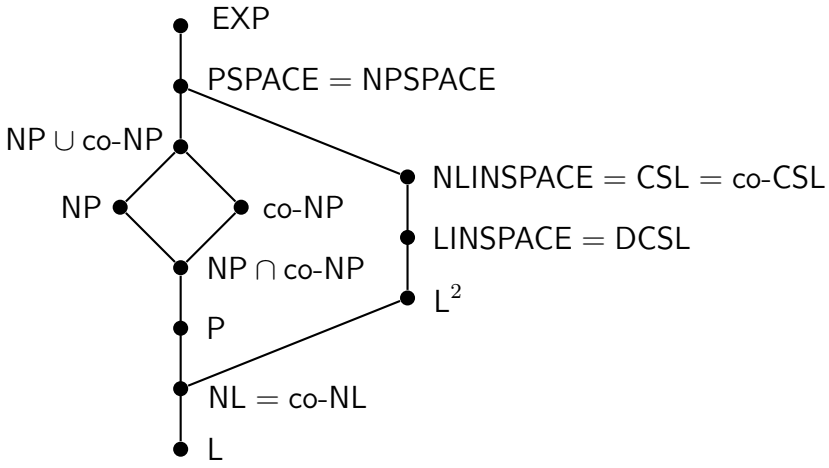
Korollar 30.

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

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



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



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

4 Hierarchiesätze

4.1 Unentscheidbarkeit mittels Diagonalisierung

Wir benutzen folgende Kodierung (Gödelisierung) von 1-DTMs $M = (Q, \Sigma, \Gamma, \delta, q_0)$. O.B.d.A. sei $Q = \{q_0, q_1, \ldots, q_m\}$, $\{0, 1, \#\} \subseteq \Sigma$ und $\Gamma = \{a_1, \ldots, a_\ell\}$ (also z.B. $a_1 = \sqcup$, $a_2 = \triangleright$, $a_3 = 0$, $a_4 = 1$ etc.). Dann kodieren wir jedes $\alpha \in Q \cup \Gamma \cup \{q_\text{h}, q_\text{ja}, q_\text{nein}, L, R, N\}$ 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:

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

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 31. *Die Diagonalsprache*

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

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

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

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

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

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

so führt dies wegen

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

zu einem Widerspruch. ■

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

Beweis. Betrachte die Diagonalsprache

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

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

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

Dies führt jedoch auf einen Widerspruch:

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

■

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

4.2 Das Gap-Theorem

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

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

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

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

Betrachte hierzu das Prädikat

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

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 32 definierte Sprache D_g nicht in Zeit $h(g(n))$ entscheidbar.

4.3 Zeit- und Platzhierarchiesätze

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

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

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

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

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

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

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

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

Korollar 35. *(Zeithierarchiesatz)*
Für jede echte Komplexitätsfunktion $g(n) \geq n+2$ 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 34. Dabei vermindert M' die Anzahl der Einsen auf dem 4. Band nach jedem simulierten Schritt von $M(x)$ um 1. M' bricht die Simulation ab, sobald M stoppt oder der Zähler auf Band 4 den Wert 0 erreicht. M' hält genau dann im Zustand q_{ja}, wenn die Simulation von M im Zustand q_{ja} endet. Nun ist leicht zu sehen, dass M' $\mathcal{O}(n \cdot g(n)^2)$-zeitbeschränkt ist und die Sprache D_g entscheidet. ■

Korollar 36.

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

Beweis.

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

■

Für Platzklassen erhalten wir eine noch feinere Hierarchie.

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

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

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

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

Betrachte folgende Offline-DTM M:

