Inhaltsverzeichnis

Kapitel 1

Turingmaschinen und deterministische Komplexitätsklassen

1.1 Turingmaschinen

Das einfachste Modell einer Turingmaschine (TM) ist die deterministische 1-Band-Turingmaschine. Für den Begriff der *Berechenbarkeit* und auch für viele Komplexitätsbetrachtungen ist dieses Modell trotz seiner Einfachheit ausreichend und zweckmässig. Im Skript *Mathematische Logik*, Kapitel 6, werden die Grundbegriffe der Berechenbarkeitstheorie auf der Basis dieses Modells erläutert.

Für die Komplexitätstheorie benötigen wir auch etwas allgemeinere Modelle mit folgenden Optionen:

- ein separates Eingabeband auf dem nur gelesen wird;
- ein separates Ausgabeband auf dem nur geschrieben wird;
- ein allgemeinerer Arbeitsspeicher, z.B.
  - k lineare Bänder (für $k \geq 1$),
  - mehrdimensionale Speicher,
  - baumförmige Speicher.

Die entsprechenden Definitionen von Konfigurationen, Berechnungen etc. müssen dem jeweiligen Modell angepasst werden. Wir erläutern dies hier exemplarisch für ein bestimmtes Modell.

Definition 1.1. Eine *(deterministische) Turingmaschine mit separatem Inputband, separatem Outputband und k linearen Arbeitsbändern* ist gegeben durch ein Tupel

$$M = (Q, \Gamma_{in}, \Gamma_{out}, \Sigma, q_0, F, \delta).$$

Dabei ist

- Q eine endliche Menge von Zuständen,

- Σ das Arbeitsalphabet mit einem ausgezeichnetem Symbol $\square$ (Blank),
- $\Gamma_{in}, \Gamma_{out}$ das Input- bzw. Outputalphabet,
- $q_0 \in Q$ der Anfangszustand,
- $F \subseteq Q$ die Menge der Endzustände und
- $\delta : (Q - F) \times \Gamma_{in} \times \Sigma^k \to Q \times \{-1, 0, 1\} \times \Sigma^k \times \{-1, 0, 1\}^k \times (\Gamma_{out} \cup \{*\})$ die Übergangsfunktion.

Eine *Konfiguration* ist die vollständige Beschreibung aller relevanten Daten zu einem bestimmten Zeitpunkt in einer Berechnung (Zustand, Speicherinhalt, Input, etc.) Es ist zweckmässig, zwischen *partiellen* und *totalen* Konfigurationen zu unterscheiden.

Definition 1.2. Sei M eine TM. Eine *partielle Konfiguration* von M ist ein Tupel $C = (q, w_1, \ldots, w_k, p_0, p_1, \ldots, p_k) \in Q \times (\Sigma^*)^k \times \mathbb{N}^{k+1}$ welches folgende Daten enthält:

- den aktuellen Zustand q,
- die Inschriften $w_1, \ldots, w_k$ der Arbeitsbänder,
- die Position p_0 auf dem Inputband und
- die Positionen $p_1, \ldots, p_k$ auf den Arbeitsbändern.

Die Inschrift des i-ten Arbeitsbandes ist durch ein endliches Wort $w_i = w_{i0} \cdots w_{im} \in \Sigma^*$ gegeben. Auf den Feldern $j > m$ des als unendlich angenommenen Bandes stehen dann ausschliesslich Blanks.

Werden zusätzlich zu einer partiellen Konfiguration die Inschriften des Ein- und Ausgabebandes angegeben, so erhält man eine *totale Konfiguration* von M.

Die *totale Anfangskonfiguration* $C_0(x)$ von M auf $x \in \Gamma_{in}^*$ ist gegeben durch

- den Anfangszustand q_0,
- leeren Speicher, d.h. $w_1 = w_2 = \ldots = w_k = \lambda$, (wobei λ das leere Wort bezeichnet),
- die Position 0 auf allen Bändern,
- die Inschrift x auf dem Inputband sowie
- die Inschrift λ auf dem Outputband.

Bemerkung. Eine *Endkonfiguration* ist eine Konfiguration $C = (q, \overline{w}, \overline{p}, x, y)$ mit $q \in F$. Das Wort y (die Inschrift auf dem Ausgabeband) ist der *Output* der Endkonfiguration C.

Nachfolgekonfiguration. Sei $C = (q, w_1, \ldots, w_k, p_0, p_1, \ldots, p_k, x, y)$ eine (totale) Konfiguration einer Turingmaschine M. Der Übergang zur nächsten Konfiguration ist bestimmt durch den Wert der Übergangsfunktion δ auf dem aktuellen Zustand q und den in C gelesenen Symbolen x_{p_0} auf dem Inputband und $w_{1,p_1}, \ldots, w_{k,p_k}$ auf den Arbeitsbändern. Sei

$$\delta(q, x_{p_0}, w_{1p_1}, \ldots, w_{kp_k}) = (q', m_0, a_1, \ldots, a_k, m_1, \ldots, m_k, b).$$

$\Delta(C) = (q', \overline{w}', \overline{p}', x, y')$ ist die Nachfolgekonfiguration von C, wenn

- w_i' aus w_i hervorgeht durch Ersetzen des Symbols w_{ip_i} durch a_i,
- $p_i' = p_i + m_i$ (für $i = 0, \ldots, k$) und
- $y' = \begin{cases} y & \text{wenn } b = * \\ yb & \text{wenn } b \in \Gamma_{out} \end{cases}$

Notation. $C \vdash_M C'$, wenn $C' = \Delta(C)$.

Definition 1.3. Eine *Berechnung* von M auf x ist eine Folge $C_0, C_1, \ldots$ von totalen Konfigurationen von M, so dass $C_0 = C_0(x)$ und $C_i \vdash_M C_{i+1}$ für alle $i \geq 0$. Die Berechnung ist vollständig, wenn sie unendlich ist oder in einer Endkonfiguration endet.

Die von M berechnete Funktion ist eine partielle Funktion $f_M : \Gamma_{in}^* \longrightarrow \Gamma_{out}^*$. Dabei ist $f_M(x) = y$ genau dann, wenn die vollständige Berechnung von M auf x endlich ist und in einer Endkonfiguration mit Output y endet.

Definition 1.4. Ein *k-Band Akzeptor* ist eine k-Band-Turingmaschine M deren Endzustandsmenge F zerlegt ist in eine Menge F^+ von *akzeptierenden* Zuständen und eine Menge F^- von *verwerfenden* Zuständen. M *akzeptiert* x genau dann, wenn M auf x in einem Zustand $q \in F^+$ hält. M *verwirft* x genau dann, wenn M auf x in einem Zustand $q \in F^-$ hält.

Definition 1.5. Sei $L \subseteq \Gamma_{in}^*$. M *entscheidet* L, falls M alle $x \in L$ akzeptiert und alle $x \in \Gamma_{in}^* - L$ verwirft. L ist *entscheidbar*, wenn es einen Akzeptor M gibt, welcher L entscheidet. Mit $L(M)$ bezeichnen wir die Menge der von M akzeptierten Inputs.

Im Folgenden werden wir oft auch k-Band Turingmaschinen ohne separate Input- und Outputbänder verwenden. Das erste Arbeitsband spielt dann gleichzeitig die Rolle des Inputbandes, irgendein Band (oder auch mehrere) dient als Outputband.

Konventionen (sofern nicht anderes explizit festgelegt ist): Eine Turingmaschine (TM) ist eine k-Band-TM (für beliebiges $k \geq 1$). Dabei steht k für die totale Anzahl der Bänder, d.h. eventuell vorhandene separate Input- oder Outputbänder werden mitgezählt. Γ steht im folgenden für das Inputalphabet.

1.2 Zeit- und Platzkomplexitätsklassen

Definition 1.6. Sei M eine TM, x eine Eingabe. Dann ist $time_M(x)$ die Länge der vollständigen Berechnung von M auf x, und $\text{space}_M(x)$ die Anzahl der im Verlauf der vollständigen Berechnung von M auf x besuchten Elemente des Arbeitsspeichers. Seien $T, S : \mathbb{N} \to \mathbb{R}$ monoton wachsende Funktionen. Eine TM M ist *T-zeitbeschränkt* bzw. *S-platzbeschränkt*, falls für alle Eingaben $x \in \Gamma^*$ gilt: $\text{time}_M(x) \leq T(|x|)$ bzw. $\text{space}_M(x) < S(|n|)$.

Definition 1.7. *(i)* $\text{DTIME}_k(T)$ ist die Menge aller Sprachen L, für die es eine T-zeitbeschränkte k-Band-TM gibt, welche L entscheidet.

(ii) $\text{DTIME}(T) = \bigcup_{k \in \mathbb{N}} \text{DTIME}_k(T(n))$

(iii) $\text{DSPACE}_k(S)$ ist die Menge aller Sprachen L, für die es eine S-platzbeschränkte k-Band-TM gibt, welche L entscheidet.

(iv) $\textsc{Dspace}(S) = \bigcup_{k\in\mathbb{N}} \textsc{Dspace}_k(S)$.

(v) $\textsc{Dtime-space}_k(T,S)$ ist die Menge aller Sprachen L, für die es eine T-zeitbeschränkte und S-platzbeschränkte k-Band-TM gibt, welche L entscheidet.

(vi) $\textsc{Dtime-space}(T,S) = \bigcup_{k\in\mathbb{N}} \textsc{Dtime-space}_k(T,S)$

Wichtige Komplexitätsklassen sind insbesondere

- $\textsc{Logspace} := \bigcup_{d\in\mathbb{N}} \textsc{Dspace}(d\log n)$
- $\mathrm{P} := \bigcup_{d\in\mathbb{N}} \textsc{Dtime}(n^d)$
- $\textsc{Pspace} := \bigcup_{d\in\mathbb{N}} \textsc{Dspace}(n^d)$
- $\textsc{Exptime} := \bigcup_{d\in\mathbb{N}} \textsc{Dtime}(2^{n^d})$
- $\textsc{Expspace} := \bigcup_{d\in\mathbb{N}} \textsc{Dspace}(2^{n^d})$

Vorsicht: In der Literatur wird manchmal auch die Definition $\textsc{Exptime} := \bigcup_{d\in\mathbb{N}} \textsc{Dtime}(2^{dn})$ verwendet.

Über den Zusammenhang von Zeit- und Platzkomplexität erhält man folgende elementare Beobachtungen.

Satz 1.8. *(a) Für alle $T : \mathbb{N} \to \mathbb{N}$ gilt $\textsc{Dtime}(T) \subseteq \textsc{Dspace}(O(T))$.*

(b) Für alle $S\colon \mathbb{N} \to \mathbb{N}$ mit $S(n) \geq \log n$ gilt $\textsc{Dspace}(S) \subseteq \textsc{Dtime}(2^{O(S)})$.

Beweis.

(a) Eine k-Band Turingmaschine kann in jedem Schritt höchstens k neue Speicherzellen besuchen.

(b) Da $L \in \textsc{Dspace}(S)$ können wir annehmen, dass L durch eine TM M mit Eingabeband und k Arbeitsbändern in Platz S entschieden wird. Für jede Eingabe x gilt, dass in der Berechnung von M auf x keine partielle Konfiguration mehr als einmal vorkommt. Denn andernfalls würde M in eine unendliche Schleife laufen und L nicht entscheiden. Die Anzahl der partiellen Konfigurationen mit Platzbedarf $S(n)$ ist beschränkt durch

$$|Q| \cdot (n+1) \cdot S(n)^k \cdot |\Sigma|^{S(n)} = 2^{O(S(n))}, \text{ falls } S(n) \geq \log n.$$

Dabei ist $(n+1)$ die Anzahl möglicher Kopfpositionen auf dem Eingabeband, $S(n)^k$ die Anzahl der Kopfpositionen auf den Arbeitsbändern und $|\Sigma|^{S(n)}$ die Anzahl möglicher Speicherinhalte. Also gilt $\mathrm{time}_M(x) \leq 2^{O(S(n))}$.

□

Korollar 1.9. $\textsc{Logspace} \subseteq \mathrm{P} \subseteq \textsc{Pspace} \subseteq \textsc{Exptime}$.

Satz 1.10 (Bandreduktion). *Sei $S(n) \geq n$. Dann ist*

$$\textsc{Dtime-space}(T,S) \subseteq \textsc{Dtime-space}_1(O(T\cdot S), S).$$

Beweis. (Simulation von k-Band-TM durch 1-Band-TM) Sei M eine T-zeitbeschränkte und S-platzbeschränkte k-Band-TM, welche L entscheidet. Die Idee des Beweises liegt darin, die k Bänder von M auf $2k$ Spuren eines einzigen Bandes einer 1-Band-TM M' zu simulieren. Dabei soll Spur $2j-1$ des Bandes von M' die Inschrift von Band j von M enthalten und Spur $2j$ eine Marke $(*)$ genau dort, wo der j-te Kopf von M gerade steht.

Vor der Simulation eines Schrittes von M steht der Kopf von M' auf der Marke, die am weitesten links steht. Zur Simulation des Schrittes werden dann drei Phasen durchlaufen:

- M' geht nach rechts bis zur letzten Marke und speichert („in der Zustandsmenge“) die Symbole, auf denen die Köpfe von M stehen, also die Informationen die zur Bestimmung des nächsten Schrittes von M notwendig sind. Zeitaufwand: höchstens $S(n)$ Schritte,
- M' bestimmt den Schritt von M. Zeitaufwand: ein Schritt.
- M' geht zurück zur ersten Markierung und führt unterwegs die notwendigen Änderungen auf dem Band durch. Zeitaufwand: höchstens $S(n)$ Schritte.

M' akzeptiert (bzw. verwirft) genau dann, wenn M akzeptiert (bzw. verwirft). Es sind höchstens $S(n)$ Felder beschriftet, die Marken liegen also höchstens $S(n)$ Felder auseinander. Um einen Schritt von M nachzuvollziehen, braucht die simulierende 1-Band-TM also $O(S(n))$ Schritte und nicht mehr Platz als die simulierte Maschine, woraus die Behauptung folgt. □

Wo wird benutzt, dass $S(n) \geq n$? Sei M eine 2-Band-TM, wobei das erste Band ein separates Eingabeband ist, und sei M S-beschränkt mit $S(n) < n$. Sofern M den gesamten Input liest wird die simulierende 1-Band-TM auf der ersten Spur ganz nach rechts gehen müssen, um dort Marken zu setzen. Die Marken können dann mehr als $S(n)$ Felder auseinanderliegen.

Korollar 1.11. $\textsc{Dtime}(T) \subseteq \textsc{Dtime}_1(O(T^2))$.

Dies folgt aus Satz 1.10 mittels der Tatsache, dass $\text{space}_M(x) \leq O(\text{time}_M(x))$ für alle M und alle x. Ausserdem folgt ebenso:

Korollar 1.12. $\textsc{Dspace}(S) \subseteq \textsc{Dspace}_1(S)$ *für* $S(n) \geq n$.

1.3 Speed-Up und Platzkompression

Definition 1.13. Für Funktionen $f, g : \mathbb{N} \to \mathbb{R}$ bedeutet $f = o(g)$, dass $\lim_{n\to\infty} f(n)/g(n) = 0$.

Satz 1.14 (Speed-Up Theorem). *Sei $k > 1$ und $\varepsilon > 0$. Dann gilt für alle $T : \mathbb{N} \to \mathbb{R}^{\geq 0}$ mit $n = o(T(n))$, dass* $\textsc{Dtime}_k(T) \subseteq \textsc{Dtime}_k(\max(n, \varepsilon \cdot T(n)))$.

Beweis. Sei M eine k-Band-TM welche L in Zeit $T(n)$ entscheidet. Sei weiter m so gewählt, dass $\varepsilon \cdot m \geq 16$. Sei Σ das Arbeitsalphabet von M. Wir beschreiben eine k-Band TM M', welche über dem Alphabet $\Sigma \cup \Sigma^m$ arbeitet. Sie kann damit m Symbole von M durch ein einziges Symbol kodieren und so die Berechnung beschleunigen.

(1) M' kopiert die Eingabe auf ein anderes Band um und komprimiert dabei m-Symbole in eines. Nachher fasst M' dieses Arbeitsband als Eingabeband auf. Zeitaufwand: n Schritte für das Umkopieren und $\lceil \frac{n}{m} \rceil$ Schritte, um den Lesekopf auf das erste Symbol der komprimierten Eingabe zurückzuführen.

(2) M' simuliert jeweils m Schritte von M in 8 Schritten. Auf jedem Band werden folgende Operationen ausgeführt:

   (a) M' speichert „in der Zustandsmenge“ den Inhalt der beiden Nachbarfelder des gerade betrachteten Feldes. Dazu werden 4 Schritte benötigt. 1 mal links, 2 mal rechts und wieder 1 mal links.

   (b) M' bestimmt des Resultat der nächsten m Schritte von M. Dies ist fest in der Übergangsfunktion von M' kodiert. In m Schritten kann M nur den Inhalt von Feldern benutzen bzw. verändern, die $\leq m$ Schritte weit weg sind, also nur den Inhalt des aktuellen Feldes von M' sowie der beiden Nachbarfelder. M' benötigt 4 Schritte, um diese Änderung durchzuführen.

   (c) M' akzeptiert bzw. verwirft genau dann, wenn M akzeptiert bzw. verwirft.

Sei x eine Eingabe der Länge n. Dann gilt

$$\text{time}_{M'}(x) \leq n + \lceil n/m \rceil + 8 * \lceil T(n)/m \rceil \leq n + n/m + 8T(n)/m + 2.$$

Da $n = o(T(n))$, gibt es zu jedem $d > 0$ ein n_d, so dass $T(n)/n \geq d$ für alle $n \geq n_d$. Also $n \leq T(n)/d$ für $n \geq n_d$. Für $n \geq \max(2, n_d)$ ist dann $2n \geq n + 2$. Also macht M' höchstens

$$2n + \frac{n}{m} + 8\frac{T(n)}{m} \leq T(n)(\frac{2}{d} + \frac{1}{md} + \frac{8}{m}) = T(n)(\frac{2m + 8d + 1}{md})$$

Schritte. Setze $d = \frac{2m+1}{8}$. Dann ist die Schrittzahl von M' höchstens

$$T(n)\frac{8(2m + 1 + 2m + 1)}{m(2m + 1)} = \frac{16}{m}T(n) \leq \varepsilon T(n)$$

für alle $n \geq \max(2, n_d)$. Die endlich vielen Inputs der Länge $< n_d$ können in Zeit n_d akzeptiert werden. □

Korollar 1.15. *Für alle $T : \mathbb{N} \to \mathbb{R}$ mit $n = o(T(n))$ und alle $\varepsilon > 0$ gilt* $\text{DTIME}(T(n)) = \text{DTIME}(max(n, \varepsilon T(n))$.

Analog, aber einfacher, zeigt man:

Satz 1.16 (Platzkompression). *Für jede Funktion $S : \mathbb{N} \to \mathbb{R}^{\geq 0}$ und alle $\varepsilon > 0$ gilt:* $\text{DSPACE}(S) \subseteq \text{DSPACE}(max(1, \varepsilon \cdot S(n))$.

1.4 Das Gap-Theorem

In diesem und dem folgenden Abschnitt wird die Frage behandelt, ob man immer mit mehr zur Verfügung stehenden Ressourcen auch mehr Probleme lösen kann. Wenn S_2 schneller wächst als S_1, ist dann auch $\text{DSPACE}(S_2) \supsetneq \text{DSPACE}(S_1)$ (und analog für Zeitkomplexität)? Wir zeigen, dass dies im allgemeinen nicht gilt. Als Vorbereitung beweisen wir folgendes Lemma.

Lemma 1.17. *Sei M ein k-Band-Akzeptor mit $\max_{|x|=n} \text{space}_M(x) \leq S(n)$ für fast alle n, und sei $L(M)$ die Menge der von M akzeptierten Inputs. Dann ist $L(M) \in \text{DSPACE}(S)$.*

„Fast alle“ bedeutet „alle, bis auf endlich viele“.

Beweis. Nach Voraussetzung ist die Menge $X = \{x : \text{space}_M(x) > S(|x|)\}$ endlich. Man kann daher für alle $x \in X$ die Antwort, ob $x \in L(M)$, fest in der Übergangsfunktion einer TM kodieren, wodurch $L(M) \cap X$ ohne Benutzung des Speichers und $L(M) \cap \bar{X}$ durch Simulation von M entschieden werden kann. Damit wird ganz $L(M)$ mit Platz $S(n)$ entschieden. □

Satz 1.18 (Gap-Theorem). *Für jede totale berechenbare Funktion $g\colon \mathbb{N} \to \mathbb{N}$ mit $g(n) \geq n$ gibt es eine totale berechenbare Funktion $S : \mathbb{N} \to \mathbb{N}$, so dass* $\textsc{Dspace}(S) = \textsc{Dspace}(g \circ S)$.

Dabei bedeutet $(g \circ S)(n) := g(S(n))$. Das Gap-Theorem besagt also, dass es beliebig grosse Lücken (englisch: gaps) in der Hierarchie der Platzkomplexitätsklassen gibt. Wir benutzen dabei die auch für die Berechenbarkeitstheorie fundamentale Beobachtung, dass man jeder Turingmaschine M ein Wort $\rho(M) \in \{0,1\}^*$ so zuordnen kann, dass das Berechnungsverhalten von M aus $\rho(M)$ effektiv (mittels einer universellen Turingmaschine) rekonstruiert werden kann (siehe Logik-Skript). Insbesondere gewinnt man so eine Aufzählung $M_0, M_1, M_2, \ldots$ aller Turingmaschinen. Für jede entscheidbare Menge L gibt es dann eine Zahl $i \in \mathbb{N}$, so dass M_i L entscheidet.
Beweis. Sei $M_0, M_1, \cdots$ eine Aufzählung aller Turingmaschinen und $S_i(n) := \max_{|x|=n} \text{space}_{M_i}(x)$. Man beweist nun zuerst:

Lemma 1.19. *Die Menge $\{(i,n,m) \mid S_i(n) = m\}$ ist entscheidbar.*

Beweis. Für jedes Tripel (i,n,m) gibt es eine Zeitschranke $t \in \mathbb{N}$, so dass M_i auf Eingaben der Länge n nicht mehr als t Schritte machen kann, ohne mehr als m Felder zu gebrauchen oder eine Konfiguration zweimal zu durchlaufen. Dabei ist t aus (i,n,m) berechenbar. Also kann durch Simulation von t Schritten von M_i auf den endlich vielen Eingaben der Länge n entschieden werden, ob $S_i(n) = m$. □

Man benutzt dies zur Konstruktion einer Funktion $S : \mathbb{N} \to \mathbb{N}$ derart, dass für jedes $i \in \mathbb{N}$ gilt:

Entweder $S_i(n) \leq S(n)$ für fast alle n,
oder $S_i(n) \geq g(S(n))$ für unendlich viele n.

Zur Konstruktion von S sei nun $P = \{(i,n,y) \in \mathbb{N}^3 \mid y < S_i(n) \leq g(y)\}$. Aus Lemma 1.19 und aus der Berechenbarkeit von g folgt, dass P entscheidbar ist. Die Funktion $S : \mathbb{N} \to \mathbb{N}$ wird durch folgenden Algorithmus definiert:

Input: n
$y := 1$
while es existiert $i \leq n$ mit $(i,n,y) \in P$
  do wähle das kleinste solche i
  setze $y := S_i(n)$
  od
$S(n) := y$

Da P entscheidbar ist, ist S eine totale berechenbare Funktion. Es bleibt noch zu zeigen, dass

$$\textsc{Dspace}(g \circ S) - \textsc{Dspace}(S) = \varnothing.$$

Sei also $L \in \text{DSPACE}(g \circ S)$. Dann ist $L = L(M_i)$ für ein $i \in \mathbb{N}$. Da $L \in \text{DSPACE}(g \circ S)$, gilt $S_i(n) \leq g(S(n))$ für alle $n \in \mathbb{N}$. Aus der Konstruktion von S folgt, dass $S_i(n) \leq S(n)$ für alle $n \geq i$ gilt, denn andernfalls wäre $S(n) < S_i(n) \leq g(S(n))$ für $i \leq n$, was durch den Algorithmus ausgeschlossen wurde.

Also ist $S_i(n) \leq S(n)$ für fast alle n, woraus aber nach Lemma 1.17 folgt, dass $L = L(M_i) \in \text{DSPACE}(S)$. □

Völlig analog beweist man

Satz 1.20 (Gap-Theorem für Zeitkomplexität). *Für jede totale berechenbare Funktion g gibt es eine totale berechenbare Funktion T mit* $\text{DTIME}(T) = \text{DTIME}(g \circ T)$.

Folgerung:
Es gibt berechenbare Funktionen f, g, h mit

- $\text{DTIME}(f) = \text{DTIME}(2^f)$.
- $\text{DTIME}(g) = \text{DTIME}(2^{2^g})$.
- $\text{DTIME}(h) = \text{DTIME}(2^{\overset{\left.\cdot^{\cdot^{2}}\right\}h(n) \text{ mal}}})$.

1.5 Hierarchiesätze

Im letzten Abschnitt wurde gezeigt, dass die Erhöhung einer Komplexitätsschranke nicht immer ausreicht, um mehr Probleme entscheiden zu können. Nun soll untersucht werden, unter welchen Voraussetzungen eine Komplexitätsklasse echt in einer anderen enthalten ist. Dazu wird die Diagonalisierungsmethode verwendet werden, analog zum Beweis der Unentscheidbarkeit des Halteproblems für Turingmaschinen (siehe Logik-Skript).

Der Beweis wird sehr allgemein, nicht auf Zeit- oder Platzkomplexität beschränkt, geführt werden:

Sei $\mathcal{M}$ eine Klasse von abstrakten Maschinen (z.B. 2-Band-TM), R eine für Maschinen aus $\mathcal{M}$ definierte Ressource (z.B. Zeit oder Platz), so dass für jede Maschine $M \in \mathcal{M}$ und für jede Eingabe x die Grösse $R_M(x) \in \mathbb{N} \cup \{\infty\}$ definiert ist. Für eine Funktion $T\colon \mathbb{N} \to \mathbb{N}$ ist dann $R(T)$ die Komplexitätsklasse aller Probleme, welche mit Maschinen aus $\mathcal{M}$ mit T-beschränkter Ressource R entscheidbar sind, also

$$R(T) = \{L : \text{es gibt ein } M \in M, \text{ welches } L \text{ entscheidet, so dass } R_M(x) \leq T(|x|) \text{ für alle } x\}.$$

Weiterhin wird angenommen, dass eine Kodierung ρ festgelegt ist, welche Maschinen aus $\mathcal{M}$ durch Worte über $\{0,1\}$ kodiert, so dass die Struktur und das Berechnungsverhalten von M aus $\rho(M)$ effektiv erschlossen werden kann.

Seien $T, t\colon \mathbb{N} \to \mathbb{N}$ Funktionen, $\mathcal{M}_1$ und $\mathcal{M}_2$ Klassen von Akzeptoren und R, r Ressourcen, die für $\mathcal{M}_1$ und $\mathcal{M}_2$ definiert sind. Damit erhält man die Komplexitätsklassen $R(T)$ und $r(t)$.

Definition 1.21. $R(T)$ erlaubt Diagonalisierung über $r(t)$, wenn eine Maschine $D \in \mathcal{M}_1$ existiert, für die gilt:

- D ist in der Ressource R T-beschränkt und hält auf jede Eingabe. Also ist $L(D) \in R(T)$.

- Für jede Maschine $M \in \mathcal{M}_2$, die in der Ressource r t-beschränkt ist, gilt für fast alle $x \in \{0,1\}^*$:

$$\rho(M)\#x \in L(D) \Leftrightarrow \rho(M)\#x \notin L(M).$$

Satz 1.22 (Allgemeiner Hierarchiesatz). *Wenn $R(T)$ Diagonalisierung über $r(t)$ erlaubt, dann ist $R(T) - r(t) \neq \varnothing$.*

Beweis. Sei D die Diagonalisierungsmaschine aus Definition 1.21. Man zeigt, dass $L(D) \notin r(t)$. Andernfalls gäbe es eine in der Ressource r t-beschränkte Maschine M mit $L(D) = L(M)$. Das kann aber nicht sein, da für fast alle x gilt:

$$\rho(M)\#x \in L(D) \Leftrightarrow \rho(M)\#x \notin L(M).$$

Also ist $L(M) \neq L(D)$. □

Definition 1.23. Eine Funktion $T\colon \mathbb{N} \to \mathbb{N}$ heisst *voll zeitkonstruierbar*, falls eine Turingmaschine M existiert, so dass $\text{time}_M(x) = T(|x|)$ für alle x. Analog ist $S : \mathbb{N} \to \mathbb{N}$ *voll platzkonstruierbar*, wenn $\text{space}_M(x) = S(|x|)$ für eine TM M und alle x.

Zeit- bzw. platzkonstruierbare Funktionen sind „anständige“ Funktionen, deren Komplexität nicht wesentlich grösser ist als ihre Werte. Die meisten üblicherweise verwendeten Funktionen sind voll zeit- und platzkonstruierbar. Insbesondere gilt dies für n^k, 2^n und $n!$. Wenn die Funktionen f und g diese Eigenschaft besitzen, dann auch die Funktionen $f + g$, $f \cdot g$, 2^f und f^g.

Satz 1.24. *Seien $T, t : \mathbb{N} \longrightarrow \mathbb{R}^{\geq 0}$, so dass $T(n) \geq n$, T voll zeitkonstruierbar und $t = o(T)$. Dann gilt für alle $k \in \mathbb{N}$: $\text{DTIME}_k(t) \subsetneq \text{DTIME}(T)$.*

Beweis. Wir zeigen, dass $\text{DTIME}(T)$ Diagonalisierung über $\text{DTIME}_k(t)$ erlaubt. Dazu sei D eine TM mit folgenden Eigenschaften:

(a) Auf die Eingabe $\rho(M)\#x$, wobei M eine k-Band TM und $x \in \{0,1\}^*$ ist, simuliert D die Berechnung von M auf $\rho(M)\#x$.

(b) Für jedes M gibt es eine Konstante c_M, so dass D für die Simulation jedes Schrittes von M höchstens c_M Schritte benötigt.

(c) Gleichzeitig simuliert D auf separaten Bändern eine TM N, welche auf Eingaben der Länge n genau $T(n)$ Schritte macht. Diese TM existiert aufgrund der Zeitkonstruierbarkeit von T.

(d) Nach $T(n)$ ($n = |\rho(M)\#x|$) Schritten hält D und akzeptiert $\rho(M)\#x$ genau dann, wenn die simulierte Berechnung von M auf $\rho(M)\#x$ bereits zur Verwerfung geführt hat. Wenn M bereits akzeptiert hat oder die simulierte Berechnung noch nicht vollständig ist, verwirft D.

Sei $L(D) = \{\rho(M)\#x : D \text{ akzeptiert } \rho(M)\#x\}$. Es gilt

- $L(D) \in \text{DTIME}(T)$.

- Für alle M gilt $T(n) \geq c_M \cdot t(n)$ für fast alle n (da $t = o(T)$). Also kann D für fast alle Eingaben $\rho(M)\#x$ in $T(n)$ Schritten die Berechnung von M auf $\rho(M)\#x$ vollständig simulieren. Daher gilt $\rho(M)\#x \in L(D) \iff \rho(M)\#x \notin L(M)$. Nach dem allgemeinen Hierarchiesatz folgt die Behauptung.

□

Korollar 1.25 (Zeithierarchiesatz). *Falls $T(n) \geq n$ voll zeitkonstruierbar und $t^2 = o(T)$, dann ist* $\text{DTIME}(T(n)) - \text{DTIME}(t(n)) \neq \varnothing$.

Beweis. Nach Korollar 1.11 gibt es eine Konstante c, so dass $\text{DTIME}(t) \subseteq \text{DTIME}_1(ct^2)$. Wenn $t^2 = o(T)$ ist, dann ist auch $ct^2 = o(T)$. Aufgrund des Satzes 1.24 gibt es also eine Sprache

$$L \in \text{DTIME}(T) - \text{DTIME}_1(c \cdot t^2) \subseteq \text{DTIME}(T) - \text{DTIME}(t).$$

□

Anwendungen. Da $2^{n \log n}$ zeitkonstruierbar ist und $(2^n)^2 = 2^{2n} = o(2^{n \log n})$ folgt z.B.

$$\text{DTIME}(2^n) \subsetneq \text{DTIME}(2^{n \cdot \log n}).$$

Insbesondere gilt

Korollar 1.26. *Für jede monoton wachsende Funktion f mit $\lim_{n\to\infty} f(n) = \infty$ gilt, dass* P *(die Klasse der in Polynomialzeit entscheidbaren Probleme) echt in* $\text{DTIME}(n^{f(n)}$ *enthalten ist. Insbesondere ist $P \subsetneq$* EXPTIME.

Verwendet man anstelle von Korollar 1.11 ($\text{DTIME}(t) \subseteq \text{DTIME}_1(t^2)$) eine Simulation von k-Band TM durch 2-Band TM, so folgt eine schärfere Form des Zeithierarchiesatzes.

Satz 1.27. *Für $T(n) \geq n$ gilt:*

$$\text{DTIME}(T) \subseteq \text{DTIME}_2(T \cdot \log T).$$

Beweis. (Skizze)
Eine k-Band TM M wird durch eine 2-Band TM M' simuliert:

- für jedes Band von M werden 2 Spuren auf dem ersten Band von M' angelegt.
- Das zweite Band von M' wird nur als Zwischenspeicher für Kopieroperationen verwendet.

Das erste Band von M' ist in Blöcke $\ldots, B_{-i}, B_{-i+1}, \ldots, B_{-1}, B_0, B_1, \ldots, B_i$ eingeteilt, wobei $|B_0| = 1, |B_j| = 2^{|j|-1}$ für $j \neq 0$. Die von M gerade gelesenen Zeichen stehen alle im Block B_0. Wenn auf einem Band der Kopf von M nach links geht, dann bewegt M' die Inschrift auf den entsprechenden beiden Spuren nach rechts, so dass das aktuelle Zeichen wieder in B_0 steht. Eine geschickte Implementierung dieser Idee führt zu einer Simulation mit höchstens logarithmischem Zeitverlust: Wenn M T-zeitbeschränkt ist, dann ist M' $T \cdot \log T$ zeitbeschränkt. □

Der Beweis findet sich z.B. in J. Hopcraft, J. Ullmann: *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley 1979, pp. 292-295.

Korollar 1.28 (Zeithierarchiesatz, schärfere Version). *Sei $T(n) \geq n$, T zeitkonstruierbar und $t \cdot \log t = o(T)$. Dann ist* $\text{DTIME}(t) \subsetneq \text{DTIME}(T)$.

Der Beweis verläuft analog zu Satz 1.25 unter Verwendung obiger Konstruktion.

Satz 1.29 (Platzhierarchiesatz). *Sei $S, s : \mathbb{N} \to \mathbb{N}$. Sei $S(n) \geq \log n$, S platzkonstruierbar und $s = o(S)$. Dann ist* $\textsc{Dspace}(S) - \textsc{Dspace}(s) \neq \varnothing$.

Beweis. Da wir nach Satz 1.10 die Anzahl der Arbeitsbänder auf eins reduzieren können ohne den Platzverbrauch zu vergrössern, reicht es, eine TM mit einem Eingabeband und einem Arbeitsband zu betrachten. Da M s-platzbeschränkt ist, gibt es höchstens $|Q| \cdot (n+1) \cdot |\Sigma|^{s(n)} \cdot s(n) = (n+1)2^{O(s(n))}$ verschiedene partielle Konfigurationen von M. M hält also entweder nach $\leq t(n) = 2^{c_M s(n) + \log(n+1)}$ Schritten oder gar nicht. Dabei bezeichnet c_M eine Konstante, welche nur von M, nicht aber von n abhängt. Da S platzkonstruierbar ist, existiert eine TM N mit $\text{space}_N(x) = S(|x|)$ für alle x. Zu zeigen ist: $\textsc{Dspace}(S)$ erlaubt Diagonalisierung über $\textsc{Dspace}(s)$.

D operiert auf den Input $\rho(M)\#x$ wie folgt:

(a) Zuerst wird durch Simulation von N ein Bereich von $S(n)$ Speicherzellen markiert. Die folgenden Operationen werden innerhalb dieses Bereiches durchgeführt. Bei Überschreiten hält D sofort und verwirft die Eingabe.

(b) D initialisiert einen Zähler auf $t(n)$. Dieser wird auf einen speziellen Speicher des Bandes geschrieben.

(c) D simuliert die Berechnung von M auf $\rho(M)\#x$ und verringert den Zähler bei jedem simulierten Schritt um 1.

(d) Falls die Simulation mehr als die markierten Speicherzellen benötigt, oder M nicht in $t(n)$ Schritten hält, verwirft D $\rho(M)\#x$. Ebenso verwirft D, wenn M $\rho(M)\#x$ akzeptiert. Wenn D erfolgreich eine verwerfende Berechnung von M auf $\rho(M)\#x$ simulieren kann, dann akzeptiert D $\rho(M)\#x$.

