# Vorlesung zur Komplexitätstheorie

**Hans Ulrich Simon**

**Sommersemester 2004**

# 1 Grundbegriffe der Komplexitätstheorie

## 1.1 Das Standardmodell der Turing-Maschine

**Definition 1.1 (deterministische 1-Band Turing-Maschinen)** *Eine deterministische 1-Band Turing-Maschine (kurz: 1-Band DTM) $M$ besteht aus den folgenden Komponenten:*

1. *endliche Zustandsmenge $Z$*
2. *Eingabealphabet $\Sigma$*
3. *Arbeitsalphabet $\Gamma \supseteq \Sigma$*
4. *Blank (Leerzeichen) $\square \in \Gamma \setminus \Sigma$*
5. *Startzustand $z_0 \in Z$*
6. *Überführungsfunktion $\delta : Z \times \Gamma \to Z \times \Gamma \times \{L, N, R\}$*
7. *Menge $Z_+ \subseteq Z$ der akzeptierenden Zustände*

*Schreibweise:* $M = (Z, \Sigma, \Gamma, \square, z_0, \delta, Z_+)$.

Anschaulich gesprochen (vgl. Abbildung 1) besteht eine DTM $M$ aus einem zweiseitig unendlichen Band, das in (ganzzahlig durchnumerierte) Zellen unterteilt ist, einem Schreib-Lese-Kopf (kurz: Kopf) und einer durch Zustandsmenge $Z$ gegebenen „endlichen Kontrolle“. Wir beschreiben im Folgenden die Arbeitsweise von $M$.

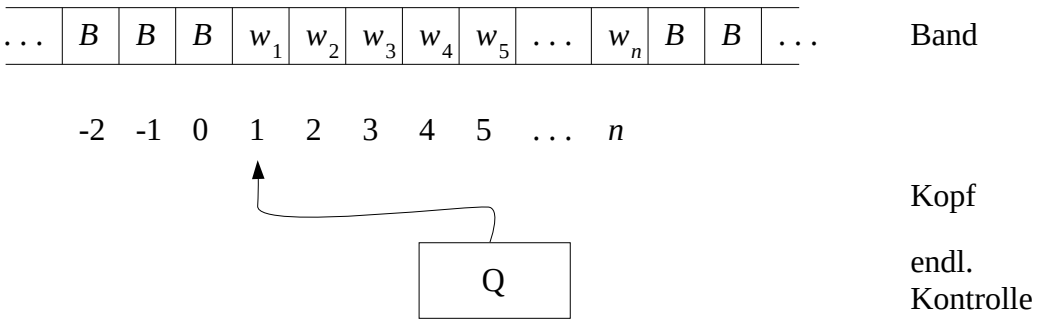



Abbildung 1: Standardmodell der Turing Maschine.

**Beginn einer Rechnung** Anfangs befindet sich auf dem Band (in den Zellen $1, \ldots, n$) ein Eingabewort $w = (w_1, \ldots, w_n) \in \Sigma^n$. In allen anderen Zellen befindet sich das Leerzeichen $\square$. Der Kopf ist auf Zelle 1 positioniert (und liest folglich das Symbol $w_1$) und die DTM $M$ wird im Zustand $z_0$ gestartet.

**Folge von Rechenschritten** Aus der beschriebenen „Startkonfiguration“ heraus vollzieht $M$ eine Folge von Rechenschritten, die durch die Überführungsfunktion $\delta$ fest gelegt sind. Genauer gesagt gilt folgendes. Wenn $M$ im Zustand $z \in Z$ das Symbol $X \in \Gamma$ liest und

$$(z', X', d) = \delta(z, X) \ ,$$

dann ersetzt $M$ das Symbol $X$ durch $X' \in \Gamma$, bewegt den Kopf in Richtung $d \in \{L, N, R\}$ und geht in den Zustand $z' \in Z$ über. Hierbei bedeutet $d = L$ bzw. $d = R$ eine Kopfbewegung auf die linke bzw. rechte Nachbarzelle; im Falle von $d = N$ verharrt der Kopf in seiner alten Position.

**Ende einer Rechnung** Ein Spezialfall eines „Rechenschrittes“ liegt vor, wenn $z' = z$, $X' = X$ und $d = N$. In diesem Fall ergeben sich durch weitere Anwendungen der Überführungsfunktion $\delta$ keine Veränderungen von Zustand, Bandinschrift oder Kopfposition. Wenn ein solcher „Rechenschritt“ zur Anwendung kommt, sagen wir, dass die Rechnung von $M$ auf der Eingabe $w$ *stoppt*. Ein Zustand $z_* \in Z$ heißt *Stoppzustand*, falls

$$\forall X \in \Gamma : \delta(z_*, X) = (z_*, X, N) \ .$$

Wenn die Rechnung von $M$ also einen Stoppzustand erreicht, dann wird sie in jedem Falle stoppen (egal welches Symbol $X$ vom Kopf gerade gelesen wird).