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


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

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

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

Korollar 38.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

- die Sprache

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

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

Betrachte folgende NTM M:

```
1  input: 0^n
2    x := ε; s := 0
3    while h(s + |x|) ≤ n do
4      s := h(s + |x|)
5      x := x^+
6    if n < h(s + |x|) − 1 then (* s = s(x) ≤ n < s(x^+) − 1 *)
7      akz. falls M_{i(x)}(0^{n+1}) in ≤ f(n)/|⟨M_{i(x)}⟩| Schritten akz.
8    else (* s = s(x) ≤ n = s(x^+) − 1 *)
9      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 $time_{M_i}(0^n) \leq cg(n)$ (siehe Übungen; Simulation von NTMs durch 2-NTMs). Wählen wir nun $k \geq 0$ so groß, dass für $x = 0^k \# \langle M_i \rangle$ und alle $n \geq s(x)$

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

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

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

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

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

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

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

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

5 Reduktionen

5.1 Logspace-Reduktionen

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

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

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

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

Definition 43.

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

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

c) *$\mathcal{C}$ heißt **abgeschlossen unter $\leq$**, falls für alle Sprachen A, B gilt:*

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

Lemma 44.

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

$$\mathcal{C} = \{A \mid A \leq L\}.$$

Beweis. Siehe Übungen. ■

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

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

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

Hamiltonkreisproblem für Digraphen (DiHam):

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

Erfüllbarkeitsproblem für boolesche Formeln (Sat):

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

Beispiel 46. *Um DiHam auf Sat zu reduzieren, benötigen wir eine Funktion $f \in \mathsf{FL}$, die einen Digraphen $G = (V, E)$ so in eine Formel $f(G) = F_G$ transformiert, dass F_G genau dann erfüllbar ist, wenn G hamiltonsch ist.*

Wir konstruieren F_G über den Variablen $x_{1,1}, x_{1,2}, \ldots, x_{n,n}$, wobei $x_{i,j}$ für die Aussage steht, dass $j \in V = \{1, \ldots, n\}$ der i-te Knoten auf dem Hamiltonkreis ist. Die folgenden Klauseln stellen dann sicher, dass die Relation $\pi = \{(i, j) \mid x_{i,j} = 1\}$ eine Permutation in S_n ist, die eine Rundreise in G beschreibt:

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

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

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

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

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) verifizieren, dass π eine Funktion $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$ ist. Bedingung c) besagt, dass π surjektiv (und damit auch bijektiv) ist, und d) sorgt dafür, dass der durch π beschriebene Kreis entlang der Kanten von G verläuft. Bilden wir daher $F_G(x_{1,1}, \ldots, x_{n,n})$ als Konjunktion dieser

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

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

5.2 Polynomielle Schaltkreiskomplexität

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

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

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

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

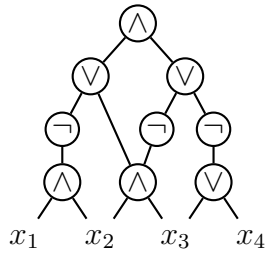
Der Schaltkreis c berechnet die boolesche Funktion $\boldsymbol{c : a \mapsto g_m(a)}$. *Er heißt* ***erfüllbar****, wenn es eine Eingabe $a \in \{0,1\}^n$ mit $c(a) = 1$ gibt.*

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

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

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

Beispiel 48.



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

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

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

Definition 49.

a) *Eine Sprache $L \subseteq \{0,1\}^*$ hat* ***polynomielle Schaltkreiskomplexität*** *(kurz: $\boldsymbol{L \in \mathsf{PSK}}$), falls es ein $c \geq 1$ und eine Folge von booleschen Schaltkreisen c_n, $n \geq 0$, mit n Eingängen der Größe $size(c_n) \leq n^c + c$ gibt, so dass für alle $x \in \{0,1\}^*$ gilt:*

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

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

Die Turingmaschine ist ein ***uniformes*** Rechenmodell, da alle Instanzen eines Problems von einer einzigen Maschine entschieden werden. Im Gegensatz hierzu stellen Schaltkreise ein ***nichtuniformes*** Berechnungsmodell dar, da für jede Eingabegröße n ein anderer Schaltkreis c_n verwendet wird. Um mit Schaltkreisen eine unendliche Sprache entscheiden zu können, wird also eine unendliche Folge c_n, $n \geq 0$, von Schaltkreisen benötigt.

5.3 P-vollständige Probleme

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

Auswertungsproblem für boolesche Schaltkreise (CirVal):

Eingabe: 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):

Eingabe: Ein boolescher Schaltkreis c mit n Eingängen.
Gefragt: Ist c erfüllbar?

Im folgenden Beispiel führen wir die Lösung des Erreichbarkeitsproblems in gerichteten Graphen auf die Auswertung von booleschen Schaltkreisen zurück. In den Übungen werden wir sehen, dass Reach NL-vollständig ist.

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

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

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

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

Dabei soll gelten:

$g_{i,j,k'} = 1$ $\Leftrightarrow$ *in G existiert ein Pfad von i nach j, der keinen Knoten $l > k'$ durchläuft,*

$h_{i,j,k} = 1$ $\Leftrightarrow$ *in G existiert ein Pfad von i nach j, der den Knoten k und keinen Knoten $l > k$ durchläuft.*

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

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

und für $k = 1, 2, \ldots, n$ gilt

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

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

Der in Beispiel 50 konstruierte Schaltkreis hat Tiefe $2n$. In den Übungen werden wir sehen, dass sich Reach auch auf die Auswertung eines Schaltkreises der Tiefe $O(\log^2 n)$ reduzieren lässt. Als nächstes leiten wir Vollständigkeitsresultate für CirVal und CirSat her.

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

Beweis. Es ist leicht zu sehen, dass CirVal $\in$ 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

$$\begin{aligned} &t_{i,j,k} = \\ &\quad c_k(t_{i-1,j-1,1} \cdots t_{i-1,j-1,m}, t_{i-1,j,1} \cdots t_{i-1,j,m}, t_{i-1,j+1,1} \cdots t_{i-1,j+1,m}). \end{aligned}$$

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

- Für jeden der $n^c + c + 2 + 2(n^c + c - 1) = 3(n^c + c)$ Randeinträge $T_{i,j}$ mit $i = 1$ oder $j \in \{1, n^c + c + 2\}$ enthält c_x m konstante Gatter $c_{i,j,k} = t_{i,j,k}$, $k = 1, \ldots, m$, die diese Einträge kodieren.

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

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

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

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

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

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

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

Korollar 53 (Savage 1972)**.**
Es gilt $\mathsf{P} \subseteq \mathsf{PSK}$.

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 $\mathsf{EXPSPACE}$ 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. $\mathsf{PSK} \not\subseteq \mathsf{RE}$.

5.4 NP-vollständige Probleme

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Im Beweis von Satz 58 haben wir gezeigt, dass für jede NP-Sprache A eine (k,q)-balancierte Sprache $B \subseteq \Sigma^*$ in P mit

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

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

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

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

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

■

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 $C_i = \bigvee_{j=1}^{k_i} \ell_{ij}$ von **Literalen** $\ell_{ij} \in \{x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n\}$ 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 $C = \bigvee_{j=1}^{k} \ell_j$ von Literalen wird auch als ***Klausel*** bezeichnet. Klauseln werden meist als Menge $C = \{\ell_1, \ldots, \ell_k\}$ der zugehörigen Literale und KNF-Formeln als Menge $F = \{C_1, \ldots, C_m\}$ ihrer Klauseln dargestellt.

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

Eingabe: 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.

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

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

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

Not-All-Equal-Satisfiability (NAESAT):

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

Satz 63. NaeSat $\in$ NPC.

Beweis. NaeSat $\in$ 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.

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

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

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

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

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

Dabei heißen zwei Literale **komplementär**, wenn das eine die Nega-

tion des anderen ist. Nun gilt

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

■

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. ■

5.5 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 $\{\ell_1, \ell_2\}$ von F die beiden Kanten $(\bar{\ell}_1, \ell_2)$ und $(\bar{\ell}_2, \ell_1)$ 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 RE ∪ co-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. co-NP $\subseteq$ PP.

Beweis. Sei $A \in$ co-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 $x \in \Sigma^*$:

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

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

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

■

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 $A \in$ 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 $A \in$ 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 $k = \text{kgV}(2, \ldots, c)$ 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

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

■

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):

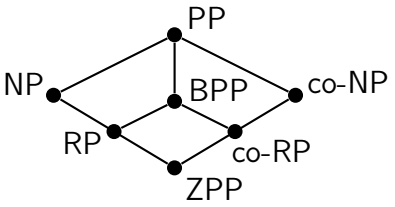
Eingabe: Eine boolsche Formel $F(x_1, \ldots, x_n)$.
Gefragt: Wird F von mindestens 2^{n-1} Belegungen erfüllt?

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

Beweis. Die Inklusion von links nach rechts ist klar, da

- wir eine ZPP-TM leicht in eine RP-TM verwandeln können, indem wir q_{h} durch q_{nein} ersetzen, und da
- $\mathsf{ZPP} = \mathsf{co\text{-}ZPP}$ ist.