Es gilt:

- $L(D) \in \textsc{Dspace}(S)$.
- Es bleibt zu zeigen: wenn M s-platzbeschränkt ist, dann kann D für fast alle x die Berechnung von M auf $\rho(M)\#x$ vollständig (oder $t(n)$ Schritte lang) simulieren.
  - da $t(n) = 2^{O(s(n)+\log n)}$, $s = o(S)$ und $S(n) \geq \log n$) kann der Zähler $t(n)$ für hinreichend grosse n durch ein Wort der Länge $S(n)$ kodiert werden.
  - M habe ein Alphabet mit d Symbolen. D braucht dann $\leq \log d$ Felder um ein Symbol von M zu kodieren (man beachte, dass der Faktor $\log d$ nur von M abhängt, nicht aber von der Länge der Eingabe).
  - D braucht zur Simulation von M höchstens Platz $\log d \cdot \text{space}_M(\rho(M)\#x) \leq \log d \cdot s(n) \leq S(n)$ für fast alle n.

Für hinreichend lange x gilt also: $\rho(M)\#x \in L(D) \iff \rho(M)\#x \notin L(M)$. Also erlaubt $\textsc{Dspace}(S)$ Diagonalisierung über $\textsc{Dspace}(s)$. Mit dem allgemeinen Hierarchiesatz folgt die Behauptung.

□

Folgerung. LOGSPACE $\subsetneq$ PSPACE.

Bemerkung. Es gilt LOGSPACE $\subseteq$ P $\subseteq$ PSPACE. Es ist nicht bekannt, ob LOGSPACE $\subsetneq$ P oder P $\subsetneq$ PSPACE. Aus dem Platzhierarchiesatz folgt, dass mindestens eine der Inklusionen echt ist, aber nicht, welche !

Kapitel 2

Nichtdeterministische Komplexitätsklassen

2.1 Nichtdeterministische Turingmaschinen

Nichtdeterministische Turingmaschinen (NTM) sind analog zu deterministischen definiert, ausser dass die Übergangsfunktion im allgemeinen mehrere Übergänge zulässt.

Auch hier ist das wichtigste Modell die k-Band TM. Die Übergangsfunktion ist hier also eine Funktion $\delta : Q \times \Sigma^k \to \mathrm{Pot}(Q \times \Sigma^k \times \{-1, 0, 1\}^k)$ (entsprechend abgewandelt, wenn ein Band ein Eingabeband ist). Auch hier werden wir meist *Akzeptoren* behandeln, also mit Endzustandmenge $F = F^+ \cup F^-$. Zu einer Konfiguration gibt es bei einer nichtdeterministischen TM M im allgemeinen mehrere Nachfolgekonfigurationen. Sei $\mathrm{Next}(C) = \{C' : C \vdash_M C'\}$ die Menge der Nachfolgekonfigurationen von C. Dabei existiert für jede NTM M eine Zahl $r \in \mathbb{N}$, so dass $|\mathrm{Next}(C)| \leq r$ für alle Konfigurationen C von M. Zu einer Eingabe x gibt es jetzt nicht nur eine (vollständige) Berechnung, sondern einen *Berechnungsbaum* $\mathfrak{T}_{M,x}$. Die Wurzel von $\mathfrak{T}_{M,x}$ ist die Eingabekonfiguration $C_0(x)$. Die Kinder jedes Knotens C sind die Elemente von $\mathrm{Next}(C)$. Eine Berechnung von M auf x ist eine Folge $C_0, \ldots, C_t$ von Konfigurationen mit $C_0 = C_0(x)$ und $C_{i+1} \in \mathrm{Next}(C_i)$, also ein Pfad durch $\mathfrak{T}_{M,x}$ der an der Wurzel beginnt.

Definition 2.1. Eine nichtdeterministische TM ist T-zeitbeschränkt (bzw. S-platzbeschränkt), wenn *keine* Berechnung von M auf Inputs der Länge n länger als $T(n)$ Schritte dauert (bzw. mehr als $S(n)$ Speicherplätze braucht).

Definition 2.2. Sei M eine NTM, x die Eingabe. M *akzeptiert* x, wenn es eine Berechnung von M auf x gibt, welche in einer akzeptierenden Konfiguration endet. $L(M) = \{x : M \text{ akzeptiert } x\}$ ist die von M akzeptierte Sprache.

Definition 2.3. $\textsc{Ntime}(T) := \{L : \text{es gibt eine } T\text{-zeitbeschränkte TM mit } L(M) = L\}$.
$\textsc{Nspace}(S) := \{L : \text{es gibt eine } S\text{-platzbeschränkte TM mit } L(M) = L\}$.
Andere Klassen, etwa $\textsc{Ntime}_k(T)$ werden analog definiert.

Bemerkung. In informellen Beschreibungen von nichtdeterministischen Algorithmen werden nichtdeterminische Schritte sehr oft durch "Erraten" beschrieben. "Errate ein $y \in \Sigma^m$ " bezeichnet dabei eine Folge von m nichtdeterministischen Schritten, wobei im i-ten Schritt das i-te Symbol von y aus den $|\Sigma|$ Möglichkeiten nichtderministisch bestimmt wird. Dem Befehl entsprcht also ein Berechnungsbaum der Tiefe m, mit den $|\Sigma|$ vielen Nachfolgern an jedem innern Knoten, dessen $|\Sigma|^m$ Blätter mit $|\Sigma|^m$ möglichen Strings $y \in \Sigma^m$ beschriftet sind.

Beispiel. Ein nichtdeterministischer Algorithmus für das Labyrinthproblem

Input: $G = (V, E)$, ein gerichteter Graph, $a, b \in V$ $(|V| = n)$
$x := a$
wiederhole n Mal: **do**
if $x = b$ akzeptiere, **end**
else errate $y \in V$ mit $(x, y) \in E$
$x := y$
od
verwerfe, **end**

Wenn ein Pfad in G von a nach b existiert, dann auch einer der Länge $\leq n$. Also gibt es eine akzeptierende Berechnung des Algorithmus genau dann, wenn ein Pfad von a nach b existiert. Der Algorithmus braucht $\leq 3 \cdot \log n$ Platz (jeweils $\log n$ für x, y und den Zähler). Das bedeutet, dass das Labyrinthproblem in $\text{NLOGSPACE} = \text{NSPACE}(O(\log n))$ liegt. In einer Übung wurde gezeigt, dass das Labyrinthproblem auch in $\text{DSPACE}(O(\log^2 n))$ liegt.

2.2 Elementare Eigenschaften nichtdeterministischer Komplexitätsklassen

Satz 2.4. *Für alle* $T : \mathbb{N} \to \mathbb{R}^+$ *gilt:* $\text{DTIME}(T) \subseteq \text{NTIME}(T) \subseteq \text{DTIME}(2^{O(T)})$

Beweis. Die Aussage $\text{DTIME}(T) \subseteq \text{NTIME}(T)$ ist trivial. Für die zweite Aussage nehmen wir an, dass M eine nichtdeterministische T-zeitbeschränkte TM ist. Jede Konfiguration in $\mathfrak{T}_{M,x}$ kann mit Platz $O(T(n))$ aufgeschrieben werden und es gibt höchstens $2^{O(T(n))}$ verschiedene Konfigurationen in $\mathfrak{T}_{M,x}$. Die Idee der Beweises ist es, mit einem deterministischen Algorithmus eine Liste von erreichbaren Konfigurationen zu verwalten und zu entscheiden, ob M je eine akzeptierende Konfiguration erreicht. Es ist darauf zu achten, dass diese Liste jede Konfiguration nur einmal enthält, auch wenn sie in $\mathfrak{T}_{M,x}$ mehrfach auftritt.

Input: x
$L := \{C_0(x)\}$, d.h. L enthält die Inputkonfiguration von M auf x
while (L enthält unmarkierte Konfiguration) **do**
wähle $C \in L$ unmarkiert
if (C akzeptierende Konfiguration) akzeptiere x, **end**
else $L := L \cup \text{Next}(C)$
markiere C
od
verwerfe x

Dieser deterministische Algorithmus akzeptiert x genau dann, wenn eine akzeptierende Konfiguration C_a von M existiert, mit $C_0(x) \vdash_M^* C_a$. Der Algorithmus benötigt $2^{O(T(n))}$ Durchläufe der while-Schleife und führt $2^{O(T(n))}$ Operationen pro Durchlauf aus. Man beachte, dass "$L := L \cup \text{Next}(C)$" erfordert, dass für jedes $D \in \text{Next}(C)$ überprüft wird, ob D bereits in L enthalten ist. Daher hat der Algorithmus eine Zeitkomplexität $\leq 2^{O(T(n))} \cdot 2^{O(T(n))} = 2^{O(T(n))}$ □

Satz 2.5 (Satz von Savitch). *Sei* $S(n) \geq \log n$ *platzkonstruierbar. Dann ist* $\text{NSPACE}(S) \subseteq \text{DSPACE}(S^2)$.

Beweis. Sei M eine S-platzbeschränkte NTM. Dann existiert ein d, so dass M auf Inputs der Länge n höchstens $2^{dS(n)}$ verschiedene Konfigurationen erreicht. Wenn M überhaupt eine akzeptierende Berechnung auf x hat, dann auch eine mit höchstens $2^{d \cdot S(n)}$ Schritten.

Jede Konfiguration von M auf x wird durch ein Wort der Länge $c \cdot S(n)$ dargestellt, wobei c eine Konstante ist. Dabei wird benutzt, dass $S(n) \geq \log n$ ist. Sei

$$\text{Conf}[S(n)] := \{C : C \text{ ist eine Konfiguration von } M \text{ mit Platzverbrauch } S(n)\}.$$

Wir definieren eine rekursive, deterministische Prozedur $\mathbf{Reach}(C_1, C_2, k)$, welche, gegeben zwei Konfigurationen $C_1, C_2 \in \text{Conf}[S(n)]$ und eine Zahl $k \in \mathbb{N}$, den Output

$$\mathbf{Reach}(C_1, C_2, k) = \begin{cases} 1 & \text{wenn } M \text{ die Konfiguration } C_2 \text{ ausgehend von } C_1 \\ & \text{in höchstens } 2^k \text{ Schritten ereichen kann} \\ 0 & \text{sonst} \end{cases}$$

berechnet.

Reach(C_1, C_2, k)
if $k = 0$ **then**
  if [$C_1 = C_2 \vee C_2 \in \text{Next}(C_1)$] **then output** 1 **end**
  else output 0 **end**
if $k > 0$ **then**
  for all $C \in \text{Conf}[S(n)]$ **do**
    if [$\text{Reach}(C_1, C, k-1) = 1 \wedge \text{Reach}(C, C_2, k-1) = 1$]
      then output 1 **end**
    od
  output 0 **end**

Sei $f(n,k) = \max\{\text{space}_{\text{Reach}}(C_1, C_2, k) : C_1, C_2 \in \text{Conf}[S(n)]\}$. Dann gilt

- $f(n, 0) = 0$
- $f(n, k+1) \leq c \cdot S(n) + f(n, k)$. Dabei ist $c \cdot S(n)$ der benötigten Platz um C hinzuschreiben. (Platzkonstruierbarkeit von S)

Daraus folgt $f(n, k) \leq k \cdot c \cdot S(n)$.

$L(M)$ kann nun durch folgenden Algorithmus M_{det} entschieden werden:

Input: x
for all $C_a \in \text{Conf}[S(n)], C_a$ akzeptierende Konf. **do**
  if $\text{Reach}(C_0(x), C_a, d \cdot S(n)) = 1$
    then akzeptiere, **end**
  od
verwerfe

Es gilt:

$$\text{space}_{M_{\text{det}}}(x) = O(S(n)) + f(n, d \cdot S(n)) = O(S^2(n)).$$

Daraus folgt, dass $L(M) \in \text{DSPACE}(O(S^2)) = \text{DSPACE}(S^2)$. □

Satz 2.6. *Für $S(n) \geq \log n$, S platzkonstruierbar, gilt:* $\text{NSPACE}(S) \subseteq \text{DTIME}(2^{O(S)})$.

Beweis. Wie beim Beweis des Satzes von Savitch, wird auch hier das Labyrinthproblem angewendet. Sei M eine S-platzbeschränkte NTM und $\text{Conf}[S(n)]$ die Menge der Konfigurationen von M, welche höchstens $S(n)$ Speicherplätze benötigen. Dann ist $G = (\text{Conf}[S(n)], \vdash_M)$ ein gerichteter Graph. M akzeptiert x genau dann, wenn es einen Pfad in G von $C_0(x)$ zu einer akzeptierenden Konfiguration gibt. In Zeit $2^{O(S(n))}$ kann man mit einem deterministischem Algorithmus

(a) G konstruieren und

(b) für alle akzeptierenden Konfigurationen C_a entscheiden, ob in G ein Pfad von $C_0(x)$ zu C_a in G existiert.

Dies folgt aus der Tatsache, dass das Labyrinthproblem durch einen deterministischen Polynomialzeit-Algorithmus gelöst werden kann. □

Korollar 2.7. *(i)* $\text{NLOGSPACE} \subseteq \text{P}$

(ii) $\text{NPSPACE} = \text{PSPACE}$

(iii) $\text{NP} \subseteq \text{PSPACE}$

Beweis.

(i) $\text{NLOGSPACE} := \text{NSPACE}(O(\log n)) \subseteq \text{DTIME}(2^{O(\log n)}) = \text{DTIME}(n^{O(1)}) = \text{P}$

(ii) $\text{NSPACE} := \bigcup_{d\in\mathbb{N}} \text{NSPACE}(n^d) \subseteq \bigcup_{d\in\mathbb{N}} \text{DSPACE}(n^{2d}) = \text{PSPACE}$

(iii) $\text{NP} := \bigcup_{d\in\mathbb{N}} \text{NTIME}(n^d) \subseteq \text{NPSPACE} = \text{PSPACE}$

□

2.3 Der Satz von Immerman und Szelepcsényi

Definition 2.8. Sei $\mathcal{C}$ eine Klasse von Sprachen (z.B. eine Komplexitätsklasse). Dann ist Co-$\mathcal{C} := \{\overline{L} : L \in \mathfrak{C}\}$, wobei $\overline{L}$ das Komplement von L bezeichnet.

Deterministische Komplexitätsklassen $\text{DTIME}(T)$ und $\text{DSPACE}(T)$ sind offensichtlich abgeschlossen unter

- Vereinigung: $L, L' \in \mathcal{C} \implies L \cup L' \in \mathcal{C}$
- Durchschnitt: $L, L' \in \mathcal{C} \implies L \cap L' \in \mathcal{C}$
- Komplement: $L \in \mathcal{C} \implies \overline{L} \in \mathcal{C}$, (d.h. $\mathcal{C}$ =Co-$\mathcal{C}$).

Auch nichtdeterministische Komplexitätsklassen $\text{NTIME}(T)$ und $\text{NSPACE}(S)$ sind abgeschlossen unter Vereinigung und Durchschnitt. Die Abgeschlossenheit unter Komplement ist dagegen nicht offensichtlich und in vielen Fällen vermutlich falsch. Die Vermutung geht dahin, dass die Klasse $\text{NTIME}(T)$ nicht unter Komplement abgeschlossen ist. Dasselbe wurde lange auch für $\text{NSPACE}(S)$ vermutet, bis Immerman und Szelepcsényi 1988 die Fachwelt mit folgendem Resultat überraschten:

Satz 2.9 (Satz von Immerman und Szelepcsényi). *Sei* $S(n) \geq \log n$ *platzkonstruierbar. Dann ist* $\text{NSPACE}(S) = \text{Co-NSPACE}(S)$.

Die Idee des Beweises liegt im nichtdeterministischem „induktiven Zählen" aller erreichbaren Konfigurationen. Wenn die Zahl $R_x(t)$ der in t Schritten erreichbaren Konfigurationen bekannt ist, kann für jede Konfiguration C entschieden werden, ob sie in $t+1$ Schritten erreichbar ist. Falls ja, kann dies durch Raten einer entsprechenden Berechnung zu C überprüft werden. Andernfalls kann man nachprüfen, dass $C \notin \text{Next}(D)$ für $R_x(t)$ Konfigurationen D, welche alle in t Schritten erreichbar sind. Allgemein wird von einem nichtdeterministischem Entscheidungsverfahren für L nur verlangt, dass $x \in L$ genau dann, wenn es eine akzeptierende Berechnung von M auf x gibt. Insbesondere kann es also verwerfende Berechnungen auf x geben, obwohl $x \in L$. Deshalb führen wir einen schärferen Begriff ein, den eines *irrtumfreien* nichtdeterministischen Berechnungs- bzw. Entscheidungsverfahrens.

Definition 2.10. Ein *irrtumfreies nichtdeterministisches Berechnungsverfahren* für eine Funktion f ist eine nichtdeterministische TM M mit folgenden Eigenschaften:

- Jede Berechnung von M auf x hält und produziert als Ausgabe entweder $f(x)$ oder ? (weiss nicht).
- Mindestens eine Berechnung von M ergibt das Resultat $f(x)$.

Ein irrtumfreies nichtdeterministisches Entscheidungsverfahren für eine Sprache L ist ein irrtumfreies nichtdeterministisches Berechnungsverfahren für χ_L.

Wir beweisen nun folgenden Satz, welcher Satz 2.9 impliziert:

Satz 2.11. *Sei* $S(n) \geq \log n$ *platzkonstruierbar. Dann gibt es für jedes* $L \in \text{NSPACE}(S)$ *auch ein irrtumfreies* S*-platzbeschränktes Entscheidungsverfahren.*

Insbesondere gibt es dann auch eines für $\overline{L}$ und es folgt, dass $\overline{L} \in \text{NSPACE}(S)$.
Beweis. Sei M eine S-platzbeschränkte NTM, welche L entscheidet. Sei $C_0(x)$ die Anfangskonfiguration von M auf x und $\text{Conf}[S(n)]$ die Menge der Konfigurationen von M mit Platzverbrauch $\leq S(n)$. Da $S(n) \geq \log n$, lässt sich jede Konfiguration $C \in \text{Conf}[S(n)]$ durch ein Wort der Länge $S(n)$ beschreiben. Sei

$$\text{Reach}_x(t) := \{C \in \text{Conf}[S(n)] : C \text{ ist von } C_0(x) \text{ aus in} \leq t \text{ Schritten erreichbar}\}$$

und sei $R_x(t) := |\text{Reach}_x(t)|$.

1. Es gibt eine nichtdeterministische Prozedur M_0 mit Eingaben x, r, t, C, wobei x die Eingabe für M ist, $r, t \in \mathbb{N}$ und $C \in \text{Conf}[S(n)]$ $(n = |x|)$, so dass gilt: Wenn $r = R_x(t)$, dann entscheidet M_0 irrtumfrei mit Platz $O(S(n))$ ob $C \in \text{Reach}_x(t+1)$. Dabei ist es unerheblich, wie sich M_0 auf (x, r, t, C) mit $r \neq R_x(t)$ verhält.

$M_0(x, r, t, C)$
m := 0
forall $D \in \text{Conf}[S(n)]$ **do**
    simuliere (nichtdet.) höchstens t Schritte von M auf x
    if (D wurde erreicht) **then do** $m = m + 1$
        if $C \in \text{Next}(D)$ **output** 1 **end**

od
od
if $m = r$ **then** **output** 0 **end**
else **output** ? **end**

Bemerkung. Die nichtdeterministische Simulation von höchstens t Schritten geht mit Platz $O(S(n))$ für $t = 2^{O(S(n))}$, z.B. wie folgt:

$C' = C_0(x)$
repeat t **times**
(**if** $D \neq C'$ **then** **do**
errate $C'' \in \mathrm{Next}(C')$
$C' := C''$
od)

Sei $r = R_x(t)$. Dann gilt:

- Ist $C \in \mathrm{Reach}_x(t+1)$, so gibt es eine Berechnung mit Antwort 1. Weiterhin gibt keine Berechnung die Antwort 0, da keine Berechnung alle in $t+1$ Schritten erreichbaren Konfigurationen findet ohne spätestens im $(t+1)$-ten Schritt zu C zu kommen.
- Ist $C \notin \mathrm{Reach}_x(t+1)$, so gibt es eine Berechnung von M_0 mit Antwort 0, nämlich diejenige, die für alle in t Schritten erreichbaren Konfigurationen D die entsprechenden Berechnungswege findet und nachprüft, dass $C \notin \mathrm{Next}(C)$. Keine Berechnung liefert allerdings die Antwort 1.

2. Es gibt eine Funktion $t(n) = 2^{O(S(n))}$, so dass M entweder in $t(n)$ Schritten hält, oder aber gar nicht hält.

Lemma 2.12. *Es gibt ein irrtumfreies nichtdeterministisches $O(S(n))$-platzbeschränktes Berechnungsverfahren M_1 für die Funktion $x \mapsto R_x(t(|x|))$.*

Beweis. Wir beschreiben M_1 wie folgt:

Eingabe: x
$r := 1$
for $t = 0$ **to** $t(|x|)$ **do**
$m := 0$
for all $C \in \mathrm{Conf}[S(n)]$ **do**
$z := M_0(x, r, t, C)$ % Aufruf der nichtdeterministischen Prozedur M_0
if $z = 1$ **then** $m := m + 1$
if $z =$? **then** **output** ? **end**
od
$r := m$
od
output r **end**

Beachte, dass M_1 eine Prozedur ist, welche die nichtdeterministische Prozedur M_0 (im allgemeinen mehrfach) aufruft und deshalb selbst nichtdeterministisch ist. Immer wenn $M_0(x, r, t, C)$ von M_1 aufgerufen wird, gilt für die aktuellen Werte von r und t, dass $r = R_x(t)$, denn

- $t = 0 : \ r = 1 = R_x(0)$
- $t > 0 : \ r = |\{C :$ es gibt eine Berechnung von M_0 auf Eingabe $(x, R_x(t-1), t-1, C)$ mit Ausgabe 1 $\}| = R_x(t)$.

Insbesondere ist der Wert von r am Ende einer erfolgreichen Berechnung von M_1 gleich $R_x(t(|x|))$. Da es für alle r, t mit $r = R_x(t)$ eine Berechnung von M_0 auf (x, r, t, C) gibt, welche nicht die Ausgabe ? produziert, gibt es auch eine Berechnung von M_1, welche die Zahl $R_x(t(|x|))$ berechnet. Damit ist das Lemma bewiesen. □

3. Schliesslich geben wir ein irrtumfreies nichtdeterministisches Entscheidungsverfahren für $L = L(M)$ an.

$\tilde{M}$
Eingabe: x
$r := M_1(x)$ % Aufruf von M_1
if $r = ?$ **then output** ? **end**
else for all ($C_a \in \mathrm{Conf}[S(n)], C_a$ akzeptierend) **do**
$z := M_0(x, r, t(|x|), C_a)$
if $z = 1$ **then output** "$x \in L$" **end**
if $z = ?$ **then output** ? **end**
od
output "$x \notin L$" **end**

- Sei $x \in L$: Dann gibt es eine Berechnung von M_1 mit Resultat $r = R_x(t(|x|))$. Es gibt eine akzeptierende Konfiguration $C_a \in \mathrm{Reach}_x(t(|x|))$ und also eine Berechnung von M_0 auf $(x, r, t(|x|), C_a)$ mit Ausgabe 1. Also gibt es eine Berechnung von $\tilde{M}$ mit Antwort "$x \in L$". Umgekehrt ist klar, dass die Antwort "$x \in L$" nur gegeben wird, wenn eine akzeptierende Konfiguration C_a existiert mit $C_0(x) \vdash_M^x C_a$, wenn also tatsächlich $x \in L$. Also haben wir gezeigt: $x \in L$ genau dann, eine Berechnung von $\tilde{M}$ mit Antwort "$x \in L$" existiert.
- Sei $x \notin L$: Wieder gibt es eine Berechnung von M_1 mit Resultat $r = R_x(t(|x|))$. Da keine akzeptierende Konfiguration C_a von $C_0(x)$ aus erreichbar ist, gibt es für jedes C_a eine Berechnung von M_0 auf $(x, r, t(|x|), C_a)$ mit Antwort 0. Also gibt es eine Berechnung von $\tilde{M}$ mit Antwort "$x \notin L$". Umgekehrt kann diese Antwort nur gegeben werden, wenn M_0 für jedes C_a die Antwort 0 gegeben hat, wenn also tatsächlich kein C_a erreichbar ist von $C_0(x)$, d.h. wenn $x \notin L$.

Damit ist gezeigt, dass $\tilde{M}$ ein irrtumfreies nichtdeterministisches Entscheidungsverfahren für $L = L(M)$ ist, und damit auch für $\overline{L}$. Offensichtlich ist $\tilde{M}$ $O(S(n))$-platzbeschränkt. Mit dem Platzkompressionstheorem folgt, dass $\overline{L} \in \mathrm{NSPACE}(S)$. □

Insbesondere ist also Co-NLOGSPACE = NLOGSPACE.

Kapitel 3

NP-vollständige Probleme

3.1 Die Klassen P und NP

Definition 3.1. P $(=$ PTIME$) = \bigcup_{k\in\mathbb{N}}$ DTIME(n^k)

(i) P wird als die Klasse der effizient (oder praktisch) lösbaren Probleme angesehen, im Gegensatz zu den Problemen, die zwar prinzipiell lösbar sind, jedoch exponentielle Komplexität besitzen.

(ii) P ist sehr robust. Da (fast) alle vernünftigen deterministischen Maschinenmodelle sich gegenseitig mit polynomialem Zeitbedarf simulieren können, definieren sie alle die gleiche Klasse der in polynomialer Zeit entscheidbaren Probleme.

(iii) P hat folgende Abschlusseigenschaften: $A, B \in \mathrm{P} \Rightarrow A \cup B\,,\ A \cap B\,,\ \bar{A} \in \mathrm{P}$

P ist ebenso abgeschlossen unter polynomialer Reduktion:

Definition 3.2. Sei $A \subseteq \Sigma^*, B \subseteq \Gamma^*$. Eine *polynomiale Reduktion von A nach B* ist ist eine in polynomialer Zeit berechenbare Funktion $f : \Sigma^* \to \Gamma^*$ so dass für alle $x \in \Sigma^*$ gilt: $x \in A \iff f(x) \in B$. Falls eine solche Reduktion existiert sagen wir, A ist polynomial auf B reduzierbar (kurz $A \leq_p B$).

Offensichtlich gilt:

(i) $A \leq_p B\,,\ B \in \mathrm{P} \Rightarrow A \in \mathrm{P}$.

(ii) $A \leq_p B\,,\ B \leq_p C \Rightarrow A \leq_p C$.

Viele wichtige kombinatorische Probleme scheinen jedoch nicht in polynomialer Zeit lösbar zu sein.

Beispiele. (1) **Das Travelling Salesman Problem (als Entscheidungsproblem).** Wir erinnern daran, dass beim Problem TSP(D) für Eingaben (D, k), bestehend aus einer Distanzmatrix D von n Städten, und eine Zahl $k \in \mathbb{N}$ zu entscheiden ist, ob eine Tour π existiert, welche höchstens die Länge k hat.

Es ist nicht bekannt, ob TSP(D) in P liegt. Der offensichtlichste Algorithmus, nämlich alle möglichen Touren durchzutesten, ist nicht polynomial beschränkt, denn es gibt genau $(n-1)!$ mögliche Touren und $(n-1)! = 2^{\Omega(n \log n)}$. Hingegen ist das Problem

$$\textsc{Tour} = \{(D, k, \pi) : D, k \text{ wie bei TSP }, \pi \text{ ist eine Tour der Länge höchstens } k\}$$

offensichtlich in P. Wenn $(D, k, \pi) \in \textsc{Tour}$, dann ist π polynomialer „Zeuge" dafür, dass $(D, k) \in$ TSP. Da der entsprechende Suchraum exponentiell gross ist, ist es (vielleicht) schwierig, eine geeignete Permutation π zu *finden*, aber wenn sie gefunden ist, dann ist es einfach zu *verifizieren*, dass π tatsächlich die geforderte Eigenschaft hat.

(2) **CLIQUE** . Sei $G = (V, E)$ ein (ungerichteter) Graph mit Knotenmenge V und Kantenmenge E. Eine Clique in G ist eine Menge $C \subseteq V$ derart, dass alle verschiedenen $u, v \in C$ durch eine Kante verbunden sind.

$$\textsc{Clique} = \{(G, k) : G \text{ Graph }, k \in \mathbb{N}, \text{ es gibt eine } k\text{-punktige Clique } C \text{ in } G\}.$$

Ob Clique in P liegt, ist noch unbekannt. Es gibt $\binom{n}{k}$ k-punktige Teilmengen C in einer n-punktigen Menge V. Für $k = n/2$ wären das also mehr als $2^{\frac{n}{2}}$ mögliche Kandidaten. Jedoch liegt $\{(G, C) \mid C \text{ ist Clique in } G\}$ offensichtlich in P.

Dies legt nahe, die Klasse derjenigen Probleme zu betrachten, die (wie in den beiden Beispielen) nach einem Objekt fragen, dessen Grösse selbst polynomial beschränkt ist (Tour, Clique), so dass für jede Eingabe und jedes „vorgeschlagene" Objekt in polynomialer Zeit *verifiziert* werden kann, ob es die geforderten Bedingungen erfüllt. Diese Klasse heisst NP.

Definition 3.3. Für $L \subseteq \Sigma^*$ ist L genau dann in NP, wenn es ein Polynom $p(n)$ und ein $L_0 \in P$ gibt, so dass

$$L = \{x \mid \exists y \, (|y| \leq p(|x|) \ \wedge \ x\#y \in L_0)\}$$

Das y aus der obigen Definition heisst "polynomialer Zeuge" (oder "polynomialer Beweis") dafür, dass $x \in L$. Anstatt $(\exists y \, (|y| \leq p(|x|) \cdots))$ benutzen wir auch die Notation $\exists^p y$ benutzt.

- $\text{TSP}(D) \in \text{NP}$
- $\textsc{Clique} \in \text{NP}$

Eine andere Charakterisierung von NP ist gegeben durch

Satz 3.4. $\text{NP} = \bigcup_{k \in \mathbb{N}} \textsc{Ntime}(n^k)$

Beweis. $\subseteq$: Sei $L = \{x \mid \exists^p y \, (x\#y) \in L_0\} \in \text{NP}$, wobei $p(n)$ ein Polynom , $L_0 \in \text{P}$, und M_0 ein deterministischer, polynomial zeitbeschränkter Algorithmus ist, welcher L_0 entscheidet. Sei M der folgende nichtdeterministische Algorithmus:

Eingabe: x
errate y mit $|y| < p(n)$
$M_0(x\#y)$

Offensichtlich entscheidet M die Menge L. Wenn M_0 q-zeitbeschränkt ist, dann ist M $(p(n) + q(n + 1 + p(n)))$-zeitbeschränkt. Wenn $q(n)$ ein Polynom ist, dann ist aber auch $p(n) + q(n + 1 +_p (n))$ ein Polynom.

$\supseteq$: Sei $L = L(M)$, p ein Polynom und M eine p-zeitbeschränkte, nichtdeterministische Prozedur. Eine Berechnung von M auf eine Eingabe der Länge n ist eine Folge von höchstens $p(n)$ Konfigurationen der Länge $\leq p(n)$. Also kann eine Berechnung von M durch eine $p(n) \times p(n)$-Tabelle

mit Einträgen aus $Q \times \Sigma \cup \Sigma$ beschrieben werden, also auch durch ein Wort der Länge $p^2(n)$. Setze

$$L_0 = \{x\#y \mid y \text{ akzeptierende Berechnung von } M \text{ auf } x\}$$

Es ist leicht einzusehen, dass $L_0 \in \mathrm{P}$ und $x \in L$ genau dann, wenn $\exists y\ |y| \leq p^2(n)$, $x\#y \in L_0$. Also ist $L \in \mathrm{NP}$. □

Satz 3.5. *(i)* $\mathrm{P} \subseteq \mathrm{NP}$.
(ii) $A \leq_p B$, $B \in \mathrm{NP} \Rightarrow A \in \mathrm{NP}$.

Als das wichtigste, noch ungelöste Problem der Komplexitätstheorie gilt die **Cook'sche Hypothese:** $\mathrm{P} \neq \mathrm{NP}$.

3.2 NP-Vollständigkeit und das Erfüllbarkeitsproblem der Aussagenlogik

Wenn auch bisher nicht bewiesen werden konnte, dass es Probleme in $\mathrm{NP}-\mathrm{P}$ gibt, so konnten doch "schwierigste" Probleme in NP identifiziert werden, in dem Sinne, dass diese Probleme nur dann in P sein können, wenn $\mathrm{P} = \mathrm{NP}$.

Definition 3.6. Ein Problem B heisst NP-*hart* (oder NP-*schwer*), wenn für alle $A \in \mathrm{NP}$ gilt: $A \leq_p B$. Ein Problem B heisst NP-*vollständig*, wenn $B \in \mathrm{NP}$ und B NP-hart ist.

Satz 3.7. *Sei B NP-vollständig. Dann ist* $\mathrm{P} = \mathrm{NP}$ *genau dann, wenn* $B \in \mathrm{P}$.

Beweis. Sei $B \in P$ und A ein beliebiges Problem aus NP. Da $A \leq_p B$, folgt $A \in \mathrm{P}$. Also ist $\mathrm{P} = \mathrm{NP}$. □

Zur Erinnerung werden im folgenden noch einmal die Grundbegriffe der Aussagenlogik wiederholt.

Wir fixieren eine abzählbare Menge $\tau = \{X_i : i \in \mathbb{N}\}$ von Aussagenvariablen. Die Menge AL der *Formeln der Aussagenlogik* ist induktiv definiert wie folgt:

- $\tau \subseteq \mathrm{AL}$ (Jede Aussagenvariable ist eine Formel)
- $\psi, \varphi \in \mathrm{AL} \implies (\psi \wedge \varphi), (\psi \vee \varphi), \neg\psi \in \mathrm{AL}$.

Für jede Formel $\psi \in \mathrm{AL}$ sei $\tau(\psi)$ die Menge der Aussagenvariablen, die in ψ vorkommen. Eine *Belegung* einer Menge $\sigma \subseteq \tau$ von Variablen ist eine Abbildung $\mathfrak{I} : \sigma \to \{0, 1\}$. Diese definiert einen Wahrheitswert $\mathfrak{I}(\psi)$ für jede Formel ψ mit $\tau(\psi) \subseteq \sigma$ wie folgt:

- für $\psi = X \in \sigma$ ist $\mathfrak{I}(\psi) = \mathfrak{I}(X)$ bereits definiert.
- $\mathfrak{I}(\psi \vee \varphi) = \max(\mathfrak{I}(\psi), \mathfrak{I}(\varphi))$
- $\mathfrak{I}(\psi \wedge \varphi) = \min(\mathfrak{I}(\psi), \mathfrak{I}(\varphi))$
- $\mathfrak{I}(\neg\psi) = 1 - \mathfrak{I}(\psi)$

Eine Formel ψ heisst *erfüllbar* genau dann, wenn es eine Belegung $\mathfrak{I} : \tau(\psi) \to \{0,1\}$ gibt, mit $\mathfrak{I}(\psi) = 1$. Eine *Tautologie* ist eine Formel ψ so, dass $\mathfrak{I}(\psi) = 1$ für alle Belegungen $\mathfrak{I} : \tau(\psi) \to \{0,1\}$ gilt.

Offensichtlich ist ψ erfüllbar genau dann, wenn $\neg\psi$ keine Tautologie ist. Zwei Formeln ψ und φ heissen äquivalent ($\psi \equiv \varphi$), wenn für alle $\mathfrak{I} : \tau(\psi) \cup \tau(\varphi) \to \{0,1\}$ gilt, dass $\mathfrak{I}(\psi) = \mathfrak{I}(\varphi)$. Eine Formel φ folgt aus ψ (kurz $\psi \models \varphi$), wenn für jede Belegung $\mathfrak{I} : \tau(\psi) \cup \tau(\varphi) \to \{0,1\}$ mit $\mathfrak{I}(\psi) = 1$ auch $\mathfrak{I}(\varphi) = 1$ gilt.

Bemerkungen. Wie üblich verwenden wir auch die Ausdrücke $(\psi \to \varphi)$ bzw. $(\psi \leftrightarrow \varphi)$ als andere Schreibweisen für $(\neg\psi \vee \varphi)$ bzw. $((\psi \wedge \varphi) \vee (\neg\psi \wedge \neg\varphi))$ und lassen für das Verständnis überflüssige Klammern in der Regel weg. Da $\wedge$ und $\vee$ bzgl. ihrer Bedeutung assoziativ sind, können wir für Konjunktionen bzw. Disjunktionen über Mengen $\{\psi_i : i \in I\}$ die Schreibweisen $\bigwedge_{i\in I} \psi_i$ bzw. $\bigvee_{i\in I} \psi_i$ verwenden. Um Formeln der Aussagenlogik als Worte über einem festen Alphabet kodieren zu können, fixieren wir die Variablenmenge $\tau = \{X_i : i \in \mathbb{N}\}$ und kodieren X_i durch $X(\text{bin } i)$, also das Symbol X gefolgt von der Binärdarstellung des Indexes i. Jede Formel der Aussagenlogik ist dann eine Wort über dem Alphabet $\Sigma = \{X, 0, 1, \wedge, \vee, \neg, (,)\}$.