**Ergebnis einer Rechnung** Bleibt zu klären, was wir nach dem Stoppen von $M$ als das Ergebnis der Rechnung auffassen. Wir unterscheiden dabei DTMs, die als Akzeptoren von Sprachen $L \subseteq \Sigma^*$ auftreten und DTMs, die als Berechner von (partiell definierten) Funktionen $f : \Sigma^* \to \Sigma^*$ auftreten.
Im ersten Fall sind wir nur daran interessiert, ob die Eingabe $w$ von $M$ akzeptiert oder verworfen wird. Wir sagen, dass $M$ das Eingabewort $w$ *akzeptiert*, wenn die Rechnung von $M$ auf Eingabe $w$ nach endlichen vielen Schritten in einem akzeptierenden Zustand $z_+ \in Z_+$ stoppt. Falls dies nicht passiert, sagen wir $M$ *verwirft* das Eingabewort $w$. Beachte, dass es zwei qualitativ unterschiedliche Arten des Verwerfens gibt:

- Stoppen in einem (verwerfenden) Zustand $z_- \in Z \setminus Z_+$,
- Vollziehen einer (niemals stoppenden) Endlosrechnung.[1]

$M$ heißt dann *Akzeptor* der Sprache

$$L_M := \{w \in \Sigma^* | \ M \text{ akzeptiert } w\} \ .$$

Die Menge

$$H_M := \{w \in \Sigma^* | \ M \text{ stoppt auf Eingabe } w \text{ nach endlich vielen Schritten}\}$$

heißt der *Haltebereich* von $M$. Offensichtlich gilt $L_M \subseteq H_M$.
Betrachten wir nun noch den Fall, dass $M$ als Funktionsberechner verwendet wird. Wir

[1] die in deutschen Amtsstuben präferierte Art des Verwerfens

sagen, dass $M$ auf Eingabe $x \in \Sigma^*$ nach endlich vielen Schritten *Ausgabe* $y \in \Sigma^*$ produziert, falls $M$ auf Eingabe $x$ nach endlich vielen Schritten stoppt und zu diesem Zeitpunkt die Bandbeschriftung $y$ (eingerahmt von Blanks) vorliegt und der Kopf sich auf dem ersten Symbol von $y$ befindet. Die von $M$ berechnete (partiell definierte) Funktion ist dann gegeben durch

$$f_M(x) = y :\Leftrightarrow \ M \text{ produziert auf Eingabe } x \text{ die Ausgabe } y \ .$$

Wenn $M$ auf Eingabe $x$ keine Ausgabe produziert, dann betrachten wir $f(x)$ als undefiniert.

Wir haben anhand des anschaulichen Modells der Turing-Maschine (unter Verwendung der Begriffe „Band“, „Kopf“ und „endliche Kontrolle”) definiert, wie eine „Rechnung“ der DTM $M$ auf einer Eingabe $x$ (gesteuert durch die Überführungsfunktion $\delta$) sich entfaltet. Eine solche Rechnung kann auch formaler als Folge von Strings beschrieben werden, wobei jeder String eine sogenannte „Konfiguration“ oder „Momentaufnahme“ des Rechenprozesses repräsentiert. Intuitiv gesprochen sollte eine Momentaufnahme alle Informationen enthalten, die es ermöglichen die Rechnung zu unterbrechen und irgendwann später weiter laufen zu lassen. Im Falle der Turing-Maschine genügt es, sich den aktuellen Zustand, die aktuelle Bandposition und die aktuelle Bandbeschriftung zu merken. Dies führt zu folgender

**Definition 1.2 (Konfiguration)** *Eine Konfiguration einer 1-Band DTM $M$ ist ein String von der Form $\alpha z \beta$ mit $\alpha, \beta \in \Gamma^*$ und $z \in Z$.*

Interpretation: $M$ befindet sich im Zustand $z$, die Bandbeschriftung ist $uv$ (eingerahmt von Leerzeichen) und der Kopf ist auf dem ersten Zeichen von $v$ positioniert (bzw. auf einem Leerzeichen positioniert, falls $v = \epsilon$). Die folgenden Definitionen ergeben sich auf natürliche Weise aus unserer anschaulichen Vorstellung von der Rechnung einer DTM:

**Definition 1.3 (Spezielle Konfigurationen)** *Der String $z_0 w$ bezeichnet die* Anfangskonfiguration *der DTM $M$ bei Eingabe $x$. Ein String der Form $\alpha z \beta$ mit $\beta = X\beta'$ für ein $X \in \Gamma$, $\beta' \in \Gamma^*$ bezeichnet eine* Endkonfiguration *von $M$, falls $\delta(z, X) = (z, X, N)$. Gilt darüber hinaus, dass $z \in Z_+$ bzw. $z \in Z \setminus Z_+$, so sprechen wir von einer* akzeptierenden *bzw. von einer* verwerfenden Endkonfiguration.