Für die umgekehrte Richtung sei A eine Sprache in $\mathsf{RP} \cap \mathsf{co\text{-}RP}$. 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. Dann gilt

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

Da M_A nur Eingaben $x \in A$ und $M_{\bar{A}}$ nur Eingaben $x \notin A$ akzeptieren kann, tritt die Kombination $M_A(x) = 1$ und $M_{\bar{A}}(x) = 1$ nicht auf. Zudem ergeben sich für die PP-TM M, die bei Eingabe x die beiden PP-TMs $M_A(x)$ und $M_{\bar{A}}(x)$ simuliert und sich gemäß der Tabelle

	$M_A(x) = 0$	$M_A(x) = 1$
$M_{\bar{A}}(x) = 0$	?	1
$M_{\bar{A}}(x) = 1$	0	–

verhält, folgende Äquivalenzen:

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

Somit erhalten wir die folgenden Implikationen:

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

Dies zeigt, dass M eine ZPP-TM für A ist, da

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

für alle Eingaben x gilt. ■

6.2 Anzahl-Operatoren

Viele Beweise über probabilistische Komplexitätsklassen lassen sich sehr elegant führen (und auch verallgemeinern), wenn wir sie als Anzahlklassen charakterisieren.

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

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

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

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

Zudem definieren wir die Operatoren R und BP durch

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

und

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

Wie wir bereits wissen, gilt $\exists^p \cdot \mathsf{P} = \mathsf{NP}$, was $\forall^p \cdot \mathsf{P} = \mathsf{co\text{-}NP}$ 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 $\Gamma_k = \{0, \ldots, k-1\}$, um die Rechnungen von M bei Eingabe $x \in \Sigma^n$ zu beschreiben. Hierzu ordnen wir dem String $y = y_1 \cdots y_{q(n)} \in \Gamma_k^{q(n)}$ 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

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

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

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

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

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

also ist $\#B(x) \notin (k^{q(n)}/3, 2 \cdot k^{q(n)}/3)$, d.h. B ist zweiseitig und es folgt $A = \exists^{\geq 1/2} B \in \mathsf{BP} \cdot \mathsf{P}$. Ist M eine RP-TM, so gilt

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

also ist $\#B(x) \notin [1, k^{q(n)}/2)$, d.h. B ist einseitig und es folgt $A = \exists^{\geq 1/2} B \in \mathsf{R} \cdot \mathsf{P}$. ∎

6.3 Verstärkung der Korrektheit

In diesem Abschnitt zeigen wir, dass sich die ***Korrektheitswahrscheinlichkeit*** $\Pr[M(x) = A(x)]$ für RP-, ZPP- und BPP-Maschinen M auf einen Wert $\geq 1 - 2^{-r(|x|)}$ vergrößern lässt, wobei r ein beliebig wählbares Polynom ist.

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 Reduzierbarkeit*** $A \leq_c B$ *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 $\mathsf{co\text{-}NP}$ unter diesen Reduktionen abgeschlossen sind. Als nächstes zeigen wir, wie sich die Korrektheit von RP-TMs verstärken lässt.

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

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

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

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

Dann ist die Sprache

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

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

$$x \in A \Rightarrow \Pr_{y \in_R \Gamma_k^{q(n)}}[x\#y \notin B] < 1/2 \Rightarrow \Pr_{z \in_R \Gamma_k^{p(n)}}[x\#z \notin C] < 2^{-r(n)}$$
$$x \notin A \Rightarrow \Pr_{y \in_R \Gamma_k^{q(n)}}[x\#y \in B] = 0 \Rightarrow \Pr_{z \in_R \Gamma_k^{p(n)}}[x\#z \in C] = 0. \qquad \blacksquare$$

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

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

Als Folgerung erhalten wir die Inklusion $\mathsf{RP} \subseteq \mathsf{BPP}$. Ganz analog lässt sich die Zuverlässigkeit einer ZPP-TM M verbessern, indem wir sie $r(n)$-mal laufen lassen und nur dann ? ausgeben, wenn $M(x)$ jedesmal ? ausgibt.

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*.*

Um die Korrektheit von BPP-TMs zu verstärken, benötigen wir das folgende Lemma.

Lemma 83. *Sei* E *ein Ereignis, das mit Wahrscheinlichkeit* $1/2 - \epsilon$ *für ein* $\epsilon \geq 0$ *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 $i > m/2$,

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

Wegen

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

erhalten wir somit

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

■

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

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

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

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

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

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

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

$$\begin{aligned}\Pr[A(x) \neq C(x\#z)] &= \Pr[|\{i : A(x) \neq B(x\#y_i)\}| > t(n)] \\ &\leq 1/2 \underbrace{(1 - 4/36)}_{8/9 = 2^{-\log_2(9/8)}}^{t(n)} \leq 2^{-r(n)}\end{aligned}$$

■

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 existiert für das Polynom $r(n) = n + 1$ eine (k,p)-balancierte Sprache $C \in \mathcal{C}$, so dass für alle $x \in \{0,1\}^*$, $|x| = n$, gilt:

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

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

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

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 52 eine

Funktion $f \in \mathsf{FL}$ die bei Eingabe 1^n einen Schaltkreis $f(1^n) = c'_n$ ausgibt mit

$$c'_n(xbin(\#z)) = C'(xbin(\#z)) \text{ für alle } x \in \{0,1\}^n \text{ und } z \in \Gamma_k^{p(n)}.$$

Folglich sind auch die n-stelligen booleschen Funktionen

$$f_{z_n} \colon x \mapsto c'_n(xbin(\#z_n)) = B(x)$$

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 $\mathcal{C}$ 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 $A \subseteq \Sigma^*$ eine Sprache in $\mathsf{co}\text{-}\exists^p \cdot \mathcal{C}$ und sei B eine (k,q)-balancierte Sprache in $\mathcal{C}$ mit $\bar{A} = \exists^p B$. Betrachte die Sprache

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

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

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

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

(ii) Sei $A \subseteq \Sigma^*$ eine Sprache in $\mathsf{co}\text{-}\mathsf{BP} \cdot \mathcal{C}$ und sei $B \in \mathcal{C}$ eine (k,q)-balancierte zweiseitige Sprache mit $\bar{A} = \exists^{\geq 1/2} B$. Dann ist die (k,q)-balancierte Sprache

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

in $\mathsf{co}\text{-}\mathcal{C}$ ebenfalls zweiseitig und es gilt $A = \exists^{\geq 1/2} \hat{B} \in \mathsf{BP} \cdot \mathsf{co}\text{-}\mathcal{C}$.

(iii) Sei $A \subseteq \Sigma^*$ eine Sprache in $\oplus \cdot \mathcal{C}$ und sei $B \in \mathcal{C}$ eine (k,q)-balancierte Sprache mit $A = \oplus B$. Betrachte die $(k, q+1)$-balancierte Sprache

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

Dann gilt $\#B'(x) = \#B(x) + 1$, d.h. $\#B(x)$ und $\#B'(x)$ haben unterschiedliche Parität. Dies zeigt $A = \overline{\oplus B'} \in \mathsf{co}\text{-}\oplus \cdot \mathcal{C}$.

Da wir o.B.d.A. annehmen können, dass $q \geq 1$ (da $\mathcal{C}$ unter $\leq_m^{log}$ abgeschlossen ist) und k (und somit auch $k^{q(n)}$) gerade ist, haben $\#B(x)$ und $\#\hat{B}(x)$ gleiche Parität und es folgt $A = \oplus B = \oplus \hat{B} \in \oplus \cdot \mathsf{co}\text{-}\mathcal{C}$. ■

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

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

Dann gilt $B', B'' \leq_m^{log} B$, also $B', B'' \in \mathcal{C}$. 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

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

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

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

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

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

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

$$\begin{aligned} x \in A' &\Leftrightarrow f(x) \in A \\ &\Leftrightarrow \#B(f(x))/k^{q(|f(x)|)} \geq 1/2 \\ &\Leftrightarrow \#B''(x)/k^{p(|x|)} \geq 1/2. \end{aligned}$$

Da zudem mit B auch B'' einseitig ist, folgt $A' = \exists^{\geq 1/2} B' \in \mathsf{R} \cdot \mathcal{C}$. Die Fälle $\mathsf{Op} \in \{\forall^p, \mathsf{BP}, \exists^{\geq 1/2}\}$ folgen analog, wobei für $\mathsf{Op} = \mathsf{BP}$ noch zu beachten ist, dass mit B auch B'' zweiseitig ist. ■

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

Beweis. Siehe Übungen. ■

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

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

Beweis. Sei $A \in \mathsf{Op} \cdot \mathcal{C}$ mittels einer (k,q)-balancierten Sprache $B \in \mathcal{C}$ und und gelte $A' \leq_{maj} A$ mittels einer FL-Funktion $f : x \mapsto \langle y_1, \ldots, y_{m(|x|)} \rangle$. Da $\mathcal{C}$ unter majority-Reduktionen (und damit auch unter $\leq_m^{log}$) abgeschlossen ist, ist nach Satz 89 auch $\mathsf{Op} \cdot \mathcal{C}$ 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

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

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

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

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

ist. Daher gilt für alle x,

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

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

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

Satz 92. *Für jede Klasse $\mathcal{C}$, die unter majority-Reduktionen abgeschlossen ist, gilt*

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

Beweis. Sei $A\in\mathsf{BP}\cdot\mathsf{BP}\cdot\mathcal{C}$. Da mit $\mathcal{C}$ auch $\mathsf{BP}\cdot\mathcal{C}$ unter majority-Reduktionen abgeschlossen ist, existiert eine (k_1,q_1)-balancierte Sprache $B\in\mathsf{BP}\cdot\mathcal{C}$, 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 existiert eine (k_2,q_2)-balancierte Sprache $B'\in\mathcal{C}$, 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 $\mathcal{C}$, 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 $A\in\mathsf{BP}\cdot\mathcal{C}$. ■

Korollar 93. *Sei $\mathcal{C}$ eine unter majority-Reduktionen abgeschlossene Sprachklasse. Dann sind die Klassen $\mathsf{BP}\cdot\exists^p\cdot\mathcal{C}$ und $\mathsf{BP}\cdot\forall^p\cdot\mathcal{C}$ 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 $\mathcal{C}$, die unter disjunktiven Reduktionen abgeschlossen ist, die Gleichheit $\mathsf{R}\cdot\mathsf{R}\cdot\mathcal{C}=\mathsf{R}\cdot\mathcal{C}$.

7 Die Polynomialzeithierachie

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

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

Die Vereinigung aller Stufen der Polynomialzeithierachie bezeichnen wir mit PH,

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

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

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

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