Definition 3.8. SAT:= $\{\psi \in \text{AL} : \psi \text{ erfüllbar }\}$

Satz 3.9 (Cook, Levin). SAT *ist* NP*-vollständig.*

Beweis. Es ist klar, dass SAT in NP ist, da $\{\psi\#\mathfrak{I} : \mathfrak{I} : \tau(\psi) \to \{0,1\}, \mathfrak{I}(\psi) = 1\} \in P$.

Sei A nun ein beliebiges Problem in NP. Wir zeigen, dass $A \leq_p$SAT. Sei dazu $M = (Q, \Sigma, q_0, F, \delta)$ mit $F = F^+ \cup F^-$ ein nichtdeterministischer 1-Band Akzeptor, welcher A in polynomialer Zeit $p(n)$ entscheidet. Wir nehmen an, dass jede Berechnung von M in einer akzeptierenden oder verwerfenden Endkonfiguration endet, d.h. C ist Endkonfiguration gdw. Next$(C) = \varnothing$. Sei $w = w_0 \cdots w_{n-1}$ eine Eingabe für M. Wir konstruieren eine Formel $\psi_w \in$ AL, welche erfüllbar ist genau dann, wenn M den Input w akzeptiert.

Dabei enthält ψ_w folgende Aussagenvariabeln:

- $X_{q,t}$ für $q \in Q, 0 \leq t \leq p(n)$.
- $Y_{a,i,t}$ für $a \in \Sigma, 0 \leq i, t \leq p(n)$
- $Z_{i,t}$ für $0 \leq i, t \leq p(n)$.

mit folgenden Interpretationen:

- $X_{q,t}$: Zur Zeit t ist M im Zustand q.
- $Y_{a,i,t}$: Zur Zeit t steht auf Feld i das Symbol a
- $Z_{i,t}$: Zur Zeit t steht der Kopf von M auf Feld i

$$\psi_w := \text{START} \wedge \text{COMPUTE} \wedge \text{END}$$

$$\begin{aligned}
\text{START} &:= X_{q_0,0} \wedge \bigwedge_{i=0}^{n-1} Y_{w_i,i,0} \wedge \bigwedge_{i=n}^{p(n)} Y_{\Box,i,0} \wedge Z_{0,0} \\
\text{COMPUTE} &:= \varphi_1 \wedge \varphi_2 \\
\varphi_1 &:= \bigwedge_{t<p(n),a\in\Sigma,i\neq j} (Z_{i,t} \wedge Y_{a,j,t} \to Y_{a,j,t+1}) \\
\varphi_2 &:= \bigwedge_{t<p(n),i,a,q} \Big((X_{q,t} \wedge Y_{a,i,t} \wedge Z_{i,t}) \to \bigvee_{\substack{(q',b,m)\in\delta(q,a) \\ 0\leq i+m\leq p(n)}} (X_{q',t+1} \wedge Y_{b,i,t+1} \wedge Z_{i+m,t+1}) \Big) \\
\text{END} &:= \bigwedge_{t\leq p(n),q\in F^-} \neg X_{q,t}
\end{aligned}$$

Dabei 'kodiert' START die Inputkonfiguration zur Zeit 0. φ_1 stellt sicher, dass dort, wo der Kopf nicht ist, auch keine Veränderungen am Speicherinhalt vorgenommen werden. φ_2 repräsentiert die Übergangsfunktion.

Offensichtlich ist die Abbildung $w \mapsto \psi_w$ in polynomialer Zeit berechenbar.

(1) Sei $w \in L(M)$. Jede Berechnung von M induziert eine Interpretation der Aussagenvariablen $X_{q,t}, Y_{a,i,t}, Z_{i,t}$. Eine akzeptierende Berechnung von M auf w induziert eine Interpretation, welche ψ_w wahr macht. Also ist $\psi_w \in \text{SAT}$.

(2) Sei $C = (q, y, p)$ eine Konfiguration von M, $t \leq p(n)$. Setze

$$\text{CONF}[C,t] := X_{q,t} \wedge \bigwedge_{i=0}^{p(n)} Y_{y_i,i,t} \wedge Z_{p,t}.$$

Man beachte, dass START $= \text{CONF}[C_0(w), 0]$ ist. Also gilt

$$\psi_w \models \text{CONF}[C_0(w), 0].$$

Für jede Nicht-Endkonfiguration C von M und alle $t < p(n)$ folgt (wegen der Teilformel COMPUTE von ψ_w) :

$$\psi_w \wedge \text{CONF}[C,t] \models \bigvee_{C'\in \text{Next}(C)} \text{CONF}[C', t+1].$$

(3) Sei $\mathfrak{I}(\psi_w) = 1$. Aus (1) und (2) folgt, dass mindestens eine Berechnung $C_0(w) = C_0, C_1, \ldots, C_r = C_{\text{end}}$ mit $r \leq p(n)$ von M auf w existiert, mit $\mathfrak{I}(\text{CONF}[C_t, t]) = 1$ für alle $t = 0, \ldots, v$. Da wegen der Teilformel END von ψ_w gilt, dass $\psi_w \models \neg\text{CONF}[C,t]$ für alle verwerfenden Endkonfigurationen C von M und alle t, folgt, dass C_{end} akzeptierend ist. Also akzeptiert M den Input w.

Damit ist gezeigt, dass $\psi_w \in \text{SAT}$ genau dann, wenn $w \in A$. □

Bemerkung. Die Reduktion $w \mapsto \psi_w$ ist besonders einfach. Insbesondere ist sie mit *logarithmischem Platz* berechenbar.

Definition 3.10. Sei $A \subseteq \Gamma^*, B \subseteq \Sigma^*$. Dann ist $A \leq_{\log} B$, wenn eine Funktion $f : \Gamma^* \to \Sigma^*$ existiert, so dass

- $x \in A \iff f(x) \in B$
- f ist berechenbar durch eine $O(\log n)$-platzbeschränkte Turingmaschine.

Sei $\mathcal{C}$ eine Komplexitätsklasse. B ist $\mathcal{C}$-vollständig via logspace-Reduktion, wenn

- $B \in \mathcal{C}$,
- $A \leq_{\log} B$ für alle $A \in \mathcal{C}$.

Übung 3.1. Zeigen Sie, dass $\leq_{\log}$ abgeschlossen ist unter Komposition: $A \leq_{\log} B, B \leq_{\log} C \implies A \leq_{\log} C$.

Eine Folgerung aus dem Beweis von Satz 3.9 ist, dass SAT NP-vollständig via logspace-Reduktion ist.

3.3 Varianten von SAT

Die NP-Vollständigkeit von SAT schliesst nicht aus, dass das Erfüllbarkeitsproblem für gewisse interessante Formelklassen $S \subseteq$ AL doch polynomiell lösbar ist. Wir werden zeigen, dass für gewisse Klassen $S \subseteq$ AL, $S \cap \text{SAT} \in P$ ist, für andere jedoch $S \cap \text{SAT}$ NP-vollständig.

Zur Erinnerung. Ein *Literal* ist eine Aussagenvariable oder deren Negat. Eine Formel $\psi \in$ AL ist in *disjunktiver Normalform (DNF)*, wenn sie die Form $\psi := \bigvee_{i=1}^{n} \bigwedge_{j=1}^{m_i} Y_{ij}$ hat, wobei die Y_{ij} Literale sind. ψ ist in *konjunktiver Normalform (KNF)*, wenn sie die Gestalt $\psi := \bigwedge_{i=1}^{n} \bigvee_{j=1}^{m_i} Y_{ij}$ hat. Eine Disjunktion $\bigvee_j Y_{ij}$ wird auch *Klausel* genannt. Jede Formel $\psi \in$ AL ist äquivalent zu einer Formel ψ_D ind DNF und einer Formel ψ_K in KNF.

$$\psi \equiv \psi_D := \bigvee_{\substack{\mathfrak{I}:\tau(\psi)\to\{0,1\} \\ \mathfrak{I}(\psi)=1}} \bigwedge_{X \in \tau(\psi)} X^{\mathfrak{I}}$$

mit

$$X^{\mathfrak{I}} = \begin{cases} X & \text{wenn } \mathfrak{I}(X) = 1 \\ \neg X & \text{wenn } \mathfrak{I}(X) = 0 \end{cases}$$

Analog für KNF.

Die Übersetzungen $\psi \mapsto \psi_D, \psi \mapsto \psi_K$ sind berechenbar, aber im Allgemeinen nicht in polynomieller Zeit. Die Formeln ψ_D und ψ_K können exponentiell länger sein als ψ, da es $2^{|\tau(\psi)|}$ mögliche Abbildungen $\mathfrak{I} : \tau(\psi) \to \{0,1\}$ gibnt. Mit SAT-DNF:= $\{\psi$ in DNF $: \psi$ erfüllbar$\}$ wird die Menge aller erfüllbaren DNF-Formeln und mit SAT-KNF := $\{\psi$ in KNF $: \psi$ erfüllbar $\}$ die Menge der erfüllbaren KNF-Formeln bezeichnet.

Satz 3.11. SAT-DNF $\in$ LOGSPACE $\subseteq P$

Beweis. $\psi = \bigvee_i \bigwedge_{j=1}^{m_i} Y_{ij}$ ist genau dann erfüllbar, wenn ein i existiert, so dass in $\{Y_{ij} : j = 1, \cdots, m_i\}$ keine Variable sowohl positiv als auch negativ auftritt. □

Satz 3.12. SAT-KNF *ist* NP*-vollständig via logspace-Reduktion.*

Beweis. Der Beweis folgt aus dem Beweis des Satzes 3.9. Betrachtet man die Formel

$$\psi_w = \text{START} \wedge \text{COMPUTE} \wedge \text{END}$$

aus dem Beweis, so sind START und END schon in KNF. Genauso ist die Teilformel φ_1 von COMPUTE schon in KNF. Es bleibt φ_2. φ_2 ist eine Konjunktion von Formeln der Form

$$\alpha : X \wedge Y \wedge Z \to \bigvee_{j=1}^{r} X_j \wedge Y_j \wedge Z_j.$$

Dabei ist $r \leq \max_{(q,a)} |\delta(q,a)|$ fest, also unabhängig von n, w. Es gilt aber

$$\alpha \equiv (X \wedge Y \wedge Z \to \bigvee_{j=1}^{r} U_j) \wedge \bigwedge_{j=1}^{r} (U_j \to X_j) \wedge (U_j \to Y_j) \wedge (U_j \to Z_j))$$

Also gilt $A \leq_{\log}$ SAT-KNF für alle $A \in$ NP. □

Horn-Formeln

Eine (aussagenlogische) *Horn-Formel* ist eine Formel $\psi = \bigwedge_i \bigvee_j Y_{i_j}$ in KNF, wobei jede Disjunktion $\bigvee_j Y_j$ höchstens ein positives Literal enthält. Horn-Formeln können mit Hilfe folgender Äquivalenzen auch als Konjunktion von Implikationen geschrieben werden:

$$\neg X_1 \vee \cdots \vee \neg X_k \vee X \equiv (X_1 \wedge \cdots \wedge X_k) \to X$$

$$\neg X_1 \vee \cdots \vee \neg X_k \equiv (X_1 \wedge \cdots \wedge X_k) \to 0$$

Sei HORN-SAT $= \{\psi \ : \ \psi$ aussagenlogische Horn-Formel, ψ erfüllbar $\}$. Aus der Vorlesung „Mathematische Logik“ ist bekannt:

Satz 3.13. HORN-SAT $\in P$.

Dies folgt z.B. aus der Einheitsresolution oder dem Markierungsalgorithmus.

Satz 3.14. HORN-SAT *ist P-vollständig via logspace-Reduktion.*

Beweis. Sei $A \in P$ und M eine deterministische 1-Band Turing-Maschine, welche A in Zeit $p(n)$ entscheidet. Betrachte die Reduktion $w \mapsto \psi_w$ aus dem Beweis von Satz 3.9. Die Formeln START, φ_1 und END sind schon Horn-Formeln. Da M deterministisch angenommen wurde, d.h. $|\delta(q,a)| = 1$ nimmt φ_2 die Form $(X \wedge Y \wedge Z) \to (X' \wedge Y' \wedge Z')$ an. Dies ist äquivalent zur Horn-Formel $(X \wedge Y \wedge Z) \to X' \wedge (X \wedge Y \wedge Z) \to Y' \wedge (X \wedge Y \wedge Z) \to Z'$. Also haben wir eine logspace berechenbare Funktion $w \mapsto \widehat{\psi}_w$, so dass

- $\widehat{\psi}_w$ ist eine Horn-Formel
- M akzeptiert w gdw. $\widehat{\psi}_w$ ist erfüllbar.

Also $A \leq_{\log}$ HORN-SAT. □

Definition 3.15. Eine Formel ist in r-KNF, wenn sie in KNF ist und jede Klausel höchstens r Literale enthält: $\psi = \bigwedge_{i=1}^{n} \bigvee_{j=1}^{m_i} Y_{i_j}$ mit $m_i \leq r$ für alle i.
Weiterhin ist r-SAT $:= \{\psi$ in r-KNF $\ : \ \psi$ erfüllbar $\}$.

Satz 3.16. 3-SAT *ist* NP*-vollständig.*

Beweis. 3-SAT $\in$ NP, da SAT $\in$ NP. Wir zeigen, dass SAT-KNF $\leq_p$ 3-SAT. Sei $\psi := \bigwedge_i C_i$ (mit $C_i = \bigvee_{i=1}^{m_i} Y_{i_j}$) in KNF. Der folgende Algorithmus transformiert ψ in 3-KNF:

Input: ψ
$\varphi := \psi$
while (φ nicht in 3-KNF) **do**
ersetze erste Klausel $C_i = \bigvee_{j=1}^{m_i} Y_{i_j}$ mit $m_i > 3$ durch die beiden Klauseln
$C_i' := Y_{i_1} \vee Y_{i_2} \vee Z$
$C_i'' := \neg Z \vee \bigvee_{j=3}^{m_i} Y_{i_j}$ wobei $Z \notin \tau(\varphi)$ neue Aussagenvariable
od
output φ

Dieser Algorithmus transformiert die KNF-Formel ψ in polynomieller Zeit in die 3-KNF-Formel φ. Es bleibt zu zeigen, dass ψ genau dann erfüllbar ist, wenn φ erfüllbar ist. Dazu reicht es zu zeigen, dass die Ersetzung von C_i durch $C_i' \wedge C_i''$ Erfüllbarkeit erhält:

- Sei die gegebene Formel durch die Interpretation $\mathfrak{I}$ erfüllt, d.h. $\mathfrak{I}(C_i) = 1$:
  (a) $\mathfrak{I}(Y_{i_1} \vee Y_{i_2}) = 1$. Erweitere $\mathfrak{I}$ durch $\mathfrak{I}(Z) = 0$. Daraus folgt $\mathfrak{I}(C_i') = \mathfrak{I}(C_i'') = 1$.
  (b) $\mathfrak{I}(Y_{i_1}) = \mathfrak{I}(Y_{i_2}) = 0$. Also $\mathfrak{I}(Y_{i_j}) = 1$ für mindestens ein $j > 2$. Erweitere $\mathfrak{I}$ durch $\mathfrak{I}(Z) = 1$. Daraus folgt $\mathfrak{I}(C_i') = \mathfrak{I}(C_i'') = 1$.

Sei die neue Formel durch die Interpretation $\mathfrak{I}$ erfüllt.

(a) $\mathfrak{I}(Z) = 1 \implies \mathfrak{I}(Y_{i,j}) = 1$ für ein $j > 2$.

(b) $\mathfrak{I}(Z) = 0 \implies \mathfrak{I}(Y_{i,j}) = 1$ für $j = 1$ oder $j = 2$.

In beiden Fällen ist $\mathfrak{I}(C) = 1$. □

r-SAT ist also NP-vollständig für alle $r \geq 3$. Es bleibt die Frage nach der Komplexität von 2-SAT. Mittels Resolution folgt leicht, dass 2-SAT in P ist.

- Die Resolvente zweier Klauseln mit ≤ 2 Literalen hat höchstens 2 Literale.
- Mit n Variablen können höchstens $O(n^2)$ Klauseln mit ≤ 2 Literalen gebildet werden.

Daraus folgt, dass $\mathrm{Res}^*(\psi)$ für eine Formel ψ in 2-KNF in polynomieller Zeit ausgerechnet werden kann.

Ein stärkeres Resultat ergibt

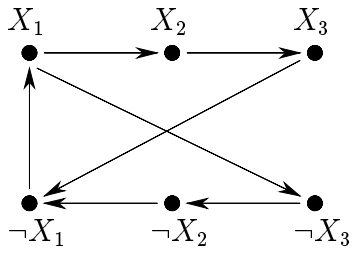
Satz 3.17. 2-SAT *ist in* NLOGSPACE.

Beweis. Wir zeigen folgende Behauptung: $\{\psi \ : \ \psi$ in 2-KNF, ψ unerfüllbar $\} \in$ NLOGSPACE.

Aus dem Satz von Immerman und Szelepcsényi folgt, dass dann auch 2-SAT $\in$ NLOGSPACE.

Dazu bilden wir eine Formel $\psi \in$ 2-KNF auf folgenden gerichteten Graphen $G = (V, E)$ ab:

- $V = \{X, \neg X \ : \ X \in \tau(\psi)\}$ sind die Literale von ψ.
- $E = \{(Y, Z) \ : \ Y, Z$ Literale, es gibt in ψ eine Klausel, welche zu $(Y \to Z)$ äquivalent ist$\}$.



Beispiel.

Abbildung 3.1: Beispielgraph zu $\psi := X_1 \wedge (\neg X_1 \vee X_2) \wedge (X_3 \vee \neg X_2) \wedge (\neg X_3 \vee \neg X_1)$

Lemma 3.18 (Krom-Kriterium). *Sei ψ in 2-KNF. ψ ist unerfüllbar gdw. ein $X \in \tau(\psi)$ existiert, so dass in G_ψ Pfade von X nach $\neg X$ und von $\neg X$ nach X existieren.*

Sei also $L = \{(G, a, b) \ : \ G$ gerichteter Graph, es gibt in G Pfade von a nach $b\ \}$. Das entspricht dem Labyrinthproblem. ψ ist also unerfüllbar gdw. es ein $X \in \tau(\psi)$ gibt, mit $(G_\psi, X, \neg X) \in L$ und $(G_\psi, \neg X, X) \in L$. Da $L \in$ NLOGSPACE, folgt die Behauptung.

Es bleibt noch das Krom-Kriterium zu beweisen. Im folgenden soll $Y \to_\psi^* Z$ bedeuten, dass ein in G_ψ ein Pfad $Y \to Z$ existiert.

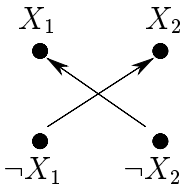
Sei $\mathfrak{I}$ eine Interpretation mit $\mathfrak{I}(\psi) = 1$. Dann gilt: $\mathfrak{I}(Y) = 1, Y \to_\psi^* Z \implies \mathfrak{I}(Z) = 1$. Also folgt: Wenn $X \to_\psi^* \neg X \to_\psi^* X$, dann ist ψ unerfüllbar.

Umgekehrt gilt für alle $X \in \tau(\psi)$ entweder nicht $X \to_\psi^* \neg X$ oder nicht $\neg X \to_\psi^* X$. Man kann nun effektiv eine Bewertung $\mathfrak{I}$ mit $\mathfrak{I}(\psi) = 1$ konstruieren:

$U := \tau(\psi) \cup \neg\tau(\psi)$
while $U \neq \varnothing$ **do**
wähle $Y \in U$, so dass nicht $Y \to_\psi^* \neg Y$.
setze $\mathfrak{I}(Y) = 1, U = U - \{Y, \neg Y\}$
für alle Z mit $Y \to_\psi^* Z$ setze $\mathfrak{I}(Z) = 1, U = U - \{Z, \neg Z\}$
od

Diese Prozedur ergibt keine Konflikte der Art, dass sowohl $Y \to_\psi^* Z$ als auch $Y \to_\psi^* \neg Z$. Dann dies ergäbe auch $\neg Z \to_\psi^* \neg Y$ und $Z \to_\psi^* \neg Y$, also $Y \to_\psi^* Z \to_\psi^* \neg Y$. Y wurde aber gerade so gewählt, dass dies nicht gilt.

Die Prozedur bestimmt also eine Belegung $\mathfrak{I}$, indem für jede Variable $X \in \tau(\psi)$ entweder $\mathfrak{I}(X) = 1$ oder $\mathfrak{I}(\neg X) = 1$ gesetzt wird. Dabei ist zu beachten, dass die Prozedur nicht eindeutig bestimmt ist, da Y jeweils verschieden gewählt werden kann.



Beispiel.

Abbildung 3.2: G_ψ zu $\psi = (X_1 \vee X_2)$

Sei $\mathfrak{I}$ eine so konstruierte Bewertung. Zu zeigen: $\mathfrak{I}$ erfüllt jede Klausel $(Z \vee Z')$ von ψ und somit ψ selbst.

Andernfalls ist $\mathfrak{I}(Z) = \mathfrak{I}(Z') = 0$. Dann ist $\mathfrak{I}(\neg Z) = 1$. Also wird in der Prozedur einmal ein Literal Y ausgewählt, mit $Y \to_\psi^* \neg Z$ und nicht $Y \to_\psi^* \neg Y$. Da $\neg Z \to_\psi^* Z'$, folgt dann aber auch $Y \to_\psi^* Z'$ und also $\mathfrak{I}(Z') = 1$, was den gewünschten Widerspruch ergibt. □

Bemerkung.

- Formeln in 2-KNF heissen auch *Krom-Formeln.*
- Es gilt sogar: $2 - SAT$ ist NLOGSPACE-vollständig via logspace-Reduktion.

3.4 NP-vollständige Probleme aus der Graphentheorie

Wir werden zeigen, dass eine Reihe graphentheoretischer Probleme NP-vollständig sind:

a) 3-Färbbarkeit

Ein Graph $G = (V, E)$ ist k-färbbar, wenn eine Funktion $f : V \to \{1, \cdots, k\}$ existiert, so dass $f(x) \neq f(y)$ für alle $(x, y) \in E$. Die Zahl $\chi(G) := \min\{k \ : \ G \text{ ist } k\text{-färbbar } \}$ wird die *chromatische Zahl* von G genannt. Folgende Probleme treten im Zusammenhang mit der Färbbarkeit eines Graphen auf:

- Gegeben ist ein Graph $G = (V, E)$. Gesucht ist eine Färbung $f : V \to \{1, \cdots, \chi(G)\}$ mit minimaler Anzahl von Farben.
- Gegeben ist ein Graph G. Gesucht ist die chromatische Zahl $\chi(G)$.
- Gegeben ist ein Graph G und eine Zahl $k \in \mathbb{N}$. Gefragt ist, ob $\chi(G) \leq k$, d.h. ob G k-färbbar ist.

Keines dieser Probleme ist effizient lösbar, es sei denn P = NP. Es gilt sogar:

Satz 3.19. *3-F* $:= \{G \ : \ G$ *3-färbbar* $\}$ *ist* NP*-vollständig.*

Beweis. Es ist klar, dass 3-F $\in$ NP, da $\{(G, f) \ : \ f : V \to \{1, 2, 3\}$ ist korrekte Färbung $\} \in$ P. Zum Beweis der Vollständigkeit reduzieren wir 3-SATauf 3-F. Sei $\psi := \bigwedge_{i=1}^{m}(Y_{i,1} \vee Y_{i,2} \vee Y_{i,3})$ in 3-KNF. Dabei nehmen wir o.E. an, dass jede Klausel von ψ genau 3 Literale enthält. Zu ψ konstruieren wir einen Graphen $G_\psi = (V, E)$ mit

$$V := \{0, 2\} \cup \underbrace{\tau(\psi) \cup \neg\tau(\psi)}_{\text{Literale von } \psi} \cup \underbrace{\{P_{i,j} \ : \ 1 \leq i \leq m, 1 \leq j \leq 5\}}_{\text{für jede Klausel von } \psi \text{ 5 Knoten}}$$

$$\begin{aligned} E := \{(0,2)\} \ &\cup \ \{(2, Y) \ : \ Y\text{Literal}\} \\ &\cup \ \{(X, \neg X) \ : \ X \in \tau(\psi)\} \\ &\cup \ \{(0, P_{i,4}), (0, P_{i,5}) \ : \ i \leq m\} \\ &\cup \ \{(P_{i,j}, P_{i,k}) \ : \ 1 \leq i \leq m, (j,k) \in \{(1,2), (2,5), (5,3), (3,4), (4,1)\} \\ &\cup \ \{(P_{i,j}, Y_{i,j}) \ : \ 1 \leq i \leq m, 1 \leq j \leq 3\} \end{aligned}$$

Behauptung. ψ erfüllbar gdw. G_ψ 3-färbbar.

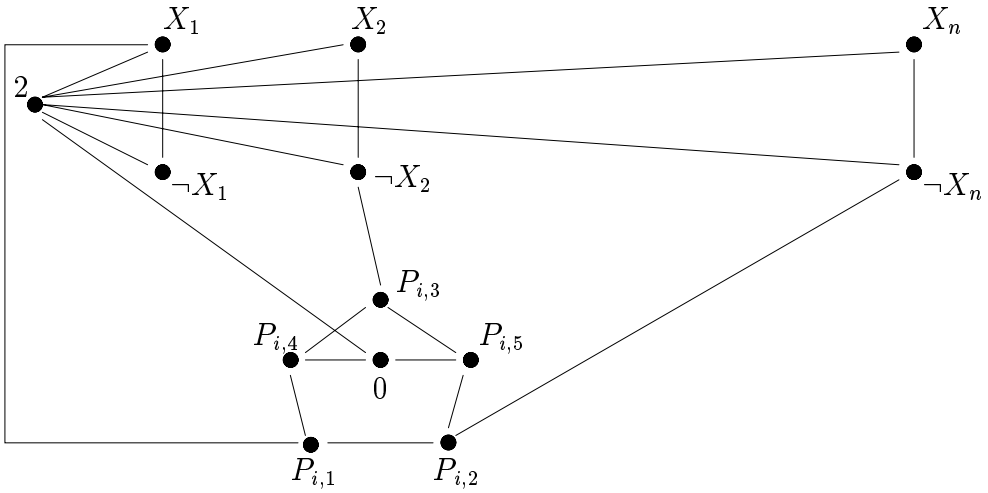


Abbildung 3.3: Beispielgraph G_ψ zu einer 3-KNF Formel ψ mit $\tau(\psi) = \{X_1, \ldots, X_n\}$. Dargestellt sind die P_i für eine Klausel $C_i = X_1 \vee \neg X_n \vee \neg X_2$

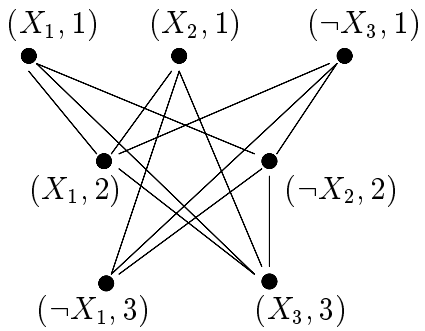
- sei $\mathfrak{I}(\psi) = 1$. Definiere eine Färbung $f : V \to \{0, 1, 2\}$ wie folgt:
  - $f(0) = 0, f(2) = 2$
  - $f(Y) = \mathfrak{I}(Y)$ für $Y \in \tau(\psi) \cup \neg\tau(\psi)$
  - Die Färbung von $P_{i,1}, \ldots, P_{i,5}$ wird über eine Fallunterscheidung definiert. Dazu sei $\psi = \bigwedge_i (Y_{i,1} \vee Y_{i,2} \vee Y_{i,3})$
    (a) $\mathfrak{I}(Y_{i,1}) = 1 : f(P_{i,1}) = 0, f(P_{i,2}) = 2, f(P_{i,3}) = 2, f(P_{i,4}) = 1, f(P_{i,5}) = 1$
    (b) $\mathfrak{I}(Y_{i,1}) = 0, \mathfrak{I}(Y_{i,2}) = 1 : f(P_{i,1}) = 2, f(P_{i,2}) = 0, f(P_{i,3}) = 2, f(P_{i,4}) = 1, f(P_{i,5}) = 1$
    (c) $\mathfrak{I}(Y_{i,1}) = \mathfrak{I}(Y_{i,2}) = 0, \mathfrak{I}(Y_{i,3}) = 1 : f(P_{i,1}) = 1, f(P_{i,2}) = 2, f(P_{i,3}) = 0, f(P_{i,4}) = 2, f(P_{i,5}) = 1$

    Daraus folgt, das G_ψ 3-färbbar ist.
  - Sei G_ψ korrekt gefärbt durch $f : V \to \{0, 1, 2\}$. O.E. sei $f(0) = 0, f(2) = 2$. Daraus folgt $f(Y) \in \{0, 1\}$ für $Y \in \tau(\psi) \cup \neg\tau(\psi)$. Setze $\mathfrak{I}(X) := f(X)$. Zu zeigen bleibt $\mathfrak{I}(\psi) = 1$. Sonst wäre aber $\mathfrak{I}(Y_{i,1}) = \mathfrak{I}(Y_{i,2}) = \mathfrak{I}(Y_{i,3}) = 0$ für ein $i \leq m$. Dann wäre $f(P_{i,1}) = 1$ und $f(P_{i,2}) = 2$ oder umgekehrt. Da aber $f(P_{i,3}) \neq 0$, folgt, dass $P_{i,4}$ oder $P_{i,5}$ nicht korrekt gefärbt sein kann. Dies ergibt den gewünschten Widerspruch.

□

b) Das *Cliquenproblem*

Bei dem Problem MAX CLIQUE handelt es sich um ein Optimierungproblem. Gegeben ist ein Graph $G = (V, E)$. Gesucht ist eine Clique C in G von maximaler Grösse. $C \subseteq V$ ist eine Clique von G gdw. für alle $x, y \in C, x \neq y$ gilt: $(x, y) \in E$.
Das zugehörige Berechnungsproblem ist: Gegeben G, bestimme die Cliquenzahl $\omega(G) := \max\{k : G \text{ enthält eine } k\text{-punktige Clique}\}$.
Das Entscheidungsproblem ist die Menge CLIQUE $= \{(G, k) : G \text{ enthält } k\text{-punktige Clique}\} = \{(G, k) : k \leq \omega(G)\}$.



Beispiel.

Abbildung 3.4: Beispielgraph G_ψ zu $\psi = (X_1 \vee X_2 \vee \neg X_3) \wedge (X_1 \vee \neg X_2) \wedge (\neg X_1 \vee X_3)$

Satz 3.20. CLIQUE *ist NP-vollständig.*

Beweis. Offensichtlich ist CLIQUE $\in$ NP. Wir zeigen: 3-SAT $\leq_p$CLIQUE. Sei $\psi = \bigwedge_{i=1}^{m} C_i$ in 3-KNF, d.h. die Disjunkte C_i enthalten höchstens 3 Literale. Wir konstruieren zu ψ einen Graphen $G_\psi = (V, E)$ durch

- $V := \{(Y, i) \ : \ Y \text{ Literal, } Y \text{ tritt in } C_i \text{ auf }\}$
- $E := \{((Y, i), (Y', j)) \ : \ i \neq j, Y \neq \neg Y', \neg Y \neq Y'\}$.

Behauptung: $(G_\psi, m) \in$ CLIQUE gdw. ψ erfüllbar.

- Sei $\mathfrak{I}(\psi) = 1$. In jeder Klausel C_i kann ein Literal Y_i gewählt werden, mit $\mathfrak{I}(Y_i) = 1$. Dann ist $\{(Y_i, i) \ : \ i = 1, \ldots, m\}$ eine m-punktige Clique von G_ψ.
- Sei K eine m-punktige Clique in G_ψ. Da in G_ψ nur Kanten zwischen $(Y, i), (Y', j)$ mit $i \neq j$ existieren, gibt es zu jedem $i \leq m$ genau ein Y_i mit $(Y_i, i) \in K$.
  Setze $\mathfrak{I}(X) = \begin{cases} 1 & \text{falls } (X, i) \in K \text{ für ein } i \leq m \\ 0 & \text{falls } (\neg X, i) \in K \text{ für ein } i \leq m \end{cases}$. Tritt keiner der beiden Fälle auf, kann $\mathfrak{I}(X)$ beliebig gewählt werden. $\mathfrak{I}$ ist wohldefiniert, da (X, i) und $(\neg X, j)$ nicht verbunden sind, also nicht beide in K liegen können. Daraus folgt $\mathfrak{I}(\psi) = 1$.

□

c) INDEPENDENT SET

Ein Independent Set ist ein diskreter oder unabhängiger Untergraph. Die Menge INDEPENDENT SET ist definiert als

$$\text{INDEPENDENT SET} = \{(G, k) \ : \ \begin{array}{l} G = (V, E) \text{ Graph, es gibt } U \subseteq V, |U| = k, \\ U \times U \cap E = \varnothing\}. \end{array}$$

d) VERTEX COVER (Sternüberdeckung)

Sei $G = (V, E)$ ein Graph, $v \in V$. Dann ist $S_v := \{\{v, w\} : (v, w) \in E\}$ der Stern von v. VERTEX COVER (VC) ist definiert als

$$\text{VERTEX COVER} = \{(G, k) \ : \ \text{es existiert } U \subseteq V, |U| = k \text{ mit } E = \bigcup_{u \in U} S_u\}.$$

e) HAMILTON-ZYKLUS

Sei $G = (V, E)$ ein gerichteter Graph, $|V| = n$. Ein Hamilton-Zyklus H von G ist eine Reihenfolge $v_1, \ldots, v_n$ der Knoten von G (also eine Nachfolgerelation auf V), so dass $(v_i, v_{i+1}) \in E$ für alle $i < n$ und $(v_n, v_1) \in E$, mit $v_i \neq v_j$ für $i \neq j$. Die Mengen HAMILTON-ZYKLUS (HZ) und HAMILTON-KREIS (HK) sind definiert als:

$$\text{HAMILTON-ZYKLUS} = \{G \ : \ G \text{ ger. Graph, } G \text{ besitzt einen Hamilton-Zyklus}\}$$

$$\text{HAMILTON-KREIS} = \{G \ : \ G \text{ unger. Graph, } G \text{ besitzt einen Hamilton-Zyklus}\}$$

f) TSP(D)

Es ist offensichtlich, dass die Probleme c) - f) in NP sind. Wir zeigen die Vollständigkeit durch die Reduktionskette CLIQUE $\leq_p$INDEPENDENT SET $\leq_p$VERTEX COVER $\leq_p$HAMILTON ZYKLUS $\leq_p$HAMILTON-KREIS $\leq_p$TSP(D).

Satz 3.21. INDEPENDENT SET *ist NP-vollständig.*

Beweis. Wir reduzieren CLIQUE $\leq_p$INDEPENDENT SET. Dazu bilden wir ein Element $((V, E), k)$ auf $((V, \overline{E}), k)$ ab, wobei $\overline{E} = \{(x, y) \ : \ x \neq y, (x, y) \notin E\}$ den komplementären Graphen bezeichnet. Eine Clique in (V, E) ist aber gerade eine unabhängige Menge in $(V, \overline{E})$. □

Satz 3.22. VERTEX COVER *ist NP-vollständig.*

Beweis. Wir reduzieren INDEPENDENT SET $\leq_p$VC. Dazu wird (G, k) auf $(G, n - k)$ abgebildet, wobei $n = |V|$ die Anzahl der Knoten von G bezeichnet. $U \subseteq V$ ist unabhängig in G gdw. jede Kanten von G mindestens einen Punkt aus $V - U$ enthält, was genau dann der Fall ist, wenn $V - U$ eine Sternüberdeckung ist. □

Satz 3.23. HAMILTON-ZYKLUS *ist NP-vollständig.*

Beweis. Wir reduzieren VC $\leq_p$HZ. Sei $G = (V, E)$ ein Graph, $k \leq |V|$. Dazu definieren wir aus (G, k) einen gerichteten Graphen $G' = (V', E')$ mit

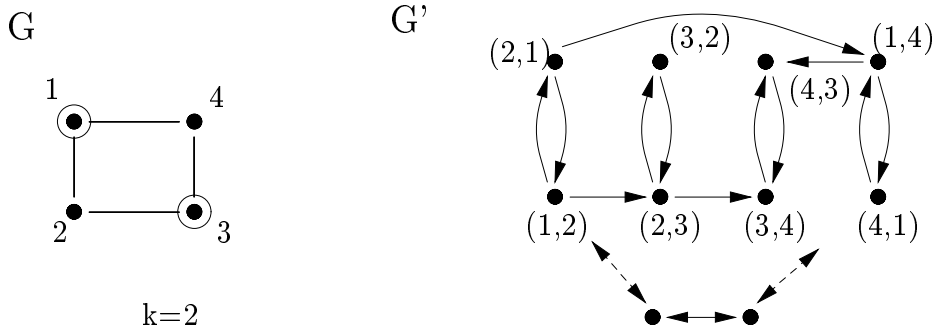
$$V' = Z \cup K = \underbrace{\{1, \ldots, k\}}_{\text{Zahlpunkte}} \cup \underbrace{\{(x, y) \in V \times V \ : \ (x, y) \in E\}}_{\text{Kantenpunkte (in beiden Richtungen)}}$$

Aus einer Kante (x, y) in G werden also zwei Knoten (x, y) und (y, x) in G'.