Offensichtlich wird eine Endkonfiguration genau dann erreicht, wenn die Rechnung der DTM stoppt.

**Definition 1.4 (Folgekonfigurationen)** *Die Konfiguration $\alpha' z' \beta'$ heißt* direkte Folgekonfiguration *von $\alpha z \beta$, falls sie sich durch einen Rechenschritt von $M$ aus der Konfiguration $\alpha z \beta$ ergibt.*[2] *In Zeichen: $\alpha z \beta \vdash_M \alpha' z' \beta'$.*
*Die Konfiguration $\alpha' z' \beta'$ heißt* Folgekonfiguration *von $\alpha z \beta$, falls sie sich durch eine (evtl. leere) Folge von Rechenschritten von $M$ aus der Konfiguration $\alpha z \beta$ ergibt. In Zeichen: $\alpha z \beta \vdash_M^* \alpha' z' \beta'$.*

Wenn die DTM $M$ sich aus dem Kontext ergibt, dann schreiben wir im Folgenden einfach „$\vdash$“ statt „$\vdash_M$“ bzw. „$\vdash^*$“ statt „$\vdash_M^*$“. Offensichtlich ist „$\vdash^*$“ die reflexive-transitive Hülle der Relation „$\vdash$“. Aufbauend auf dem Konzept der Konfiguration können wir nun die Definition der von einer DTM akzeptierten Sprache bzw. ihres Haltebereiches formulieren wie folgt:

[2] Die formalen Details dieser (etwas saloppen) Definition sind leicht einzufüllen (s. Übung).

**Definition 1.5 (Sprache und Haltebereich einer DTM)** *Die* Sprache *der von DTM $M$ akzeptierten Wörter ist gegeben durch*

$$L_M := \{w \in \Sigma^* |\ \exists \alpha, \beta \in \Gamma^*, \exists z_+ \in Z_+ : z_0 w \vdash_M^* \alpha z_+ \beta \text{ und } \alpha z_+ \beta \text{ ist eine Endkonfiguration}\} \ .$$

*Der* Haltebereich *der DTM $M$ örter ist gegeben durch*

$$H_M := \{w \in \Sigma^* |\ \exists \alpha, \beta \in \Gamma^*, \exists z \in Z : z_0 w \vdash_M^* \alpha z \beta \text{ und } \alpha z \beta \text{ ist eine Endkonfiguration}\} \ .$$

In analoger Weise könnte man die Definition von $f_M$ auf dem Konzept der Konfigurationen aufbauen.

Die formale Beschreibung einer Rechnung als Folge von Konfigurationen (anstelle der anschaulichen Beschreibung) ist keine rein mathematische Spielerei. Vielmehr ermöglicht die Darstellung einer Rechnung als Folge von Strings, Rechnungen ihrerseits maschinell zu verarbeiten. In der Praxis ist dies zum Beispiel bei Betriebssystemen relevant, die ja gleichzeitig mehrere laufende Prozesse verwalten (und zu diesem Zweck die Prozesse durch Konfigurationen repräsentieren). Bei Turing-Maschinen werden wir Konfigurationen mitunter benutzen, um eine Maschine mit Hilfe einer anderen Maschine zu simulieren.

## 1.2 Varianten des Standardmodells

### 1.2.1 Mehrband Turing-Maschinen

Eine *deterministische $k$-Band Turing-Maschine* (kurz: $k$-Band DTM) unterscheidet sich von unserem Standardmodell einer 1-Band DTM nur dadurch, dass sie über $k$ Bänder (mit jeweils einem Kopf) verfügt, so dass $k$ Bandbeschriftungen parallel manipuliert werden können. Formal drückt sich das dadurch aus, dass die Überführungsfunktion von der Form

$$\delta : Z \times \Gamma^k \to Z \times \Gamma^k \times \{L, N, R\}^k$$

ist. Die Gleichung

$$(z', X_1', \ldots, X_k', d_1, \ldots, d_k) = \delta(z, X_1, \ldots, X_k)$$

hat die offensichtliche Interpretation (nämlich?).
Die Eingabe einer $k$-Band DTM steht auf Band 1. Falls eine Ausgabe produziert wird, so ist diese auf Band $k$ zu finden. Die Definitionen im Zusammenhang mit Konfigurationen, Sprache, Haltebereich und der von einer DTM berechneten Funktion gelten analog.