Als Folgerung hieraus ergibt sich, dass eine NP-vollständige Sprache nicht in P (bzw. $\text{co-}\mathsf{NP}$) enthalten ist, außer wenn PH auf P (bzw. NP) kollabiert. In den Übungen werden wir sehen, dass unter der Voraussetzung $\mathsf{PH} \neq \Sigma_2^p$ keine NP-vollständige Sprache in PSK enthalten ist. Allgemeiner liefert die Polynomialzeithierarchie eine Folge von stärker werdenden Hypothesen der Form $\mathsf{PH} \neq \Sigma_k^p$ für $k = 0, 1, 2, \ldots$, die mit $\Sigma_0^p \subsetneq \Sigma_1^p \subsetneq \cdots \subsetneq \Sigma_k^p \subsetneq \Sigma_{k+1}^p$, also für $k = 0$ mit $\mathsf{P} \neq \mathsf{NP}$ und für $k = 1$ mit $\mathsf{NP} \neq \text{co-}\mathsf{NP}$ 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 $\mathcal{C}$, die unter majority-Reduktionen abgeschlossen ist, gilt*

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

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

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

Setzen wir $B_x = \{y \in \Gamma_k^{p(n)} \mid x\#y \in B\}$, 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_k, d.h. $i \oplus j = (i + j) \bmod k$, und für zwei Strings $y, z \in \Gamma_k^*$ derselben Länge $|y| = |z| = l$ sei

$$y_1 \cdots y_\ell \oplus z_1 \cdots z_\ell = v_1 \cdots v_\ell \in \Gamma_k^l \text{ mit } v_i = y_i \oplus z_i \text{ für } i = 1, \ldots, l$$

Für $v \in \Gamma_k^{p(n)}$ sei $B_x \oplus v = \{y \oplus v \mid y \in B_x\}$. Dann hat die Menge $B_x \oplus v$ die gleiche Mächtigkeit wie B_x. Zudem gilt

$$u \in B_x \oplus v \Leftrightarrow \exists y \in B_x \colon \underbrace{u = y \oplus v}_{\Leftrightarrow v = u \ominus y} \Leftrightarrow v \in u \ominus B_x,$$

wobei der Operator $\ominus$ analog zu $\oplus$ definiert ist und auch die Menge $u \ominus B_x = \{u \ominus y \mid y \in B_x\}$ die gleiche Mächtigkeit wie B_x hat.
Betrachte die Sprachen

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

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

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

in $\forall^p \cdot \mathcal{C}$ ist (wegen $B' \leq_d B$ folgt $B' \leq_{maj} B$, also $B' \in \mathcal{C}$).
Sei also $x \in A$ mit $|x| = n \geq 2$ und sei $z \in \Gamma_k^{p(n)}$ beliebig. Da $\#B(x) \geq (1 - k^{-n})k^{p(n)}$ ist, gilt für zufällig gewählte $u_1, \dots, u_{p(n)} \in_R \Gamma_k^{p(n)}$,

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

Daher gilt für ein zufällig gewähltes $u \in_R \Gamma_k^{p(n)^2}$,

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

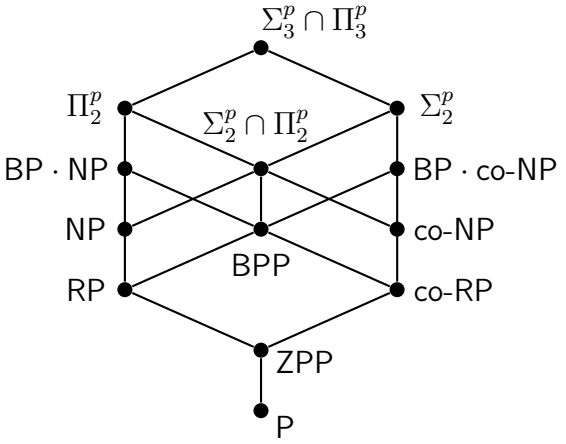
und somit $\#B''(x) \geq k^{p(n)^2}/2$.
Für den Nachweis der 2. Implikation nehmen wir an, dass $\#B''(x) > 0$ für ein x der Länge n mit $k^n > p(n)$ ist. Dann existiert eine Folge von Wörtern $u_1, \dots, u_{p(n)} \in \Gamma_k^{p(n)}$, so dass jedes $z \in \Gamma_k^{p(n)}$ in mindestens einer der Mengen $B_x \oplus u_i$ enthalten ist. Da die Mengen $B_x \oplus u_i$ 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 für $\mathcal{C} = \mathsf{co\text{-}NP}$ die Inklusion $\mathsf{BP} \cdot \mathsf{co\text{-}NP} \subseteq \mathsf{R} \cdot \forall^p \cdot \mathsf{co\text{-}NP} = \mathsf{R} \cdot \mathsf{co\text{-}NP}$. Wegen $\mathsf{R} \cdot \mathsf{co\text{-}NP} \subseteq \mathsf{BP} \cdot \mathsf{co\text{-}NP}$ führt dies auf die folgende Inklusionen.

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



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)
$A \in \mathsf{co\text{-}NP}$	$\mathsf{PH} = \mathsf{NP}$ (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 Fragen an das Orakel A, deren Beantwortung jeweils nur einen Rechenschritt kostet, abgerufen werden. Auf diese Weise erhalten wir zu jeder Komplexitätsklasse $\mathcal{C}$ eine relativierte Version $\mathcal{C}^A$, in der alle Probleme enthalten sind, die relativ zum Orakel A innerhalb der durch $\mathcal{C}$ 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 (dies geschieht innerhalb eines einzigen Rechenschritts). Die unter dem Orakel A arbeitende OTM wird mit M^A bezeichnet und die von M^A **akzeptierte Sprache** ist $L(M^A)$.*

*Anstelle von Sprachorakel können funktionale Orakel f benutzt werden. In diesem Fall ist M mit einem zusätzlichen read-only **Orakelantwortband** ausgestattet, auf dem jede Orakelfrage y innerhalb eines Rechenschrittes mit $f(y)$ beantwortet wird (in diesem Fall gilt immer $q = q_+$). Wir nennen M **nichtadaptiv**, falls die Fragen von M nicht von den Antworten auf zuvor gestellte Fragen abhängen.*

Wir verlangen, dass OTMs vorgegebene Ressourcenschranken unab-

hängig vom benutzten Orakel einhalten.

Definition 98. *Die **Rechenzeit** einer OTM M bei Eingabe $x \in \Sigma^*$ ist*

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

wobei augehend von einer Fragekonfiguration $K_?$ alle möglichen Antwortkonfigurationen als Folgekonfiguration zulässig sind.
*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 ODTM bzw. ONTM) M mit Orakel A in Zeit $t(n)$ entscheidbar sind, in den relativierten Komplexitätsklassen

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

und

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

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 ODTM (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. Für $A \in \mathsf{P}^B$ schreiben wir auch $A \leq_T^P B$ und sagen A ***ist Turing-reduzierbar auf*** B.
Ebenso wie DTMs und NTMs lassen sich auch PTMs (also probabilistische TMs) oder Transducer mit einem Orakelmechanismus ausstatten, wodurch wir OPTMs bzw. 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 $\mathcal{C}$ sei

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

Für $\mathsf{P}^{\mathcal{C}}$ (bzw. P^A) schreiben wir auch $\mathsf{P}(\mathcal{C})$ (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) $\mathsf{P}^{\mathsf{NP} \cap \mathsf{co\text{-}NP}} = \mathsf{NP} \cap \mathsf{co\text{-}NP}$ *und* $\mathsf{NP}^{\mathsf{NP} \cap \mathsf{co\text{-}NP}} = \mathsf{NP}$,

(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 $\mathsf{NP}^{\mathsf{NP} \cap \mathsf{co\text{-}NP}} \subseteq \mathsf{NP}$. Sei $L = L(M^A)$ für eine NP-OTM M und sei A ein Orakel in $\mathsf{NP} \cap \mathsf{co\text{-}NP}$.

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 sobald 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. Andernfalls 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 $\Sigma_{k+1}^p \subseteq \mathsf{NP}^{\Sigma_k^p}$, da nun $A \in \Pi_k^p$ bzw. $\bar{A} \in \Sigma_k^p$ ist und daher $L = L(M^{\bar{A}}) \in \mathsf{NP}^{\Sigma_k^p}$ folgt.
Für die Inklusion von $\mathsf{NP}^{\mathsf{NP}}$ in Σ_2^p sei $L = L(M^A)$ für eine NP-OTM M und ein NP-Orakel A. Zu A existieren ein Polynom q und eine (k,q)-balancierte Sprache $B \in \mathsf{P}$ mit

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

Zudem sei p eine polynomielle Zeitschranke für M und sei $k = \text{kgV}(1,2,\ldots,c)$, wobei $c \geq 1$ der maximale Verzweigungsgrad von M ist. 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.
Dann gilt

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

wobei C alle Strings $x\#r z_1 \ldots z_{p(n)}$ enthält, so dass

- $r \in \Gamma^{p(n)}$ eine akzeptierende Rechnung α von $M(x)$ beschreibt, bei 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.

Wegen $A \in \mathsf{NP}$ liegt C in $\text{co-}\mathsf{NP}$ und es folgt $L \in \exists^p \cdot \text{co-}\mathsf{NP} = \Sigma_2^p$.
Mit demselben Argument folgt auch die Inklusion $\mathsf{NP}^{\Sigma_k^p} \subseteq \Sigma_{k+1}^p$, da nun A in Σ_k^p (bzw. $\bar{A}$ in Π_k^p) und B in Π_{k-1}^p liegen. Folglich liegt C nun in Π_k^p und somit ist $L = L(M^A) = \exists^p C$ in $\exists^p \cdot \Pi_k^p = \Sigma_{k+1}^p$ enthalten. ■

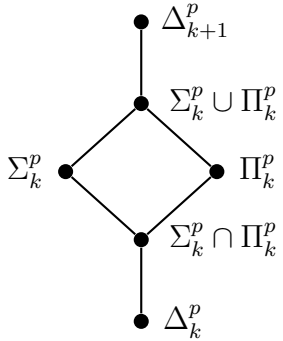
Der vorige Satz liefert folgende Charakterisierung für die k-te Stufe der Polynomialzeithierachie ($k \geq 1$):

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

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

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 entscheidbare 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 $L(B) \in \mathsf{NP}^B$ für jedes Orakel B gilt, genügt es, mittels Diagonalisierung ein Orakel B mit $L(B) \notin \mathsf{P}^B$ zu konstruieren.

Sei M_1,M_2,... eine Aufzählung von P-OTMs, so dass $\{L(M_i^C) \,|\, i \geq 1\}$ alle Sprachen in P^C über dem Alphabet $\{0\}$ enthält. Dabei nehmen wir an, dass die Funktion $0^i \mapsto \langle M_i \rangle$ berechenbar und die Laufzeit von M_i durch das Polynom $n^i + i$ beschränkt ist:

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

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

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

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

Für $i \geq 1$ wird B_i in Stufe i so definiert, dass 0^{n_i} in $L(M_i^{B_{i-1}}) \triangle L(B_i)$ enthalten ist. Da $M_i^{B_{i-1}}(0^{n_i})$ zudem keine Orakelfrage in $B_i \setminus B_{i-1}$ und wegen $n_j > (n_i)^i + i$ für alle $j \geq i+1$ auch keine Orakelfrage in $B_j \setminus B_i$ stellt, folgt $0^{n_i} \in L(M_i^B) \triangle L(B)$ für alle $i \geq 1$ 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 Inklusionen

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

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 co-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.

a) Die Anzahl der akzeptierenden Rechnungen einer NTM M bei Eingabe x bezeichnen wir mit $\#M(x)$.

b) Die Funktionenklasse $\{\#M \mid M$ ist eine NP-TM$\}$ bezeichnen wir mit $\#\mathsf{P}$.

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

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

d) Eine Sprache $L \subseteq \Sigma^*$ 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 $\oplus$P-vollständig ist (siehe Übungen).

Eingabe: 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 Der Satz von Valiant und Vazirani

Bei manchen Anwendungen ist der Bereich der tatsächlich auftretenden 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 für (A, B), 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 (1SAT, SAT) *zu lösen, genügt es, die Erfüllbarkeit für alle Eingaben $F \in$ 1SAT richtig zu entscheiden. Somit ist jede Sprache L mit* USAT $\subseteq L \subseteq$ SAT *eine Lösung für* (1SAT, SAT). *Neben* USAT *und* SAT *ist z.B. auch* $\oplus$SAT *eine Lösung.* ◁

Als nächstes zeigen wir, dass SAT (und damit jedes NP Problem) auf jede Lösung von (1SAT, 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:*

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

Beispiel 107. *Für jede einseitige $(2,q)$-balancierte Sprache B ist $A = \exists^p B$ auf B randomisiert reduzierbar mittels $f : x\#y \to x\#y$:*

$$\begin{aligned} x \in A &\Rightarrow \Pr_{y \in_R \{0,1\}^{q(n)}}[x\#y \in B] \geq 1/2, \\ x \notin A &\Rightarrow \Pr_{y \in_R \{0,1\}^{q(n)}}[x\#y \in B] = 0. \end{aligned}$$

Dies gilt auch, wenn B nur schwach einseitig ist.

Ist umgekehrt eine Sprache A auf B randomisiert reduzierbar, so folgt $A = \exists^p B' \in \mathsf{R}' \cdot \{B'\}$ für die schwach einseitige Sprache

$$B' = \{x\#y \mid y \in \{0,1\}^{q(n)}, f(x\#y) \in B\}. \qquad \triangleleft$$

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

Definition 108. *$Lin(n,k)$ 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 109. *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 110. *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 $H_0 = \{h \mid h_i(x) = 0\}$ und $H_1 = \{h \mid h_i(x) = 1\}$ 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 die beiden Werte $h(x)$ und $h(x')$ im Fall $x \neq x'$ 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 $H'_0 = \{h \mid h_i(x') = y_i \wedge h_i(x) = 0\}$ und $H'_1 = \{h \mid h_i(x') = y_i \wedge h_i(x) = 1\}$ gleich groß, woraus

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