Zur Definition der Kantenrelation wählen wir eine beliebige aber feste totale Ordnung $\leq$ auf V.

$$\begin{aligned} E' := \ & Z \times K \cup K \times Z \\ & \cup \{(i, j) \in Z \times Z \ : \ j = i + 1 \ (\text{mod } k)\} \\ & \cup \{((x, y), (y, z)) \in K \times K \ : \ x \leq z\} \end{aligned}$$

Zu zeigen ist, dass G genau dann eine k-punktige Sternüberdeckung hat, wenn G' einen Hamilton-Zyklus besitzt.



Beispiel. k=2

Abbildung 3.5: Graph G mit Vertex Cover $\{1,3\}$ und zugehöriger Graph G'

- Sei $U = \{u_1, \ldots, u_k\}$ eine Sternübderdeckung von G. Man betrachtet eine Aufzählung der Kanten in E:

$$\begin{array}{lllll} \{u_1, v_1^1\}, & \{u_1, v_1^2\}, & \ldots, & \{u_1, v_1^{n_1}\} & (= S_{u_1}) \\ \{u_2, v_2^1\}, & \{u_2, v_2^2\}, & \ldots, & \{u_2, v_2^{n_2}\} & (= S_{u_2} - S_{u_1}) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \{u_k, v_k^1\}, & \{u_k, v_k^2\}, & \ldots, & \{u_k, v_k^{n_2}\} & (= S_{u_k} - \bigcup_{j<k} S_{u_j}) \end{array}$$

  so dass $v_i^j < v_i^{j+i}$ für alle i, j. Dann gilt

  - $\Sigma_{i=1}^k n_i$ = Anzahl der Kanten in G,
  - $\{\{u_i, v_i^1\}, \ldots, \{u_i, v_i^{n_i}\}\} = S_{u_i} - \bigcup_{j<i} S_{u_j}$, d.h. es werden nur Paare genommen, welche nicht schon zu einem u_j für $j < i$ gehören.

  In dem Graph G' konstruieren wird nun folgenden Hamilton-Zyklus:

$$\begin{array}{lll} 1 & (u_1, v_1^1) & (v_1^1, u_1)\ (u_1, v_1^2)\ (v_1^2, u_1)\ \ldots\ (u_1, v_1^{n_1})\ (v_1^{n_1}, u_1) \\ 2 & (u_2, v_2^1) & \ldots \\ & \vdots & \\ k-1 & (u_{k-1}, v_{k-1}^1) & \ldots \\ k & (u_k, v_k^1) & \ldots \\ 1 & & \end{array}$$

- Sei umgekehrt $H = (z_1, \ldots, z_r)$ ein Hamilton-Zyklus von G', d.h. $z_i \in V'$, $(z_i, z_{i+1}) \in E'$ und $(z_r, z_1) \in E'$. Für $x, y \in V'$ heisst y *Nachfolger* von x bezüglich H, wenn y im Hamilton-Zyklus unmittelbar auf x folgt, d.h. $y = z_j, x = z_i$ und $j = i + 1$ (mod r). Sei $C := \{u \in V$: es gibt $v \in V$, so dass der Kantenpunkt (u, v) Nachfolger eines Zahlpunktes ist $\}$.

  1. $|C| \leq k$, da nur k Zahlpunkte existieren.
  2. Für alle $(u, v) \in K$ gilt: Ist (u, v) nicht Nachfolger von (v, u), so ist $u \in C$. Zur Begründung wähle man zu festem u das kleinste v mit $(u, v) \in K$ und (u, v) ist nicht Nachfolger von (v, u). Für alle $v' < v$ mit $(u, v') \in K$ ist also (u, v') Nachfolger von (v', u), also ist (u, v) nicht Nachfolger von (v', u). Da (u, v) nur von Kantenpunkten (v', u) mit $v' \leq v$ in einem Schritt erreichbar ist, muss also (u, v) Nachfolger eines Zahlpunktes sein. Daher ist $u \in C$.

3. C ist Sternüberdeckung von G. Sei (u, v) eine Kante von G. Mindestens einer der Kantenpunkte $(u, v), (v, u)$ ist nicht Nachfolger des anderen. Also ist $u \in C$ oder $v \in C$.

□

Satz 3.24. HAMILTON-KREIS *ist NP-vollständig.*

Beweis. Wir reduzieren HZ $\leq_p$HK. Dazu bilden wir den gerichteten Graph $G = (V, E)$ auf den ungerichteten Graph $G' = (V', E')$, mit

- $V' = V \times \{0, 1, 2\}$
- $E' = \{\{(u, i), (u, j)\} : u \in V, \{i, j\} = \{0, 1\} \text{ oder } \{i, j\} = \{1, 2\}\} \cup \{\{(u, 2), (v, 0)\} : (u, v) \in E\}$.

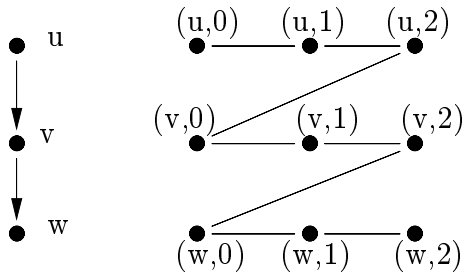


Abbildung 3.6: Umwandlung des gerichteten Graphen in einen ungerichteten.

- Sei $z_1, \ldots, z_n$ ein Hamilton-Zyklus von G. Dann ist $(z_1, 0)(z_1, 1)(z_1, 2)(z_2, 0), \ldots, (z_n, 0)$ $(z_n, 1)$ $(z_n, 2)$ ein Hamiltonkreis von G'.
- Sei $z_1, \ldots, z_{3n}$ ein Hamiltonkreis von G'. Sei weiterhin $z_i = (u, 1)$. Dann gilt $z_{i-1} = (u, 0), z_{i+1} = (u, 2)$ oder umgekehrt, da dies die einzigen Nachbarn von $(u, 1)$ sind. Daraus folgt $z_{i+2} = (v, 0)$ mit $(u, v) \in E$, $z_{i+3} = (v, 1)$, $z_{i+4} = (v, 2), \ldots$

  Also hat $z_1, \cdots, z_{3n}$ die Form $(v_1, 0)(v_1, 1), (v_1, 2), (v_2, 0)(v_2, 1), \ldots, (v_n, 0), (v_n, 1), (v_n, 2)$, so dass $v_1, v_2, \ldots, v_n$ ein Hamilton-Zyklus von G ist.

□

Satz 3.25. TSP(D) *ist NP-vollständig.*

Beweis. Wir reduzieren HAMILTON-KREIS $\leq_p$TSP(D). Dazu konstruieren wir aus $G = (V, E)$ das Paar (D, k), wobei D eine $n \times n$-Matrix über $\mathbb{N}$ mit $n = |V|$ und $k \in \mathbb{N}$ ist. Sei $V = \{v_1, \ldots, v_n\}$. Setze $D = (d_{i,j})_{1 \leq i,j \leq n}$ mit $d_{i,j} = \begin{cases} 1 & \text{wenn } (v_i, v_j) \in E \\ 2 & \text{sonst} \end{cases}$. Dabei ist $k = n$.

Dann hat G genau dann einen Hamilton-Kreis, wenn D eine Tour der Länge n erlaubt. □

Kapitel 4

PSPACE und die poylnomiale Hierarchie

4.1 Ein PSPACE-vollständiges Problem

Wir erinnern nochmals kurz an zwei wichtige Eigenschaften der Komplexitätsklasse PSPACE := $\bigcup_k$ DSPACE(n^k).

- PSPACE = $\bigcup_{k\in\mathbb{N}}$ NSPACE(n^k) = NPSPACE, da nach dem Satz von Savitch gilt: NSPACE(s) $\subseteq$ DSPACE(s^2),
- NP $\subseteq$ PSPACE, da gilt: NTIME(n^k) $\subseteq$ NSPACE(n^k) $\subseteq$ DSPACE(n^{2k}) $\subseteq$ PSPACE.

Ein Problem A ist PSPACE-hart, wenn für alle $B \in$ PSPACE gilt: $B \leq_p A$. A ist PSPACE-vollständig, wenn $A \in$ PSPACE und A PSPACE-hart ist.

Als Beispiel eines PSPACE-vollständigen Problems wird hier das Auswertungsproblem für *Quantifizierte Bool'sche Formeln (QBF)* vorgestellt.

Zur Definition der *Quantifizierten Aussagenlogik* fixiert man eine Menge $\tau = \{X_i : i \in \mathbb{N}\}$ von Aussagenvariablen.

Definition 4.1. QAL ist die kleinste Menge mit

- AL $\subseteq$ QAL,
- $\psi, \varphi \in$ QAL $\implies (\psi \vee \varphi), (\psi \wedge \varphi), \neg\psi \in$ QAL,
- $\psi \in$ QAL, X Aussagenvariable $\implies \exists X\psi, \forall X\psi \in$ QAL.

Beispiel. $\exists X(\forall Y(X \vee Y) \wedge \exists Z(X \vee Z))$

Mit Frei(ψ) sei die Menge der in ψ frei auftretenden Aussagenvariablen bezeichnet. Jede aussagenlogische Interpretation $\mathfrak{I} : \sigma \longrightarrow \{0,1\}$ mit $\sigma \subseteq \tau$ definiert Wahrheitswerte $\mathfrak{I}(\psi)$ für alle $\psi \in$ QAL mit Frei(ψ) $\subseteq \sigma$. Sei $\mathfrak{I}$ eine Bewertung und $X \in \tau$ eine Aussagenvariable. $\mathfrak{I}[X = 1]$ sei die Bewertung, welche mit $\mathfrak{I}$ auf allen $Y \in \tau, Y \neq X$ übereinstimmt und X mit 1 bewertet. Analog dazu sei $\mathfrak{I}[X = 0]$ die Belegung mit $\mathfrak{I}[X/0](Y) = \mathfrak{I}(Y)$ für $Y \neq X$ und $\mathfrak{I}[X/0](X) = 0$. Es ist also $\mathfrak{I}(\exists X\psi) = 1$ genau dann, wenn $\mathfrak{I}[X/0](\psi) = 1$ oder $\mathfrak{I}[X/1](\psi) = 1$. Analog ist $\mathfrak{I}(\forall X\psi) = 1$ genau dann, wenn $\mathfrak{I}[X/0](\psi) = 1$ und $\mathfrak{I}[X/1](\psi) = 1$.

Ist Frei(ψ) = $\varnothing$, so ist der Wahrheitswert $\mathfrak{I}(\psi) \in \{0,1\}$ eindeutig bestimmt, unabhängig von einer konkreten Interpretation. Zum Beispiel ist die Formel $\exists X(\forall Y(X \vee Y) \wedge \exists Z(X \vee Z))$ wahr.

Definition 4.2. QBF $= \{\psi \in \text{QBF} : \text{Frei}(\psi) = \varnothing\ ,\ \psi \text{ wahr}\}$.

Bemerkung. Sei $\psi = \psi(X_1, \ldots, X_n)$ eine aussagenlogische also quantorenfreie Formel. Dann gilt

$$\psi \in \text{SAT} \iff \exists X_1 \cdots \exists X_n \psi \in \text{QBF}.$$

QBF ist also mindestens so schwer wie SAT. Tatsächlich ist QBF PSPACE-vollständig.

Satz 4.3. QBF $\in$ PSPACE.

Beweis. Um dies beweisen zu können, wird eine rekursive Prozedur $\text{Eval}(\psi, \mathfrak{I})$ konstruiert, welche für $\psi \in \text{QBF}$ und $\mathfrak{I}$: $\text{Frei}(\psi) \to \{0,1\}$ den Wert $\mathfrak{I}(\psi)$ berechnet.

Eval$(\psi, \mathfrak{I})$
Input: $\psi, \mathfrak{I}$
if $\psi = X \in V$ **then output** $\mathfrak{I}(X)$ **end**
if $\psi = (\varphi_1 \vee \varphi_2)$ **then**
  if Eval$(\varphi_1, \mathfrak{I}) = 1$ **then output** 1 **end**
  else output Eval$(\varphi_2, \mathfrak{I})$ **end**
if $\psi = (\varphi_1 \wedge \varphi_2)$ **then**
  if Eval$(\varphi_2, \mathfrak{I}) = 0$ **then output** 0 **end**
  else output Eval$(\varphi_2, \mathfrak{I})$ **end**
if $\psi = \neg\varphi$ **output** $1 -$ Eval$(\varphi, \mathfrak{I})$ **end**
if $\psi = \exists X \varphi$ **then**
  if Eval$(\varphi, \mathfrak{I}[X = 0]) = 1$ **then output** 1 **end**
  else output Eval$(\varphi, \mathfrak{I}[X = 1])$ **end**
if $\psi = \forall X \varphi$ **then**
  if Eval$(\varphi, \mathfrak{I}[X = 0]) = 0$ **then output** 0
  else output Eval$(\varphi, II[X = 1])$ **end**

Die Prozedur hat einen Platzbedarf von $O(n^2)$. Falls $\text{Frei}(\psi) = \varnothing$ ist, so hängt $\text{Eval}(\psi, \mathfrak{I})$ nur noch von ψ ab und $\mathfrak{I}(\psi)$ wird korrekt berechnet. □

Satz 4.4. QBF *ist* PSPACE*-hart.*

Beweis. Sei $A \in$ PSPACE und M eine n^k-platzbeschränkte 1-Band TM mit $L(M) = A$. Jede Konfiguration von M auf einem Input w der Länge n wird beschrieben durch ein Tupel $\bar{X}$ von Aussagenvariablen bestehend aus:

X_q	(q Zustand von M)	: M befindet sich im Zustand q.
$X'_{a,i}$	(a Bandsymbol, $i \leq n^k$)	: Im Feld i steht das Zeichen a.
X''_j	($j \leq n^k$)	: M liest Feld j.

Analog zum Beweis der NP-Vollständigkeit von SAT werden Formeln $\text{Conf}(\bar{X})$, $\text{Next}(\bar{X}, \bar{Y})$, $\text{Input}_w(\bar{X})$ und $\text{Acc}(\bar{X})$ mit den folgenden Interpretationen konstruiert:

$\text{Conf}(\bar{X})$: $\bar{X}$ kodiert eine Konfiguration, d.h. es ist genau ein X_q, zu jedem i genau ein $X'_{a,i}$ und genau ein X''_j wahr.

$\mathrm{Input}_w(\bar{X})$: $\bar{X}$ kodiert die Input-Konfiguration von M auf $w = w_0 \cdots w_{n-1}$:

$$\mathrm{Input}_w(\bar{X}) := \mathrm{Conf}(\bar{X}) \wedge X_{q_0} \wedge \bigwedge_{i=0}^{n-1} X'_{w_i,i} \wedge \bigwedge_{i=n}^{n^k} X'_{\Box,i} \wedge X''_0.$$

$\mathrm{Acc}(\bar{X})$: $\bar{X}$ ist eine akzeptierende Konfiguration:

$$\mathrm{Acc}(\bar{x}) := \mathrm{Conf}(\bar{X}) \wedge \bigvee_{q \in E^+} X_q.$$

$\mathrm{Next}(\bar{X}, \bar{Y})$: $\bar{Y}$ ist Nachfolgekonfiguration von $\bar{X}$:

$$\mathrm{Next}(\bar{X}, \bar{Y}) := \bigwedge_i \{X''_i \to [\bigwedge_{a,j \neq i} (Y'_{a,j} \leftrightarrow X_{a,j}) \wedge \bigwedge_{\substack{\delta(q,a)=(q',b,m) \\ 0 \leq m+i \leq n^k}} (X_q \wedge X'_{a,i} \to Y_{q'} \wedge Y'_{b,i} \wedge Y''_{i+m})]\}.$$

Diese Formeln können in polynomialer Zeit aus w konstruiert werden. Außerdem sei noch

$$\mathrm{Eq}(\bar{X}, \bar{Y}) \equiv \bigwedge_q (X_q \leftrightarrow Y_q) \wedge \bigwedge_{a,i} (X'_{a,i} \leftrightarrow Y'_{a,i}) \wedge \bigwedge_j (X''_j \leftrightarrow Y''_j).$$

Nun wird eine Formel $\mathrm{Reach}_m(\bar{X}, \bar{Y})$ konstruiert, die folgende Interpretation hat: $\bar{X}$ und $\bar{Y}$ kodieren Konfigurationen und $\bar{Y}$ ist von $\bar{X}$ aus in höchstens 2^m Schritten erreichbar. Für $m = 0$ sei

$$\mathrm{Reach}_0(\bar{X}, \bar{Y}) := \mathrm{Conf}(\bar{X}) \wedge \mathrm{Conf}(\bar{Y}) \wedge (\mathrm{Eq}(\bar{X}, \bar{Y}) \vee \mathrm{Next}(\bar{X}, \bar{Y})).$$

Ein erster, naheliegender Ansatz zur Definition von Reach_{m+1} wäre

$$\mathrm{Reach}_{m+1} := \exists \bar{Z}(\mathrm{Reach}_m(\bar{X}, \bar{Z}) \wedge \mathrm{Reach}_m(\bar{Z}, \bar{Y})).$$

Das Problem liegt darin, dass $|\mathrm{Reach}_{m+1}| \geq 2 \cdot |\mathrm{Reach}_m|$. Daher ist $|\mathrm{Reach}_m| \geq 2^m$. Wir können aber Reach_{m+1} auch anders konstruieren, so dass durch Verwendung von universellen Quantoren dieses exponentielle Wachstum der Formellänge vermieden wird:

$$\begin{aligned}\mathrm{Reach}_{m+1}(\bar{X}, \bar{y}) :=& \exists \bar{Z} \forall \bar{U} \forall \bar{V} \Big(\big((\mathrm{Eq}(\bar{X}, \bar{U} \wedge \mathrm{Eq}(\bar{Z}, \bar{V})) \vee (\mathrm{Eq}(\bar{Z}, \bar{U}) \wedge \mathrm{Eq}(\bar{Y}, \bar{V})) \big) \\ & \to \mathrm{Reach}_m(\bar{U}, \bar{V}) \Big)\end{aligned}$$

Nun gilt:

$|\mathrm{Reach}_0| = O(n^k)$ für ein geeignetes k,

$|\mathrm{Reach}_{m+1}| = |\mathrm{Reach}_m| + O(n^k)$.

Also ist $|\mathrm{Reach}_m| = O(m \cdot n^k)$.

Wenn M den Input w mit Platz n^k akzeptiert, dann auch in Zeit $\leq 2^{c \cdot n^k}$ für eine geeignete Kostante c. Setze $m := c \cdot n^k$ und

$$\psi_w := \exists \bar{X}\ \exists \bar{Y} (\mathrm{Input}(\bar{X}) \wedge \mathrm{Acc}(\bar{Y}) \wedge \mathrm{Reach}_m(\bar{X}, \bar{Y})).$$

Offensichtlich ist ψ_w aus w in polynomialer Zeit konstruierbar und es gilt: $\psi_w \in$ QBF genau dann, wenn $w \in L(M)$. Also ist QBF PSPACE-vollständig. □

4.2 Orakel-Turingmaschinen

Definition 4.5. Eine deterministische bzw. nichtdeterministische *Orakel-TM* ist eine TM mit einem besonderen Orakel-Band und drei ausgezeichneten Zuständen "? (Frage)", "J (ja)" und "N (nein)". Die Übergangsfunktion ist für alle Paare (q, a) mit $q \neq ?$ wie gewöhnlich definiert.

Die Konfiguration C des Orakel-Bandes einer Orakel-TM mit k Arbeitsbändern hat die Form

$$C = (q, p_0, \cdots, p_k, w_0, \cdots, w_k),$$

wobei

- q der Zustand der TM ist,
- $p_0, \cdots, p_k$ die Kopfpositionen auf den Bändern sind (p_0 ist die Kopfposition des Orakelbandes) und
- $w_0 \cdots, w_k$ die Bandinschriften sind (w_0 ist die Inschrift des Orakelbandes).

Die Berechnung (bzw. der Berechnungsbaum) einer Orakel-TM ist abhängig von einer vorher festgelegten Orakelmenge $A \subseteq \Sigma^*$ (Σ ist das Alphabet von M). Die Nachfolgekonfigurationen einer Konfiguration C sind wie üblich definiert für $q \neq ?$. Für $q = ?$ ist die Nachfolgekonfiguration C' definiert als:

$$C' = \begin{cases} (J, 0, p_1, \cdots, p_k, \lambda, w_1, \cdots w_k) & \text{falls } w_0 \in A \\ (N, 0, p_1, \cdots, p_k, \lambda, w_1, \cdots, w_k) & \text{falls } w_0 \notin A \end{cases}$$

wobei λ das leere Wort ist. Das Orakel beantwortet also die Frage, ob w_0 (die Inschrift des Orakelbandes) in A ist. Dann geht die Maschine in den entsprechenden Zustand (J oder N) und die Inschrift des Orakelbandes wird gelöscht.

Definition 4.6. Sei M eine Orakel-Turingmaschine, $A \subseteq \Sigma^*$ eine Orakelmenge. Dann ist

$$L(M^A) := \{x : M \text{ akzeptiert den Input } x \text{ mit Orakel } A\}.$$

Ausserdem definiert eine Orakelmenge A die folgenden Komplexitätsklassen:

(i) $\mathrm{P}^A := \{L$: es gibt eine deterministische Orakel-TM M, welche L mit Orakel A in polynomialer Zeit entscheidet.$\}$

(ii) $\mathrm{NP}^A := \{L$: es gibt eine nichtdeterministische Orakel-TM M, welche L mit Orakel A in polynomialer Zeit entscheidet.$\}$

Sei $\mathcal{C}$ eine Klasse von Sprachen, z.B. eine Komplexitätsklasse. Dann ist

$$\mathrm{P}^{\mathcal{C}} = \bigcup_{A \in \mathcal{C}} P^A \qquad \text{und} \qquad \mathrm{NP}^{\mathcal{C}} = \bigcup_{A \in \mathcal{C}} \mathrm{NP}^A.$$

Beispiele. (1) Sei $B \in \mathrm{NP}$. Dann ist $B \in P^{\mathrm{SAT}}$. Es gibt eine polynomial berechenbare Funktion f mit $x \in B \iff f(x) \in \mathrm{SAT}$. Folgender Orakel-Algorithmus entscheidet dann B:

Input: x
berechne $f(x)$
Orakelfrage: $f(x) \in$ SAT
wenn ja: akzeptiere
wenn nein: verwerfe

(2) Vermutlich ist NP$\subsetneq P^{\mathrm{SAT}}$, denn auch jedes $B \in$ Co-NP ist in P^{SAT}. Dazu kann man obigen Algorithmus benutzen und vertauscht das Verhalten für die Antworten Ja und Nein.

(3) Sei $B := \{(G,k) : G \text{ Graph }, \omega(G) = k\}$, wobei $\omega(G)$ die maximale Anzahl der Knoten von Cliquen in G ist. Zur Erinnerung: Das Problem CLIQUE $:= \{(G,k) : k \leq \omega(G)\})$ ist NP-vollständig. Es ist leicht einzusehen, dass $B \in P^{\mathrm{CLIQUE}}$:

Input: G, k
Orakelabfrage: $(G,k) \in$ CLIQUE ?
wenn nein, verwerfe
wenn ja: Orakelabfrage: $(G, k+1) \in$ CLIQUE ?
wenn nein, akzeptiere
wenn ja, verwerfe

4.3 Die polynomiale Hierarchie

Definition 4.7. Wir definieren für alle $k \in \mathbb{N}$ die Komplexitätsklassen Σ_k^p , Π_k^p , Δ_k^p:

- $\Sigma_0^p := \Pi_0^p := \Delta_0^p := \mathrm{P}$
- $\Sigma_{k+1}^p := \mathrm{NP}^{\Sigma_k^p}$
- $\Pi_k^p := \text{Co-}\Sigma_k^p = \{\overline{A} : A \in \Sigma_k^p\}$
- $\Delta_{k+1}^p := \mathrm{P}^{\Sigma_k^p}$

Satz 4.8. *Die Klassen Σ_k^p , Π_k^p , Δ_k^p haben folgende elementare Eigenschaften:*

(i) $\Delta_1^p = P$.

(ii) $\Sigma_1^p = \mathrm{NP}$, $\Pi_1^p = Co\text{-NP}$.

(iii) $\Sigma_{k+1}^p = \mathrm{NP}^{\Pi_k^p} = \mathrm{NP}^{\Delta_{k+1}^p}$.

(iv) $P^{\Delta_k^p} = \Delta_k^p$.

Beweis.

(i) $\Delta_1^p = \mathrm{P}^{\mathrm{P}} = \mathrm{P}$.

(ii) $\Sigma_1^p = \mathrm{NP}^{\mathrm{P}} = \mathrm{NP}$, $\Pi_1^p = \text{Co-}\Sigma_1^p = \text{Co-NP}$.

(iii) Sei $B \in \Sigma_{k+1}^p$, $B = L(M^A)$ und $A \in \Sigma_k^p$. Sei ferner M' gleich M nur mit vertauschten "J" und "N". Offensichtlich gilt auch $B = L(M'^{\overline{A}})$ und daher $B \in \mathrm{NP}^{\Pi_k^p}$. Ausserdem gilt $\mathrm{NP}^{\Delta_{k+1}^p} = \mathrm{NP}^{P^{\Sigma_k^p}} = \mathrm{NP}^{\Sigma_k^p} = \Sigma_{k+1}^p$.

(iv) $k = 0 : P^{\Delta_0^p} = \mathrm{P}^{\mathrm{P}} = \mathrm{P} = \Delta_0^p$.
$k > 0 : \mathrm{P}^{\Delta_k^p} = \mathrm{P}^{\mathrm{P}^{\Sigma_{k-1}^p}} = \mathrm{P}^{\Sigma_{k-1}^p} = \Delta_k^p$.

□

Satz 4.9. *Für alle k gilt:* $\Sigma_k^p \cup \Pi_k^p \subseteq \Delta_{k+1}^p \subseteq \Sigma_{k+1}^p \cap \Pi_{k+1}^p$.

Beweis. Für $k = 0$ besagt die Behauptung: $\mathrm{P} \subseteq \mathrm{P} \subseteq \mathrm{NP} \cap \text{Co-NP}$. Dies ist offensichtlich richtig. Für $k > 0$ gilt:

- $\Sigma_k^p \subseteq \mathrm{P}^{\Sigma_k^p}$ und daher auch $\Pi_k^p \subseteq \mathrm{P}^{\Sigma_k^p}$, weil $\mathrm{P}^{\Sigma_k^p}$ komplementabgeschlossen ist. Daraus folgt, dass $\Sigma_k^p \cup \Pi_k^p \subseteq \Delta_{k+1}^p$.
- $\Delta_{k+1}^p = \mathrm{P}^{\Sigma_k^p} = \text{Co-P}^{\Sigma_k^p} \subseteq \text{Co-NP}^{\Sigma_k^p} = \Pi_{k+1}^p$.
  $\Delta_{k+1}^p = \mathrm{P}^{\Sigma_k^p} \subseteq \mathrm{NP}^{\Sigma_n^p} = \Sigma_{k+1}^p$. Also ist $\Delta_{k+1}^p \subseteq \Sigma_{k+1}^p \cap \Pi_{k+1}^p$.

□

Satz 4.10. *Falls ein k existiert, so dass $\Sigma_{k+1}^p = \Sigma_k^p$, dann gilt $\Sigma_{k+i}^p = \Pi_{k+i}^p = \Sigma_k^p$ für alle $i > 0$.*

Beweis. Für $i = 1$ gilt $\Sigma_{k+i}^p = \Sigma_{k+1}^p = \Sigma_k^p$ nach Voraussetzung. Nach Induktionsvoraussetzung sei nun $\Sigma_{k+i}^p = \Sigma_k^p$. Dann folgt:

$$\Sigma_{k+i+1}^p = \mathrm{NP}^{\Sigma_{k+i}^P} = \mathrm{NP}^{\Sigma_k^p} = \Sigma_{k+1}^p = \Sigma_k^p$$

und daher auch

$$\Pi_{k+1}^p \subseteq \Sigma_{k+i+1}^p = \Sigma_k^p \quad \text{für alle } i.$$

Insbesondere gilt dann $\Pi_k^p \subseteq \Sigma_k^p$. Es bleibt zu zeigen, dass $\Sigma_k^p \subseteq \Pi_k^p$. Wenn $B \in \Sigma_k^p$, dann gilt $\overline{B} \in \Pi_k^p \subseteq \Sigma_k^p$, also $B \in \Pi_k^p$. □

Falls dieser Fall eintritt, sagt man, daß die polynomiale Hierarchie zu Σ_k^p kollabiert.

Korollar 4.11. *Falls ein $k > 0$ existiert, mit $\Sigma_k^p \neq \mathrm{P}$, dann ist $\mathrm{P} \neq \mathrm{NP}$.*

Definition 4.12. $\mathrm{PH} := \bigcup_{k \in \mathbb{N}} \Sigma_k^p$ ist die *polynomiale Hierarchie.*

Satz 4.13. $\mathrm{PH} \subseteq \textsc{Pspace}$.

Beweis. Per Induktion über k beweist man, dass $\Sigma_k^p \subseteq \textsc{Pspace}$ für alle $k \in \mathbb{N}$:

$$\Sigma_0^p = \mathrm{P} \subseteq \textsc{Pspace}$$
$$\Sigma_{k+1}^p = \mathrm{NP}^{\Sigma_k^p} \subseteq \mathrm{NP}^{\textsc{Pspace}} \subseteq \textsc{Pspace}^{\textsc{Pspace}} = \textsc{Pspace}$$

Mit $\textsc{Pspace}^{\textsc{Pspace}}$ ist dabei die Klasse der Sprachen gewählt, die von einer deterministischen polynomial platzbeschränkten Orakel-Turingmaschine mit einem Orakel in Pspace entschieden werden können. (Der Platz auf dem Orakelband wird hier beim Platzverbrauch mitgerechnet.) □

Wenn PH = PSPACE gilt, dann kollabiert die polynomiale Hierarchie:

Satz 4.14. *Gilt* PH = PSPACE*, so existiert ein* k *mit* PH $= \Sigma_k^p$.

Beweis. Falls PH = PSPACE, so gilt QBF $\in$ PH. Also existiert ein k so dass QBF $\in \Sigma_k^p$. Für jedes $A \in$ PSPACE gilt dann $A \leq_m^p$ QBF und damit auch $A \in \Sigma_k^p$.
Daraus folgt PH $= \Sigma_k^p$. □

Man vermutet, dass $\Sigma_k^p \subsetneq \Sigma_{k+1}^p$ für alle k, dass also die polynomielle Hierarchie tatsächlich eine strikte Hierarchie ist und somit PH $\subsetneq$ PSPACE.

Ergänzungen (ohne Beweis)

1. Zunächst zwei vollständige Probleme für Σ_k^p und Π_k^p:
   Es ist Σ_k-QBF $= \{\psi = (\exists \overline{X_1})(\forall \overline{X_2}) \dots (Q_k \overline{X_k})\varphi \in \text{QAL} \ : \ \varphi \in \text{AL}, \psi \text{ wahr } \}$. Dabei ist Q_k ein All-Quantor, falls k gerade ist, ansonsten ein Existenz-Quantor.
   Π_k-QBF ist analog definiert, nur beginnen die Formeln mit All-Quantoren.
   Das Problem Σ_k-QBF ist Σ_k^p-vollständig, analog ist Π_k-QBF Π_k^p-vollständig. Dies verallgemeinert die NP-Vollständigkeit von SAT.

2. Als weitere Ergänzung soll hier noch eine andere Darstellung von Σ_k^p und Π_k^p gegeben werden. Zur Erinnerung hier noch einmal die Definition von NP. Es ist $A \in$ NP genau dann, wenn es ein $B \in$ P und ein Polynom $p(n)$ gibt, mit $A = \{x \ : \ \exists^p y(x\#y \in B)\}$. Dabei ist $\exists^p y$ eine Abkürzung für $\exists y \ : \ |y| \leq p(|x|)$.
   Nun zur Charakterisierung von Σ_k^p und Π_k^p:

   - Es ist $A \in \Sigma_k^p$ genau dann, wenn es ein $B \in$ P und ein Polynom $p(n)$ gibt, mit $A = \{x \ : \ (\exists^p y_1)(\forall^p y_2)(\exists^p y_3) \dots (Q_k^p y_k)x\#y_1\#y_2\# \dots \#y_k \in B\}$. Dabei ist Q_k ein Existenz-Quantor, wenn k ungerade ist, ansonsten ein All-Quantor.
   - Analog kann Π_k^p charakterisiert werden, nur beginnen die Formelm mit einem All-Quantor.

4.4 Relativierungen

Es ist ein ungelöstes Problem, ob P = NP. Eine interessante Frage ist, ob es Orakel A, B gibt, mit

- $P^A = \text{NP}^A$
- $P^B \neq \text{NP}^B$.

Die erste Frage ist sehr leicht zu beantworten, denn es gilt $\text{P}^{\text{QBF}} = \text{NP}^{\text{QBF}} = \text{PSPACE}$. Die zweite Frage ist hingegen nicht so leicht zu beantworten:

Satz 4.15. *Es gibt ein Orakel* B*, so dass* $\text{P}^B \neq \text{NP}^B$.

Beweis. Bekanntlich können Turingmaschinen rekursiv aufgezählt werden, ebenso können auch polynomial zeitbeschränkte Orakel-TM rekursiv aufgezählt werden. Man wählt nun eine rekursive Aufzählung $\{M_i : i \in \mathbb{N}\}$ der deterministischen, polynomialen Orakel-TM, so dass für alle Orakel A gilt:

1. $P^A = \{L(M_i^A) \ : \ i \in \mathbb{N}\}$
2. Es existiert eine Folge $\{p_i(n) \mid i \in \mathbb{N}\}$ von Polynomen, mit:

   (i) M_i ist p_i-zeitbeschränkt,

   (ii) $p_i(n) \leq p_{i+1}(n)$ für alle i, n.

Eine gegebene Folge $\{q_i(n) \ : \ i \in \mathbb{N}\}$ von Zeitschranken für $\{M_i \mid i \in \mathbb{N}\}$ kann zu einer solchen Folge $p_{i+1}(n)$ modifiziert werden durch:

- $p_0(n) := q_0(n)$,
- $p_{i+1}(n) := \max(p_i(n), q_{i+1}(n))$.

Für jedes $C \subseteq \{0,1\}^*$ sei $S(C) = \{0^n \ : \ \text{es existiert ein } x \in C \text{ mit } |x| = n.\}$

Offensichtlich gilt

Lemma 4.16. *Für alle B gilt: $S(B) \in \mathrm{NP}^B$.*

Das Ziel besteht nun darin, ein B zu finden, so dass $S(B) \notin \mathrm{P}^B$. Dieses B wird folgendermaßen konstruiert: Zu Beginn initialisiere $B_0 := \varnothing$ und $k_0 := 0$.
Für $n > 0$ konstruiere B_n, k_n folgendermaßen:

- Setze k_n die kleinste natürliche Zahl mit $2^{k_n} > p_n(k_n)$ und $k_n > p_{n-1}(k_{n-1})$.
- Wenn $0^{k_n} \in L(M_n^{B_{n-1}})$, dann setze $B_n := B_{n-1}$. Andernfalls sei $w(n)$ das lexikographisch erste Wort in $\{0,1\}^{k(n)}$, das nicht als Orakelabfrage in der Berechnung von $M_n^{B_{n-1}}$ auf Input 0^{k_n} vorkommt. Ein solches Wort existiert, da $p_n(k_n) < 2^{k_n}$. Setze $B_n = B_{n-1} \cup \{w(n)\}$.

Setze $B := \bigcup_{n \in \mathbb{N}} B_n$.

Lemma 4.17. *Für alle n gilt: $0^{k_n} \in L(M_n^B) \iff 0^{k_n} \in L(M_n^{B_{n-1}})$.*

In der Rechnung von M_n^B auf 0^{k_n} kommt kein $w \in B - B_{n-1}$ als Orakelabfrage vor:

- für $w = w(n)$ nach Konstruktion und
- für $w = w(m)$ mit $m > n$, weil $|w(m)| = k(m) > p_n(k_n)$.

Lemma 4.18. *$S(B) \notin P^B$.*

Sonst gäbe es ein $n \in \mathbb{N}$ mit $S(B) = L(M_n^B)$. Ein solches n gibt es aber nicht, denn nach Definition ist $0^{k_n} \in S(B)$ genau dann, wenn ein w existiert mit $|w| = k_n$ und $w \in B$. Nach Konstruktion von B ist dies aber genau dann der Fall, wenn $0^{k_n} \notin L(M_n^{B_{n-1}})$. Dies folgt daraus, dass genau dann ein Wort der Länge k_n zu B hinzugefügt wird, wenn $0^{k_n} \notin L(M_n^{B_{n-1}})$. Nach Lemma 4.17 ist $0^{k_n} \notin L(M_n^{B_{n-1}})$ äquivalent zu $0^{k_n} \notin L(M_n^B)$, also $S(B) \neq L(M_n^B)$. □

Für sehr viele Komplexitätsklassen $\mathcal{C}_1$ und $\mathcal{C}_2$ ist das Problem, ob $\mathcal{C}_1 = \mathcal{C}_2$ ist, ungelöst. Für die meisten diese ungelösten Probleme gibt es Orakel A und B, so dass:

$$\mathcal{C}_1^A = \mathcal{C}_2^A \qquad \text{und} \qquad \mathcal{C}_1^B \neq \mathcal{C}_2^B.$$

In vielen Fällen wurde außerdem noch gezeigt, daß für fast alle Orakel D gilt: $\mathcal{C}_1^D \neq \mathcal{C}_2^D$. Solche „widersprüchlichen Relativierungen“ zeigen, dass im Hinblick auf das Problem $\mathcal{C}_1 \neq \mathcal{C}_2$ alle diejenigen Beweistechniken versagen, welche „relativieren“, d.h. unabhängig von Orakeln sind.