### 1.2.2 Nicht-deterministische Turing-Maschinen

Eine *nicht-deterministische* 1*-Band Turing-Maschine* (kurz: 1-Band NTM) unterscheidet sich von unserem Standardmodell einer 1-Band DTM dadurch, dass sie in einem Rechenschritt i.A. mehrere Alternativen hat. Formal drückt sich das dadurch aus, dass die Überführungsfunktion von der Form

$$\delta : Z \times \Gamma^k \to \mathcal{P}\left(Z \times \Gamma \times \{L, N, R\}\right)$$

ist, wobei $\mathcal{P}(S)$ die Potenzmenge einer Menge $S$ bezeichnet. Die Bedingung

$$(z', X', d) \in \delta(z, X)$$

hat die folgende Interpretation: wenn $M$ im Zustand $z \in Z$ das Symbol $X \in \Gamma$ liest, dann hat sie die Option, $X$ durch $X' \in \Gamma$ zu ersetzen, den Kopf in Richtung $d$ zu bewegen und in den Zustand $z' \in Z$ überzugehen. Eine der $|\delta(z, X)|$ vielen Optionen muss sie andererseits auswählen.
$k$-Band NTMs sind in Analogie zu $k$-Band DTMs definiert (NTMs mit $k$ Bändern und einem Kopf pro Band).

**Beobachtung** Eine NTM hat auf einer Eingabe $w \in \Sigma^n$ i.A. viele potenzielle Rechnungen, da sich pro Rechenschritt mehrere Alternativen bieten können. Daher ist die direkte Folgekonfiguration einer Konfiguration i.A. nicht mehr eindeutig bestimmt und „$\vdash_M$“ sowie „$\vdash_M^*$“ sind im Falle einer NTM $M$ echte Relationen.

Dies wirft die Frage auf, wann eine Eingabe als akzeptiert gilt. Die folgende Definition klärt uns darüber auf:

**Definition 1.6 (Sprache einer NTM)** *Die* Sprache *der von NTM $M$ akzeptierten Wörter ist gegeben durch*

$$L_M := \{w \in \Sigma^* \mid \exists \alpha, \beta \in \Gamma^*, \exists z_+ \in Z_+ : z_0 w \vdash_M^* \alpha z_+ \beta \text{ und } \alpha z_+ \beta \text{ ist eine Endkonfiguration}\} \ .$$

Definitionen 1.6 und 1.5 unterscheiden sich nur in einem Buchstaben (NTM statt DTM). Dennoch ist der Unterschied zwischen beiden Definitionen dramatisch. Für NTMs hat nämlich der Existenzquantor innerhalb der Definition von $L_M$ eine völlig neue Qualität bekommen: bei DTMs ist die Konfiguration, die am Ende einer Rechnung auf Eingabe $x$ erreicht wird, eindeutig bestimmt. Bei NTMs kann es neben einer akzeptierenden Rechnung auf Eingabe $x$ zahlreiche verwerfende Rechnungen geben. Definition 1.6 legt nun fest, dass $x$ als akzeptiert gilt, wenn mindestens eine akzeptierende Rechnung existiert; $x$ gilt somit nur dann als verworfen, wenn alle Rechnungen auf Eingabe $x$ verwerfend sind.

**Beispiel 1.7** *COMPOSITES sei die Sprache aller Zahlen aus $\mathbb{N}_0$ (in Binärkodierung), die einen echten Teiler besitzen (also die von Primzahlen und $0, 1$ verschiedenen Zahlen). Eine NTM für COMPOSITES könnte auf einer Eingabe $w \in \{0,1\}^n$ (aufgefasst als Zahl aus $\mathbb{N}_0$) vorgehen wie folgt:*

**Raten** *„Rate nichtdeterministisch“ zwei Binärfolgen $u, v$ der Länge $n$ (evtl. auch mit führenden Nullen). Dabei bedeutet das Raten eines Bits, dass $M$ eine neue Zelle besucht und die zwei Alternativen besitzt, entweder $0$ oder $1$ aufzuschreiben.*