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 Strings $y, y' \in \{0,1\}^k$ 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 111. *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*

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

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 110 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 112 (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. Betrachte die Reduktionsfunktion

$$f : F \# z \mapsto F_z = F \wedge \bigwedge_{i=1}^{k_z} \bigoplus_{a_{ij}=1} x_j,$$

wobei die ersten $m = \lceil \log_2(n+1) \rceil$ Bit von $z \in \{0,1\}^{m+(n+1)n}$ eine Zahl

$$k_z = 1 + \Sigma_{i=1}^{m} z_i 2^{i-1} \in \{1, \ldots, 2^m\}$$

und die restlichen $(n+1)n$ Bit von z eine Matrix

$$A_{h_z} = (a_{ij}) \in \{0,1\}^{(n+1) \times n}$$

kodieren.

Dann wird F_z unter einer Belegung $a \in \{0,1\}^n$ genau dann wahr, wenn $F(a) = 1$ ist und die ersten k_z Bit von $h_z(a)$ den Wert 1 haben.

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

$$\begin{aligned} \Pr[f(F_z) \in L] &\geq \Pr[f(F_z) \in \text{USAT}] \\ &\geq \underbrace{\Pr[k_z = k]}_{2^{-m} > 1/2(n+1)} \cdot \underbrace{\Pr[F_z \in \text{USAT} \,|\, k_z = k]}_{\geq 2/9 \text{ (nach Lemma 111)}} \\ &> 1/9(n+1). \end{aligned}$$

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

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

Korollar 113. SAT *ist auf* USAT *und* $\oplus$SAT *randomisiert reduzierbar.*

Korollar 114. *Falls das Promise-Problem* $(1\text{SAT}, \text{SAT})$ *eine Lösung in* $\mathcal{C}$ *hat und* $\mathcal{C}$ *unter disjunktiven Reduktionen abgeschlossen ist, folgt* $\mathsf{NP} \subseteq \mathsf{R} \cdot \mathcal{C} \subseteq \mathsf{BP} \cdot \mathcal{C}$.

Beweis. Sei $B \in \mathcal{C}$ eine Lösung von $(1\text{SAT}, \text{SAT})$ und $f \in \mathsf{FL}$ eine randomisierte Reduktion von SAT auf B. Dann ist die Sprache $B' = \{x\#y \mid y \in \{0,1\}^{q(n)}, f(x\#y) \in B\}$ eine schwach einseitige Sprache in $\mathcal{C}$ und es folgt $\text{SAT} = \exists^p B' \in \mathsf{R}' \cdot \mathcal{C}$. Da $\mathcal{C}$ unter disjunktiven Reduktionen abgeschlossen ist, folgt $\mathsf{NP} \subseteq \mathsf{R}' \cdot \mathcal{C} = \mathsf{R} \cdot \mathcal{C} \subseteq \mathsf{BP} \cdot \mathcal{C}$. $\blacksquare$

Korollar 115. *Falls* $\mathcal{C}$ *unter* $\leq_m^{log}$ *und* $\oplus \cdot \mathcal{C}$ *unter disjunktiven Reduktionen abgeschlossen ist, gilt* $\exists^p \cdot \mathcal{C} \subseteq \mathsf{R} \cdot \oplus \cdot \mathcal{C} \subseteq \mathsf{BP} \cdot \oplus \cdot \mathcal{C}$.

Beweis. Sei $A = \exists^p B \in \exists^p \cdot \mathcal{C}$ für eine balancierte Sprache B. Wir konstruieren eine balancierte Sprache $B' \leq_m^{log} B$, so dass A auf $\oplus B' \in \oplus \cdot \mathcal{C}$ randomisiert reduzierbar und somit $A \in \mathsf{R}' \cdot \oplus \cdot \mathcal{C}$ ist. Wir können O.B.d.A. annehmen, dass B $(2,q)$-balanciert für ein Polynom q und $x\#0^{q(n)}$ für kein x in B ist. Sei B' die Sprache

$$B' = \{x\#z\#y \mid \begin{array}{c} x\#y \in B, z = bin_m(k)A_h \in \{0,1\}^{m+q(n)(q(n)+1)} \\ \text{und } 1^{k+1} \text{ ist ein Präfix von } h(y) \end{array}\},$$

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 es folgt für $z \in_R \{0,1\}^{m+q(n)(q(n)+1)}$,

$$\begin{aligned} x \in A &\Rightarrow \Pr[\#B'(x\#z) = 1] > 1/9(q(n)+1), \\ x \notin A &\Rightarrow \Pr[\#B'(x\#z) \geq 1] = 0. \end{aligned}$$

Folglich gilt für jede Lösung L des Promise-Problems

$$(\{x\#z \mid \#B'(x\#z) \leq 1\}, \{x\#z \mid \#B'(x\#z) \geq 1\})$$

wegen $\{x\#z \mid \#B'(x\#z) = 1\} \subseteq L \subseteq \{x\#z \mid \#B'(x\#z) \geq 1\}$,

$$\begin{aligned} x \in A &\Rightarrow \Pr[x\#z \in L] > 1/9(q(n)+1), \\ x \notin A &\Rightarrow \Pr[x\#z \in L] = 0, \end{aligned}$$

d.h. L ist schwach einseitig und es gilt $A = \exists^p B = \exists^p L$. Insbesondere folgt für $L = \oplus B'$, dass $A = \exists^p \oplus B' \in \mathsf{R}' \cdot \oplus \cdot \mathcal{C} \subseteq \mathsf{R} \cdot \oplus \cdot \mathcal{C} \subseteq \mathsf{BP} \cdot \oplus \cdot \mathcal{C}$, wobei die Inklusionen $\mathsf{R}' \cdot \oplus \cdot \mathcal{C} \subseteq \mathsf{R} \cdot \oplus \cdot \mathcal{C} \subseteq \mathsf{BP} \cdot \oplus \cdot \mathcal{C}$ aus der Abgeschlossenheit von $\oplus \cdot \mathcal{C}$ unter disjunktiven Reduktionen folgen (siehe Übungen). $\blacksquare$

Der folgende Satz zeigt, dass die Klasse $\oplus \cdot \mathcal{C}$ tatsächlich sehr robust ist, sofern $\mathcal{C}$ unter konjunktiven Reduktionen abgeschlossen ist. Als Folge davon genügt in vorigem Korollar die Voraussetzung, dass $\mathcal{C}$ unter konjunktiven Reduktionen abgeschlossen ist.

Satz 116. *Für jede Klasse* $\mathcal{C}$*, die unter konjunktiven Reduktionen abgeschlossen ist, gilt*

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

Beweis. Es reicht, die Inklusion $\oplus \mathsf{P}^{\oplus \cdot \mathcal{C}} \subseteq \oplus \cdot \mathcal{C}$ zu zeigen. Sei $L \in \oplus \mathsf{P}^{\oplus \cdot \mathcal{C}}$ via einer NP-OTM M und einem Orakel $A \in \oplus \cdot \mathcal{C}$. Da mit $\mathcal{C}$ auch $\oplus \cdot \mathcal{C}$ 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. Sei $r(n)$ eine polynomielle Zeitschranke für M und c der maximale Verzweigungsgrad von M. Sei B eine (k,p)-balancierte Sprache in $\mathcal{C}$ mit $A = \oplus B$, wobei wir $k \geq \text{kgV}(1,2,\ldots,c)$ annehmen, und seien B_0 und B_1 $(p+1)$-balancierte Sprachen in $\mathcal{C}$ mit $\#B_1(y) \equiv_2 A(y)$ und $\#B_0(y) \equiv_2 \bar{A}(y)$, also z.B.

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

Betrachte nun die $(k, r(n) + q(n)p(q(n)))$-balancierte Sprache

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

Dann ist B' konjunktiv auf B reduzierbar und somit in $\mathcal{C}$. Zudem gilt

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

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

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

genau dann ungerade, wenn alle Antworten korrekt sind. Daher folgt

$$\begin{aligned} x \in L &\Leftrightarrow \#M^A(x) \text{ ist ungerade}\\ &\Leftrightarrow \#B'(x) \text{ ist ungerade}\end{aligned}$$

und somit $L = \oplus B' \in \oplus \cdot \mathcal{C}$. ∎

10.2 Der 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 117. *Für $\mathsf{Op} \in \{\exists^p, \forall^p, \exists^{\geq 1/2}, \oplus\}$ und jede unter majority-Reduktionen abgeschlossene Klasse $\mathcal{C}$ gilt $\mathsf{Op} \cdot \mathsf{BP} \cdot \mathcal{C} \subseteq \mathsf{BP} \cdot \mathsf{Op} \cdot \mathcal{C}$.*

Beweis. Zu $L \in \mathsf{Op} \cdot \mathsf{BP} \cdot \mathcal{C}$ existiert eine (k, q)-balancierte Sprache $A \in \mathsf{BP} \cdot \mathcal{C}$ mit $L = \mathsf{Op}A$. Zu A existiert eine (k', p)-balancierte Sprache $B \in \mathcal{C}$ 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 $B' = \{x\#z\#y \mid x\#y\#z \in B\}$ in $\mathcal{C}$ 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 $L = \exists^{\geq 1/2}\mathsf{Op}B' \in \mathsf{BP} \cdot \mathsf{Op} \cdot \mathcal{C}$ ist. ∎

Satz 118 (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 117), $\exists^p \cdot \oplus\mathsf{P} \subseteq \mathsf{BP} \cdot \oplus\mathsf{P}$ (Kor. 115) und $\mathsf{BP} \cdot \mathsf{BP} \cdot \oplus\mathsf{P} = \mathsf{BP} \cdot \oplus\mathsf{P}$ (Kor. 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 119. *Seien $f_1, f_2 \in \#\mathsf{P}$, $t \in \mathsf{FP}$, p ein Polynom und $A \in \mathsf{P}$ eine (k, q)-balancierte Sprache. Dann sind folgende Funktionen in $\#\mathsf{P}$:*

(i) $f_1 + f_2$
(ii) $f_1 \cdot f_2$
(iii) $x \mapsto \sum_{y, x\#y \in A} f_1(x\#y)$
(iv) $x \mapsto f_1(x)^{p(n)}$
(v) $x \mapsto f_1(t(x))$

Beweis. Zu f_i existiert eine (k_i, q_i)-balancierte Sprache $B_i \in \mathsf{P}$ mit $f_i(x) = \#B_i(x)$. Dann ist $f_1(x) + f_2(x) = \#C_1(x)$ für die $(\max\{2, k_1, k_2\}, q_1 + q_2 + 1)$-balancierte Sprache

$$C_1 = \{x\#y0^{q_2(n)+1} \,|\, x\#y \in B_1\} \cup \{x\#0^{q_1(n)}z1 \,|\, x\#z \in B_2\} \in \mathsf{P}.$$

Weiter ist $f(x)g(x) = \#C_2(x)$ für die $(\max\{k_1, k_2\}, q_1+q_2)$-balancierte Sprache

$$C_2 = \{x\#yz \,|\, x\#y \in B_1 \wedge x\#z \in B_2\} \in \mathsf{P}.$$

Um zu zeigen, dass $h(x) = \sum_{y, x\#y\in A} f_1(x\#y)$ in $\#\mathsf{P}$ ist, beobachten wir, dass $h(x) = \#C_3(x)$ für die $(\max\{k, k_1\}, q(n) + q_1(n + 1 + q(n))$-balancierte Sprache

$$C_3 = \{x\#yz \,|\, x\#y \in A, x\#y\#z \in B_1\} \in \mathsf{P}$$

ist. Die Zugehörigkeit von $h'(x) = f_1(x)^{p(n)}$ zu $\#\mathsf{P}$ folgt mittels der (k_1, pq_1)-balancierten Sprache

$$C_4 = \{x\#y_1 \cdots y_{p(n)} \,|\, x\#y_1, \ldots, x\#y_{p(n)} \in B_1\} \in \mathsf{P}.$$

Schließlich folgt $g(x) = f_1(t(x))$ in $\#\mathsf{P}$ mittels der (k_1, s)-balancierten Sprache

$$C_5 = \{x\#z0^{s(|x|)-q_1(|t(x)|)} \,|\, t(x)\#z \in B_1\},$$

wobei s ein Polynom mit $q_1(|t(x)|) \leq s(|x|)$ ist. ■

Lemma 120. *Für jede Funktion $g \in \#\mathsf{P}$ ist die Sprache $A = \{x\#bin(\ell) \,|\, g(x) \geq \ell\}$ 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

$$B' = \left\{ x\#bin(\ell)\#y_{q(n)} \ldots y_0 \;\middle|\; \begin{array}{l} \sum_{i=0}^{q(n)} y_i k^i < k^{q(n)+1}/2 - \ell \text{ oder} \\ y_{q(n)} = k - 1 \wedge x\#y_{q(n)-1} \ldots y_0 \in B \end{array} \right\}$$

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

Weiterhin benötigen wir das Konzept der Modul-verstärkenden Polynome. Nach Definition von $\oplus\mathsf{P}$ existiert für jede Sprache $A \in \oplus\mathsf{P}$ eine Funktion $h \in \#\mathsf{P}$ mit $h(x) \equiv_2 A(x)$. Mithilfe des nächsten Lemmas können wir zeigen, dass sich der Modul 2 in dieser Kongruenz für jedes Polynom p auf $2^{p(n)}$ verstärken lässt.

Lemma 121. *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

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

■

Nun können wir den Modul 2 in der Kongruenz $h(x) \equiv_2 A(x)$ auf $2^{p(n)}$ verstärken.

Lemma 122. *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 $\#\mathsf{P}$-Funktion mit $h(x) \equiv_2 A(x)$. Eine viermalige Anwendung von Lemma 119 zeigt, dass dann auch die Funktion

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

in $\#\mathsf{P}$ ist. Mit Lemma 121 folgt $A(x) \equiv_{2^{p(n)}} f(x)$. ■

Satz 123 (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 122 ex. eine Funktion $f \in \#\mathsf{P}$ mit

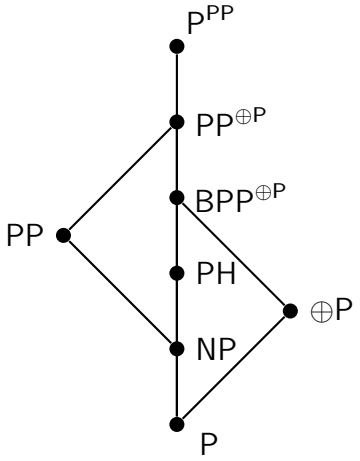
$$f(x\#y) \equiv_{2^{p(n)}} A(x\#y)$$

Betrachte nun die Funktion $g(x) = \sum_{y \in \Gamma^{q(n)}} f(x\#y)$, die nach Lemma 119 in $\#\mathsf{P}$ 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 120) 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 124. *Es gilt*
$\mathsf{PH} \subseteq \mathsf{BP} \cdot \oplus\mathsf{P} = \mathsf{BPP}^{\oplus\mathsf{P}} \subseteq \mathsf{PP}^{\oplus\mathsf{P}} \subseteq \mathsf{P}^{\mathsf{PP}}$.



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 NP 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 125.

- *Ein **Multi-Prover Interactive Proof System (MIPS)** besteht aus einer PTM V (**Verifier**) und PTMs $P_1, \ldots, P_k$ (**Prover**), die alle Zugriff auf ein gemeinsames read-only Eingabeband haben. V und $P_1, \ldots, P_k$ verfügen zusätzlich über private Zufallsgeneratoren, mit denen sich gleichverteilte Zufallszahlen ziehen lassen. Hierzu verwenden wir die Anweisung **guess randomly** $i \in_R \{1, \ldots, \ell\}$, wobei ℓ als Unärzahl generiert werden muss.*

  *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 Zufallszahlen zieht.*