Kapitel 5

Parallele Berechnungsmodelle I : Alternierende Turingmaschinen

Sei M eine nichtdeterministische Turingmaschine, x eine Eingabe und $\mathfrak{T}_{M,x}$ der Berechnungsbaum von M auf x. M akzeptiert x, wenn es einen Pfad in $\mathfrak{T}_{M,x}$ von der Wurzel bis zu einer akzeptierenden Konfiguration gibt. Eine andere Methode, die gleiche Situation zu beschreiben wäre die folgende: jeder Knoten $C \in \mathfrak{T}_{M,x}$ definiert den *Unterbaum* T_C, dessen Wurzel C ist. Man definiert induktiv, dass T_C akzeptierend ist, wenn

- C eine akzeptierende Endkonfiguration ist, oder
- eine Nachfolgekonfiguration C' von C existiert, so dass T_C akzeptierend ist.

M akzeptiert x, wenn $\mathfrak{T}_{M,x} = T_{C_0}$ akzeptierend ist, wobei C_0 die Wurzel von $\mathfrak{T}_{M,x}$ ist.

Definition 5.1. Eine alternierende TM ist eine nichtdeterministische TM, deren Zustandsmenge Q in die vier Klassen $Q_\exists$, $Q_\forall$, Q_{acc} und Q_{rej} unterteilt ist.

Die Zustände unterteilen sich also in existentielle, universelle, akzeptierende und verwerfende Zustände. $Q_{acc} \cup Q_{rej}$ sind dann die Endzustände. Eine Konfiguration von M heißt existentiell, universell, akzeptierend bzw. verwerfend, wenn der zugehörige Zustand existentiell, universell, akzeptierend bzw. verwerfend ist.

Der Berechnungsbaum $\mathfrak{T}_{M,x}$ einer alternierenden TM ist definiert wie der einer nichtdeterministischen TM. Die Akzeptanzbedingung ist aber komplizierter: T_C ist ein akzeptierender Unterbaum, wenn

- C eine akzeptierende Endkonfiguration ist, oder
- C eine existentielle Konfiguration ist und es eine Nachfolgekonfiguration C' von C gibt, so dass $T_{C'}$ ein akzeptierender Unterbaum ist, oder
- C eine universelle Konfiguration ist und für alle Nachfolgekonfigurationen C' von C gilt: $T_{C'}$ ist ein akzeptierender Unterbaum.

M akzeptiert die Eingabe x, wenn $T_{C_0} = \mathfrak{T}_{M,x}$ (C_0 ist die Wurzel von $\mathfrak{T}_{M,x}$) akzeptierend ist. Eine andere Möglichkeit die Akzeptanzbedingungen zu beschreiben ergibt sich aus einem Spiel auf dem Berechnungsbaum. Daran nehmen die zwei Spieler $\exists$ und $\forall$ teil, und es wird nach den

folgenden Regeln gespielt:
Man beginnt bei der Wurzel C_0 von $\mathfrak{T}_{M,x}$. An jeder existentiellen Konfiguration wählt $\exists$ ein $C' \in \text{Next}(C)$, an jeder universellen Konfiguration wählt $\forall$ ein $C' \in \text{Next}(C)$. Sobald eine akzeptierende Konfiguration erreicht wird, hat $\exists$ gewonnen, bei Erreichen einer verwerfenden Konfiguration gewinnt $\forall$. Offensichtlich gilt: M akzeptiert x genau dann, wenn $\exists$ eine Gewinnstrategie für das Spiel auf $\mathfrak{T}_{M,x}$ hat.

Zeit- und Platzkomplexität sind wie bei einer NTM definiert. Es kommt jedoch ein zusätzliches Komplexitätsmaß hinzu: die Alternationen. M macht höchstens $A(n)$ Alternationen, wenn auf keinem Berechnungspfad mehr als $A(n)$ Wechsel von existentiellen zu universellen Konfigurationen (oder umgekehrt) stattfinden. Daraus ergeben sich alternierende Komplexitätsklassen:

(i) $\text{ATIME}(T)$ enthält alle Sprachen L, für die es eine T-zeitbeschränkte ATM M gibt, mit $L = L(M)$,

(ii) $\text{ASPACE}(S)$ enthält alle Sprachen L, für die es eine S-platzbeschränkte ATM M gibt, mit $L = L(M)$,

(iii) $\text{ATIME-ALT}(T, A)$ ist die Menge aller Spachen L, für die es eine T-zeitbeschränkte ATM M mit $L = L(M)$ gibt, welche weniger als $A(n)$ Alternationen macht.

Von besonderem Interesse sind die Klassen:

- $\text{ALOGSPACE} = \bigcup_{d\in\mathbb{N}} \text{ASPACE}(d \cdot \log n)$,
- $\text{APTIME} = \bigcup_{d\in\mathbb{N}} \text{ATIME}(n^d)$,
- $\text{APSPACE} = \bigcup_{d\in\mathbb{N}} \text{ASPACE}(n^d)$.

Beispiel. $\text{QBF} \in \text{ATIME}(O(n))$. Ohne Einschränkung können wir annehmen, dass die Negation nur in Literalen auftritt. Ein alternierende Version von $\text{Eval}(\psi, \mathfrak{I})$ würde nun folgendermaßen aussehen:

Eingabe : $(\psi, \mathfrak{I})$ wobei $\psi \in \text{QBF}$ und $\mathfrak{I}$: $\text{Frei}(\psi) \to \{0,1\}$
if $\psi = Y$ **then**
if $\mathfrak{I}(Y) = 1$ **then** akzeptiere
else verwerfe
if $\psi = \varphi_1 \vee \varphi_2$ **then** "$\exists$" errät $i \in \{1,2\}$, $\text{Eval}(\varphi_i, \mathfrak{I})$
if $\psi = \varphi_1 \wedge \varphi_2$ **then** "$\forall$" wählt $i \in \{1,2\}$, $\text{Eval}(\varphi_i, \mathfrak{I})$
if $\psi = \exists X\varphi$ **then** "$\exists$" errät $j \in \{0,1\}$, $\text{Eval}(\varphi, \mathfrak{I}[X = j])$
if $\psi = \forall X\varphi$ **then** "$\forall$" wählt $j \in \{0,1\}$, $\text{Eval}(\varphi, \mathfrak{I}[X = j])$

Satz 5.2. *Sei $S(n)$ platzkonstruierbar mit $S(n) \geq n$. Dann ist*

$$\text{NSPACE}(S) \subseteq \text{ATIME}(S^2).$$

Beweis. Der Beweis beruht auf einer ähnlichen Idee, wie der Beweis des Satzes von Savitch: L werde von einer nichtdeterministischen TM M mit einem Platzbedarf von $S(n)$ und in der Zeit $2^{c\cdot S(n)}$ entschieden. Sei $\text{Conf}[S(n)]$ die Menge der Konfigurationen von M mit Platzverbrauch $\leq S(n)$. Folgender alternierender Algorithmus $\text{Reach}(C_1, C_2, t)$ entscheidet, ob die Konfiguration $C_2 \in \text{Conf}[S(n)]$ von der Konfiguration $C_1 \in \text{Conf}[S(n)]$ in höchstens 2^t Schritten erreichbar ist:

Input C_1, C_2, t
if $t = 0$ **do**
if $C_1 = C_2$ oder $C_2 \in \text{Next}(C_1)$ **then** akzeptiere

else verwerfe
if $t > 0$ **do**
existentieller Schritt: errate $C \in \mathrm{Conf}[S(n)]$
universeller Schritt: wähle $(D_1, D_2) = (C_1, C)$ oder $(D_1, D_2) = (C, C_2)$
$\mathrm{Reach}(D_1, D_2, t-1)$

Der Algorithmus ist korrekt, denn C_2 ist von C_1 aus in $\leq 2^t$ Schritten erreichbar, wenn ein C existiert, so dass $\mathrm{Reach}(C_1, C, t-1)$ und $\mathrm{Reach}(C, C_2, t-1)$ akzeptieren.

Sei $f(t) = \max_{C_1, C_2 \in \mathrm{Conf}[S(n)]} \mathrm{time}_{\mathrm{Reach}}(C_1, C_2, t)$. Es gilt: $f(0) = O(S(n))$ und für $t > 0$ gilt $f(t) = O(S(n)) + f(t)$. Also ist

$$f(t) = (t+1) \cdot O(S(n)).$$

L wird nun wie folgt entschieden: Für einen Input x wird zunächst die Inputkonfiguration C_0 von M auf x konstruiert. Dann wird eine akzeptierende Endkonfiguration C_a von M erraten. Akzeptiert wird, falls $\mathrm{Reach}(C_0, C_a, S(n))$ akzeptiert.
Dieser Algorithmus benötigt

$$(c \cdot S(n) + 1)O(S(n)) = O(S^2(n))$$

Schritte. Aufgrund des linearen Speed-Up Theorems, daß auch für alternierende TM gilt, ist $L \in \mathrm{ATIME}(S^2)$. □

Satz 5.3. *Sei T platzkonstruierbar und $T(n) \geq n$. Dann gilt:* $\mathrm{ATIME}(T) \subseteq \mathrm{DSPACE}(T^2)$.

Beweis. Sei $L \in \mathrm{ATIME}(T)$ und M eine alternierende TM, welche L mit Zeitbedarf $T(n)$ akzeptiert.
Es gibt nun ein r, so dass für alle Konfigurationen C von M gilt: $|\mathrm{Next}(C)| \leq r$. Der deterministische Algorithmus F berechnet zu jeder Konfiguration C in $\mathfrak{T}_{M,x}$, ob der Unterbaum T_C akzeptierend (Antwort: 1) oder verwerfend (Antwort: 0) ist.

Input : C
if C akzeptierend **then output** 1 **end**
if C verwerfend **then output** 0 **end**
if C existentiell **then**
for $i = 1, \cdots, r$ **do**
berechne i-te Nachfolgekonfiguration C_i von C
if $F(C_i) = 1$ **then output** 1 **end**
od
output 0 **end**
if C universell **then**
for $i = 1, \cdots, r$ **do**
berechne i-te Nachfolgekonfiguration C_i von C
if $F(C_i) = 0$ **then output** 0 **end**
od
output 1 **end**

Offensichtlich arbeitet der Algorithmus korrekt. $F(C_0(x))$ entscheidet, ob M x akzeptiert, ist also ein deterministisches Entscheidungsverfahren für L.

Wieviel Speicherplatz braucht der Algorithmus ?

Sei C ein Knoten in $\mathfrak{T}_{M,x}$ der Höhe t, d.h. alle von C ausgehenden Berechnungen von M dauern höchstens t Schritte. Dann gilt:

$$\text{space}_F(C) = \begin{cases} 0 & \text{wenn } t = 0 \\ \max_{C_i \in \text{Next}(C)}(|C_i| + \text{space}_F(C_i)) & \text{wenn } t > 0 \end{cases}$$

Da $C_i \in \text{Next}(C)$ Höhe $t-1$ hat, folgt: $\text{space}_F(C) \leq t \cdot T(n)$, also $\text{space}_F(C_o) \leq T^2(n)$. □

Damit folgt

Satz 5.4. *(Parallele Zeitkomplexität = sequentielle Platzkomplexität)*

- APTIME = PSPACE.
- AEXPTIME = EXPSPACE.

Beweis.

- $\text{ATIME}(n^d) \subseteq \text{DSPACE}(n^{2d}) \subseteq \text{PSPACE}$,
  $\text{DSPACE}(n^d) \subseteq \text{NSPACE}(n^d) \subseteq \text{ATIME}(n^{2d}) \subseteq \text{APTIME}$.
- $\text{ATIME}(2^{n^d}) \subseteq \text{DSPACE}(2^{2n^d}) \subseteq \text{EXPSPACE}$,
  $\text{DSPACE}(2^{n^d}) \subseteq \text{ATIME}(2^{2n^d}) \subseteq \text{AEXPTIME}$.

□

Satz 5.5. *Sei $T(n) \geq n$ und $c > 0$. Dann ist* $\text{DTIME}(T) \subseteq \text{ASPACE}(c \cdot \log T)$.

Beweis. Sei $L \in \text{DTIME}(T)$. Dann gibt es eine deterministische 1-Band-TM $M = (Q, \Sigma, q_0, F, \delta)$, welche L in der Zeit T^2 entscheidet. Sei $\Gamma = \Sigma \cup (Q \times \Sigma) \cup \{*\}$ und $t = T^2(n)$.
Jede Konfiguration $C = (q, i, w)$ (in einer Berechnung auf einen Input der Länge n) kann beschrieben werden durch ein Wort

$$\underline{c} = *w_0 \cdots w_{i-1}(qw_i)w_{i+1} \cdots w_t * \in \Gamma^{t+2}.$$

Das i-te Symbol der Nachfolgekonfiguration hängt nur von den Symbolen ab, welche an den Positionen $i-1$, i und $i+1$ stehen. Also gibt es eine Funktion $f_M : \Gamma^3 \to \Gamma$, so dass gilt:
Wenn eine Konfiguration $\underline{c}$ an den Positionen $i-1$, i und $i+1$ die Symbole a_{-1}, a_0 und a_1 hat, dann hat die Nachfolgekonfiguration $\underline{c'}$ an Position i das Symbol $f_M(a_{-1}, a_0, a_1)$.

Wir geben einen alternierende Algorithmus A an, welcher entscheidet, ob $x \in L(M)$:

Existentieller Schritt: Errate $s \leq T^2(n) = t$,
errate $(q^+a) \in Q^+_{acc} \times \Sigma$, $i \in \{0, \ldots, s\}$
$b := (q^+a)$
for $j = 1 \ldots s$ **do**
  Existentieller Schritt: Errate $a_{-1}, a_0, a_1 \in \Gamma^3$
  prüfe, ob $f_M(a_{-1}, a_0, a_1) = b$

wenn nein, dann verwerfe.
Universeller Schritt: Wähle $k \in \{-1, 0, 1\}$
$b := a_k$
$i := i + k$
od
Prüfe, ob i-tes Symbol der Inputkonfiguration von M auf x gleich b ist.
Wenn ja, akzeptiere, andernfalls verwerfe.

Der Algorithmus A hat einen Platzbedarf von $O(\log T(n))$. Falls M den Input x akzeptiert, hat $\exists$ folgende Gewinnstrategie auf $\mathfrak{T}_{M,x}$:
Die Zeit, nach der M die Eingabe x akzeptiert, wird mit s bezeichnet. $(q^+a), i$ wird so gewählt, dass die Konfiguration von M zur Zeit s die Gestalt

$$*w_0 \cdots w_{i-1}(q^+a)w_{i+1} \cdots w_t*$$

hat. Beim j-ten Durchlauf der Schleife (also bei Konfiguration $s - j$) werden für a_{-1}, a_0, a_1 die Symbole gewählt, die an den Positionen $i-1, i, i+1$ der Konfiguration von M zur Zeit $s-j$ stehen.

Falls M den Input x nicht akzeptiert, ist das i-te Symbol der Konfiguration zur Zeit s nicht (q^+a). Für alle j gilt dann:
Bei jeder Wahl von $\exists$ für a_{-1}, a_0, a_1 ist immer entweder $f(a_{-1}, a_0, a_1) \neq b$ oder es gibt ein $k \in \{-1, 0, 1\}$, für welches das $(i + k)$-te Symbol der Konfiguration $s - j$ verschieden von a_k ist.
$\forall$ wählt dann eben dieses k.
Letztlich ist dann a_k verschieden vom i-ten Symbol der Inputkonfiguration. □

Satz 5.6. *Für $S(n) \geq \log n$ gilt:* $\text{ASPACE}(S) \subseteq \bigcup_{c>0} \text{DTIME}(2^{c \cdot S})$.

Beweis. Sei $L \in \text{ASPACE}(S)$ und M eine alternierende TM, welche L mit Platz $S(n)$ entscheidet. Dann gibt es eine Konstante d, so dass es höchstens $2^{dS(n)}$ verschiedene Konfigurationen von M mit einem Platzbedarf von $S(n)$ gibt. Jede dieser Konfigurationen ist beschreibbar durch ein Wort der Länge $S(n)$.
Eine deterministische Maschine M_0 konstruiert zuerst einen gewichteten Graphen, dessen Knoten die Konfigurationen von M auf x sind.

Der Algorithmus zu M lautet:

Konstruiere die Inputkonfiguration C_0 von M auf x
Setze $P = \{C_0\}$, $L = \varnothing$, $S = \varnothing$, $Q = \varnothing$
while $Q \neq P$ **do**
  nimm $C \in P - Q$
  if C akzeptierend **then** setze $F(C) = 1$, $L = L \cup \{C\}$
  if C verwerfend **then** setze $F(C) = 0$, $L = L \cup \{C\}$
    else konstruiere Nachfolgekonfigurationen $C'_1, \cdots C'_r$ von C
  $P = P \cup \{C'_1\}$, $E = E \cup \{(C, C'_i) \mid i = 1, \cdots, r\}$, $Q = Q \cup \{C\}$
  od
while $C_0 \notin L$ **do**
  for all $C \in P - L$ **do**
  if (C existentiell und $\exists C' \in L$ mit $(C, C') \in E \wedge F(C') = 1$)
    setze $F(C) = 1$, $L = L \cup \{C\}$

if (C existentiell und $\forall C' : (C, C') \in E \Rightarrow C' \in l \wedge F(C') = 0$)
setze $F(C) = 0$, $L = L \cup \{C\}$
if (C universell und $\exists C' : (C, C') \in E \wedge C' \in L \wedge F(C') = 0$)
setze $F(C)$, $L = L \cup \{C\}$
if (C universell und $\forall C' : (C, C') \in E \Rightarrow C' \in L \wedge F(C') = 1$)
setze $F(C) = 1$, $L = L \cup \{C\}$
od
if $F(C_0) = 1$ **then** akzeptiere, **else** verwerfe.

□

Korollar 5.7. *Es gilt:*

(i) ALOGSPACE $= P$

(ii) APSPACE $=$ EXPTIME

Letztlich gelten also die folgenden Beziehungen zwischen Komplexitätsklassen:

$$\begin{array}{ccccccccccc}
\text{LOGSPACE} & \subseteq & \text{P} & \subseteq & \text{PSPACE} & \subseteq & \text{EXPTIME} & \subseteq & \text{EXPSPACE} & \subseteq & \ldots \\
 & & \| & & \| & & \| & & \| & & \\
 & & \text{ALOGSPACE} & \subseteq & \text{APTIME} & \subseteq & \text{APSPACE} & \subseteq & \text{AEXPTIME} & \subseteq & \ldots
\end{array}$$

Kapitel 6

Parallele Berechnungsmodelle II : Schaltkreise

6.1 Schaltkreiskomplexität

Eine (n-stellige) Boolesche Funktion ist eine Funktion $f : \{0,1\}^n \to \{0,1\}$. Sei B^n die Menge aller n-stelligen Booleschen Funktionen und $B = \bigcup_{n\in\mathbb{N}} B^n$. B^0 enthält die konstanten Funktionen 0 und 1. B^1 enthält vier Funktionen $f_{00}, f_{01}, f_{10}, f_{11}$ mit

$$f_{00}(0) = f_{00}(1) = 0, \qquad f_{11}(0) = f_{11}(1) = 1,$$
$$f_{01}(x) = x, \qquad f_{10}(x) = 1 - x.$$

B^n enthält 2^{2^n} verschiedene Funktionen.

Definition 6.1. Sei $\Omega \subseteq B$. Ein Schaltkreis über Ω mit n Inputs und m Outputs ist gegeben durch:

(1) Einen gerichteten azyklischen Graphen $C = (P, E)$ mit $P = \{X_1, \ldots, X_n\} \cup \{g_1, \ldots, g_r\}$. Die Knoten $X_1, \ldots, X_n$ sind die Inputknoten. Sie haben keine eingehenden Kanten (ingrad$(X_i) = 0$). Die Knoten $g_1, \ldots, g_r$ heissen interne Knoten.

(2) Eine Funktion $\omega : \{g_1, \ldots, g_r\} \to \Omega$, welche für jeden internen Knoten angibt, welche Boolesche Funktion dort berechnet wird. Wenn $\omega(g) \in B^k$, dann hat g genau k einfallende Kanten, welche mit $1, \ldots, k$ beschriftet sind. Der Anfangspunkt der j-ten Kante nach g heisst j-ter Vorgänger $p_j(g)$ von g.

(3) Eine Funktion $o : \{1, \ldots, m\} \to P$, welche die Knoten $o(1), \ldots, o(m)$ als Outputknoten auszeichnet.

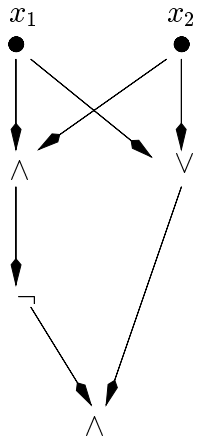
An jedem Knoten $g \in P$ wird eine Funktion $res_g \in B^n$ berechnet. Für alle $x = (x_1, \ldots, x_n) \in \{0,1\}^n$ ist

- $res_{X_i}(x) := x_i$
- $res_{g_i}(x) := \omega(g_i)(res_{p_1(g_i)}(x), \ldots, res_{p_m(g_i)}(x))$

Die von C berechnete Funktion ist das m-Tupel der an den output-Knoten berechneten Funktion: $f_C = (res_{o(1)}, \ldots, res_{o(m)}) : \{0,1\}^n \to \{0,1\}^m$.

Wir werden vor allem Schaltkreise mit nur einem Output betrachten. Wenn nichts anderes explizit festgelegt ist, gilt im folgenden also $m = 1$.

Beispiel. $\Omega = \{\wedge, \vee, \neg\}$ (Standardbasis)



Komplexitätsmaße für Schaltkreise

- Die Grösse von C, bezeichnet durch $|C|$, ist die Anzahl der internen Knoten von C.
- Die Tiefe von C, kurz $D(C)$, ist die Länge des längsten Pfades in C.

Der Schaltkreis im oben angegebenen Bespiel hat Grösse $|C| = 4$ und Tiefe $D(C) = 3$.

Definition 6.2. Die *Schaltkreiskomplexität* einer Boolschen Funktion $f \in B$ bzgl. Ω ist

$$C_\Omega(f) := \min\{|C| \ : \ C \text{ Schaltkreis über } \Omega \text{ mit } f_C = f\}.$$

Analog ist $D_\Omega(f) = \min\{D(C) \ : \ C \text{ Schaltkreis über } \Omega \text{ mit } f_C = f\}$.

Eine Menge Ω von Booleschen Funktionen ist *funktional vollständig*, wenn für jedes $f \in B$ ein Schaltkreis C über Ω existiert, so dass $f_C = f$.

Beispiel. $\{\wedge, \vee, \neg\}$, $\{\wedge, \oplus, 1\}$ und $\{\text{NAND}\}$ sind funktional vollständig.

Satz 6.3. *Sei Ω endlich und Ω' funktional vollständig. Dann gibt es Konstanten c, d, so dass für alle $f \in B$*

- $C_{\Omega'}(f) \leq c \cdot C_\Omega(f)$
- $D_{\Omega'}(f) \leq d \cdot D_\Omega(f)$

Beweis. Sei $c = \max\{C_{\Omega'}(w) \ : \ w \in \Omega\}, d = \max\{D_{\Omega'}(w) \ : \ w \in \Omega\}$. In jedem Schaltkreis C über Ω können die Ω-Knoten durch Schaltkreise der Grösse $\leq c$ bzw. der Tiefe $\leq d$ über Ω' ersetzt werden. Dies ergibt Schaltkreise über Ω', der Grösse $\leq c \cdot |C|$ bzw. der Tiefe $\leq d \cdot D(C)$, die dieselbe Funktion wie C berechnen. □

Wir zeigen nun, dass es Boolesche Funktionen mit exponentieller Schaltkreiskomplexität gibt.

Satz 6.4. *Sei $\Omega \subseteq B^2$. Für hinreichend grosse n gibt es Funktionen $f \in B_n$, so dass*

$$C_\Omega(f) > \frac{2^n}{n}.$$

Wir beschränken uns auf eine Skizze des Beweises. Es gibt 2^{2^n} Funktionen in B^n. Sei $F(n,r)$ die Anzahl der Booleschen Funktionen $f \in B^n$, welche durch Schaltkreise der Grösse r über $\Omega = B^2$ berechnet werden können.

Lemma 6.5.

$$F(n,r) \leq S(n,r) := \frac{((r+n-1)^{2r} \cdot 16^r \cdot r)}{r!}.$$

Beweis. In B^2 gibt es $2^{2^2} = 16$ Funktionen. In einem Schaltkreis der Grösse r mit n Inputs gibt es für jeden inneren Knoten höchstens $16(r+n-1)^2$ Möglichkeiten, seinen Typ und seine beiden Vorgänger festzulegen. Also gibt es nicht mehr als $16^r(r+n-1)^{2r}$ Schaltkreise mit n Inputs und r inneren Knoten. Jeder Schaltkreis berechnet r Funktionen. Jede Funktion ist aber jetzt $r!$ Mal gezählt worden, nämlich für alle $r!$ Umbenennungen von $g_1, \ldots, g_r$. □

Annahme: Sei $r \leq 2^n/n$ und $C_\Omega(f) \leq r$ für alle $f \in B^n$. Also ist $F(n,r) = 2^{2^n}$ und daher $\log S(n,r) \geq \log F(n,r) = 2^n$ für ein $r \leq 2^n/n$.

Eine langwierige (wenn auch nicht schwierige) Rechnung zeigt aber, dass

$$\log S(n, \frac{2^n}{n}) \leq 2^n - \frac{2^n}{n} \log\log n < 2^n$$

für alle hinreichend grossen n. Insbesondere gilt also für $r \leq 2^n/n$, dass $\log S(n,r) < 2^n$, im Widerspruch zur Annahme.

Damit ist Satz 6.4 bewiesen. Aus der Abschätzung für $\log S(n, 2^n/n)$ erhalten wir aber noch eine stärkere Aussage:

$$S(n, \frac{2^n}{n}) \leq 2^{2^n} \cdot \frac{1}{2^{\frac{2^n}{n}\log\log n}}.$$

Für $r(n) := \frac{2^n}{n}\log\log n$ gilt also $S(n, \frac{2^n}{n}) \leq |B_n| \cdot 2^{-r(n)}$. Es folgt also, dass sogar *fast alle* Boolsche Funktionen exponentielle Schaltkreiskomplexiät haben.

Satz 6.6. *Für hinreichend grosse n gilt: Mindestens $|B^n|(1-2^{-r(n)})$ Funktionen in B_n haben Schaltkreiskomplexität grösser als $2^n/n$.*

Leider gibt uns dieses Zählargument keine explizit beschriebene Funktion mit exponentieller Schaltkreiskomplexität in die Hand. Für explizite, natürliche Funktionen sind keine nichtlinearen Schranken für $C_\Omega(f)$ bekannt.

Schaltkreiskomplexität von Sprachen $A \subseteq \{0,1\}^*$**.** Sei $A \subseteq \{0,1\}^*$. Daraus ergibt sich eine Familie $(F_{A,n})_{n\in\mathbb{N}}$ von Boolschen Funktionen. Dabei ist $F_{A,n} \in B^n$ die charakteristische Funktion von $A \cap \{0,1\}^n$:

$$F_{A,n}(x) := \begin{cases} 1 & \text{wenn } x \in A \\ 0 & \text{wenn } x \notin A. \end{cases}$$

Definition 6.7. Die *Schaltkreiskomplexität* einer Sprache $A \subseteq \{0,1\}^n$ (über Ω) ist die Funktion $C_\Omega^A : \mathbb{N} \to \mathbb{N}$ mit $C_\Omega^A(n) = C_\Omega(F_{A,n})$.

Die Aussage, A habe Schaltkreiskomplexität $\leq C_\Omega^A$, bedeutet demnach, dass eine Familie $(C_n)_{n\in\mathbb{N}}$ von Schaltkreisen über Ω existiert, so dass $|C_n| \leq C_\Omega^A(n)$ und $f_{C_n} = F_{A,n}$.

Satz 6.8. *Sei Ω funktional vollständig. Jedes Problem $A \in \mathrm{P}$, $A \subseteq \{0,1\}^*$, hat polynomiale Schaltkreiskomplexität.*

Beweis. Sei M eine deterministische 1-Band-Turingmaschine mit Zustandsmenge Q und Alphabet Σ, welche A in Zeit n^r entscheidet. Ohne Einschränkung macht M auf Inputs der Länge n genau $m := n^r$ Schritte. Wie schon in früheren Beweisen kodieren wir eine Konfiguration $C = (q, i, w)$ durch das Wort $*w_0 \dots w_{i-1}(qw_i)w_{i+1} \dots w_m*$ über dem Alphabet $\Gamma := \Sigma \cup (Q \times \Sigma) \cup \{*\}$. Es gibt eine Funktion $f : \Gamma^3 \to \Gamma$, welche das Übergangsverhalten beschreibt: Wenn in der Konfiguration C an den Positionen $i-1, i, i+1$ die Symbole a, b, c stehen, dann steht in der Nachfolgekonfiguration an Position i das Symbol $f(a,b,c)$. Sei $|\Gamma| = d$. Jedes Symbol von Γ kann binär durch ein Wort der Länge $s = \lceil \log d \rceil$ kodiert werden. Jede Konfiguration wird jetzt beschrieben durch ein Wort aus $\{0,1\}^{s \cdot m}$ und f kann als Funktion von $\{0,1\}^{3s}$ nach $\{0,1\}^s$ aufgefasst werden. Sei F ein Schaltkreis mit $3s$ Inputs und s Outputs, der f berechnet. (Da s eine Konstante ist, ist auch $|F|$ konstant.)

Ausserdem gibt es Schaltkreise G_i mit n Inputs und s Outputs, welche das i-te Symbol der Inputkonfiguration von M auf $x_0 \dots x_{n-1}$ berechnen und schließlich einen Schaltkreis B mit $s \cdot m$ Inputs und einem Output, so dass $f(y) = 1$ genau dann, wenn y eine akzeptierende Konfiguration kodiert.

Damit kann nun ein Schaltkreis C_n zusammengesetzt werden, der das Gewünschte leistet. Dieser Schaltkreis hat $O(n^{2r})$ Knoten. □

Beachte: C_n ist mit Platz $O(\log n)$ konstruierbar.

Die Umkehrung von Satz 6.8 gilt aber nicht:

Satz 6.9. *Seien $0, 1 \in \Omega$. Es gibt unentscheidbare Mengen $A \subseteq \{0,1\}^*$ mit $C_\Omega^A(n) = 1$.*

Beweis. Sei $B \subseteq \mathbb{N}$ unentscheidbar. Dann ist auch $L_B = \{x \in \{0,1\}^* \ : \ |x| \in B\}$ unentscheidbar.

Für jedes n sei C_n einer der beiden trivialen Schaltkreise, welche die konstanten Funktionen 0 bzw. 1 berechnen.

$n \in B \Rightarrow C_n$ ist der Schaltkreis der Grösse 1 mit $f_{C_n} = 1$.
$n \notin B \Rightarrow C_n$ ist der Schaltkreis der Grösse 1 mit $f_{C_n} = 0$.

Offensichtlich entscheidet $(C_n)_{n \in \mathbb{N}}$ das Problem L_B und $|C_n| = 1$. Also ist $C_\Omega^{L_B} = 1$. □

Das Problem beim soeben geführten Beweis ist, dass C_n aus n nicht effektiv bestimmt werden kann. Wir wissen, dass C_n einer von zwei möglichen, sehr kleinen Schaltkreisen ist, aber wir können nicht entscheiden, welcher. Normalerweise verlangt man deshalb *Uniformitätseigenschaften* von Folgen $(C_n)_{n \in \mathbb{N}}$ von Schaltkreisen.

Definition 6.10. Eine Familie $(C_n)_{n \in N}$ von Schaltkreisen heisst *p-uniform* bzw. *log-uniform*, wenn ein Algorithmus existiert, welcher für jeden Input $n \in \mathbb{N}$ (eine Kodierung von) C_n in Zeit $O(|C_n|^{O(1)})$ bzw. mit Platz $O(\log |C_n|))$ berechnet.

Definition 6.11. Das *Circuit Value Problem* ist

$$\mathrm{CVP} := \{(C, x) \ : \ C \text{ ein Schaltkreis}, x \in \{0,1\}^* \text{ Input für } C \text{ mit } f_C(x) = 1\}.$$

Satz 6.12. *CVP ist P-vollständig via logspace-Reduktion.*

Beweis. C habe die internen Knoten $g_1, \ldots, g_r$, welche so nummeriert seien, dass die Vorgänger von g_j entweder Inputknoten oder innere Knoten g_i mit $i < j$ sind. Damit können der Reihe nach, für gegebenes $x \in \{0,1\}^*$ die Werte $res_{g_j}(x)$ und damit auch $f_C(x)$ berechnet werden. Also ist CVP in P.

Sei nun A ein beliebiges Problem in P. Wir können annehmen, dass $A \subseteq \{0,1\}^*$. Sei M eine deterministische 1-Band TM, welche A entscheidet. Im Beweis von Satz 6.8 wurde gezeigt, dass aus $x \in \{0,1\}^n$ durch einen logspace-Algorithmus ein Schaltkreis C_n konstruiert werden kann, mit $x \in A$ genau dann, wenn $f_{C_n}(x) = 1$. Also ist $x \mapsto (C_n, x)$ eine logspace-Reduktion von A auf CVP. □

Satz 6.13. *Sei $A \subseteq \{0,1\}^*$. Wenn eine p-uniforme Familie $(C_n)_{n\in\mathbb{N}}$ von Schaltkreisen polynomialer Grösse existiert, so dass $f_{C_n} = F_{A,n}$, (so dass also $(C_n)_{n\in\mathbb{N}}$ die Menge A entscheidet), dann ist $A \in P$.*

Beweis. Gegeben $x \in \{0,1\}^n$, konstruiere in polynomialer Zeit den Schaltkreis C_n und entscheide, ob $(C_n, x) \in$ CVP. □

6.2 Die Klasse NC

Ein Schaltkreis C der Grösse $|C|$ und mit Tiefe $D(C)$ kann aufgefaßt werden als paralleles Berechnungsmodell für f_C, wobei $|C|$ dem Hardwareaufwand und $D(C)$ der Rechenzeit entspricht. Besonders interessant ist in diesem Zusammenhang die Frage, welche Probleme aus P *effizient parallelisierbar* sind. "Effizient" bedeutet hier, mit polynomialem Hardwareaufwand und polylogarithmischer Rechenzeit. Im Folgenden sei $\Omega = B^2$.

Definition 6.14. Eine Familie $(f_n)_{n\in\mathbb{N}}$ von Boolschen Funktionen (mit $f_n \in B^n$) ist in NC^i, wenn es eine log-uniforme Familie $(C_n)_{n\in\mathbb{N}}$ von Schaltkreisen über Ω gibt, mit:

- $|C_n| = O(n^k)$ für ein $k \in \mathbb{N}$,
- $D(C_n) = O(\log^i n)$,
- $f_{C_n} = f_n$.

Eine Sprache $A \subseteq \{0,1\}^*$ ist in NC^i wenn die Familie $(F_{A,n})_{n\in\mathbb{N}} \in \mathrm{NC}^i$ ist. Schliesslich sei

$$\mathrm{NC} := \bigcup_{i\in\mathbb{N}} \mathrm{NC}^i.$$

(Eingeführt wurde die Klasse NC von S. Cook und zu Ehren von Nicolas Pippenger "Nick's Class" (NC) genannt.)

Offensichtlich ist $\mathrm{NC} \subseteq$ P.

Satz 6.15. *Für alle $i \geq 1$ gilt:* $\mathrm{NC}^i \subseteq \textsc{Dspace}(O(\log^i n))$.

Beweis. Sei $A \in \mathrm{NC}^i$ und $(C_n)_{n\in\mathbb{N}}$ eine log-uniforme Familie von Schaltkreisen der Größe $p(n)$ und der Tiefe $O(\log^i n)$, welche A entscheidet. O.B.d.A. kann man annehmen, dass die C_n Schaltkreise über {NAND} sind, da {NAND} funktional vollständig ist. Für jeden internen Knoten g von C_n bezeichen wir mit $p(g)$ und $q(g)$ die Vorgänger von g. Es ist

$$res_g(x) = \neg(res_{p(g)}(x) \wedge res_{q(g)}(x)).$$

Der Schaltkreis soll nun durch folgenden deterministischen $O(\log^i n)$-platzbeschränkten Algorithmus ausgewertet werden: Betrachte Pfade rückwärts durch den Schaltkreis, ausgehend vom Output-Knoten. Jedem solchen Pfad P wird ein Wort $u(P) \in \{\ell, r\}^*$ wie folgt zugeordnet:

$$u(\text{output}) := \lambda, \qquad u(Pp(g)) := u(P)\ell, \qquad u(Pq(g)) = u(P)r,$$

wobei $Pq(g)$ den Pfad symbolisiert, der über den Pfad P zum Knoten g führt, und von dort zum linken Vorgänger $p(g)$. Es ist durchaus möglich, dass es mehrere Pfade vom Output zum gleichen Knoten g gibt. Für $u \in \{\ell, r\}^*$ sei g^u der Endknoten des zu u gehörenden Pfades. Aufgrund der log-Uniformität von $(C_n)_{n\in\mathbb{N}}$ kann g^u aus u und n mit einem Platzbedarf von $O(\log n)$ bestimmt werden, ebenso wie seine Vorgänger $p(g^u) = g^{u\ell}$ und $q(g^u) = g^{ur}$.

Die Auswertung von C_n geht nun folgendermassen vonstatten: Sei P ein Pfad vom Output zum Knoten g. Wenn $y = res_g(x)$ schon ausgewertet ist, setzt man $v(P) = u(P)y$. Falls $res_g(x)$ noch nicht ausgewertet wurde, so setzt man $v(P) = u(P)$.

Der Algorithmus startet mit dem Output-Knoten, also mit $v = \lambda$. Wenn der aktuelle Knoten v noch nicht ausgewertet wurde, geht man so weit nach links ($v = v\ell$), bis ein Inputknoten erreicht ist, der dann ausgewertet wird. Wenn $v\ell$ zu 0 ausgewertet ist, kann man v zu 1 auswerten ($v\ell 0 \mapsto v1$). Hat $v\ell$ aber den Wert 1, dann geht man nach vr. Der Knoten vr wird nur ausgewertet, wenn $v\ell 1$ gilt. Ist dann vr auch ausgewertet, kann schließlich v ausgewertet werden, und zwar durch $vr0 \mapsto v1$ oder $vr1 \mapsto v0$. Der Algorithmus lautet:

Input: x $\quad(|x| = n)$
$v = \lambda$
while $(v \neq 0 \wedge v \neq 1)$ **do**
  if $v = v'l0 \vee v = v'r0$ **then** $v := v'1$
  if $v = v'l1$ **then** $v := v'r$
  if $v = v'r1$ **then** $v := v'0$
  if $v = 0$ **then** verwerfe
  if $v = 1$ **then** akzeptiere
  if v endet mit r oder l **then do**
    konstruiere $g = g^v$
    if $g = X_i$ **then** $v := vx_i$
    if g interner Knoten **then** $v := v\ell$
    od
  od

Der Platzbedarf des Algorithmus ist

$$|v| + |g| + O(\log n) \leq O(\log^i n) + O(\log n) = O(\log^i n).$$

□

Aus dem soeben bewiesenen Satz und dem Platzhierarchiesatz folgt:

Korollar 6.16. NC $\subsetneq$ PSPACE.

6.3 Schaltkreise und alternierende TM

Für parallele Algorithmen ergeben auch Zeitschranken kleiner als n einen Sinn (siehe NC^i). Auch für alternierende TM ist denkbar, dass nicht bei jeder Berechnung der gesamte Input gelesen werden muss. Das führt zu folgender Definition.

Definition 6.17. Eine ATM M mit Adressierungsmöglichkeit nutzt eines der Arbeitsbänder zur Adressierung von Feldern des Eingabebandes. Die Inschrift auf dem Adressband ist jeweils eine binär kodierte natürliche Zahl i. Der Lesekopf auf dem Eingabeband von M liest dann das Feld i. Wenn M den Inhalt des Adressbandes ändert, springt der Lesekopf des Eingabebandes auf die neue Adresse. Eine solche ATM kann mit logarithmischem Zeitvebrauch beliebige Bits des Inputs lesen.

Damit können jetzt auch für Funktionen T mit $T(n) < n$ die Komplexitätsklassen $\textsc{Atime}(T)$ sinnvoll definiert werden. Von Bedeutung ist insbesondere die Klasse

$$\textsc{Alogtime} := \bigcup_{d\in\mathbb{N}} \textsc{Atime}(d\log n).$$

Beispiel. Für $w = w_0 \cdots w_{n-1}$ sei $\hat{w} = w_{n-1} \cdots w_0$. Die Menge der *Palindrome* ist

$$\text{Pal} := \{w \in \{0,1\}^* : \hat{w} = w\} \in \textsc{Atime}(O(\log n)).$$

Ein alternierender Algorithmus, der entscheidet ob ein Wort in Pal liegt könnte dann etwa so funktionieren:

Eingabe: w
$\exists$: errate n
  if $(w_{n-1} \notin \{0,1\} \vee w_n \neq \square)$ verwerfe
$\forall$: wähle $i < n$
  berechne $j := n - i - 1$
  if $w_i = w_j$ **then** akzeptiere
    else verwefe

Satz 6.18. *Für alle $i \geq 2$ gilt:* $\text{NC}^i = \textsc{Aspace-time}(O(\log n), O(\log^i n))$.

Auf den Beweis des Satzes wird verzichtet.

Korollar 6.19. $\text{NC}^1 \subseteq \textsc{Logspace} \subseteq \textsc{Nlogspace} \subseteq \text{NC}^2$.

6.4 Schaltkreise mit unbeschränktem Fan-in

Die Konjunktion $\bigwedge_n$ und die Disjunktion $\bigvee_n$ von n Boolschen Werten sind Funktionen aus B^n. Ein *Schaltkreis mit unbeschränktem Fan-in* hat als Basis die Menge $\{\bigwedge_n, \bigvee_n : n \in \mathbb{N}\}$. Um auch nicht-monotone Funktionen mit solchen Schaltkreisen berechnen zu können, stehen neben $X_1, \cdots, X_n$ auch deren Negationen $\overline{X_1}, \ldots, \overline{X_n}$ als Inputknoten zur Verfügung.

Zum Beispiel kann eine aussagenlogische Formel in KNF

$$\bigwedge_{i=1}^{n} \bigvee_{j=1}^{m_i} Y_{i,j}$$

als Schaltkreis der Tiefe 2 mit unbeschränktem Fan-in aufgefasst werden.

Also kann jede Boolsche Funktion durch einen solchen Schaltkreis (allerdings im allgemeinen von exponentieller Grösse) berechnet werden.

Definition 6.20. Eine Familie $(f_n)_{n\in\mathbb{N}}$ von Boolschen Funktione (mit $f_n \in B^n$) ist in der Klasse AC^i (für $i \geq 0$) genau dann, wenn es eine log-uniforme Familie von Schaltkreisen mit unbeschränktem Fan-in gibt, so dass

- $|C_n| = O(n^k)$ für ein $k \in \mathbb{N}$,
- $D(C_n) = O(\log^i n)$,
- $f_{C_n} = f_n$.

Eine Sprache $A \subseteq \{0,1\}^*$ ist in AC^i genau dann wenn $(F_{A,n})_{n\in\mathbb{N}} \in \mathrm{AC}^i$.

Satz 6.21. *Für alle $i \geq 0$ gilt:* $\mathrm{AC}^i \subseteq \mathrm{NC}^{i+1} \subseteq \mathrm{AC}^{i+1}$.

Beweis. Es ist klar, dass $\mathrm{NC}^i \subseteq \mathrm{AC}^i$ für alle $i \geq 1$. Die andere Aussage, also $\mathrm{AC}^i \subseteq \mathrm{NC}^{i+1}$ folgt daraus, dass $\bigwedge$- und $\bigvee$-Knoten mit m Vorgängern durch Schaltkreise über $\wedge, \vee \in B^2$ mit höchstens m internen Knoten und mit Tiefe $\lfloor \log n \rfloor$ ersetzt werden können.

Also kann ein Schaltkreis mit unbeschränktem Fan-in der Grösse $O(n^k)$ und Tiefe $O(\log^i n)$ durch einen Schaltkreis über $\{\vee, \wedge, \neg\}$ der Grösse $O(n^{2k})$ und Tiefe $O(\log^{i+1} n)$ simuliert werden. □

AC^0 ist die Klasse der Funktionen, die mit Schaltkreisen polynomialer Grösse und konstanter Tiefe berechnet werden können.

Beispiel. Ein Beispiel für eine Funktion in AC^0 ist die Addition: Die Inputs x und y sind in Binärdarstellung gegeben, also

$$\mathrm{bin}(x) = x_{n-1} \qquad x_0 \in \{0,1\}^n \qquad x = \sum_{i=0}^{n-1} x_i 2^i,$$

ebenso für y.

Der Schaltkreis mit den Inputknoten $X_{n-1}, \ldots, X_0$ und $Y_{n-1}, \ldots, Y_0$, und den Outputknoten $Z_n, \ldots, Z_0$ beruht nun auf der carry-look-ahead Methode:

$$z_i = x_i \oplus y_i \oplus \bigvee_{j<i} ((x_j \wedge y_j \wedge \bigwedge_{j<k<i} x_k \vee y_k)).$$

AC^0 ist die einzige dieser Klassen, von der bisher gezeigt werden konnte, dass sie echt in P enthalten ist. Ein Beispiel einer Funktionenfamilie die in NC^1 aber nicht in AC^0 liegt, ist die Parität einer Bitfolge:

$$\textsc{Parity} := (\oplus_n)_{n\in\mathbb{N}} \text{ wobei } \oplus_n (x_1, \cdots, x_n) := \sum_{i=1}^{n} x_i \mod 2.$$

Satz 6.22. $\textsc{Parity} \notin \mathrm{AC}^0$.

Insbesondere gilt $\mathrm{AC}^0 \subsetneq \mathrm{NC}^1$. Es kann nun die folgende Hierarchie von Komplexitätsklassen erstellt werden:

$$\begin{aligned}\mathrm{AC}^0 \subsetneq \mathrm{NC}^1 \subseteq \textsc{Logspace} \subseteq \textsc{Nlogspace} \subseteq \mathrm{AC}^1 \subseteq \mathrm{NC}^2 \subseteq \cdots \subseteq \mathrm{NC} \\ \subseteq \mathrm{P} \subseteq \mathrm{NP} \subseteq \Sigma_2^p \cdots \subseteq \mathrm{PH} \subseteq \textsc{Pspace}.\end{aligned}$$

Kapitel 7

Komplexitätstheorie für Optimierungsprobleme

7.1 NP-Optimierungsprobleme

Definition 7.1. Ein *Optimierungsproblem* ist gegeben durch ein Quadrupel $Q = (I, S, w, opt)$ bestehend aus folgenden Objekten:

(1) Einer Menge I von *Inputs*.

(2) Einer Funktion S, welche jedem $x \in I$ eine endliche Menge $S(x)$ von *Lösungen* für x zuordnet.

(3) Einer *Bewertungsfunktion* $w : \{(x, y) : x \in I, y \in S(x)\} \to \mathbb{N}$, welche den Wert der Lösung y zum Input x angibt.

(4) $opt \in \{\max, \min\}$, welches das Ziel bestimmt: Maximierung oder Minimierung.

Bei einem *NP-Optimierungsproblem* (NPO-Problem) wird zusätzlich folgendes gefordert:

- Es gibt ein Polynom p, so dass $|y| \leq p(|x|)$ für alle $y \in S(x)$.
- $\{(x, y) : x \in I, y \in S(x)\}$ ist in P.
- Die Bewertungsfunktion w ist in polynomialer Zeit berechenbar.

Zu einem Optimierungsproblem Q gehört ein Berechnungsproblem und ein Entscheidungsproblem:

Berechungsproblem: Berechne für $x \in I$ den Wert einer optimalen Lösung

$$opt_Q(x) := opt\{w(x, y) : y \in S(x)\}.$$

Entscheidungsproblem $Q(D)$: Entscheide für $x \in I$ und $k \in \mathbb{N}$, ob ein $y \in S(x)$ existiert, so dass $w(x, y) \geq k$ für Maximierungsprobleme bzw. $w(x, y) \leq k$ für Minimierungsprobleme.

Lemma 7.2. *Für alle NPO-Probleme Q gilt:*

(i) $Q(D)$ ist in NP.

(ii) Wenn Q in polynomialer Zeit lösbar ist, dann auch das zugehörige Berechnungsproblem.

(iii) Das zu Q gehörende Berechnungsproblem ist in polynomialer Zeit lösbar genau dann, wenn $Q(D)$ in P ist.

(iv) Wenn das Entscheidungsproblem $Q(D)$ NP-vollständig ist, dann gibt es keinen polynomialen Algorithmus für das Optimierungsproblem Q, es sei denn P wäre gleich NP.

Beweis. Fast alle Aussagen ergeben sich unmittelbar aus den Definitionen. Bei *(iii)* ist die eine Richtung offensichtlich, für die andere verwende man binäre Suche. □

In vielen Fällen kann man auch einen polynomialen Algorithmus für das Berechungsproblem verwenden, um einen polynomialen Algorithmus für das Optimierungsproblem zu konstruieren.

Beispiel. MAX CLIQUE: In einem gegebenen Graph G finde man eine Clique maximaler Grösse.

Sei $G = (V, E)$ und $C \subseteq V$. Mit $G|_C$ bezeichnen wir den von C induzierten Untergraphen von G. Wir nehmen an, wir haben einen polynomialen Algorithmus zur Berechnung der Cliquenzahl

$$\omega(G) := \max\{|C|, C \text{ Clique von } G\}.$$

Wir gewinnen daraus den folgenden polynomialen Algorithmus für MAX CLIQUE:

Input: $G = (V, E)$
Berechne $\omega(G)$
$C := V$
for all $v \in V$ **do**
    berechne $\omega(G|_{C-\{v\}})$
    if $\omega(G|_{C-\{v\}}) = \omega(G)$ **then** $C := C - \{v\}$
    od
output C.

7.2 Approximationsalgorithmen und Approximationsschemata

Da man nicht für jedes Optimierungsproblem effiziente Algorithmen kennt, welche immer eine optimale Lösung finden (und es, wenn P $\neq$ NP, für viele NP-Optimierungprobleme gar keine solche Algorithmen geben *kann*) ist es interessant, nach polynomialen Algorithmen zu suchen, welche 'nahezu optimale' Lösungen finden, d.h. Lösungen, welche zwar nicht unbedingt optimal sind, deren Werte aber von Werten optimaler Lösungen nicht allzu stark abweichen. Solche Algorithmen nennt man Approximationsalgorithmen. Wir illustrieren dies zunächst an einem Beispiel.

Beispiel. Bei dem Optimierungsproblem MIN VERTEX COVER geht es darum, zu einem gegeben Graphen $G = (V, E)$ ein Vertex Cover (Sternüberdeckung) minimaler Grösse zu finden. Zur Erinnerung: Ein Vertex Cover ist eine Teilmenge $U \subseteq V$, so dass für jede Kante $(x, y) \in E$ entweder x oder y in U liegt.

Approximationsalgorithmus für VC (Min Vertex Cover):

Input: $G = (V, E)$
$U := \varnothing$
$F := E$
while $F \neq \varnothing$ **do**
    wähle $(u, v) \in F$
    $F := F - \{(u, v)\}$
    if $u \notin U \wedge v \notin U$ **then** $U := U \cup \{u, v\}$
    od
output U.

Die so berechnete Menge U ist offensichtlich ein Vertex Cover.

Behauptung. $|U| \leq 2 \cdot opt_{VC}(G)$.

U überdeckt $|U|/2$ disjunkte Kanten (da jeweils beide Endpunkte einer Kante zu U hinzugefügt wurden). Jedes (auch ein optimales) Vertex Cover muss diese Kanten überdecken, enthält also mindestens $|U|/2$ Knoten.

Definition 7.3. Sei $Q = (I, S, w, opt)$ ein NPO-Problem. Die *Qualität* einer Lösung $y \in S(x)$ ist

$$R(x,y) := \begin{cases} \frac{opt_Q(x)}{w(x,y)} & \text{wenn } Q \text{ Maximierungsproblem} \\ \frac{w(x,y)}{opt_Q(x)} & \text{wenn } Q \text{ Minimierungsproblem.} \end{cases}$$

Ein *ε-Aproximationsalgorithmus* für Q ist ein Algorithmus A, der zu jedem Input $x \in I$ eine Lösung $A(x) \in S(x)$ findet, mit $R(x, A(x)) \leq 1 + \varepsilon$.

Das bedeutet

- Für Minimierungsprobleme:

$$\frac{w(x, A(x))}{opt_Q(x)} \leq 1 + \varepsilon.$$

- Für Maximierungsprobleme:

$$\frac{opt_Q(x)}{w(x, A(x))} \leq 1 + \varepsilon.$$

Das Problem Min Vertex Cover erlaubt also einen polynomialen 1-Approximationsalgorithmus.

Bemerkung. In manchen Lehrbüchern und -arbeiten wird anstelle der Qualität ein anderer Parameter (z.B. relativer Fehler) verwendet. Die folgenden Approximationsklassen sind aber robust.

Definition 7.4. Sei Q ein NPO-Problem.

- Q ist in APX, wenn es zu einer Konstante $\varepsilon \leq \infty$ einen polynomialen ε-Approximationsalgorithmus für Q gibt.
- Q ist in PTAS wenn es zu jedem $\varepsilon > 0$ einen polynomialen ε-Approximationsalgorithmus A_ε für Q gibt. Die Algorithmen A_ε bilden ein *polynomiales Approximationsschema* für Q.

Sei PO die Klasse der in polynomialer Zeit lösbaren NPO-Probleme. Offensichtlich ist

$$\text{PO} \subseteq \text{PTAS} \subseteq \text{APX} \subseteq \text{NPO}.$$

Wir haben gezeigt, dass Min Vertex Cover in APX liegt. Da Vertex Cover NP-vollständig ist, folgt, dass PO $\neq$ APX, wenn P $\neq$ NP.

Ist jedes NPO-Problem in APX ? Nein, es sei denn, P ist gleich NP.

Satz 7.5. *Wenn* P $\neq$ NP, *dann ist das* TSP *nicht in* APX.

Beweis. Sei A ein ε-Approximationsalgorithmus für das TSP. Wir konstruieren daraus einen polynomialen Algorithmus für HAMILTONKREIS:

Gegeben sei ein Graph $G = (V, E)$ mit $V = \{1, ..., n\}$. Konstruiere eine $n \times n$-Matrix $D = (d_{i,j})_{1 \leq i,j \leq n}$ mit

$$d_{ij} := \begin{cases} 1 & \text{für } (i,j) \in E \\ \lceil \varepsilon n \rceil + 2 & \text{für } (i,j) \notin E \end{cases}$$

und wende den ε-Approximationsalgorithmus A auf D an. Wenn G einen Hamiltonkreis hat, dann erlaubt D eine Tour der Länge n. Touren, welche nicht zu Hamiltonkreisen gehören, haben eine Länge $\geq n - 1 + \varepsilon n + 2 > (1 + \varepsilon)n$. Wenn also D eine Tour der Länge n hat, dann muss der Approximationsalgorithmus eine solche finden, denn jede längere Tour wäre zu weit vom Optimum entfernt. Also hat G einen Hamiltonkreis genau dann, wenn A eine Tour der Länge n findet. □

Bemerkung. Wir haben die NP-Vollständigkeit des TSP(D) mit einer Reduktion von HAMILTONKREIS auf TSP-Inputs mit ausschließlich Distanzen 1 und 2 bewiesen. Also ist TSP(D) auch noch NP-vollständig, wenn nur Distanzen 1 und 2 zugelassen sind. Für solche TSP-Inputs gibt es offensichtlich einen ε-Approximationsalgorithmus, denn keine Tour ist mehr also doppelt so lang wie die optimale.

Polynomiale Approximationsalgorithmen gibt es für viele wichtige Eischränkungen des TSP. Die wichtigste ist das Δ-TSP, bei dem nur Distanzmatrizen als Input zugelassen sind, deren Distanzen der Dreiecksungleichung genügen.

Wir zeigen nun, daß es tatsächlich Probleme mit polynomialen Approximationsschemata gibt.

MAX KNAPSACK: Ein Input I besteht aus n Objekte mit Gewichten $w_1, ..., w_n$ und Werten $v_1, ..., v_n$, sowie aus einer Gewichtsschranke W (mit $W \geq w_i$ für alle i). Lösungen sind Teilmengen $S \subseteq \{1, ..., n\}$, so dass $W(S) := \sum_{i \in S} w_i \leq W$. Ziel ist es, den Wert $V(S) := \sum_{i \in S} v_i$ zu maximieren.

Satz 7.6. MAX KNAPSACK $\in$ PTAS.

Beweis. Für einen Input $I = (w_1, \ldots, w_n, W, v_1, \ldots, v_n)$ sei $V := \max\{v_1, ..., v_n\}$. Für $i \leq n$ und $v \leq nV$ sei $W(i, v)$ das minimale Gewicht $W(S)$ für alle $S \subseteq \{1, ..., i\}$ mit $V(S) = v$. Zudem sei $W(0, 0) := 0$ und $W(0, v) = \infty$ für $v > 0$. Offensichtlich gilt

$$W(i+1, v) = \min\{W(i, v), W(i, v - v_{i+1}) + w_{i+1}\}.$$

Wenn die Werte $W(i, v')$ für $v' < v$ bereits berechnet sind, dann kann man also $W(n, v)$ mit $O(n)$ arithmetischen Operationen (Additionen, Subtraktionen, Vergleiche) berechnen. Der Wert $opt(I)$ einer optimalen Lösung für Input I ist das grösste $v \leq nV$ mit $W(n, v) \leq W$. Diesen Wert, und damit gleichzeitig auch eine optimale Lösung S, kann man also mit $O(n^2 V)$ arithmetischen Operationen bestimmen. Man beachte dabei, dass V exponential gross bezüglich der Länge der Eingabe I sein kann. Das beschriebene Verfahren zur Lösung von MAX KNAPSACK ist also kein polynomialer Algorithmus, führt aber zur folgenden Idee zur Konstruktion von Approximationsalgorithmen.

Wir setzen, für einen geigneten Parameter b der erst später bestimmt wird, die letzten b Bits in der Binärdarstellung der Werte v_i auf 0 (Tradeoff Schnelligkeit gegen Genauigkeit). Aus dem

Input $I = (w_1, ..., w_n, W, v_1, ..., v_n)$ definiert man also einen neuen Input $I' = (w_1, ..., w_n, W, v'_1, ..., v'_n)$ mit $v'_i := 2^b \lfloor \frac{v_i}{2^b} \rfloor$ (gleich v_i mit letzten b Bits auf 0).

Der Wert einer optimalen Lösung S‘ für I' kann mit dem oben beschriebenen Algorithmus mit $O(\frac{n^2 V}{2^b})$ arithmetischen Operationen gefunden werden, wenn man die auf 0 gesetzten Bits einfach ignoriert. Sei S eine optimale Lösung für I. Da S optimal für I, S' optimal für I' sind, und da $v_i \geq v'_i \geq v_i - 2^b$, folgt:

$$\sum_{i \in S} v_i \geq \sum_{i \in S'} v_i \geq \sum_{i \in S'} v'_i \geq \sum_{i \in S} v'_i \geq \sum_{i \in S} v_i - 2^b \geq \sum_{i \in S} v_i - n2^b.$$

Also ist $opt(I) = \sum_{i \in S} v_i \leq \sum_{i \in S'} v_i + n2^b$. Betrachte nun S' als Approximationslösung für I, wobei wir ohne Beschränkung der Allgemeinheit annehmen können, dass $V(S') = \sum_{i \in S'} \geq V$ ist (andernfalls ersetzen wir S' durch die Lösung $S' = \{i\}$ für $v_i = V$).

Die *Qualität* von S' für I ist

$$R(I, S') = \frac{opt(I)}{W(S')} = \frac{\sum_{i \in S} v_i}{\sum_{i \in S'} v_i} \leq \frac{\sum_{i \in S'} v_i + n2^b}{\sum_{i \in S'} v_i} = 1 + \frac{n2^b}{\sum_{i \in S'} v_i} \leq 1 + \frac{n2^b}{V}.$$

Daraus gewinnen wir das folgende Approximationsschema: Gegeben sei $\varepsilon > 0$. Setze $b := \log(\varepsilon V/n)$. Berechne mit $O(\frac{n^2 V}{2^b}) = O(\frac{n^3}{\varepsilon})$ arithmetischen Operationen eine Lösung S' der Qualität $\leq 1 + \varepsilon$. □

Bemerkung. Das beschriebene Approximationsschema für MAX KNAPSACK ist nicht nur polynomial in der Inputlänge sondern sogar polynomial in $1/\varepsilon$. Mann nennt dies ein *voll-polynomiales Approximationsschema.*

Wir wissen, dass PO $\subseteq$ PTAS $\subseteq$ APX $\subseteq$ NPO ist, und für viele Optimierungsprobleme sind Approximationsresultate bekannt. Wir kennen folgende Beispiele:

- MAX FLOW ist in PO (siehe Einleitung).
- MAX KNAPSACK ist in PTAS.
- MIN VERTEX COVER und ΔTSP sind Probleme in APX.
- Das TSP ist in NPO, aber nicht in APX, es sei denn P ist gleich NP.

Wichtige Fragen, die sich aus diesen Resultaten ergeben sind etwa:

- Wie sieht eine geeignete *Reduktionstheorie* zwischen Optimierungsprobleme aus, entsprechend den polynomialen (oder logspace) Reduktionen zwischen Entscheidungsproblemen.
- Gibt es strukturelle Kriterien für Approximierbarkeit? Wie kann man einem Problem ansehen, ob es in APX liegt?

7.3 Approximationserhaltende Reduktionen

Wir haben gesehen, dass NPO-Probleme sehr unterschiedliche Approximationseigenschaften haben können, auch wenn die zugehörigen Entscheidungsprobleme alle NP-vollständig sind. Polynomiale Reduktionen zwischen den Entscheidungsproblemen bewahren demnach die Approximationseigenschaften der Optimierungsprobleme nicht.

Approximationserhaltende Reduktionen zwischen Optimierungsproblemen (sogenannte L-Reduktionen) haben eine kompliziertere Struktur. Sie bilden nicht nur Inputs eines Problems Q auf Inputs eines andern Problems Q' ab, sondern auch umgekehrt Lösungen für Q' auf Lösungen für Q.

Definition 7.7. Seien $Q = (I, S, w, opt)$ und $Q' = (I', S', w', opt')$ NPO-Probleme. Eine *L-Reduktion* von Q nach Q' besteht aus einem Paar von polynomial berechenbaren Funktionen f, g mit

- $f : I \to I'$ und es existiert eine Konstante α, so dass für alle $x \in I$ gilt:

$$opt_{Q'}(f(x)) \leq \alpha \cdot opt_Q(x).$$

- Für alle $x \in I$ und $y' \in S'(f(x))$ ist $g(x, y') \in S(x)$, und es gibt eine Konstante β mit

$$|opt_Q(x) - w(x, g(x, y')| \leq \beta |opt_{Q'}(f(x)) - w'(f(x), y')|.$$

Wir schreiben $Q \leq_L Q'$ wenn eine L-Reduktion von Q nach Q' existiert.

Beispiel. Betrachte die triviale Reduktion von MAX INDEPENDENT SET auf MIN VERTEX COVER, so dass für jeden Inputgraphen $G = (V, E)$

$$f : G \mapsto G$$
$$g : (G, C) \mapsto (V - C).$$

Diese Reduktion bildet zwar Vertex Covers C für G auf unabhängige Mengen $V - C$ von G ab (wenn C ein Vertex Cover ist, dann hat jede Kante von G mindestens einen Endpunkt in C, also ist $V - C$ unabhängig), ist aber *keine* L-Reduktion. Ein optimales Vertex Cover von G kann beliebig viel grösser sein als eine maximale unabhängige Menge in G (z.B. wenn G eine grosse Clique ist). Die erste Bedingung für L-Reduktionen ist also verletzt.

Betrachte hingegen die Probleme MAX k-DEG IS und MAX k-DEG VC, dieselben Probleme für Inputgraphen mit maximalem Grad k. (Der maximale Grad eines Graphen $G = (V, E)$ ist $\Delta(G) = \max_{v \in V} \deg(v) = \max_{v \in V} |\{w : (v, w) \in E\}|$.)

Die Reduktion (f, g) ist eine L-Reduktion von MAX k-DEG I.S. nach MAX k-DEG VC, denn:

- Wenn $\Delta(G) = k$, dann ist $opt_{IS}(G) \geq |V|/(k+1)$. Also ist

$$opt_{VC}(G) \leq |V| \leq \frac{1}{k+1} opt_{IS}(G).$$

- Für alle Vertex Covers C gilt:

$$|opt_{VC}(G) - |C|| = |opt_{IS}(G) - |V - C|.|$$

Also erfüllen $\alpha = 1/(k+1)$ und $\beta = 1$ die geforderten Bedingungen.

Lemma 7.8. *Wenn $Q \leq_L Q'$ und $Q' \leq_L Q''$, dann auch $Q \leq_L Q''$.*

Beweis. Seien (f, g) und (f', g') L-Reduktionen von Q nach Q' bwz. von Q' nach Q'' mit den zugehörigen Konstanten α, β bzw. α', β'. Setze $f'' := f' \circ f$ und $g''(x, y'') := g(x, g'(f(x), y''))$. Man verifiziert leicht, dass (f'', g'') eine L-Reduktion von Q nach Q'' ist, mit zugehörigen Konstanten $\alpha'' = \alpha\alpha'$ und $\beta'' = \beta\beta'$. □

Satz 7.9. *Sei (f, g) eine L-Reduktion von Q nach Q' mit Konstanten α, β. Wenn ein polynomialer ε-Approximationsalgorithmus für Q' existiert, dann auch ein polynomialer $\alpha\varepsilon\beta$-Approximationsalgorithmus für Q.*

Beweis. Sei A' ein ε-Approximationsalgorithmus für Q'. Der gewünschte Approximationsalgorithmus A für Q berechnet zu Input x die Lösung

$$y := A(x) := g(x, A'(f(x))).$$

Wir betrachten den Fall, dass Q ein Mininimierungsproblem ist (Maximierungsprobleme werden analog behandelt). Die Qualität von y ist

$$\begin{aligned} R(x,y) &= \frac{w(x,y)}{opt_Q(x)} = 1 + \frac{w(x,y) - opt_Q(x)}{opt_Q(x)} \leq 1 + \frac{\beta |w(f(x), A'(f(x))) - opt_{Q'}(f(x))|}{opt_Q(x)} \\ &\leq \frac{\beta\varepsilon\, opt_{Q'}(f(x))}{opt_Q(x)} \leq 1 + \frac{\beta\varepsilon\alpha\, opt_Q(x)}{opt_Q(x)} = 1 + \alpha\varepsilon\beta. \end{aligned}$$

□

Damit folgt, dass L-Reduktionen tatsächlich einen geeigneten Reduktionsbegriff für die Approximationsklassen APX und PTAS definieren.

Korollar 7.10. *Seien Q und Q' NPO-Probleme mit $Q \leq_L Q'$. Dann gilt:*

(i) Wenn $Q' \in$ APX, dann ist auch $Q \in$ APX.

(ii) Wenn $Q' \in$ PTAS, dann ist auch $Q \in$ PTAS.

7.4 Ein logisches Kriterium für Approximierbarkeit: die Klasse MAX SNP

Wir beschreiben eine Klasse von logisch definierten Maximierungsproblemen und zeigen, dass alle diese Probleme einen polynomialen Approximationsalgorithmus erlauben. Dieser Ansatz wurde Ende der 80er Jahre von Papadimtriou und Yannakakis vorgeschlagen, motiviert durch die logische Charakterisierung von NP durch die existentielle Logik zweiter Stufe (Satz von Fagin).

Definition 7.11. MAX Σ_0 ist die Klasse aller NP-Maximierungsprobleme $Q = (I, S, w, \max)$ deren Inputs endliche Strukturen einer festen Signatur τ sind, und für die eine quantorenfreie Formel $\psi(\bar{x}, \bar{S})$ der Signatur $\tau \cup \{\bar{S}\}$ existiert ($\bar{S}$ ist ein Tupel von Relationssymbolen), so dass für alle Inputs $\mathfrak{A}$ folgendes gilt:

(1) $opt_Q(\mathfrak{A}) = \max_{\bar{S}} |\{\bar{x} : (\mathfrak{A}, \bar{S}) \models \psi(\bar{x}, \bar{S})\}|$

(2) Aus $\mathfrak{A}$ und $\bar{S}$ kann in polynomialer Zeit eine Lösung $S_0 \in S(\mathfrak{A})$ gefunden werden, mit $w(\mathfrak{A}, S_0) = |\{\bar{x} : (\mathfrak{A}, \bar{S}) \models \psi(\bar{x}, \bar{S})\}|$.

Bemerkung. Die erste Bedingung folgt aus der zweiten. In der Regel ist die gewünschte Lösung S_0 eine der Relationen aus dem dem Tupel $\bar{S}$.

Die Klasse MAX SNP ist der Abschluss von MAX Σ_0 unter L-Reduktionen: Ein NPO Problem Q ist in MAX SNP wenn $Q \leq_L Q'$ für ein $Q' \in$ MAX Σ_0. (Damit kann die Klasse MAX SNP auch Minimierungsprobleme enthalten).

Beispiele. (1) Das Problem MAX CUT besteht darin, zu einem gegebenen Graphen $G = (V, E)$ eine Teilmenge $U \subseteq V$ zu finden, so dass die Anzahl der Kanten zwischen U und $V - U$ möglichst gross wird. Dieses Problem ist in MAX SNP, denn

$$opt_{\text{MAX CUT}}(G) = \max_U |\{(x, y) : G \models Exy \wedge Ux \wedge \neg Uy\}|$$

(2) Für MAX CLIQUE ist die natürliche Definition einer optimalen Lösung dagegen nicht quantorenfrei, sondern durch eine universelle Formel gegeben:

$$opt_{\text{MAX CLIQUE}}(G) = \max_C |\{x : (G, C) = \psi(x, C)\}|$$

mit $\psi(x, C) = Cx \wedge \forall y \forall z(Cy \wedge Cz \to y = z \vee Eyz)$.

Satz 7.12 (Papadimitriou, Yannakakis). MAX SNP $\subseteq$ APX.

Beweis. Aufgrund von Korollar 7.10 reicht es zu zeigen, dass MAX $\Sigma_0 \subseteq$ APX. Sei Q in MAX Σ_0 mit

$$opt_Q\mathfrak{A}) = \max_{\bar{S}} |\{\bar{x} : (\mathfrak{A}, \bar{S}) \models \psi(\bar{x}, \bar{S})\}|.$$

Betrachte eine feste Inputstruktur $\mathfrak{A}$. Wir bezeichnen mit Ω die Menge aller möglichen Interpretationen von $\bar{S}$ auf $\mathfrak{A}$ und fassen Ω als endlichen Wahrscheinlichkeitsraum mit der uniformen Verteilung auf (d.h. alle Interpretationen $\bar{S} \in \Omega$ sind gleich wahrscheinlich). Durch

$$F(\bar{S}) := |\{\bar{x} : (\mathfrak{A}, \bar{S}) \models \psi(\bar{x}, \bar{S})\}|$$

wird eine Zufallsvariable $F : \Omega \to \mathbb{N}$ definiert. Offensichtlich ist $opt_Q(\mathfrak{A}) = \max F$. Sei $E(F)$ der *Erwartungswert* von F auf Ω.

Lemma 7.13. *Es gibt ein $\varepsilon > 0$, so dass*

$$opt_Q(\mathfrak{A}) = \max F \geq E(F) \geq \varepsilon \cdot \max F.$$

Um dies einzusehen schreiben wir F als Summe von Zufallsvariabeln mit Werten 0 und 1.

$$F(\bar{S}) := \sum_{\bar{a} \in A^r} F_{\bar{a}}(\bar{S}) \text{mit}$$

$$F_{\bar{a}}(\bar{S}) := \begin{cases} 1 & \text{wenn } (\mathfrak{A}, \bar{S}) \models \psi(\bar{a}, \bar{S}) \\ 0 & \text{andernfalls} \end{cases}$$

Lemma 7.14. $E(F) = \sum_{\bar{a} \in A^r} E(F_{\bar{a}})$.

Dies ist eine allgemeine und elementare Tatsache (Linearität des Erwartungswerts):

$$E(F) = \frac{1}{|\Omega|} \sum_{\bar{S} \in \Omega} F(\bar{S}) = \frac{1}{|\Omega|} \sum_{\bar{S} \in \Omega} \sum_{\bar{a}} F_{\bar{a}}(\bar{S}) = \sum_{\bar{a}} \frac{1}{|\Omega|} \sum_{\bar{S} \in \Omega} F_{\bar{a}}(\bar{S}) = \sum_{\bar{a}} E(F_{\bar{a}}).$$

Sei $B := \{\bar{a} \in A^r : F_{\bar{a}}(\bar{S}) = 1 \text{ für mindestens ein } \bar{S}\}$. Dann gilt

- $\max F \leq |B|$.
- $E(F) = \sum_{\bar{a} \in A^r} E(F_{\bar{a}}) = \sum_{\bar{a} \in B} E(F_{\bar{a}})$.