  V darf insgesamt nur polynomiell viele Rechenschritte ausführen und die Laufzeit der Prover ist unbegrenzt (aber endlich).
- *Für das Ereignis, dass V bei Eingabe x und Interaktion mit $P_1, \ldots, P_k$ akzeptiert, schreiben wir kurz $V \leftrightarrow P_1, \ldots, P_k(x) = 1$. Ein **MIPS** $(V, P_1, \ldots, P_k)$ **entscheidet eine Sprache** $A \subseteq \Sigma^*$, falls für alle $x \in \Sigma^*$ gilt:*

  $$x \in A \Rightarrow \Pr[V \leftrightarrow P_1, \ldots, P_k(x) = 1] \geq 2/3 \qquad \textit{(Vollständigkeit)}$$
  $$x \notin A \Rightarrow \forall P_1', \ldots, P_k' : \Pr[(V \leftrightarrow P_1', \ldots, P_k'(x) = 1] \leq 1/3 \qquad \textit{(Korrektheit)}$$
- *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* $\mathsf{MIP}[r(n)]$ ($\mathsf{IP}[r(n)]$).
- *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.*

  $\mathsf{AM}[r(n)]$ ($\mathsf{MA}[r(n)]$) *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* $\mathsf{AM}[2]$ *bzw.* $\mathsf{MA}[3]$ *etc. wird auch einfach* AM *bzw.* MAM *etc. geschrieben.*

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

Proposition 126.

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

Dagegen erfordern folgende Beziehungen teilweise umfangreiche Beweise.

Satz 127.

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

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 128.

a) 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 $E^\varphi = \{\{u^\varphi, v^\varphi\} \,|\, \{u, v\} \in E\}$.

*b) 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):

Eingabe: 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 129. *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):

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

Lemma 130. *Für jeden Graphen $G \in \mathcal{G}_n$ 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 in $\mathcal{G}_n$. 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),

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

■

11.2 Ein interaktives Beweissystem für $\overline{\text{GI}}$

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

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

1 **input:** zwei Graphen $G_1, G_2 \in \mathcal{G}_n$
2 $\boldsymbol{V}$**:** **guess randomly** $(i,\pi) \in_R \{1,2\} \times S_n$
3 $H := G_i^\pi$
4 $\boldsymbol{V} \to \boldsymbol{P}$**:** H
5 $\boldsymbol{P}$**:** **if** $H \cong G_1$ **then** $j := 1$ **else** $j := 2$
6 $\boldsymbol{P} \to \boldsymbol{V}$**:** j
7 $\boldsymbol{V}$**:** **if** $i = j$ **then** accept **else** reject

Behauptung 131.

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

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

Beweis. *i)* klar.

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

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

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 $\Pr[Y = H] = \Pr[Y = H | X = i]$ für $i = 1,2$. Daher sind X und Y stochastisch unabhängig.

Da Z nur von Y (und den Eingabegraphen) abhängt, ist mit Y auch Z von X stochastisch unabhängig. Folglich kann P' den Verifier V höchstens mit Wahrscheinlichkeit

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

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). ■

Man beachte, dass die Laufzeit des ehrlichen Provers polynomiell beschränkt ist, falls er ein GI-Orakel befragen kann.

Die Korrektheit des Protokolls hängt allerdings 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 $\overline{\text{GI}}$ angeben.

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 $\overline{\text{GA}}$

1	**input:**	ein Graph $G = (V, E) \in \mathcal{G}_n$
2	$\boldsymbol{V}$**:**	**guess randomly** $\pi \in_R S_n$
3		$H := G^\pi$
4	$\boldsymbol{V} \to \boldsymbol{P}$**:**	H
5	$\boldsymbol{P}$**:**	compute $\varphi \in \mathit{Iso}(G, H)$
6	$\boldsymbol{P} \to \boldsymbol{V}$**:**	φ
7	$\boldsymbol{V}$**:**	**if** $\pi = \varphi$ **then** accept **else** reject

Behauptung 132.

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

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

Beweis. *i)* Klar, da es in diesem Fall genau einen Isomorphismus in $\mathit{Iso}(G, H)$ gibt.

ii) Sei X die Zufallsvariable, die die Wahl der Zufallspermutation π beschreibt, und sei Y die Zufallsvariable, die die Wahl des Zufallsgraphen H beschreibt. Dann gilt für jede Permutation $\pi \in S_n$ und für jeden Graphen $H \in W(Y) = \{H \mid H \cong G\}$,

$$\Pr[X = \pi | Y = H] = \frac{1}{|Aut(G)|} \leq 1/2.$$

Da die Antwort Z eines beliebigen Provers P' nur über Y von X abhängt, kann es P' höchstens mit Wahrscheinlichkeit $1/2$ gelingen, die von V gewählte Zufallspermutation π zu erraten. 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. In den Übungen wird gezeigt, dass sich ein Isomorphismus $\varphi \in \mathit{Iso}(G, H)$ in Polynomialzeit mit nichtadaptiven Orakelfragen an GA berechnen lässt, falls $G \notin \text{GA}$ ist. Der ehrliche Prover P in obigem Protokoll ist als in FP^{GA} berechenbar.

11.3 Ein Public-Coin-Protokoll für $\overline{\text{GI}}$

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

Public-Coin-Protokoll für $\overline{\text{GI}}$

1	**input:**	zwei Graphen $G_1, G_2 \in \mathcal{G}_n$
2	$\boldsymbol{V}$**:**	**guess randomly** $(H, \pi) \in_R \mathcal{G}_n \times S_n$
3	$\boldsymbol{V} \to \boldsymbol{P}$**:**	(H, π)
4	$\boldsymbol{P}$**:**	$I := \mathit{Iso}(G_1, H) \cup \mathit{Iso}(G_2, H)$
5		**if** $I \neq \emptyset$ **then** compute $\varphi \in I$ **else** $\varphi := id$
6	$\boldsymbol{P} \to \boldsymbol{V}$**:**	φ
7	$\boldsymbol{V}$**:**	**if** $H \in \{G_1^\varphi, G_2^\varphi\} \wedge H^\pi = H$ **then** accept **else** reject

Dieses Protokoll hat folgende Eigenschaften.

Behauptung 133.

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

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

Beweis. Sei $X(G_i) = \{(H, \pi) \in \mathcal{G}_n \times S_n \mid H \cong G_i, \pi \in Aut(H)\}$. Dann gilt für jeden Prover P':

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

Nach Lemma 130 haben die Mengen $X(G_i)$ die Mächtigkeit $|X(G_i)| = n!$ und somit folgt

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

Zudem folgt für jeden Prover P'

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

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 134.

a) Eine Funktion $g : \Sigma^* \to \mathbb{N}^+$ ist in FP berechenbar, falls ein FP-Transducer T existiert, der bei Eingabe x die Binärdarstellung von $g(x)$ ausgibt.

b) Für eine Sprachklasse $\mathcal{C}$ enthält die Klasse $\widetilde{\mathsf{BP}} \cdot \mathcal{C}$ alle Sprachen A, für die Funktionen $f \in \# \cdot \mathcal{C}$ und $g : \Sigma^* \to \mathbb{N}^+$ in FP existieren mit

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

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 $\widetilde{\mathsf{BP}} \cdot \mathsf{NP} = \mathsf{BP} \cdot \mathsf{NP}$ ist, siehe Satz 136).

Satz 135. *Es gilt $\overline{\mathrm{GI}} \in \widetilde{\mathsf{BP}} \cdot \mathsf{NP}$.*

Beweis. Wählen wir für f die Funktion $f(x) = \#B(x)$ in $\# \cdot \mathsf{NP}$, wobei B die NP-Sprache

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

und o.B.d.A. $|\langle H, \pi \rangle| = |\langle G_1, G_2 \rangle|$ ist, sowie für g die Funktion $g(\langle G_1, G_2 \rangle) = 2n!$ in FP, so gilt für alle $x = \langle G_1, G_2 \rangle$ mit $G_1, G_2 \in \mathcal{G}_n$,

$$\begin{aligned} G_1 \ncong G_2 &\Rightarrow \#B(x) = g(x) \\ G_1 \cong G_2 &\Rightarrow \#B(x) = g(x)/2 \end{aligned}$$ ■

Satz 136. *Es gilt $\widetilde{\mathsf{BP}} \cdot \mathsf{NP} \subseteq \mathsf{BP} \cdot \mathsf{NP}$.*

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

$$f(x) = \#A(x) = |\{y \in \Gamma_k^{q(n)} \,|\, x \# y \in A\}|,$$

wobei wir o.B.d.A. annehmen können, dass $k = 2$ ist und A keine Wörter der Form $x \# 0^{q(n)}$ enthält. Weiter sei $p(n) = 5q(n)$ und B sei die $(2, p)$-balancierte NP-Sprache

$$B = \{x \# y_1 \ldots y_5 \,|\, \text{für } i = 1, \ldots, 5 \text{ ist } x \# y_i \in A\}$$

Dann enthalten die Mengen $B_x = \{y \in \{0,1\}^{p(n)} \,|\, x \# y \in B\}$ 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

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

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

Dann ist für einen zufällig gewählten String $z \in_R \{0,1\}^{r(n)}$ 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 110 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. $\lfloor \log_2 g(x)^5 \rfloor \leq 2$, also $g(x)^5 < 8$ bzw. $g(x) = 1$). Dann hat $h_z(y)$ für alle $z \in \{0,1\}^{r(|x|)}$ und alle $y \in \{0,1\}^{p(n)}$ den Wert $h(y) = 0^{k(x)} = \varepsilon$ und es folgt $Var(S_x) = 0$ sowie $E(S_x) = S_x = \#B(x)$. 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 $k(x) \geq 1$ ist $k(x) = \lfloor \log_2 g(x)^5 \rfloor - 2$, was

- $\log_2 g(x)^5 - 1 < k(x) + 2 \leq \log_2 g(x)^5$ und
- $g(x)^5/8 < 2^{k(x)} \leq g(x)^5/4$

impliziert. Da $\#B(x) \geq g(x)^5$ für alle $x \in L$ und somit $E(S_x) \geq 2^{-k(x)}g(x)^5 \geq 4$ 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 $\Pr[S_x \geq 1] \geq 3/4$ für alle $x \in L$ impliziert. Andererseits enthält B_x für alle $x \notin L$ höchstens $g(x)^5/2^5$ Strings und es folgt in diesem Fall

$$\Pr[S_x \geq 1] \leq \sum_{y \in B_x} \Pr[Z_y = 1] \leq \underbrace{2^{-k(x)}}_{<8/g(x)^5} g(x)^5/2^5 < 1/4$$

■

Aus den Beweisen der Sätze 135 und 136 erhalten wir folgendes AM-Protokoll für $\overline{\text{GI}} \in \mathsf{BP} \cdot \mathsf{NP}$.

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

1	**input:**	zwei Graphen $G_1, G_2 \in \mathcal{G}_n$
2	$\boldsymbol{V}$**:**	$k := \max\{0, \lfloor \log_2(8(n!)^5) \rfloor\}$
3		$p := 5 \cdot \lvert\langle G_1, G_2\rangle\rvert$
4		**guess randomly** $h \in_R Lin(p,k)$
5	$\boldsymbol{V} \to \boldsymbol{P}$**:**	h
6	$\boldsymbol{P}$**:**	suche $H_1, \ldots, H_5, \pi_1, \ldots, \pi_5, \varphi_1, \ldots, \varphi_5$ mit
7		$(*)\begin{cases}\varphi_i \in \mathit{Iso}(G_1, H_i) \cup \mathit{Iso}(G_2, H_i), \pi_i \in \mathit{Aut}(H_i) \text{ für} \\ i = 1, \ldots, 5 \text{ und } h(\langle H_1, \pi_1\rangle \cdots \langle H_5, \pi_5\rangle) = 0^k\end{cases}$
8	$\boldsymbol{P} \to \boldsymbol{V}$**:**	$H_1, \ldots, H_5, \pi_1, \ldots, \pi_5, \varphi_1, \ldots, \varphi_5$
9	$\boldsymbol{V}$**:**	**if** $(*)$ **then** accept **else** reject

Korollar 137. 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 $\mathcal{C}$, die unter majority-Reduktionen abgeschlossen ist,

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

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. ■

11.4 Ein Zero-Knowledge Protokoll für GI

Interaktive Beweissysteme haben den großen Vorteil, dass sie im Gegen- satz zu NP-Beweisen die Möglichkeit der Nichtreproduzierbarkeit bieten. Durch Verwendung von Zufall ist es nämlich prinzipiell möglich, dass der Verifier von der Zugehörigkeit von x zu A überzeugt wird, ohne selbst in die Lage versetzt zu werden, einen Dritten von dieser Tatsache überzeugen zu können. Zum Beispiel besteht ein NP-Beweis für die Isomorphie zweier Graphen in der Angabe eines Isomorphismus und ist daher vom Verifier problemlos einem Dritten gegenüber reproduzierbar.

Insbesondere in der Kryptografie sind jedoch sogenannte Zero-Knowledge Beweise von Interesse, aus denen der Verifier kein Zusatzwissen (außer der Tatsache, dass x zu A gehört) gewinnen kann. Formal lässt sich diese Eigenschaft durch die Bedingung beschreiben, dass sämtliche Informationen, die die (oder der) Prover einem Verifier zukommen lassen, auch von diesem selbst produziert werden könnten.