Sei k die Anzahl der $\bar{S}$-Atome in $\psi(\bar{x}, \bar{S})$. Wähle für $\bar{a} \in B$ ein $\bar{S}_0 \in \Omega$ mit $(\mathfrak{A}, \bar{S}_0) \models \psi(\bar{a}, \bar{S})$. Jedes $\bar{S}_1 \in \Omega$, das mit $\bar{S}_0$ bezüglich der Wahrheitswerte der k $\bar{S}$-Atome übereinstimmt, welche in der quantorenfreien Formel $\psi(\bar{a}, \bar{S})$ tatsächlich vorkommen, wird die Formel $\psi(\bar{a}, \bar{S})$ ebenfalls erfüllen. Damit folgt:

- Wenn $\bar{a} \in B$, dann ist $E(F_{\bar{a}}) \geq 2^{-k}$.

Also gilt

$$\max F \geq E(F) = \sum_{\bar{a} \in B} E(F_{\bar{a}}) \geq 2^{-k}|B| \geq 2^{-k} \max F.$$

Damit ist Lemma 7.13 bewiesen.

Dies bedeutet, dass der erwartete Wert von F auf einem zufällig gewählten Tupel $\bar{S}$ um höchstens einen konstanten Faktor vom Wert einer optimalen Lösung abweicht. Es bleibt zu zeigen, dass man ein Tupel $\bar{S}$ mit dieser Eigenschaft auch explizit (und einigermassen effizient) konstruieren kann. (Man nennt diesen Prozess *Derandomisierung.*)

Lemma 7.15. *Es gibt einen deterministischen Polynomialzeit-Algorithmus, welcher zu jeder Inputstruktur $\mathfrak{A}$ ein Tupel $\bar{S}_0$ findet, so dass $F(\bar{S}_0) \geq E(F)$.*

Sei $\alpha_0, \ldots, \alpha_{m-1}$ eine Aufzählung aller $\bar{S}$-Atome über A (m ist ein Polynom in $|A|$). Für $i = 0, ..., m-1$ werden sukzessive Wahrheitswerte für α_i definiert. Sei β_i die Konjunktion über die α_j bzw. $\neg\alpha_j$ für $j < i$, die bereits als wahr definiert wurden. Sei nun $E(F \mid \beta_i)$ der bedingte Erwartungswert für $F(\bar{S})$ wenn die Wahrheitswerte von α_j für $j < i$ gemäss β_i bereits definiert sind.

Lemma 7.16. *Die Erwartunggswerte $E(F \mid \beta_i \wedge \alpha_i)$ und $E(F \mid \beta_i \wedge \neg\alpha_i)$ sind in polynomialer Zeit berechenbar.*

Die folgt aus ähnlichen Überlegungen wie im Beweis von Lemma 7.13. Für jede Konjunktion γ von $\bar{S}$-Atomen gilt auch für die bedingten Erwartungswerte, dass $E(F \mid \gamma) = \sum_{\bar{a} \in A^r} E(F_{\bar{a}} \mid \gamma)$. Da aber $F_{\bar{a}}$ nur von einer festen Anzahl k von $\bar{S}$-Atomen abhängt, ist $E(F_{\bar{a}} \mid \gamma)$ in polynomialer Zeit berechenbar.

Definiere nun sukzessive, für $i = 0, \ldots, m-1$ die Wahrheitswerte von α_i wie folgt: Setze $\alpha_i := \mathit{true}$, wenn $E(F \mid \beta_i \wedge \alpha_i) \geq E(F \mid \beta_i \wedge \neg\alpha_i)$, andernfalls definiere $\alpha_i := \mathit{false}$.

Zu Beginn dieser Prozedur ist noch nichts festgelegt und es gilt $E(F \mid \beta_0) = E(F)$. Am Ende dieser Prozedur sind Wahrheitswerte für alle α_i, und damit eine Formel β_m bestimmt, welche ein Tupel $\bar{S}_0$ eindeutig festlegt. Es gilt dann $E(F \mid \beta_m) = F(\bar{S}_0)$. Für alle i gilt aber

$$E(F \mid \beta_i) = \frac{1}{2}\Big(E(F \mid \beta_i \wedge \alpha_i) + E(F \mid \beta_i \wedge \neg\alpha_i)$$

und daher

$$E(F \mid \beta_{i+1}) \geq E(F \mid \beta_i).$$

Damit folgt für das so gefundene $\bar{S}_0$:

$$F(\bar{S}_0) = E(F \mid \beta_m) \geq E(F \mid \beta_{m-1}) \geq \cdots \geq E(F \mid \beta_0) = E(f).$$

Zusammenfassend folgern wir, dass in polynomialer Zeit zu $\mathfrak{A}$ ein Tupel $\bar{S}_0$ gefunden wird, so dass

$$F(\bar{S}_0) \geq E(F) \geq 2^{-k} \max F = 2^{-k} \cdot opt_Q(\mathfrak{A}).$$

Aus $\bar{S}_0$ erhält man eine Lösung S für Q auf Input $\mathfrak{A}$ mit $w_Q(\mathfrak{A}, S) = F(\bar{S}_0)$, d.h.

$$R(\mathfrak{A}, S) \leq \frac{opt_Q(\mathfrak{A})}{2^{-k} \cdot opt_Q(\mathfrak{A})} \leq 2^k.$$

Also ist Q in APX. □

Kapitel 8

Komplexitätstheorie für probabilistische Algorithmen

8.1 Probabilistische Algorithmen

Probabilistische Algorithmen sind Algorithmen, welche an bestimmten Punkten ihrer Berechnung aus einer Anzahl von verschiedenen Möglichkeiten für die nächste Operation 'zufällig' eine auswählen können. Probabilistische Algorithmen können also als Abwandlung von nichtdeterministischen Algorithmen aufgefasst werden. Das Resultat der Berechnung eines solchen Algorithmus ist daher nicht eine feste Antwort, sondern eine Zufallsvariable: die Antwort ist abhängig von den während der Berechnung 'zufällig' getroffenen Entscheidungen. Man beachte dass dies nichts mit einer Annahme über die Verteilunng der möglichen Eingaben zu tun hat. Die Zufälligkeit betrifft nicht die Inputs, sondern nur die Auswahl der Entscheidungen während der Berechnung.

Probabilistische Algorithmen spielen eine bedeutende Rolle in verschiedensten Anwendungsgebieten. Sie sind oft einfacher und effizienter als die besten bekannten deterministischen Algorithmen für dasselbe Problem. Wichtige Gebiete, wie z.B. die algorithmische Zahlentheorie und die Kryptologie sind in ihrer heutigen Form ohne probabilistische Algorithmen gar nicht denkbar. Wir behandeln zwei Beispiele.

8.1.1 Perfektes Matching und symbolische Determinanten

Wir erinnern nochmals an die Definition des Heiratsproblem (siehe Einleitung). Gegeben ist ein bipartiter Graph $G = (U, V, E)$ mit zwei disjunkten, gleichmächtigen Knotenmengen $U = \{u_1, \ldots, u_n\}$, $V = \{v_1, \ldots, v_n\}$, und einer Kantenmenge $E \subseteq U \times V$. Gefragt ist, ob G ein *perfektes Matching* zulässt, d.h., eine Teilmenge $M \subseteq E$ so dass für alle $u \in U$ genau ein $v \in V$ und für alle $v \in V$ genau ein $u \in U$ existiert mit $(u, v) \in M$. Anders gefragt: Gibt es eine Permutation $\pi \in S_n$ so dass $(u_i, v_{\pi(i)}) \in E$ für alle $i \in \{1, \ldots, n\}$.

Wir können dies auch als ein Problem über Matrizen und Determinanten beschreiben. Der Graph $G = (U, V, E)$ wird durch eine Matrix A^G beschrieben, deren Einträge Variabeln x_{ij} oder 0 sind.

$$A^G := (z_{ij})_{1 \leq i,j \leq n} \quad \text{mit} \quad z_{ij} := \begin{cases} x_{ij} & \text{wenn } (u_i, v_j) \in E \\ 0 & \text{sonst.} \end{cases}$$

Die Determinante von A^G ist $\det A^G := \sum_{\pi \in S_n} \operatorname{sgn}(\pi) \prod_{i=1}^n z_{i\pi(i)}$ wobei $\operatorname{sgn}(\pi) = 1$, wenn π das Produkt einer geraden Anzahl von Transpositionen ist, und $\operatorname{sgn}(\pi) = -1$, andernfalls. Offensichtlich ist $\det A^G$ ein Polynom in $\mathbb{Z}[x_{11}, \ldots, x_{nn}]$ von totalem Grad n, das linear ist in jeder Variablen x_{ij}.

Eine Permutation $\pi \in S_n$ definiert ein perfektes Matching genau dann, wenn $\prod_{i=1}^n z_{ij} \neq 0$. Da alle diese Produkte paarweise verschieden sind folgt:

$$G \text{ erlaubt ein perfektes Matching} \iff \det A^G \neq 0.$$

Wenn wir also symbolische Determinanten (Determinanten von Matrizen die Variablen enthalten können) effizient berechnen könnten, dann könnten wir damit auch das Heiratsproblem lösen.

Determinanten via Gauß-Elimination. Aus der linearen Algebra ist bekannt, wie man Determinanten von numerischen Matrizen berechnen kann: Man transformiere die gegebene Matrix (etwa durch Vertauschen von Zeilen und durch Addieren von geeigneten Linearkombinationen von Zeilen zu andern Zeilen) in eine Dreiecksmatrix mit derselben Determinante und berechne das Produkt der Diagonalelemente. Dies erfordert $O(n^3)$ arithmetische Operationen. Ausserdem bleiben die Einträge der transformierten Matrizen polynomial beschränkt, da es sich um Subdeterminanten der gegebenen Matrix handelt.

Leider ist die Anwendung dieses Verfahrens auf symbolische Matrizen problematisch. Die Einträge der transformierten Matrizen sind rationale Funktionen in den Einträgen der ursprünglichen Matrix, und diese Funktionen haben im allgemeinen exponentiell viele Terme. Bereits das Problem, ob irgend ein festes Monom, z.B. $x_{11}x_{23}x_{31}$ in der Determinante von A^G auftritt, ist NP-hart. Gauß-Elimination scheint also nicht hilfreich zur Berechnung symbolischer Determinanten zu sein.

Aber wir brauchen ja gar nicht unbedingt die Determinante von A^G zu berechnen. Es reicht uns zu wissen, ob sie identisch 0 ist oder nicht. Die Idee des probabilistischen Algortihmus für das Perfect Matching Problem besteht einfach darin, ein Tupel $\bar{a} = (a_{11}, \ldots, a_{nn})$ zufällig gewählter Zahlen in die Matrix A^G einzusetzen und dann mittels Gauß-Elimination die Determinante der numerischen Matrix $A^G(\bar{a})$ auszurechnen.

Wenn sich herausstellt, dass $\det A^G(\bar{a}) \neq 0$, dann ist offensichtlich die symbolische Determinante $\det A^G$ nicht identisch 0. Die Umkehrung gilt jedoch natürlich nicht: Es kann passieren, dass $\det A^G \neq 0$, dass wir aber zufällig eine Nullstelle $\bar{a}$ von $\det A^G$ erwischt haben.

Das folgende Lemma erlaubt uns aber, durch geeignete Wahl des Bereichs aus dem wir $\bar{a}$ auswählen, die Wahrscheinlichkeit, dass wir Nullstellen eines nicht identisch verschwindenden Polynoms $\det A^G$ erwischen, zu kontrollieren.

Lemma 8.1. *Sei $p(x_1, \ldots, x_n)$ ein Polynom, so dass $p \neq 0$ und jedes x_i höchstens Grad d in p hat. Für jedes $m \in \mathbb{N}$ gilt dann, dass*

$$|\{(a_1, \ldots, a_n) \in \{0, \ldots, m-1\}^n : p(a_1, \ldots, a_n) = 0\}| \leq ndm^{n-1}.$$

Beweis. Wir führen den Beweis per Induktion über n. Für $n = 1$ ist dies eine wohlbekannte Tatsache: Kein Polynom $p \neq 0$ in einer Variablen vom Grad d hat mehr als d Nullstellen. Sei nun $n > 1$. Wir schreiben $p(x_1, \ldots, x_n)$ als ein Polynom in x_n mit Koeffizienten aus $\mathbb{Z}[x_1, \ldots, x_{n-1}]$:

$$p(x_1, \ldots, x_n) = p_0(x_1, \ldots, x_{n-1}) + p_1(x_1, \ldots, x_{n-1})x_n + \cdots + p_d(x_1, \ldots, x_{n-1})x_n^d.$$

Sei $p(a_1, \ldots, a_n) = 0$ für $(a_1, \ldots, a_n) \in \{0, \ldots, m-1\}^n$. Wir unterscheiden zwei Fälle.

(a) $p_d(a_1, \ldots, a_{n-1}) = 0$. Nach Induktionsvoraussetzung tritt dies für höchstens $(n-1)dm^{n-2}$ Tupel $(a_1, \ldots, a_{n-1}) \in \{0, \ldots, m-1\}^{n-1}$ ein. Also gibt es höchstens $(n-1)dm^{n-1}$ Nullstellen $(a_1, \ldots, a_n) \in \{0, \ldots, m-1\}^n$ von p mit $p_d(a_1, \ldots, a_{n-1}) = 0$.

(b) $p_d(a_1, \ldots, a_{n-1}) \neq 0$. Dann ist $p(a_1, \ldots, a_{n-1}, x_n)$ ein Polynom vom Grad d in einer Variablen x_n, für das demnach höchstens d Nullstellen a_n existieren. Zu den Nullstellen im Fall (a) kommen also noch höchstens dm^{n-1} zusätzliche Nullstellen hinzu.

Damit haben wir insgesamt höchstens ndm^{n-1} Nullstellen $(a_1, \ldots, a_n) \in \{0, \ldots, m-1\}^n$. □

Damit können wir nun einen probabilistischen Algorithmus für das Perfect Matching Problem angeben.

Input: Eine Matrix A^G für einen bipartiten Graphen $G = (U, V, E)$ mit $|U| = |V| = n$
ein Sicherheitsparameter $k \in \mathbb{N}$
setze $m := 2n^2$
for $i = 1, \ldots, k$ **do**
wähle zufällig Zahlen $a_{11}, \ldots, a_{nn} \in \{0, \ldots, m-1\}$
Berechne $\det A^G(\bar{a})$ mittels Gauß-Elimination
if $\det A^G(\bar{a}) \neq 0$ **then output** 'es gibt ein perfektes Matching' **end**
od
output 'es gibt vermutlich kein perfektes Matching' **end**

Da die Berechnung numerischer Determinanten mittels Gauß-Elimination in polynomialer Zeit möglich ist, ist dies ein Polynomialzeit-Algorithmus. Wenn der Algorithmus ein Tupel $\bar{a}$ findet mit $\det A^G \neq 0$, und also ausgibt 'es gibt ein perfektes Matching', dann ist dies in jedem Fall korrekt. Wenn der Algorithmus aber in k Versuchen kein solches $\bar{a}$ findet und also das Resultat 'vermutlich gibt es kein perfektes Matching' produziert, dann ist diese Antwort mit einer Unsicherheit behaftet. Diese ist aber sehr klein, denn die Wahrscheinlichkeit, dass der Algorithmus für ein nicht-verschwindendes Polynom $\det A^G$ keine Nicht-Nullstelle findet, kann mit Hilfe des soeben bewiesenen Lemmas wie folgt abgeschätzt werden. Da $\det A^G$ linear ist in jeder der n^2 Variabeln, ist der Anteil der Tupel $\bar{a} \in \{0, \ldots, m-1\}^{n^2}$, welche Nullstellen von $\det A^G$ sind, höchstens

$$\frac{n^2 dm^{n^2-1}}{m^{n^2}} = \frac{n^2 d}{m} = \frac{n^2}{2n^2} = \frac{1}{2}.$$

Die Wahrscheinlichkeit, dass in k Versuchen nur immer solche Tupel gefunden wurden ist also höchstens 2^{-k}. Man beachte, dass dies nicht eine Wahrscheinlichkeitsaussage über bipartite Graphen oder symbolische Determinanten ist. Es ist eine Aussage über die Fehlerwahrscheinlichkeit eines probabilistischen Algorithmus bezüglich seiner Zufallsentscheidungen und gilt für alle bipartiten Graphen.

8.1.2 Ein probabilistischer Primzahltest

Zwei zentrale offene Probleme der algorithmischen Zahlentheorie sind die Existenz polynomialer Algorithmen, welche

(1) testen, ob eine gegebene Zahl $n \in \mathbb{N}$ eine Primzahl ist;

(2) eine gegebene Zahl $n \in \mathbb{N}$ in ihre Primfaktoren zerlegen.

Es wird vermutet, dass das zweite Problem, das Faktorisieren natürlicher Zahlen, auch im praktischen Sinn sehr schwierig ist. Auf dieser Annahme beruhen übrigens zahlreiche moderne Public-Key Kryptosysteme. Auch für das erste Problem (zu entscheiden, ob eine Zahl prim ist) gibt es noch keine deterministischen, beweisbar korrekten, polynomialen Algorithmen. Trotzdem kann das Problem für die meisten praktischen Zwecke als gelöst betrachtet werden. Einerseits gibt es relativ gute, wenn auch nicht polynomiale deterministische Primzahltest, andererseits kennt man einfache und effiziente probabilistische Verfahren, welche mit fast beliebig grosser Wahrscheinlichkeit richtig erkennen, ob eine Zahl prim ist oder nicht. Allerdings liefern diese Verfahren für zusammengesetzte Zahlen natürlich *nicht* die Zerlegung in Primfaktoren.

Den Ansatz für solche Verfahren liefert der 'kleine Satz von Fermat'. Für $n \in \mathbb{N}$ ist

$$\mathbb{Z}_n^* := \{a \in \{1, \ldots, n-1\} : \mathrm{ggT}(a, n) = 1\}.$$

Man beachte, dass $(\mathbb{Z}_n^*, \cdot \pmod n)$ eine Gruppe ist.

Satz 8.2 (Fermat). *Sei p prim. Dann gilt für alle $a \in \mathbb{Z}_p^*$, dass $a^{p-1} \equiv 1 \pmod p$.*

Beweis. Sei $f(p, a)$ die Anzahl der verschiedenen aperiodischen Färbungen eines Zyklus der Länge p mit a Farben. Da p prim ist, und da für jede *periodische* Färbung die Periode ein Teiler von p sein muss, ist nur Periode 1 möglich, also nur monochrome Färbungen. Die Anzahl aller Färbungen von p Knoten mit a Farben ist a^p, die Anzahl der monochromen Färbungen ist a, also ist $f(p, a) = (a^p - a)/p = a(a^{p-1} - 1)/p$. Es folgt, dass p ein Teiler ist von $a^{p-1} - 1$, also $a^{p-1} \equiv p \pmod p$. □

Man könnte nun hoffen, dass auch die Umkehrung gilt, dass also für jede *zusammengesetzte* Zahl n ein $a \in \mathbb{Z}_n^*$ existiert so dass $a^{n-1} \not\equiv 1 \pmod n$. Wenn man weiter zeigen könnte, dass in diesem Fall sogar 'viele' $a \in \mathbb{Z}_n^*$ mit dieser Eigenschaft existieren, dann könnte ein probabilistischer Primzahltest so aussehen, dass man zu gegebenem n ein $a \in \mathbb{Z}_n^*$ zufällig wählt, und dann prüft, ob $a^{n-1} \equiv 1 \pmod n$.

Dabei muss man zuerst überlegen, dass die Verifikation, ob $a^{n-1} \not\equiv 1 \pmod n$, tatsächlich in polynomialer Zeit (bezüglich der Länge der Eingabe, also $\log n$) möglich ist. Dies kann durch wiederholtes Quadrieren modulo n erreicht werden: Für $k = \lfloor \log n \rfloor$ berechne man die Zahlen $b_0, \ldots, b_k$ mit $b_0 := a$, $b_{i+1} := (b_i)^2 \pmod n$, also $b_i = a^{2^i} \pmod n$. Sei $n - 1 = \sum_{i=1} u_i 2^i$ die Binärdarstellung von $n - 1$, mit $u_i \in \{0, 1\}$. Dann gilt

$$a^{n-1} = a^{\sum_i u_i 2^i} = \prod_i a^{u_i 2^i} \equiv \prod_{u_i = 1} b_i \pmod n.$$

Leider scheitert der Fermat-Test in dieser einfachen Form aber daran, dass die Umkehrung des 'kleinen Satzes von Fermat' falsch ist. Es gibt (sogar unendlich viele) zusammengesetzte Zahlen $n \in \mathbb{N}$, so dass $a^{n-1} \equiv 1 \pmod n$ für alle $a \in \mathbb{Z}_n^*$. Man nennt diese Zahlen *Carmichael-Zahlen*. Die ersten Carmichael-Zahlen sind 561 und 1729.

Für Nicht-Carmichael-Zahlen funktioniert die Idee jedoch. Für $n \in \mathbb{N}$, sei

$$F_n := \{a \in \mathbb{Z}_n^* : a^{n-1} \equiv 1 \pmod n\}.$$

Lemma 8.3. *Wenn n zusammengesetzt und keine Carmichael-Zahl ist, dann ist $|F_n| \leq |Z_n^*|/2$.*

Beweis. Es ist ganz leicht einzusehen, dass $(F_n, \cdot \pmod n)$ eine Untergruppe von $(\mathbb{Z}_n^*, \cdot \pmod n)$ ist. Da n weder prim noch eine Carmichael-Zahl ist, ist $F_n \subsetneq Z_n^*$. Die Ordnung einer Untergruppe muss immer ein Teiler der Ordnung der Gruppe sein, also $|\mathbb{Z}_n^*| = q|F_n|$ für ein $q \geq 2$. □

Die ursprüngliche Idee scheitert also ausschliesslich an den Carmichael-Zahlen. Es gibt aber Möglichkeiten, den Fermat-Test zu verfeinern und so auch mit den Carmichael-Zahlen fertig zu werden. Es gibt zwei verschieden solche probabilistische Primzahltests, den Solovay-Strassen-Test und den Rabin-Miller-Test, den wir hier beschreiben werden. Er beruht auf folgender Beobachtung.

Lemma 8.4. *Sei p prim. Dann gilt für alle $a \in \mathbb{Z}_p^*$: Wenn $a^2 \equiv 1 \pmod p$, dann ist $a \equiv \pm 1 \pmod p$.*

Beweis. Wenn p prim ist, dann ist $(\mathbb{Z}_p^*, + \pmod n, \cdot \pmod n)$ ein Körper, und in Körpern gibt es nur die trivialen Einheitswurzeln 1 und -1. □

Der Rabin-Miller-Primzahltest

Input: eine ungerade Zahl $n \in N$
ein Sicherheitsparameter k
Berechne t, w, so dass $n - 1 = 2^t w$ mit ungeradem w
Wiederhole k Mal:
Wähle zufällig ein $a \in \{1, \ldots, n-1\}$
Berechne $b_i := a^{2^i w} \pmod n$ für $i = 0, \ldots, t$
if $b_t = a^{n-1} \not\equiv 1 \pmod n$ **then output** "n zusammengesetzt" **end**
Bestimme $j := \max\{i : b_i \not\equiv 1 \pmod n\}$
if $b_j \not\equiv -1 \pmod n$ **then output** "n zusammengesetzt" **end**
output "n vermutlich prim" **end**

Satz 8.5. *(i) Der Rabin-Miller-Primzahlltest ist durchführbar in polynomialer Zeit (bzgl. $\log n$).*

(ii) Wenn n prim ist, dann gibt der Test immer die Antwort "n vermutlich prim".

(iii) Wenn n zusammengesetzt ist, dann gibt der Test mit Wahrscheinlichkeit $\geq 1 - 2^{-k}$ die Antwort "n zusammengesetzt".

Die Antwort "n zusammengesetzt" ist also immer richtig, die Antwort "n vermutlich prim" bedeutet, dass n mit sehr grosser Wahrscheinlichkeit tatsächlich prim ist.

Beweis. Die Aussage *(i)* ist offensichtlich korrekt. Die Aussage *(ii)* folgt aus Satz 8.2 und Lemma 8.4. Wenn n prim ist, dann gilt für alle im Test verwendeten a:

- $a^{n-1} \equiv 1 \pmod n$.
- $b_j \not\equiv 1 \pmod n$, aber $b_{j+1} = (b_j)^2 \equiv 1 \pmod n$. Also ist $b_j \equiv -1 \pmod n$

Daraus folgt, dass der Test die Antwort "n vermutlich prim" liefert.

Für die Aussage *(iii)* sei M_n die Menge aller $a \in \{1, \ldots, n-1\}$, so dass die Auswahl von a durch den Rabin-Miller-Test mit Input n *nicht* zur Antwort "n zusammengesetzt" führt. Offensichtlich reicht es zu zeigen, dass $|M_n| \leq (n-1)/2$ für alle zusammengesetzten, ungeraden

n. Die Wahrscheinlichkeit, dass bei k-maliger unabhängiger Wahl eines zufälligen a immer nur Elemente $a \in M_n$ erwischt werden, ist dann kleiner als 2^{-k}.

Zunächst beobachten wir, dass $M_n \subseteq \mathbb{Z}_n^*$. Wenn nämlich $a \in M_n$, dann ist $a^{n-1} \equiv 1 \pmod{n}$ und also $a^{n-2}a + rn = 1$ für ein geeignetes $r \in \mathbb{Z}$. Wenn nun a und n einen gemeinsamen Teiler $q > 1$ besitzen, dann müsste dieser auch die Summe $a^{n-2}a + rn$ teilen, was unmöglich ist, da diese gleich 1 ist. Also sind a und n teilerfremd und damit $a \in \mathbb{Z}_n^*$.

Also reicht es, folgendes zu zeigen.

Behauptung. *Es gibt eine echte Untergruppe $U_n < Z_n^*$, welche M_n umfasst.*

Dann folgt nämlich $|M_n| \leq |U_n| \leq |\mathbb{Z}_n^*|/2 \leq (n-1)/2$.

Für zusammengesetzte Nicht-Carmichael-Zahlen n ergibt sich die Behauptung unmittelbar aus Lemma 8.3, da $M_n \subseteq F_n$. Für Carmichael-Zahlen zeigt man zunächst, dass diese nicht Primpotenzen sind, dass also jede Carmichael-Zahl n als Produkt zweier teilerfremden, ungeraden Zahlen n_1, n_2 geschrieben werden kann. Fixiere ein solches $n = n_1 n_2$.

Für jedes $a \in M_n$ hat die Folge $b_0, \ldots, b_t$ (mit $b_i = a^{2^i w} \pmod{n}$) die Form

$$* * * \cdots * \; -1\, 1\, 1 \cdots 1 \quad \text{oder} \quad 1\, 1 \cdots 1.$$

Setze

$$h := \max\{i : 0 \leq i \leq t, \text{es gibt ein } a \in Z_n^* \text{ mit } a^{2^i w} \equiv -1 \pmod{n}\}.$$

Ein solches h existiert, da z.B. $(-1)^{2^0 w} = -1$. Sei nun

$$U_n := \{a \in \mathbb{Z}_n^* : a^{2^h w} \equiv \pm 1 \pmod{n}\}.$$

Offensichtlich ist U_n eine Untergruppe von Z_n^*, welche M_n umfasst. Man zeigt nun, dass $U_n \subsetneq Z_n^*$ wie folgt: Sei $b \in Z_n^*$ so, dass $b^{2^h w} \equiv -1 \pmod{n}$. Nach dem Chinesischen Restsatz gibt es ein $a \in \mathbb{Z}_n^*$ so dass

(1) $a \equiv b \pmod{n_1}$

(2) $a \equiv 1 \pmod{n_2}$

Wir zeigen, dass $a \notin U_n$, indem wir die Annahme $a \in U_n$ auf einen Widerspruch führen.

Sei $a \in U_n$, weil $a^{2^h w} \equiv 1 \pmod{n}$. Dann ist auch $a^{2^h w} \equiv 1 \pmod{n_1}$. Andererseits gilt wegen (1) $a^{2^h w} \equiv b^{2^h w} \equiv -1 \pmod{n_1}$ was unmöglich ist, da $n_1 > 2$.s

Im anderen Fall ist $a \in U_n$, weil $a^{2^h w} \equiv -1 \pmod{n}$. Dann ist auch $a^{2^h w} \equiv -1 \pmod{n_2}$. Andererseits gilt wegen (2) $a^{2^h w} \equiv 1 \pmod{n_2}$ was unmöglich ist, da $n_2 > 2$. □

Man vermutet, dass PRIM $\in$ P. Dies würde aus der *Erweiterten Riemannschen Hypothese* (ERH) folgen.

Satz 8.6 (Miller). *Die ERH impliziert, dass eine Funktion $f : \mathbb{N} \to \mathbb{N}$ existiert, so dass $f(n)$ durch ein Polynom in $\log n$ beschränkt ist und so dass für alle ungeraden $n > 2$ gilt: Wenn n* nicht prim *ist, dann trifft eine der folgenden Aussagen zu:*

(i) n ist eine Primpotenz;

(ii) es gibt ein $a < f(n)$, mit $a \notin M_n$, d.h. die Verwendung von a im Rabin-Miller Test auf Input n führt zur Antwort "n zussammengesetzt".

Korollar 8.7. *Die ERH impliziert* PRIM $\in$ *P.*

8.2 Probabilistische Turingmaschinen und Komplexitätsklassen

Für $m \in \mathbb{N}$ betrachten wir $\{0,1\}^m$ als *Wahrscheinlichkeitsraum* mit *Gleichverteilung*: Für jedes $u \in \{0,1\}^m$ ist die Wahrscheinlichkeit

$$\Pr_{y \in \{0,1\}^m}[y = u] = \frac{1}{2^m}.$$

Definition 8.8. Eine *probabilistische Turingmaschine* (PTM) ist eine Turingmaschine, deren Eingabe aus einem Paar $(x, y) \in \Sigma^* \times \{0,1\}^*$ besteht. Dabei bezeichne $x \in \Sigma^*$ die eigentliche Eingabe und $y \in \{0,1\}^*$ ein Zufallswort, das die Berechnung der Maschine steuert.

Eine PTM M heisst $p(n)$-zeitbeschränkt, wenn M auf jede Eingabe (x, y) nach höchstens $p(|x|)$ Schritten hält. Ohne Einschränkung kann dann angenommen werden, dass $|y| = p(|x|)$.

Sei M $p(n)$-zeitbeschränkt. Betrachten wir M als Akzeptor über $\Sigma^* \times \{0,1\}^*$, so erhalten wir eine Sprache $L(M) \subseteq \Sigma^* \times \{0,1\}^*$. Damit definieren wir M als *probabilistischen Akzeptor* über Σ^*. Für $x \in \Sigma^*$ mit $|x| = n$ setzen wir:

$$\begin{aligned} \Pr[M \text{ akzeptiert } x] :=& \quad \Pr_{y \in \{0,1\}^{p(n)}}[(x, y) \in L(M)] \\ =& \quad \frac{|\{y \in \{0,1\}^{p(n)} \ : \ (x, y) \in L(M)\}|}{2^{p(n)}}. \end{aligned}$$

Lemma 8.9. *Eine Sprache $A \subseteq \Sigma^*$ ist genau dann in NP, wenn es eine polynomiale PTM M gibt, so dass $A = \{x \in \Sigma^* \ : \ \Pr[M \text{ akzeptiert } x] > 0\}$.*

Beweis. Sei $A \in$ NP. Dann existiert existiert ein $B \in$ P und ein Polynom $p(n)$, so dass $A = \{x \in \Sigma^* \ : \ \exists y \ (|y| \leq p(|x|) \ \wedge \ (x, y) \in B\}$. Es ist nicht schwer, B und $p(n)$ so zu modifizieren, dass $A = \{x \in \Sigma^* \ : \ (\exists y \in \{0,1\}^{p(|x|)}) \ (x, y) \in B\}$. Sei M eine polynomiale deterministische TM über $\Sigma^* \times \{0,1\}^*$ mit $L(M) = B$. Fassen wir M als probabilistische TM über Σ^* auf, so folgt:

$$\begin{aligned} A &= \{x \in \Sigma^* \ : \ \Pr_{y \in \{0,1\}^{p(n)}}[(x, y) \in L(M)] > 0\} \\ &= \{x \in \Sigma^* \ : \ \Pr[M \text{ akzeptiert } x] > 0\}. \end{aligned}$$

Sei umgekehrt $A = \{x \in \Sigma^* \ : \ \Pr[M \text{ akzeptiert } x] > 0\}$. für eine polynomiale PTM M. Also ist, für ein geeignetes Polynom p, $A = \{x \in \Sigma^* \ : \ \Pr_{y \in \{0,1\}^{p(n)}}[(x, y) \in L(M)] > 0\}$. Dann ist $B := \{(x, y) \in \Sigma^* \times \{0,1\}^{p(|x|)} \ : \ (x, y) \in L(M)\}$ in P und daher $A = \{x \in \Sigma^* : \exists y \ (|y| \leq p(|x|) \ \wedge \ (x, y) \in B\}$ in NP. □

Die Wahrscheinlichkeit für einen Input x für ein NP-Problem durch Raten einen geeigneten Zeugen y zu finden kann sehr klein sein. „Gute“ probabilistische Algorithmen sind solche, die mit „grosser“ Erfolgsaussicht raten. Wir nennen einen probabilistischer Algorithmus für A “stabil”, wenn $\Pr[M \text{ akzeptiert } x]$ für $x \in A$ *deutlich* grösser ist als $\Pr[M \text{ akzeptiert } x]$ für $x \notin A$.

Definition 8.10. Sei $A \subseteq \Sigma^*$ eine Sprache.

- $A \in$ PP (**p**robabilistic **p**olynomial time), wenn eine polynomiale PTM M existiert, mit

  $$A = \{x \ : \ \Pr[M \text{ akzeptiert } x] > \tfrac{1}{2}\}.$$

- $A \in$ BPP (**b**ounded error **p**robabilistic **p**olynomial time), wenn eine polynomiale PTM M existiert, so dass

$$x \in A \implies \Pr[M \text{ akzeptiert } x] \geq \frac{2}{3}\} \text{ und}$$
$$x \notin A \implies \Pr[M \text{ akzeptiert } x] \leq \frac{1}{3}\}.$$

Probabilistische Algorithmen unterliegen zweierlei *Irrtumswahrscheinlichkeiten*:

(1) *Falsch positiv:* $x \notin A$ aber $\Pr[M \text{ akzeptiert } x] > 0$.

(2) *Falsch negativ:* $x \in A$ aber $\Pr[M \text{ akzeptiert } x] < 1$, d.h. $\Pr[M \text{ akzeptiert } x \text{ nicht}] = 1 - \Pr[M \text{ akzeptiert } x] > 0$.

Für die bisher definierten Komplexitäsklassen ergibt sich folgendes Bild:

BPP: beide Irrtumswahrscheinlichkeiten $\leq \frac{1}{3}$,

PP: nur die triviale Schranke, Irrtumswahrscheinlichkeit $\leq \frac{1}{2}$, die schon durch einen Münzwurf erreicht wird,

NP: kein falsch positiver Irrtum, allerdings kann $\Pr[M \text{ akzeptiert } x]$ für $x \in A \subseteq$ NP beliebig klein sein.

Definition 8.11. Der Begriff der Irrtumswahrscheinlichkeit führt uns, zusätzlich zu PP und BPP, zu folgenden probabilistischen Komplexitätsklassen:

- $A \in$ RP (**r**andom **p**robabilistic **p**olynomial time), wenn eine polynomiale PTM M existiert, mit

$$x \in A \implies \Pr[M \text{ akzeptiert } x] \geq \tfrac{2}{3}\} \text{ und}$$
$$x \notin A \implies \Pr[M \text{ akzeptiert } x] = 0\}.$$

  (keine falsch positiven Antworten).

- $A \in$ Co-RP $:\Longleftrightarrow \overline{A} \in$ RP, d.h. es existiert eine polynomiale PTM M, mit

$$x \in A \implies \Pr[M \text{ akzeptiert } x] = 1\} \text{ und}$$
$$x \notin A \implies \Pr[M \text{ akzeptiert } x] \leq \tfrac{1}{3}\}.$$

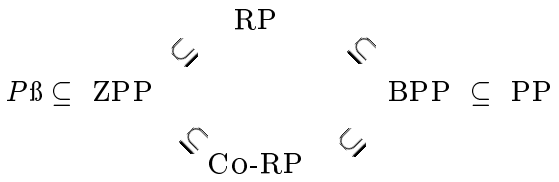
  (keine falsch negativen Antworten).

- $A \in$ ZPP (**z**ero-error **p**robabilistic **p**olynomial time), wenn $A \in$ RP und $A \in$ Co-RP.

Zur Interpretation von ZPP betrachten wir eine Sprache $A \in$ ZPP. Es gibt dann polynomiale PTM M^+ und M^- für $A \in$ RP und und $\overline{A} \in$ RP. Sei nun M eine PTM, welche die Berechnungen von M^+ und M^- parallel simuliert und die Eingabe akzeptiert, falls M^+ sie akzeptiert, bzw. verwirft, falls M^- sie akzeptiert. In dem Fall, dass M^+ verwirft und M^- akzeptiert, gibt M die Antwort „weiss nicht“. Offenbar arbeitet M irrtumsfrei, d.h. die Antworten sind immer korrekt, gibt aber mit Wahrscheinlichkeit $\varepsilon \leq 1/3$ keine befriedigende Antwort. Durch Wiederholung mit unabhängigen Zufallszügen, kann ε beliebig klein gemacht werden.

Beispiel. Der Rabin-Miller Primzahltest (RM) zeigt, dass PRIM $\in$ Co-RP. Adleman und Huang haben 1987 (das viel schwierigere Resultat) gezeigt, dass PRIM auch in RP ist. Also ist PRIM$\in$ ZPP.

Offensichtlich gelten folgende Inklusionen:



Ausserdem gilt: RP $\subseteq$ NP, Co-RP $\subseteq$ Co-NP und ZPP $\subseteq$ NP $\cap$ Co-NP.

Satz 8.12. NP $\subseteq$ PP $\subseteq$ PSPACE.

Beweis. Sei $A \in$ NP. Nach Lemma 8.9 gibt es eine PTM M mit $A = \{x \in \Sigma^* : \Pr[M \text{ akzeptiert } x] > 0\}$. Sei M' eine PTM, welche $(x, y_0y_1y_2\dots)$ genau dann akzeptiert, wenn entweder $y_0 = 1$ oder wenn M $(x, y_1y_2\dots)$ akzeptiert. Dann gilt

$$\Pr[M' \text{ akzeptiert } x] = \frac{1}{2} + \frac{1}{2}\Pr[M \text{ akzeptiert } x].$$

Damit folgt, dass $A = \{x : \Pr[M' \text{ akzeptiert } x] > \frac{1}{2}\} \in$ PP.

Sei $A \in$ PP. Es gilt

$$x \in A \iff \Pr[M \text{ akzeptiert } x] = \Pr_{y\in\{0,1\}^{p(n)}}[(x,y) \in L(M)] > \frac{1}{2}.$$

Nun kann man für Input x mit polynomialem Platz alle Berechnungen von M auf Input (x, y) mit $y \in \{0,1\}^{p(n)}$ simulieren und festellen, ob mehr als $2^{p(n)-1}$ der Paare (x, y) akzeptiert werden. □

Unklar bleibt das Verhältnis von BPP zu NP.

Als nächstes betrachten wir eine Methode zur Reduktion der Irrtumswahrscheinlichkeit eines BPP-Algorithmus. Die grundlegende Idee dabei ist, nach k-maliger Wiederholung eine Mehrheitsentscheidung zu treffen.

Sei M eine $p(n)$-zeitbeschränkte PTM mit Irrtumswahrscheinlichkeit $\leq \varepsilon < \frac{1}{2}$. Sei M^k eine PTM welche $(x, y_1y_2\dots y_k)$, mit $y_i \in \{0,1\}^{p(n)}$, genau dann akzeptiert, wenn $|\{i : (x, y_i) \in L(M)\}| \geq k/2$. Der Algorithmus M^k ist polynomial wenn $k = k(n)$ polynomial ist in n.

Um die Irrtumswahrscheinlichkeit von M^k zu berechnen, benötigen wir ein Ergebnis aus der Wahrscheinlichkeitstheorie.

Seien $X_1, \dots, X_k$ $\{0,1\}$-wertige Zufallsvariablen mit $\Pr[X_i = 1] = p$ und $\Pr[X_i = 0] = 1 - p$ für $0 < p < 1$ (Bernoulli-Zufallsvariablen). Die Summe $X = \sum_{i=1}^{k} X_i$ ist eine $\mathbb{N}$-wertige Zufallsvariable mit Binomialverteilung. Sie hat den Erwartungswert $E(X) = p \cdot k$. Der folgende Satz liefert eine Abschätzung der Wahrscheinlichkeit, dass der Wert von X um weniger als d vom Erwartungswert abweicht:

Lemma 8.13 (Chernoff). *Für $d \geq 0$ gilt*

$$\Pr[X - pk \geq d] \leq e^{-\frac{d^2}{4kp(1-p)}} \leq e^{-\frac{-d^2}{k}} \quad \textit{und} \quad \Pr[pk - X \geq d] \leq e^{-\frac{-d^2}{k}}.$$

Zurück zu unserem Problem, definieren wir für $\bar{y} = y_1 \dots y_k$ (mit $y_i \in \{0,1\}^{p(n)}$) die Zufallsvariablen

$$X_i(\bar{y}) := \begin{cases} 1 & \text{wenn } (x, y_i) \in L(M), \\ 0 & \text{sonst.} \end{cases}$$

Sei A eine Sprache, die durch einen BPP-Algorithmus M mit Irrtumswahrscheinlichkeit $\leq \varepsilon < \frac{1}{2}$ entschieden wird. Dann gilt:

(1) Für $x \notin A$ ist $p := \Pr[X_i = 1] \leq \varepsilon$. Mit $X := \sum_{i=1}^{k} X_i$ ist

$$\Pr[M^k \text{ akzeptiert } x] = \Pr[X \geq \frac{k}{2}].$$

Nach dem Lemma von Chernoff folgt

$$\Pr[X \geq k/2] = \Pr[X - pk \geq k/2 - pk] \leq e^{-(\frac{1}{2}-p)^2 k} = 2^{-\Omega(k)}.$$

Sei $q(n)$ ein beliebiges Polynom. Für $k \geq c \cdot q(n)$ (c geeignete Konstante), erhält man also eine falsch-positive Fehlerwahrscheinlichkeit $\leq 2^{-q(n)}$.

Analoges gilt für die falsch-negative Fehlerwahrscheinlichkeit:

(2) Für $x \in A$ ist $\Pr[M \text{ akzeptiert } x] = \Pr[X_i = 1] = p \geq 1 - \varepsilon$ und

$$\begin{aligned} \Pr[M^k \text{ akzeptiert } x \text{ nicht}] &= \Pr\left[X < \tfrac{k}{2}\right] = \\ &= \Pr\left[pk - X \geq (p - \tfrac{1}{2})k\right] \leq e^{-(p-\frac{1}{2})^2 k} = 2^{-\Omega(k)}. \end{aligned}$$

Damit ist gezeigt:

Satz 8.14. *Zu jeder Sprache $A \in$ BPP und jedem Polynom $q(n)$ gibt es eine polynomiale PTM M, welche A mit Fehlerwahrscheinlichkeit $\leq 2^{-q(n)}$ akzeptiert, d.h.*

$$\begin{aligned} x \in A &\implies \Pr[M^k \textit{ akzeptiert } x] \geq 1 - 2^{-q(|x|)}, \\ x \notin A &\implies \Pr[M^k \textit{ akzeptiert } x] \leq 2^{-q(|x|)}. \end{aligned}$$

Eine analoge Aussage gilt auch für die Klasse RP.

Aus Satz 8.14 ergibt sich eine interessante Folgerung über den Zusammenhang von BPP zur Schaltkreiskomplexität.

Sei M ein BPP-Algorithmus für A mit Fehlerwahrscheinlichkeit $\leq 2^{-q(n)}$. Für alle x gilt:

$$\frac{|\{y \in \{0,1\}^{p(n)} \ : \ (x,y) \in L(M) \iff x \in A\}|}{2^{p(n)}} \geq 1 - 2^{-q(n)}.$$

Daraus folgt, dass es für jede feste Inputlänge n Zufallswerte $y \in \{0,1\}^{p(n)}$ gibt, die für *alle* $x \in \Sigma^n$ das korrekte Ergebnis liefern:

$$\begin{aligned} &|\{y \in \{0,1\}^{p(n)} \ : \ y \text{ „schlecht“ für mindestens ein } x \in \Sigma^n\}| = \\ &\sum_{|x|=n} |\{y \in \{0,1\}^{p(n)} \ : \ y \text{ „schlecht“ für } x\}| \leq |\Sigma^n| \cdot 2^{-q(n)} \cdot 2^{p(n)}. \end{aligned}$$

Wählt man $q(n)$ so, dass $\lim_{n\to\infty} |\Sigma^n| \cdot 2^{-q(n)} = 0$ (z.B. $q(n) = n^2$, oder $q(n) = cn$ mit $c \geq \log|\Sigma|$), dann folgt, dass für grosse n mindestens ein $y(n) \in \{0,1\}^{p(n)}$ die richtige Entscheidung für alle $x \in \Sigma^n$ liefert;

Es gibt also demnach eine Funktion $f : \mathbb{N} \longrightarrow \{0,1\}^*$ mit folgenden Eigenschaften:

- f polynomial beschränkt: $|f(n)| = p(n)$ und
- für alle hinreichend langen $x \in \Sigma^*$ gilt:

$$x \in A \iff \underbrace{(x, f(x)) \in L(M)}_{\text{polynomial}}.$$

Definition 8.15. $A \in \Sigma^*$ ist *nicht-uniform polynomial entscheidbar* ($A \in$ nicht-uniform P), wenn eine Funktion $f : \mathbb{N} \longrightarrow \{0,1\}^*$ und eine Menge $B \in$ P existiert, so dass

- $|f(n)| \leq p(n)$ für ein Polynom p und
- $A = \{x \in \Sigma^* \ : \ (x, f(|x|)) \in B\}$.

Eine solche Funktion f wird *Advice*-Funktion genannt, da sie für jede Inputlänge n eine zusätzliche Information $f(n)$ zur Verfügung stellt welche erlaubt, A für in polynomialer Zeit zu entscheiden. Man beachte, dass f selbst nicht berechenbar sein muss. Für die Klasse nicht-uniform P wird auch die Bezeichnung P/poly, „P mit polynomialem Advice (Rat)“ verwendet. Die zusäzliche Information $f(n)$ kann als Kodierung eines polynomialen Schaltkreises aufgefasst werden, der A für die Inputlänge n entscheidet. In der Tat ist leicht einzusehen, dass

Es gilt: $A \in$ nicht-uniform P $\iff$ A wird von einer Folge von Schaltkreisen polynomialer Grösse entschieden.

Korollar 8.16. BPP $\subseteq$ nicht-uniform P. *Also haben alle Probleme in* BPP *polynomiale Schaltkreiskomplexität.*

Satz 8.17. BPP $\subseteq \Sigma_2^p \cap \Pi_2^p$

Beweis. Es reicht zu zeigen, dass BPP $\subseteq \Sigma_2^p$. Da Co-BPP = BPP folgt dann sofort, dass auch BPP $\subseteq \Pi_2^p$.

Sei $A \in$ BPP. Dann existiert nach Satz 8.14 eine polynomiale PTM M, welche A mit Irrtumswahrscheinlichkeit $< 2^{-n}$ entscheidet:

$$\begin{aligned} x \in A &\implies \Pr_{y \in \{0,1\}^{p(n)}}[M \text{ akzeptiert } x] > 1 - 2^{-n} \\ x \notin A &\implies \Pr_{y \in \{0,1\}^{p(n)}}[M \text{ akzeptiert } x] < 2^{-n} \end{aligned}$$

Insbesondere ist

$$x \in A \iff |\{y \in \{0,1\}^{p(n)} \ : \ (x,y) \in L(M)\}| > 2^{p(n)}(1 - 2^{-n}).$$

Fixiere ein x, $|x| = n$. Sei $\Omega = \{0,1\}^{p(n)}$ und sei $B \subseteq \Omega$. Wir suchen ein Kriterium für die Eigenschaft $|B| > (1 - 2^{-n})|\Omega|$. Die Idee ist, mit „wenigen“ Bildern von B unter Translation modulo 2 ganz Ω zu überdecken.

Für $y, z \in \Omega$ sei $y \oplus z := w_0 \dots w_{p(n)-1} \in \Omega$ mit $w_i = y_i \oplus z_i$ (Bitweise Addition modulo 2). Sei $B \oplus z := \{y \oplus z \ : \ y \in B\}$.

Lemma 8.18. *Für hinreichend grosse n und $B \in \{0,1\}^{p(n)}$, so dass entweder*

(i) $|B| < 2^{-n} \cdot 2^{p(n)}$

oder

(ii) $|B| > (1 - 2^{-n}) \cdot 2^{p(n)}$

gilt: $(ii) \iff \exists \overline{z} = (z_1, \ldots, z_{p(n)}) \in \Omega^{p(n)} : \bigcup_i B \oplus z_i = \{0,1\}^{p(n)}$.

Beweis. $\Rightarrow$: $\bigcup_i B \oplus z_i$ hat höchstens $p(n) \cdot |B|$ Elemente. Wenn also $\bigcup_i B \oplus z_i$ ganz Ω überdeckt, dann ist (i) unmöglich, da $p(n) \cdot 2^{-n}|\Omega| < |\Omega|$ für grosse n.

$\Leftarrow$: (mit probabilistischem Argument)
Fixiere ein $y \in \Omega$ und wähle $z \in B \oplus y$. Unter der Annahme, dass (ii) gilt, folgt:
$\Pr_{z \in \Omega}[y \in B \oplus z] = \Pr_{z \in \Omega}[z \in B \oplus y] = \Pr_{z \in \Omega}[z \in B] > 1 - 2^{-n}$. Daraus folgt:

$$\Pr_{\overline{z} \in \Omega^n}\left[\bigwedge_i y \notin B \oplus z_i\right] \leq \prod_i \Pr_{z_i \in \Omega}[y \notin B \oplus z_i] \leq 2^{-n \cdot p(n)}.$$

Damit ist die Wahrscheinlichkeit, dass ein zufälliges $\overline{z} \in \Omega^n$ die Bedingungen des Lemmas *nicht* erfüllt, wie folgt abschätzbar:

$$\begin{aligned}\Pr_{\overline{z} \in \Omega^n}\left[\bigcup_i B \oplus z_i \neq \Omega\right] &\leq \sum_{y \in \Omega} \Pr_{\overline{z} \in \Omega^n}\left[\bigwedge_i y \notin B \oplus z_i\right] \\ &\leq 2^{p(n)} \cdot 2^{-n \cdot p(n)} < 1 \quad \text{für grosse } n.\end{aligned}$$

Also muss ein „gutes“ $\overline{z}$ existieren. □

Damit können wir A wie folgt darstellen: Sei $B_x = \{y \in \Omega \ : \ (x,y) \in L(M)\}$. Dann gilt:

$$\begin{aligned} x \in A &\implies |B_x| > (1 - 2^{-n}) \cdot 2^{p(n)} \\ x \notin A &\implies |B_x| < 2^{-n} \cdot 2^{p(n)}. \end{aligned}$$

Also

$$\begin{aligned} x \in A &\iff \exists \overline{z} \in \Omega^{p(n)} : \bigcup_{i=1}^{p(n)} B_x \oplus z_i = \Omega \\ &\iff \exists \overline{z} \in \Omega^{p(n)} \forall y \in \Omega \bigwedge_{i=1}^{p(n)} \underbrace{y \in B_x \oplus z_i}_{\equiv y \oplus z \in B_x} \\ &\iff \exists \overline{z} \in \Omega^{p(n)} \forall y \in \Omega \bigwedge_{i=1}^{p(n)} \underbrace{(x, y \oplus z) \in L(M)}_{\text{in P}}. \end{aligned}$$

Also ist $A \in \Sigma_2^p$. □

8.3 Zufallsquellen

Ein grundliegendes Problem, das sich bei der Realisierung von BPP-Algorithmen stellt, besteht in der Erzeugung „echter“ Zufallsfolgen. Angestrebt wäre eine *perfekte Zufallsquelle*, die beliebig lange Folgen $y_1 y_2 \ldots y_n \cdots \in \{0,1\}^*$ produziert, so dass für alle n und alle $u \in \{0,1\}^*$ gilt $\Pr[y_1 y_2 \ldots y_n = u] = 2^{-n}$.

Es ist zweifelhaft, ob perfekte Zufallsquellen physikalisch realisierbar sind. Diese müssten folgenden Anforderungen genügen:

- *Unabhängigkeit:* $\Pr[y_i = 1]$ unabhängig von $y_1 y_2 \dots y_{i-1}$, d.h. für alle $E \subseteq \{0,1\}^{i-1}$ soll gelten $\Pr[y_1 y_2 \dots y_{i-1} \in E \wedge y_i = 1] = \Pr[y_1 y_2 \dots y_{i-1} \in E] \cdot \Pr[y_i = 1]$;
- *Fairness:* $Pr[y_i = 1] = \frac{1}{2}$.

Die Fairness-Anforderung ist dabei kein echtes Problem. Mit dem folgenden von von Neumann stammenden Verfahren, lässt sich jede nicht faire Quelle von unabhängigen Zufallsbits $y_1 y_2 \dots$ in eine perfekte Zufallsquelle umwandeln:

Zerlege die Folge $y_1 y_2 \dots$ in Paare;
Interpretiere 01 als 0,
10 als 1,
Ignoriere 00 und 11.

Beispiel.

0 0 0 1 1 0 0 0 0 1 0 0 0 1 1 1 1 0 ...
0 0|0 1|1 0|0 0|0 1|0 0|0 1|1 1|1 0| ...
1 1 0 0 1 ...

Beachte: Die Wahrscheinlichkeit $p = \Pr[y_i = 1]$ braucht nicht bekannt zu sein (solange $0 < p < 1$).

Um mit dem obigen Verfahren eine perfekte Zufallsfolge der Länge n zu erhalten, brauchen wir eine Folge der gegebenen Quelle mit *erwarteter* Länge $\frac{2}{1-c}$, wobei $c = p^2 + (1-p)^2$ die sogenannte *Koinzidenz-Wahrscheinlichkeit* darstellt.

Das echte Problem stellt die *Unabhängigkeits*-Anforderung dar. Durch physikalische Prozesse scheint Unabhängigkeit sehr schwierig zu realisieren (und nachzuweisen!).

Es gibt hier verschiedene Ansätze:

- Durch mathematische Konstruktion lassen sich *Pseudo-Zufallsreihen* erzeugen (mit besonderer Anwendung in der Kryptologie). Allerdings beruht die Unabhängigkeit dieser Reihen gemeinhin auf unbewiesenen komplexitätstheoretischen Vermutungen.
  Achtung: In vielen Computer-Systemen werden Pseudo-Zufallszahlen mit Kongruenzen erzeugt:

  Fixiere a, b, c.
  Gegeben sei ein Startwert $x_0 \in \mathbb{N}$
  Für $i > 0$: $x_i = a x_{i-1} + b \pmod{c}$.

  Es ist inzwischen bekannt, dass solche Pseudo-Zufallszahlen relativ wertlos sind. Es ist leicht, aus einer Folge $x_0 \dots x_i$ den nächsten Wert x_{i+1} (und sogar die „geheimen“ Parameter a, b, c) auszurechnen.

- *Schwache Zufallsquellen:* Sei $0 < \delta < \frac{1}{2}$ und $p : \{0,1\}^* \longrightarrow [\delta, 1-\delta]$ eine beliebige (unbekannte) Funktion. Die *δ-Zufallsquelle* S_p erzeugt Bitfolgen $y_1 y_2 \dots y_n \dots$, so dass für alle $n \in \mathbb{N}$ und $u = u_1 \dots u_n \in \{0,1\}^n$ gilt:

$$\Pr[y_1 \dots y_n = u] = \prod_{i=1}^{n} \left(u_i p(u_1 \dots u_{i-1}) + (1-u_i)(1 - p(u_1 \dots u_{i-1}))\right).$$

Die Wahrscheinlichkeit $\Pr[y_i = 1] = p(y_1 \dots y_{i-1}) \in [\delta, 1-\delta]$ hängt also in beliebig komplizierter Weise von vorher erzeugten Bits ab, aber diese Abhängigkeit von der Vergangenheit bestimmt ein Bit höchstens mit Wahrscheinlichkeit $1 - \delta < 1$. Eine perfekte Zufallsquelle wäre eine $\frac{1}{2}$-Zufallsquelle. Für $\delta < \frac{1}{2}$ nennen wir die δ-Zufalsquellen schwach.

Beispiel. Der radioaktive Zerfall, Würfel- und Münzwurf.

Leider kann man schwache Zufallsquellen nicht *direkt* zur Steuerung von BPP-Algorithmen verwenden (ohne die vom BPP-Algorithmus geforderten Eigenschaften möglicherweise zu zerstören). Allerdings ist eine indirekte Simulation möglich.

Definition 8.19. Sei $0 < \delta < \frac{1}{2}$. Eine Sprache A ist in δ-BPP, wenn eine polynomiale PTM M existiert, so dass für *alle* durch δ-Zufallsquellen definierten Wahrscheinlichkeitsverteilungen auf $\{0,1\}^{p(n)}$ gilt:

$$x \in A \implies \Pr[M \text{ akzeptiert } x] \geq \frac{2}{3}$$
$$x \notin A \implies \Pr[M \text{ akzeptiert } x] \leq \frac{1}{3}.$$

Lemma 8.20. *Es gilt:*

(i) $0\text{-BPP} = \text{P}$.

(ii) $\frac{1}{2}\text{-BPP} = \text{BPP}$.

Beweis. Eine 0-Zufallsquelle kann einem beliebigen $y \in \{0,1\}^{p(n)}$ die Wahrscheinlichkeit 1 geben. Da alle 0-Zufallsquellen die BPP-Bedingung erfüllen, muss $(x,y) \in L(M)$ für alle y oder kein y gelten. Daher ist $0\text{-BPP} = \text{P}$. Die zweite Behauptung ist offensichtlich. □

Nun stellt sich die Frage, was geschieht, wenn $0 < \delta < \frac{1}{2}$? Eine Antwort darauf gibt

Satz 8.21 (U. Vazirani, V. Vazirani). $\delta\text{-BPP} = \text{BPP}$ *für alle* $\delta > 0$.

Beweis. Sei $A \in \text{BPP}$, M ein BPP-Algorithmus für A mit Fehlerwahrscheinlichkeit $\leq \frac{1}{32}$ und M' eine PTM, die von der δ-Zufallsquelle S_p gesteuert wird. Wir setzen

$$m := p(n) \text{ und } k := \left\lceil \frac{\log(m) + 5}{2(\delta - \delta^2)} \right\rceil = O(\log(n)).$$

Ein *Block* sei ein $\{0,1\}$-Wort der Länge k. Die Menge dieser Blocks, $\{0,1\}^k$, betrachten wir als k-dimensionalen Vektorraum über $\mathbb{F}_2$. Das Skalarprodukt zweier Blocks ist dann definiert durch

$$\begin{aligned} \langle \cdot, \cdot \rangle : \quad & \mathbb{F}_2^k \times \mathbb{F}_2^k \longrightarrow \mathbb{F}_2 \\ & (u,v) \longmapsto \langle u, v \rangle = \textstyle\sum_{i=1}^k u_i v_i \pmod 2. \end{aligned}$$

Die Maschine M' arbeitet nach folgendem Algorithmus: Sei $y \in \{0,1\}^k (= \mathbb{F}_2^k)$ ein Block, generiert von der Zufallsquelle S_p. Erzeuge die 2^k Bits $\langle y, v \rangle$ für alle $v \in \mathbb{F}_2^k$. Wiederhole dies für $m = p(n)$ Generationen von y und verwende die so erzeugten 2^k Bitfolgen der Länge m, um 2^k

Berechnungen von M auf x zu simulieren. Akzeptiere/verwerfe gemäss Majoritätsvotum. Dann gilt:

$$M' \text{ akzeptiert } (x, \underbrace{y(1)y(2)\dots y(n)}_{\in\{0,1\}^{k\cdot m}}) \quad [\text{ wobei } y(i) \in \mathbb{F}_2^k \text{ }]$$
$$\text{gdw} \quad |\{v \in \mathbb{F}_2^k \ : \ (x, \underbrace{\langle y(1), v\rangle \dots \langle y(m), v\rangle}_{\in\{0,1\}^m} \in L(M)\}| \geq 2^{k-1}.$$

Dieser probabilistische Algorithmus für M' hat Zeitkomplexität $2^k \cdot p(n)$, ist also polynomial (da $k = O(\log n)$).

Betrachten wir den Input x und eine von S_p erzeugte Bitfolge $\bar{y} = y(1)\dots y(n) \in \{0,1\}^{k\cdot m}$. M' auf Input $(x, \bar{y})$ simuliert M für die 2^k $\{0,1\}$-Werte aus

$$T = \{\langle y(1), v\rangle \dots \langle y(m), v\rangle \ : \ v \in \mathbb{F}_2^k\} \subseteq \{0,1\}^m.$$

Die Menge der $y \in \{0,1\}^m$, auf denen $M(x,y)$ eine falsche Antwort liefert, nennen wir

$$B_x := \{y \in \{0,1\}^m \ : \ [(x,y) \in L(M) \oplus x \in A]\}.$$

Nach Voraussetzung gilt $|B_x| \leq \frac{1}{32}2^n = 2^{n-5}$.
Die Wahrscheinlichkeit, dass M' auf x eine falsche Antwort liefert, ist $\Pr\left[|T \cap B_x| \geq \frac{1}{2}|T|\right]$.

Behauptung: $\Pr\left[|T \cap B_x| \geq \frac{1}{2}|T|\right] \leq \frac{1}{3}$.

Betrachte dazu ein Bit $z = \langle y, v\rangle$ für $y \in \{0,1\}^k$ generiert von S_p und $v \in \mathbb{F}_2^k$. Der *Bias* dieser Grösse z sei definiert als

$$\text{bias}(z) := (Pr[z = 1] - Pr[z = 0])^2.$$

Idee: Wir werden zeigen, dass *im Durchschnitt über alle* $v \in \mathbb{F}_2^k$ der Bias von $\langle y, v\rangle$ nach oben beschränkt ist:
Sei $u \in \{0,1\}^k = \mathbb{F}_2^k$ und $p[u] := \Pr_{S_p}[y = u]$, die Wahrscheinlichkeit, dass S_p den Block u erzeugt. Dann ist die *Koinzidenz-Wahrscheinlichkeit*, d.h. die Wahrscheinlichkeit, dass bei zwei Experimenten S_p denselben Block erzeugt: $\sum_{u\in\mathbb{F}_2^k} p[u]^2$.

Lemma 8.22. *Für $y \in \{0,1\}^k$ gilt:*

$$\frac{1}{2^k}\sum_{v\in\mathbb{F}_2^k} \text{bias}(\langle y, v\rangle) = \sum_{u\in\mathbb{F}_2^k} p[u]^2.$$

„Durchschnittlicher Bias = Koinzidenz-Wahrscheinlichkeit".

Beweis. Einerseits gilt:

$$\Pr[\langle y, v\rangle = 0] - \Pr[\langle y, v\rangle = 1] = \sum_{u\in\mathbb{F}_2^k} (-1)^{\langle y,v\rangle} p[u],$$

$$\text{denn } \Pr[\langle y, v\rangle = 0] = \sum_{\langle y,v\rangle=0} p[u]; \quad (-1)^0 = -1 \text{ und}$$
$$\Pr[\langle y, v\rangle = 1] = \sum_{\langle y,v\rangle=1} p[u]; \quad (-1)^1 = -1.$$

Damit folgt:

$$\begin{aligned}\sum_{v\in\mathbb{F}_2^k} \text{bias}(\langle y,v\rangle) &= \sum_{v\in\mathbb{F}_2^k} (\Pr[\langle y,v\rangle = 0] - \Pr[\langle y,v\rangle = 1])^2 \\ &= \sum_{v\in\mathbb{F}_2^k} \left(\sum_{u\in\mathbb{F}_2^k} (-1)^{\langle y,v\rangle}\right)^2 \\ &= \sum_{v\in\mathbb{F}_2^k}\sum_{u\in\mathbb{F}_2^k} p[u]^2 + \sum_{v\in\mathbb{F}_2^k}\sum_{u,u'\in\mathbb{F}_2^k} p[u]p[u'] \\ &= 2^k \cdot \sum_{u\in\mathbb{F}_2^k} p[u]^2 + \sum_{u\neq u'} p[u]p[u'] \underbrace{\left(\sum_v (-1)^{\langle y,v\rangle}\right)}_{0}.\end{aligned}$$

□

Lemma 8.23.

$$\sum_{u\in\mathbb{F}_2^k} p[u]^2 \leq (\delta^2 + (1-\delta)^2)^k.$$

„Die Koinzidenzwahrscheinlichkeit ist beschränkt."

Beweis. (Skizze) Nach Definition ist

$$p[u] = \Pr[y = u] = \prod_{i=1}^{k}((u_i \cdot p(u_1 \dots u_{i-1}) + (1-u)(1-p(u_1 \dots u_{i-1})).$$

Für ein einzelnes Bit sei $p_i := \Pr[y_i = 1] = p(y_1 \dots y_{i-1})$.

Betrachte	$u = u_1 \dots u_{k-1}1$	$p[u] = Ap_k$
	$u' = u_1 \dots u_{k-1}0$	$p[u'] = A(1-p_k)$.

Es gilt

$$\sum_{u\in\mathbb{F}_2^k} p[u]^2 = \sum_{u\in\mathbb{F}_2^{k-1}} \left(p[u1]^2 + p[u0]^2\right) = \sum_{u\in\mathbb{F}_2^{k-1}} B \underbrace{\left(p_k^2 + (1-p_k)^2\right)}_{\substack{\text{maximal, wenn } p_k \text{ und } 1-p_k \text{ so weit} \\ \text{wie möglich ausein-} \\ \text{anderliegen: } \delta, 1-\delta}}.$$

Für die ganze Bitfolge erhält man dann:

$$\sum_{u\in\mathbb{F}_2^k} p[u]^2 \leq \sum_{i=1}^{k} \binom{k}{i} \delta^2 (1-\delta)^{2(k-i)} = (\delta^2 + (1-\delta)^2)^k.$$

□

Nach Lemma8.22 und 8.23 gilt:

(a)

$$\sum_{v \in \mathbb{F}_2^k} \text{bias}(\langle y, v \rangle) \leq 2^k (\delta^2 + (1-\delta)^2)^k$$

Wir sagen $\langle y, v \rangle$ ist *gut*, wenn $\text{bias}(\langle y, v \rangle) \leq \frac{1}{m^2}$ (für $m = p(n)$, Laufzeit von M); sonst nennen wir $\langle y, v \rangle$ *verdorben.*

(b)

$$\langle y, v \rangle \text{ gut} \quad \Longrightarrow \quad \Pr[\langle y, v \rangle = 1] \in \left[\frac{1}{2} - \frac{1}{2m}; \frac{1}{2} + \frac{1}{2m}\right],$$

$$(\text{sonst } (\Pr[\langle y, v \rangle = 1] - \Pr[\langle y, v \rangle = 0])^2 \geq \frac{1}{m^2} \quad \Longrightarrow \quad \langle y, v \rangle \text{ vedorben}).$$

(c) Es gibt höchstens $m^2 2^k (\delta^2 + (1-\delta)^2)^k$ verdorbene Bits $\langle y, v \rangle$ (für festes $y \in \{0,1\}^m$ von S_p). Insgesamt werden in der Simulation also höchstens $m^3 2^k (\delta^2 + (1-\delta)^2)^k$ verdorbene Bits verwendet.

(d)

$$\begin{aligned} m^3 2^k (\delta^2 + (1-\delta)^2)^k &= m^3 2^k (1 - 2\delta + 2\delta^2)^{\frac{3 \log m + 5}{2\delta - 2\delta^2}} \\ &= m^3 2^k 2^{\log(1-2\delta+2\delta^2) \frac{3 \log m + 5}{2\delta - 2\delta^2}} \\ &\leq m^3 2^k \underbrace{2^{-3 \log m - 5}}_{m^{-3} 2^{-5}} = 2^{k-5} \end{aligned}$$

(e) $y = \langle y(1), v \rangle \ldots \langle y(m), v \rangle$ sei *verdorben*, wenn mindestens ein $\langle y(i), v \rangle$ verdorben ist.
Setze $U \subseteq T$ die Menge der unverdorbenen Folgen.
Aus (d) folgt: T enthält höchstens 2^{k-5} verdorbene Folgen.

Lemma 8.24.

$$E(|B_x \cap T|) \leq \frac{|T|}{8}$$

Beweis.

$$\begin{aligned} E(|B_x \cap T|) &= \sum_{t_1 \ldots t_m \in T} \sum_{b_1 \ldots b_m \in B} \prod_{i=1}^{m} \Pr[b_i = t_i] \\ &\leq \underbrace{2^{k-5}}_{\substack{\text{verdorbene} \\ \text{Folgen in } T}} + \sum_{t_1 \ldots t_m \in U} \sum_{b_1 \ldots b_m \in B} \prod_{i=1}^{m} \Pr[b_i = t_i] \\ &\leq 2^{k-5} + \sum_{t_1 \ldots t_m \in U} \sum_{b_1 \ldots b_m \in B} \left(\frac{1}{2} + \frac{1}{2m}\right)^m, \\ &\qquad (\text{da für unverdorbene } t_i : \Pr[b_i = t_i] \leq \frac{1}{2} + \frac{1}{2m}) \\ &\leq 2^{k-5} + 2^k \cdot 2^{m-5} \cdot e \cdot 2^{-m} = 2^{k-5}(1+e) \leq 2^{k-3} = \frac{|T|}{8}. \end{aligned}$$

□

Zum Abschluss unseres Beweises fehlt uns nur noch ein Lemma aus der Wahrscheinlichkeitstheorie:

Lemma 8.25. *Sei X eine $\mathbb{N}$-wertige Zufallsvariable und $d > 0$. Dann gilt* $\Pr[X \geq dE(X)] \leq \frac{1}{d}$.

Für $n \in \mathbb{N}$ sei $p_n := Pr[X = n]$. Nun ist

$$E(X) = \sum_{n \in \mathbb{N}} np_n = \sum_{n < dE(X)} np_n + \sum_{n \geq E(X)} np_n \leq d \cdot E(X) \cdot \Pr[X \geq d \cdot E(X)]$$

$$\Longrightarrow \Pr[X \geq d \cdot E(X)] \leq \frac{1}{d}$$

□

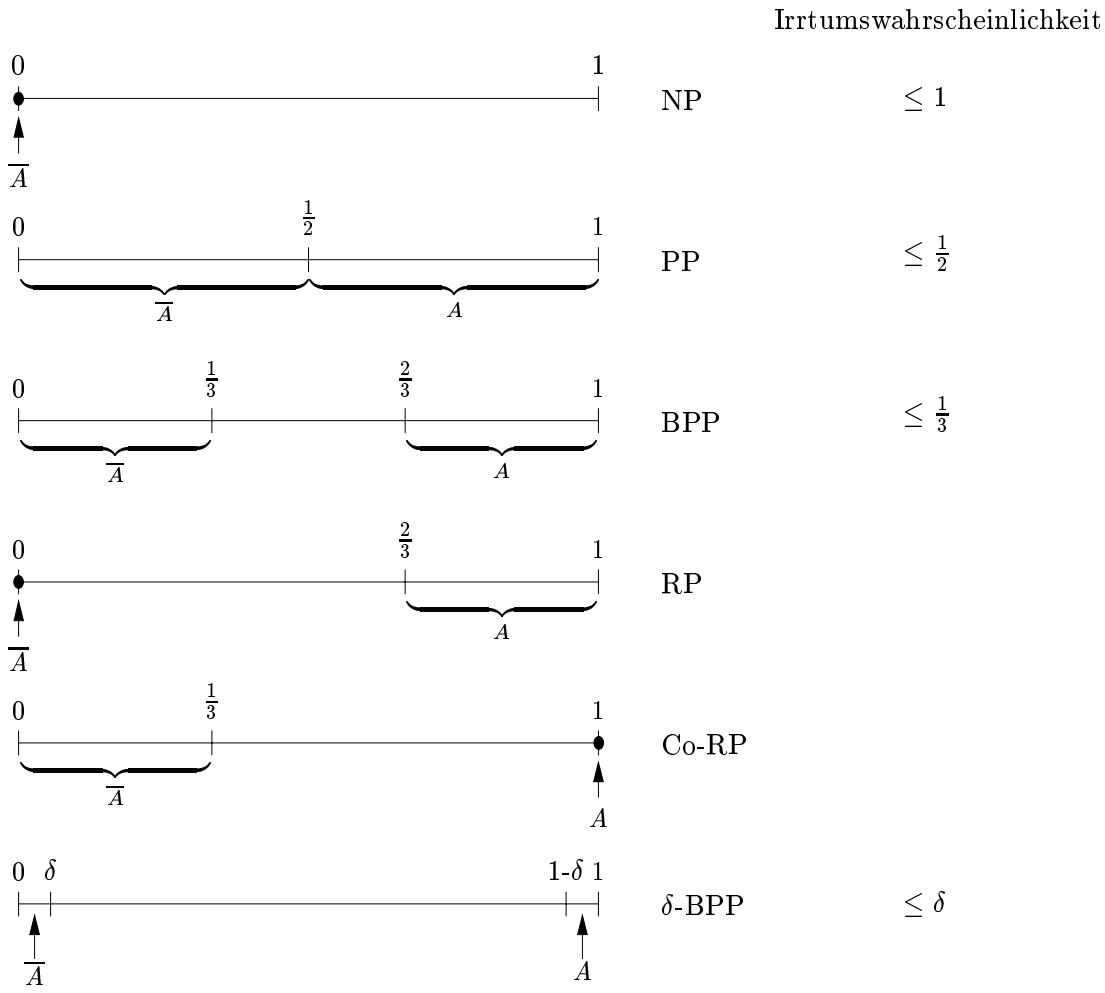


Abbildung 8.1: Übersicht der Klassen

8.4 Probabilistische Beweissysteme und Artus-Merlin Spiele

`folgt später`