Dabei hängt die Zero-Knowledge-Eigenschaft nur von den Provern $P_1, \ldots, P_k$ ab, da auch ein ***unehrlicher Verifier*** V', selbst wenn er es darauf anlegt, nicht in der Lage sein darf, sich durch die Interaktion mit $P_1, \ldots, P_k$ ein Zusatzwissen anzueignen. Im Gegensatz hierzu kommt es bei den beiden Eigenschaften Vollständigkeit und Korrektheit eines interaktiven Beweissystems $(V, P_1, \ldots, P_k)$ nur auf den Verifier V an. Dies ist bei der Korrektheitsbedingung offensichtlich, da $P_1, \ldots, P_k$ gar nicht darin vorkommen. Aber auch für die Vollständigkeitsbedingung ist die Angabe der ehrlichen Prover $P_1, \ldots, P_k$ nicht erforderlich, da sich diese aus V ableiten lassen, sofern sie existieren.

Definition 138.

a) Das ***Transkript*** $View_{V,P_1,\ldots,P_k}(x)$ *eines MIPS* $(V, P_1, \ldots, P_k)$ *bei Eingabe* x *ist die Zufallsvariable, die sämtliche während der Berechnung von* $V \leftrightarrow P_1, \ldots, P_k(x)$ *ausgetauschten Nachrichten auflistet.*

b) Ein MIPS $(V, P_1, \ldots, P_k)$ *für eine Sprache* A *hat die* ***perfekte Zero-Knowledge-Eigenschaft*** *(kurz* ***PZK****), wenn es für jeden effizienten Verifier* V' *einen probabilistischen Simulator* S *mit im Erwartungswert polynomiell beschränkter Laufzeit gibt, so dass für alle Eingaben* $x \in A$ *die Zufallsvariablen* $S(x)$ *und* $View_{V',P_1,\ldots,P_k}(x)$ *identisch verteilt sind.*

Bevor wir ein PZK-IPS für GI angeben, betrachten wir nochmals die drei in den vorangehenden Abschnitten vorgestellten Protokolle für $\overline{\text{GI}}$ und $\overline{\text{GA}}$ und überlegen, ob sie die PZK-Eigenschaft erfüllen oder nicht.

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

input: zwei Graphen $G_1, G_2 \in \mathcal{G}_n$
V: **guess randomly** $(i, \pi) \in_R \{1,2\} \times S_n$
$H := G_i^\pi$
$V \to P$: H
P: **if** $H \cong G_1$ **then** $j := 1$ **else** $j := 2$
$P \to V$: j
V: **if** $i = j$ **then** accept **else** reject

Für den ehrlichen Verifier V in diesem Protokoll lässt sich leicht ein Simulator S_V angeben.

Simulator S_V für obiges Protokoll

input: zwei Graphen $G_1, G_2 \in \mathcal{G}_n$ mit $G_1 \ncong G_2$
guess randomly $(i, \pi) \in_R \{1,2\} \times S_n$
output $G_i^\pi \# i$

Damit auch für jeden effizienten unehrlichen Verifier V' die Existenz eines Simulators $S_{V'}$ nachweisbar ist, muss das folgende funktionale Promise-Problem P_1 in erwarteter Polynomialzeit lösbar sein.

Funktionales Promise-Problem P_1:

Eingabe: Drei Graphen $G_1, G_2, H \in \mathcal{G}_n$.

Promise: $H \cong G_1 \vee H \cong G_2$.

Gesucht: Eine Zahl $j \in \{1, 2\}$ mit $H \cong G_j$.

Das Problem P_1 lässt sich leicht in ein äquivalentes Problem P_2 gemäß unserer Definition von Promise-Problemen transformieren.

Promise-Problem P_2:

Eingabe: Drei Graphen $G_1, G_2, H \in \mathcal{G}_n$.

Promise: $H \cong G_1 \oplus H \cong G_2$.

Gefragt: Gilt $H \cong G_1$?

Tatsächlich ist GI auf jede Lösung von P_2 reduzierbar (siehe Übungen).

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

1 **input:** ein Graph $G = (V, E) \in \mathcal{G}_n$
2 **V:** **guess randomly** $\pi \in_R S_n$
3 $H := G^\pi$
4 **$V \to P$:** H
5 **P:** compute $\varphi \in \mathit{Iso}(G, H)$
6 **$P \to V$:** φ
7 **V:** **if** $\pi = \varphi$ **then** accept **else** reject

Auch hier lässt sich für den ehrlichen Verifier V leicht ein Simulator S_V angeben.

Simulator S_V für obiges Protokoll

1 **input:** ein Graph $G \in \mathcal{G}_n$ mit $G \in \overline{\text{GA}}$
2 **guess randomly** $\pi \in_R S_n$
3 **output** $G^\pi \# \pi$

Um auch für jeden effizienten unehrlichen Verifier V' die Existenz eines Simulators $S_{V'}$ nachweisen zu können, muss das folgende funktionale Promise-Problem P_1' in erwarteter Polynomialzeit lösbar sein.

Promise-Problem P_1':

Eingabe: Zwei Graphen $G, H \in \mathcal{G}_n$.

Promise: $G, H \in \overline{\text{GA}}$ und $G \cong H$.

Gesucht: Ein Isomorphismus φ zwischen G und H.

Auch P_1' lässt sich in ein äquivalentes Problem P_2' gemäß unserer Definition von Promise-Problemen transformieren (siehe Übungen).

Promise-Problem P_2':

Eingabe: Drei Graphen $G, H_1, H_2 \in \mathcal{G}_n$.

Promise: $G, H_1, H_2 \in \overline{\text{GA}}$ und $G \cong H_1 \oplus G \cong H_2$.

Gefragt: Gilt $G \cong H_1$?

In diesem Fall ist GA auf jede Lösung von P_2' reduzierbar (siehe Übungen). Nun stellen wir ein 3-Runden IPS für GI mit der PZK-Eigenschaft vor.

***Zero-Knowledge-Protokoll für* GI**

1 **input:** zwei Graphen $G_1, G_2 \in \mathcal{G}_n$
2 **P:** **guess randomly** $(i, \pi) \in_R \{1, 2\} \times S_n$
3 $H := (G_i)^\pi$
4 **$P \to V$:** H
5 **V:** **guess randomly** $j \in_R \{1, 2\}$
6 **$V \to P$:** j
7 **P:** **case**
8 $j = i$: $\varphi := \pi$
9 $j = 3 - i$: compute $\tau \in \mathit{Iso}(G_j, G_i)$; $\varphi := \tau\pi$
10 **else** $\varphi :=$ id
11 **$P \to V$:** φ
12 **V:** **if** $(G_j)^\varphi = H$ **then** accept **else** reject

Man beachte, dass der Prover P polynomiell zeitbeschränkt ist, sofern er einen Isomorphismus zwischen G_1 und G_2 als Zusatzeingabe erhält.

Behauptung 139.

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

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

Um zu zeigen, dass das IPS (V, P) 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 alle 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: zwei Graphen $G_1, G_2 \in \mathcal{G}_n$ mit $G_1 \cong G_2$
  repeat
    guess randomly $(i, \pi) \in_R \{1, 2\} \times S_n$
    $H := G_i^\pi$
    simuliere V' bei Eingabe $\langle G_1, G_2 \rangle$ und erster Prover-Nachricht H
  until V' antwortet mit einer Nachricht $j \neq 3 - i$
  if $j \neq i$ **then** $\pi := \mathrm{id}$
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. 131), 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.

Definition 140. *Seien g, h Funktionen.*

*a) Die Funktion g ist **parsimonious reduzierbar** auf h (kurz $g \leq^{log}_{par} h$ oder einfach $g \leq_{par} h$), falls eine Funktion $f \in \mathsf{FL}$ existiert mit $g(x) = h(f(x))$.*

*b) Die Funktion g ist **Turing-reduzierbar** auf h (kurz $g \leq^p_T h$), falls eine FP-OTM M existiert mit $M^h(x) = g(x)$.*

c) Stellt $M^h(x)$ bei allen Eingaben x höchstens eine Frage an g, so schreiben wir $g \leq^{\mathsf{FP}[1]} h$.

Es gibt auch NP-Probleme $A = \exists^p B$, die effizient entscheidbar sind, obwohl das zugehörige Anzahlproblem $\#B$ schwierig ist. Wir werden z.B. sehen, dass das Problem #BPM, die Anzahl der perfekten Matchings in einem bipartiten Graphen zu bestimmen, vollständig für #P unter $\leq^{\mathsf{FP}[1]}$ Reduktionen ist. Ob G ein perfektes Matching hat, ist dagegen in P entscheidbar.

Ähnlich ist es bei der Anzahl #HORNSAT der erfüllenden Belegungen von Hornformeln. Auch dieses Problem ist vollständig für #P unter $\leq^{\mathsf{FP}[1]}$, obwohl die Erfüllbarkeit von Hornformeln effizient entscheidbar ist.

In manchen Fällen ist das Anzahlproblem aber auch nicht schwerer als das zugrunde liegende NP-Problem. So werden wir sehen, dass sich die Anzahl $\#\mathrm{GI}(G, H)$ von Isomorphis- men zwischen zwei Graphen G und H in Polynomialzeit durch Orakelfragen an GI berechnen lässt.

12.1 Aussagenlogische Anzahlprobleme

Ein einfaches Beispiel für ein Anzahlproblem, das schwieriger als das zugehörige Entscheidungsproblem ist (unter der Voraussetzung PP ≠ P), ist die Bestimmung der Anzahl aller erfüllenden Belegungen einer DNF-Formel.

Definition 141. *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 $D_i = \bigwedge_{j=1}^{k_i} \ell_{ij}$ von **Literalen** ist.*

#DNFSAT

Eingabe: 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 Problem DNFSAT, die Erfüllbarkeit einer DNF-Formel zu entscheiden, in P lösbar ist.

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

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

Definition 142. *Eine boolesche Formel F heißt **monoton**, falls sie nur mittels $\vee$ und $\wedge$ aus Variablen und Konstanten aufgebaut ist.*

#MonSat

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

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

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

Zum Beispiel 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.

In den Übungen werden wir sehen, dass bereits die Einschränkung #2-MonSat von #MonSat auf 2-KNF-Formeln $\#\mathsf{P}$-vollständig unter $\leq_T^P$-Reduktionen ist, d.h. $\#\mathsf{P} \subseteq \mathsf{FP}^{\#\text{2-MonSat}}$.

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

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

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

- einem Isomorphismus π_1 zwischen G_1 und G_2 und
- einem Isomorphismus π_2 zwischen G_2 und G_1

zusammen. Daher gilt für zusammenhängende Graphen

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

und somit ist $\#\text{GI} \in \mathsf{FP}_{\parallel}^{\#\text{GA}[3]}$ mit den 3 Fragen $G_1, G_2, G_1 + G_2$. Wir können die beiden Fragen G_1 und G_2 an das #GA-Orakel sogar durch eine Frage H ersetzen, d.h. $\#\text{GI} \in \mathsf{FP}_{\parallel}^{\#\text{GA}[2]}$. Hierbei entsteht H aus $K_1 + G_1 + K_1 + G_2$, indem wir den ersten K_1 mit allen Knoten von G_1 und den zweiten K_1 mit allen Knoten von G_1 und G_2 verbinden. Dann ist $\#\text{GA}(H) = \#\text{GA}(G_1) \cdot \#\text{GA}(G_2)$ (siehe Übungen), unabhängig davon ob G_1 und G_2 isomorph sind oder nicht, und daher folgt

$$\#\text{GI}(G_1, G_2) = \begin{cases} \sqrt{\#\text{GA}(H)}, & \#\text{GA}(G_1 + G_2) = 2\#\text{GA}(H) \\ 0, & \text{sonst.} \end{cases}$$

Wir zeigen nun, dass #GA und #GI Turing-reduzierbar auf GI sind, d.h. es gilt $\#\text{GA}, \#\text{GI} \in \mathsf{FP}^{\text{GI}}$. Da GI und das Graphenisomorphieproblem ColGI für gefärbte Graphen logspace-äquivalent sind

(d.h. ColGI $\equiv_m^{log}$ GI; siehe Übungen), reicht es, #GA und #GI auf ColGI zu reduzieren.

Für eine Folge $(j_1, \ldots, j_k) \in \{1, \ldots, n\}^k$ von paarweise verschiedenen Knoten j_i sei $G_{(j_1,\ldots,j_k)}$ der Graph mit der Knotenfärbung $c(j_i) = i$ für $i = 1, \ldots, k$ und $c(j) = n$ für alle $j \in V \setminus \{j_1, \ldots, j_k\}$. Den Graphen $G_{(1,\ldots,i)}$ bezeichnen wir auch kurz mit $G_{[i]}$. Es ist klar, dass $Aut(G_{[i]})$ eine Untergruppe von $Aut(G_{[i-1]})$ ist. Hierfür schreiben wir kurz $Aut(G_{[i]}) \leq Aut(G_{[i-1]})$), d.h. es gilt

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

Aus der Gruppentheorie wissen wir, dass die Menge

$$N_i = \{Aut(G_{[i]})\rho \mid \rho \in Aut(G_{[i-1]})\}$$

aller Nebenklassen von $Aut(G_{[i]})$ in $Aut(G_{[i-1]})$ eine Partition von $Aut(G_{[i-1]})$ in gleichmächtige Teilmengen bilden. Enthält N_i also k_i Nebenklassen, so ist $\#\mathrm{GA}(G_{[i-1]}) = k_i \cdot \#\mathrm{GA}(G_{[i]})$ und es folgt

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

Zudem liegen $\pi, \varphi \in Aut(G_{[i-1]})$ genau dann in derselben Nebenklasse, wenn $\pi\varphi^{-1} \in Aut(G_{[i]})$, also $i^{\pi\varphi^{-1}} = i$ bzw. $\pi(i) = \varphi(i)$ ist. Daher gilt für die Anzahl k_i der Nebenklassen

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

Es gilt also $k_i = 1 + \sum_{j=i+1}^n k_{ij}$ mit

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

Folglich lässt sich $\#\mathrm{GA}(G)$ durch $\sum_{i=1}^{n-1}(n-i) = \sum_{i=1}^{n-1} i = n(n-1)/2$ nichtadaptive Fragen an GI berechnen. Mit einer Frage mehr lässt sich auch #GI in $\mathsf{FP}_{\|}^{\mathrm{GI}}$ berechnen.

Wir fassen den $\mathsf{FP}_{\|}^{\mathrm{GI}}$ Algorithmus zur Berechnung von NumGA nochmals zusammen:

```
1 input: ein Graph G ∈ G_n
2   for i := 1 to n do
3     k_i := 1
4     for j := i + 1 to n do
5       if G_[i] ≅ G_(1,...,i−1,j) then k_i := k_i + 1
6 output: ∏_{i=1}^n k_i
```

Da sich ein Isomorphismus zwischen zwei gegebenen isomorphen Graphen in $\mathsf{FP}^{\mathrm{GI}}$ berechnen lässt (siehe Übungen), können wir auch noch eine Menge $M = \{\pi_{ij} \mid k_{ij} = 1\}$ von $\sum_{1\leq i<j\leq n} k_{ij} = \sum_{i=1}^n k_i - n$ Generatoren für $Aut(G)$ berechnen. Hierzu genügt es, für jedes Paar (i, j) mit $k_{ij} = 1$ einen Isomorphismus $\pi_{ij} \in Iso(G_{[i]}, G_{[1,\ldots,i-1,j]}) \subseteq Aut(G_{[i-1]})$ zu berechnen. Dann lässt sich jeder Automorphismus $\varphi \in Aut(G)$ als Produkt $\varphi = \pi_{n\varphi_n(n)} \cdots \pi_{1\varphi_1(1)}$ von höchstens n Generatoren aus der Menge M darstellen, wobei die Permutationen φ_i induktiv durch $\varphi_1 = \varphi$ und $\varphi_{i+1} = \varphi_i \pi_{i\varphi_i(i)}^{-1}$ für $i = 1, \ldots, n-1$ definiert sind. Wegen

$$\underbrace{\underbrace{\underbrace{\varphi_1\pi_{1\varphi_1(1)}^{-1}}_{\varphi_2} \cdots \pi_{(n-1)\varphi_{n-1}(n-1)}^{-1}}_{\varphi_{n-1}} \pi_{n\varphi_n(n)}^{-1}}_{\varphi_n} = \varphi(\pi_{n\varphi_n(n)} \cdots \pi_{1\varphi_1(1)})^{-1}$$

folgt nämlich induktiv, dass $\varphi_i \in Aut(G_{[i-1]})$ und somit $\varphi_n = id$ ist.

12.3 Anzahl von perfekten Matchings in bipartiten Graphen

In diesem Abschnitt beweisen wir die #P-Vollständigkeit von #BPM unter $\leq^{\mathsf{FP}[1]}$-Reduktionen. Für einen bipartiten Graphen $G =$

(U, W, E) mit $U = \{u_1, \ldots, u_n\}$ und $W = \{w_1, \ldots, w_n\}$ sei

$$\#\mathrm{BPM}(G) = |\{M \mid M \text{ ist ein perfektes Matching in } G\}|.$$

Stellen wir G in Form einer $(n \times n)$-Binärmatrix $A = (a_{ij})$ mit $a_{ij} = 1 \Leftrightarrow \{u_i, w_j\} \in E$ dar, so gilt

$$\#\mathrm{BPM}(G) = perm(A) := \sum_{\pi \in S_n} \prod_{i=1}^{n} a_{i,\pi(i)},$$

wobei $perm(A)$ die ***Permanente*** einer $(n \times n)$-Matrix $A = (a_{ij}) \in \mathbb{Z}^{n \times n}$ genannt wird. Während die Determinante

$$det(A) = \sum_{\pi \in S_n} sgn(\pi) \prod_{i=1}^{n} a_{i,\pi(i)}$$

von A effizient berechenbar ist, sind zur Berechnung von $perm(A)$ nur Exponentialzeitalgorithmen bekannt.

Interpretieren wir A als Adjazenzmatrix eines gerichteten Graphen $G_A = (V, E_A)$ mit $V = \{1, \ldots, n\}$ und $E_A = \{(i, j) \mid a_{i,j} = 1\}$, so gibt $perm(A)$ die Anzahl aller Kreisüberdeckungen von G_A an.

Es gilt nämlich $\prod_{i=1}^{n} a_{i,\pi(i)} = 1$, falls $C_\pi = \{(i, \pi(i)) \mid i \in V\}$ nur Kanten aus E_A enthält und somit (V, C_π) einen Subgraphen von G_A bildet. Da π eine Permutation ist, zerfällt C_π in (bis zu n) disjunkte Kreise, die alle Knoten in V überdecken.

Definition 143. *Sei (V, C) ein (spannender) Subgraph eines Digraphen $G = (V, E)$.*

*a) C heißt **Kreisüberdeckung** (engl. **cycle cover**, kurz **CC**) von G, falls jeder Knoten $u \in V$ den Ein- und Ausgangsgrad 1 in (V, C) hat, d.h. für alle Knoten u gilt $d^+_{(V,C)}(u) = d^-_{(V,C)}(u) = 1$.*

*b) Für zwei Knoten $s \neq t$ in V heißt C s-t-**Kreisüberdeckung** (kurz s-t-**CC**) von G, falls C nicht die Kante (t, s) enthält und $C \cup \{(t, s)\}$ eine CC von G ist.*

*c) Für vier paarweise verschiedene Knoten s, t, s', t' in V heißt C s-t-s'-t'-**Kreisüberdeckung** (kurz s-t-s'-t'-**CC**) von G, falls C nicht die Kante (t', s') enthält und $C \cup \{(t', s')\}$ eine s-t-CC von G ist.*

Man beachte, dass $C' = C \setminus \{(t, s)\}$ eine s-t-CC von G und von $G - (t, s)$ ist, falls C eine CC von G mit $(t, s) \in C$ ist. Daher können wir eine Permutation π mit $\pi(t) = s$ auch zur Beschreibung der s-t-CC $C' = C_\pi \setminus \{(t, s)\}$ benutzen, die wir mit s-t-C_π bezeichnen. Entsprechend können wir π im Fall $\pi(t) = s$ und $\pi(t') = s'$ zur Beschreibung der s-t-s'-t'-CC $C'' = C_\pi \setminus \{(t, s), (t', s')\}$ benutzen, die wir mit s-t-s'-t'-C_π bezeichnen.

Eine $(n \times n)$-Matrix $A = (a_{ij}) \in \mathbb{Z}^{n \times n}$ können wir als Adjazenzmatrix eines gerichteten Graphen $G_A = (V, E_A)$ mit $V = \{1, \ldots, n\}$, Kantenmenge $E_A = \{(i, j) \mid a_{i,j} \neq 0\}$ und ganzzahligen Kantengewichten $w(i, j) = a_{ij}$ auffassen. Da das Produkt $\prod_{i=1}^{n} a_{i,\pi(i)}$ genau dann ungleich Null ist, wenn C_π eine CC in G_A ist, bestimmt $perm(A) = \sum_{\pi \in S_n} w(C_\pi)$ die Summe der Gewichte $w(C_\pi)$ aller CCs von G_A, wobei das ***Gewicht*** von C_π das Produkt $w(C_\pi) = \prod_{i=1}^{n} a_{i,\pi(i)}$ der Gewichte aller Kanten $(i, \pi(i)) \in C_\pi$ ist. Der Einfachheit halber identifizieren wir im Folgenden (gewichtete) Digraphen mit ihren Adjazenzmatrizen.

Wir bezeichnen die Einschränkung von $perm$ auf Matrizen mit Koeffizienten in $\mathbb{N}$ mit $perm_{\mathbb{N}}$. Zudem bezeichne $perm_m$ die Einschränkung von $perm$ auf Matrizen mit Koeffizienten in $\{0, \ldots, m-1\}$ und $perm_{\bar{m}}$ die auf Matrizen mit Koeffizienten in $\{-1, 0, \ldots, m-1\}$.

Lemma 144.

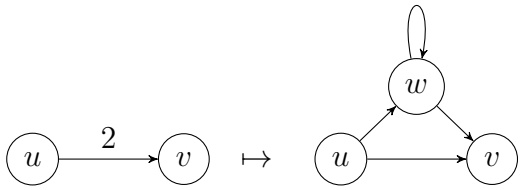
(i) $perm_3 \leq_{par} perm_2$ *und* $perm_{\bar{3}} \leq_{par} perm_{\bar{2}}$,

(ii) $perm_{\mathbb{N}} \leq_{par} perm_3$ *und* $perm \leq_{par} perm_{\bar{3}}$,

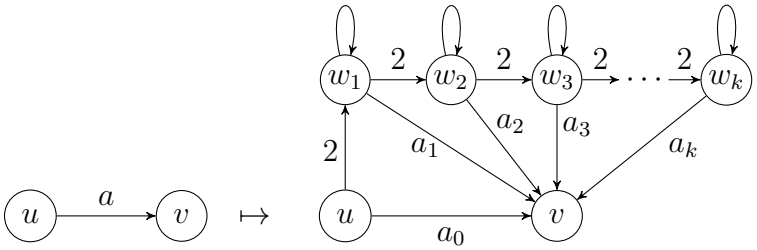
(iii) $perm_{\bar{2}} \leq^{\mathsf{FP}[1]} perm_{\mathbb{N}}$.

Beweis.

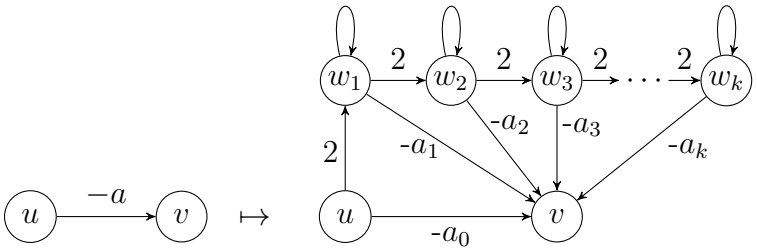
(i) Für die Reduktionen $perm_3 \leq_{par} perm_2$ und $perm_{\bar{3}} \leq_{par} perm_{\bar{2}}$ reduzieren wir das Gewicht jeder Kante (u, v) mit dem Gewicht 2 auf das Gewicht 1 und fügen für jede solche Kante einen neuen Knoten w sowie die Kanten $(u, w), (w, w), (w, v)$ hinzu:



(ii) Für die Reduktion $perm_{\mathbb{N}} \leq_{par} perm_3$ reduzieren wir das Gewicht jeder Kante (u, v) mit dem Gewicht $a = \sum_{i=0}^{k} a_i 2^i > 2$, $a_i \in \{0, 1\}$, auf das Gewicht a_0 und fügen für jede solche Kante k neue Knoten $w_1, \ldots, w_k$ sowie folgende Kanten hinzu:

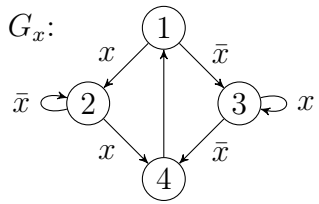


Für die Reduktion $perm \leq_{par} perm_{\bar{3}}$ ersetzen wir zusätzlich alle negative Kanten mit einem Gewicht $-a = -\sum_{i=0}^{k} a_i 2^i < -1$ wie folgt:

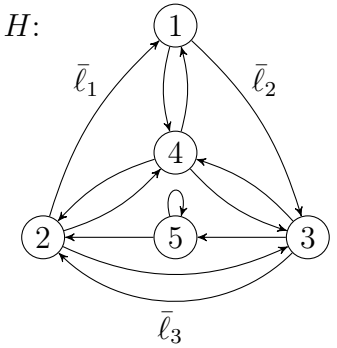


(iii) Für die Reduktion $perm_{\bar{2}} \leq^{\mathsf{FP}[1]} perm_{\mathbb{N}}$ nutzen wir aus, dass $|perm_{\bar{2}}(A)| \leq n!$ bzw. $-n! \leq perm_{\bar{2}}(A) \leq n!$ ist. Daher reicht es, $perm_{\bar{2}}(A)$ modulo $m = 2n! + 1$ zu berechnen. Ersetzen wir in der Matrix A alle Einträge $a_{ij} = -1$ durch $a_{ij} = m - 1$, so erhalten wir eine Matrix A' mit $perm_{\mathbb{N}}(A') \equiv_m perm_{\bar{2}}(A)$ und es folgt $perm_{\bar{2}}(A) = perm_{\mathbb{N}}(A') \bmod m - n!$. ■

Wir beschreiben nun die Reduktion von #3-Sat auf *perm*. Hierzu benutzen wir die folgenden drei Digraphen. Der erste Graph



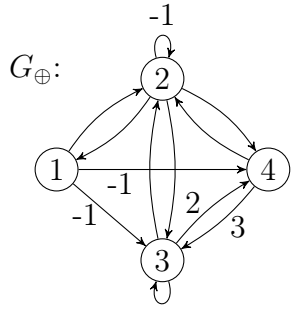
hat genau zwei CCs C_x und $C_{\bar{x}}$, wobei C_x alle mit x markierten Kanten und $C_{\bar{x}}$ alle mit $\bar{x}$ markierten Kanten enthält. Daher können wir die beiden CCs C_x und $C_{\bar{x}}$ als mögliche Belegungen der Variablen x interpretieren, die x den Wert 1 bzw. 0 zuweisen. Wir sagen auch, C_x ist mit der Belegung $x = 1$ und $C_{\bar{x}}$ mit der Belegung $x = 0$ ***konsistent***. Der abgebildete Graph hat drei x- und drei $\bar{x}$-Kanten, kann aber leicht zu einem Graphen mit mehr x- und $\bar{x}$-Kanten erweitert werden. Der zweite Digraph



hat sieben CCs und zwar für jede echte Teilmenge $T \subsetneq \{\bar{\ell}_1, \bar{\ell}_2, \bar{\ell}_3\}$ genau eine CC C_T mit $C_T \cap \{\bar{\ell}_1, \bar{\ell}_2, \bar{\ell}_3\} = T$:

- $T = \emptyset$: $(1,4)(2,3,5)$
- $T = \{\bar{\ell}_1\}$: $(1,4,3,5,2)$
- $T = \{\bar{\ell}_2\}$: $(1,3,5,2,4)$
- $T = \{\bar{\ell}_3\}$: $(1,4)(2,5,3)$
- $T = \{\bar{\ell}_1, \bar{\ell}_2\}$: $(1,3,4,2)(5)$
- $T = \{\bar{\ell}_1, \bar{\ell}_3\}$: $(1,4,3,2)(5)$
- $T = \{\bar{\ell}_2, \bar{\ell}_3\}$: $(1,3,2,4)(5)$

Jede der sieben CCs von H hat also das Gewicht 1 ist mit genau einer erfüllenden Belegung der Formel $\neg(\bar{\ell}_1 \wedge \bar{\ell}_2 \wedge \bar{\ell}_3)$ bzw. der logisch äquivalenten Formel $K = \ell_1 \vee \ell_2 \vee \ell_3$ konsistent. Schließlich hat der dritte Digraph

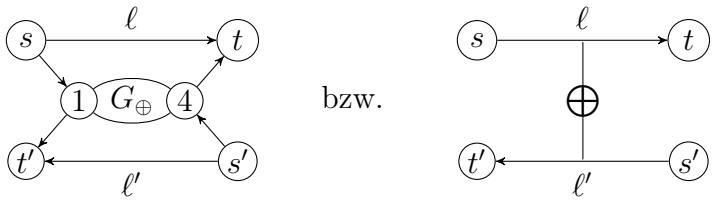


- vier CCs, deren Gewichte sich auf $6 - 3 - 2 - 1 = 0$ addieren:
  - C_1: $(1,2)(3,4)$
  - C_2: $(1,4,3,2)$
  - C_3: $(1,3,4,2)$
  - C_4: $(1,4,2)(3)$
- sechs 1-4-CCs, deren Gewichte sich auf $2 \cdot (2 + 1 - 1) = 4$ addieren:
  - C'_1: $(1,2,3,4)$
  - C'_2: $(1,3,4)(2)$
  - C'_3: $(1,4)(2)(3)$
  - C'_4: $(1,3,2,4)$
  - C'_5: $(1,4)(2,3)$
  - C'_6: $(1,2,4)(3)$
- und zwei 4-1-CCs, deren Gewichte sich ebenfalls auf $3 + 1 = 4$ addieren:
  - C'_7: $(4,3,2,1)$
  - C'_8: $(4,2,1)(3)$

Zudem hat jeder der drei induzierten Teilgraphen $G_\oplus[1,2,3]$, $G_\oplus[2,3,4]$ und $G_\oplus[2,3]$ zwei oder vier CCs, deren Gewichte sich jeweils auf $1 - 1 = 3 + 2 + 1 - 6 = 1 - 1 = 0$ addieren:

- $G_\oplus[1,2,3]$ hat zwei CCs: $(1,2)(3)$ und $(1,3,2)$
- $G_\oplus[2,3,4]$ hat vier CCs: $(2,4)$, $(2,3,4)$, $(2,4)(3)$ und $(2)(3,4)$
- $G_\oplus[2,3]$ hat zwei CCs: $(2)(3)$ und $(2,3)$

Als Folge hiervon hat der Digraph G



- sechs s-t-s'-t'-CCs, die zwar die Kante (s', t'), aber nicht die Kante (s, t) enthalten (diese entstehen aus den sechs möglichen 1-4-CCs des Teilgraphen $G_\oplus$ durch Hinzufügen der drei Kanten (s', t'), $(s, 1)$ und (t, v))
- zwei s-t-s'-t'-CCs, die zwar die Kante (s, t), aber nicht die Kante (s', t') enthalten (diese entstehen aus den zwei möglichen 4-1-CCs des Teilgraphen $G_\oplus$ durch Hinzufügen der drei Kanten (s, t), $(s', 4)$ und $(1, t')$)

Die Gewichte dieser s-t-s'-t'-CCs summieren sich in beiden Fällen auf den Wert 4. Dagegen heben sich die Gewichte aller s-t-s'-t'-CCs von G, die entweder beide Kanten (s, t) und (s', t') oder keine dieser zwei Kanten enthalten, jeweils gegenseitig auf.

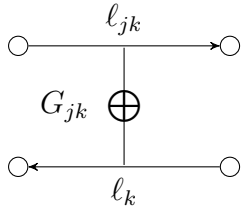
Daher ist die Summe der Gewichte aller mit einer erfüllenden Belegung $a \in \{01, 10\}$ der Formel $F = \ell \oplus \ell'$ konsistenten s-t-s'-t'-CCs von G gleich 4. Andererseits heben sich im Fall $F(a) = 0$ die Gewichte aller mit a konsistenten s-t-s'-t'-CCs von G gegenseitig auf.

Sei nun eine 3-KNF-Formel

$$F = \underbrace{(\ell_{11} \vee \ell_{12} \vee \ell_{13})}_{K_1} \wedge \cdots \wedge \underbrace{(\ell_{m1} \vee \ell_{m2} \vee \ell_{m3})}_{K_m}$$

mit Literalen $\ell_{jk} \in \{x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n\}$ gegeben, wobei Mehrfachvorkommen von Literalen in Klauseln zulässig sind. Wir konstruieren einen Digraphen G_F wie folgt:

- Für jede Variable x_i enthält G_F eine Kopie G_{x_i} des Digraphen G_x und
- für jede Klausel K_j eine Kopie H_j des Digraphen H, dessen drei Literalkanten mit den Literalen $\bar{\ell}_{j1}, \bar{\ell}_{j2}, \bar{\ell}_{j3}$ markiert werden.
- Zudem enthält G_F für jedes Vorkommen eines Literals $\ell_{jk} \in \{x_i, \bar{x}_i\}$ in einer Klausel K_j eine Kopie G_{jk} von $G_\oplus$, die eine noch nicht verknüpfte ℓ_{jk}-Kante von G_{x_i} mit der $\bar{\ell}_{jk}$-Kante von H_j mithilfe von 4 zusätzlichen Kanten verknüpft:



Die Adjazenzmatrix von G_F bezeichnen wir mit A_F.

Sei C eine CC von G_F und sei $a \in \{0,1\}^n$ eine Belegung der Variablen von F. Wir sagen, C ***ist mit*** a ***konsistent***, wenn C jede mit einem Literal ℓ markierte Kante in G_F genau dann enthält, wenn $a(\ell) = 1$ ist. Entsprechend heißt C ***konsistent***, wenn es eine Belegung a gibt, mit der C konsistent ist.

Die Menge aller CCs von G_F bezeichnen wir mit M und die Menge aller mit der Belegung a konsistenten CCs mit M_a.

Da sich die Gewichte aller inkonsistenten CCs von G_F gegenseitig aufheben, ist

$$perm(A_F) = \sum_{C \in M} w(C) = \sum_{a \in \{0,1\}^n} w(a),$$

wobei $w(a) = \sum_{C \in M_a} w(C)$ das Gesamtgewicht aller mit a konsistenten CCs ist.

Wir zeigen nun, dass $w(a)$ im Fall $F(a) = 1$ den Wert $w(a) = 4^{3m}$ und sonst den Wert $w(a) = 0$ hat. Daraus folgt dann, dass $perm(A_F) = 4^{3m} \cdot \mathsf{Num3\text{-}Sat}(F)$ bzw. $\mathsf{Num3\text{-}Sat}(F) = perm(A_F)/4^{3m}$ ist.

Da jede CC C von G_F in jedem Teilgraphen G_{x_i} entweder alle x_i-Kanten oder alle $\bar{x}_i$-Kanten enthält, ist jede konsistente CC C mit genau einer der 2^n Belegungen der Variablen $x_1, \ldots, x_n$ von F konsistent. Diese Belegung bezeichnen wir mit a_C.

Wie wir gesehen haben, kann eine CC C von G_F für keine Klausel $K_j = \ell_{j1} \vee \ell_{j2} \vee \ell_{j3}$ alle drei Literalkanten $\bar{\ell}_{j1}, \bar{\ell}_{j2}, \bar{\ell}_{j3}$ von H_j enthalten. Daher muss die Belegung a_C jeder konsistenten CC C alle Klauseln und somit auch F erfüllen, was $w(a) = 0$ für jede Belegung a mit $F(a) = 0$ impliziert.

Gilt umgekehrt $F(a) = 1$, so addieren sich die Gewichte aller mit a konsistenten CCs auf den Wert $w(a) = 4^{3m}$, da die CCs in jeder der

- n Kopien G_{x_i} von G_x den Faktor 1
- m Kopien H_j von H den Faktor 1 und in jeder der
- $3m$ Kopien G_{jk} von $G_\oplus$ den Faktor 4

zum Gesamtgewicht $w(a)$ beitragen. Damit können wir nun folgenden Satz beweisen.

Satz 145. $\#\mathsf{P} \subseteq \mathsf{FP}^{perm_2[1]}$, *d.h.* $perm_2$ *ist* $\#\mathsf{P}$*-vollständig unter* $\leq^{\mathsf{FP}[1]}$*-Reduktionen.*

Beweis. Da die Funktion $F \mapsto A_F$ in FL berechenbar ist und $\#\text{3-Sat}(F) = perm(A_F)/4^{3m}$ ist, folgt sofort $\#\text{3-Sat} \leq^{\mathsf{FP}[1]} perm$.

Zudem folgt aus Lemma 144

$$perm \leq_{par} perm_{\bar{3}} \leq_{par} perm_{\bar{2}} \leq^{\mathsf{FP}[1]} perm_{\mathbb{N}} \leq_{par} perm_3 \leq_{par} perm_2$$

und somit #3-SAT $\leq^{\mathsf{FP}[1]}$ $perm_2$, was wiederum mit der Inklusion $\#\mathsf{P} \subseteq \mathsf{FP}^{perm_2[1]}$ äquivalent ist. ■

Die Komplexität der Determinantenberechnung von ganzzahligen Matrizen wird durch folgendes Ergebnis beschrieben.

Satz 146. *Die Funktion det ist vollständig für* GapL *unter parsimonious Reduktionen* $\leq_{par}$.

Hierbei ist die Klasse

$$\mathsf{GapL} = \{\#N - \#_{rej}N \mid N \text{ ist eine NL-TM}\} = \{f - g \mid f, g \in \#\}$$

das logspace-Analogon zur Klasse GapP. In den Übungen wird die Inklusion $\# \subseteq \mathsf{FP}$ bewiesen, die $\# \subseteq \mathsf{GapL} \subseteq \mathsf{FP}$ impliziert. Zum Beispiel ist auch das Problem, für einen ***azyklischen*** Digraphen G mit $n \geq 2$ Knoten die Differenz der Anzahl der Pfade von 1 nach n und der Anzahl der Pfade von 1 nach $n-1$ zu bestimmen, GapL-vollständig. Für ***beliebige*** Digraphen G ist das Problem dagegen GapP-vollständig.