**Verifizieren** *„Verifiziere deterministisch“, dass $1 < u < w$ und dass $w = u \cdot v$. Hierzu muss die Turing-Maschine im Wesentlichen multiplizieren können, was auch deterministisch leicht zu realisieren ist.*

*Die beschriebene NTM ist ein Akzeptor von COMPOSITES, da eine akzeptierende Rechnung auf $w$ genau dann existiert, wenn $w$ einen echten Teiler besitzt.*

An diesem Beispiel lassen sich zwei Phänomene beobachten, die charakteristisch für NTMs sind:

- NTMs besitzen virtuell die Fähigkeit „auf magische Weise“ wichtige Information zu raten. (Im Beispiel wurde eine Zerlegung von $w$ in zwei echte Teiler geraten.)[3]
- NTMs gehen typischerweise nach dem Prinzip „Rate&Verifiziere“ vor (mit einer nichtdeterministischen Ratephase und einer deterministischen Verifikationsphase).

Wir kommen auf dieses wichtige Thema in der Vorlesung noch mehrfach zurück.

**Konventionen** Eine *Mehrband DTM* bzw. *Mehrband NTM* ist eine $k$-Band DTM bzw. eine $k$-Band NTM für eine Konstante $k \geq 1$. Der Ausdruck „Konstante“ soll deutlich machen, dass $k$ nicht etwa von der Länge $n$ des Eingabewortes $w \in \Sigma^n$ abhängen darf.
Unter einer DTM bzw. NTM wollen wir im Folgenden eine Mehrband DTM bzw. Mehrband NTM verstehen.

## 1.3 Komplexitätsklassen

Rechenprobleme, deren Lösungen einen “annähernd gleichen” Ressourcenverbrauch erfordern, werden zu **einer** Komplexitätsklasse zusammengefasst. Die entstehende “Landschaft der Komplexitätsklassen” ist leichter zu überschauen als der “Dschungel der einzelnen Probleme”. Die für uns wichtigsten Ressourcen sind “Platz” (Speicherplatz) und “Zeit” (Rechenzeit). Bei der Formalisierung der Begriffe “Platz” und “Zeit” ist es nicht sinnvoll, diese Größen quantitativ genau zu erfassen. Wenn man zum Beispiel die Rechenzeit in Millisekunden messsen würde, dann müsste man bei jeder verbesserten Rechnergeneration (mit einer höheren Anzahl von flops, schnellerem Speicherzugriff etc.) die Komplexitätstheorie wieder neu schreiben. Darüberhinaus ist die Landschaft der Komplexitätsklassen um so verwirrender, je feiner wir die Unterteilung gestalten. Es ist sozusagen ein “gesundes Maß an Schlampigkeit” gefragt. Bei der Definition von Komplexitätsklassen erreichen wir dies auf mehrfache Weise:

- Wir messen den Ressourcenverbrauch nur “größenordnungsmäßig” als Funktion in der Länge der Eingabe. Der Begriff der “Größenordnung” wird durch die Landau'sche O-Notation formalisiert werden.
- Wir fassen Funktionen verschiedener Größenordnungen mitunter zu noch gröberen Funktionsklassen zusammen. Zum Beispiel betrachten wir oft die Klasse aller Funktionen einer polynomiell beschränkten Größenordnung.
- Wir beziehen uns stets auf das Modell der Turing-Maschine (anstelle eines realistischeren Modells für einen modernen Rechner).

Eine Formalisierung dieser Grundgedanken folgt.

[3] Natürlich kann die NTM auch „falsch“ raten und eine verwerfende Rechnung für eine zu akzeptierende Eingabe $w$ produzieren. Da uns aber in der Definition von $L_M$ eine einzige akzeptierende Rechnung reicht, können wir uns stets auf diese zurück ziehen, um $w \in L_M$ nachzuweisen.

### 1.3.1 Grundbegriffe zur Asymptotik

In diesem Abschnitt listen wir die relevanten Definitionen nur kurz auf. Bei Verständnisschwierigkeiten sei das Selbststudium des Abschnittes zur Asymptotik im Begleitmaterial zur Vorlesung empfohlen.

Im folgenden bezeichne $\mathbb{N}_0$ die Menge der nichtnegativen ganzen Zahlen, $\Re^+$ die Menge der positiven reellen Zahlen und $f, g, \ldots$ seien Funktionen von $\mathbb{N}_0$ nach $\mathbb{N}$.

**Definition 1.8 (Asymptotische Gleichheit)**

$$f \sim g :\Leftrightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = 1 \tag{1}$$

Dies ist offensichtlich eine Äquivalenzrelation.

**Definition 1.9 (Asymptotisch langsameres bzw. schnelleres Wachstum)**

$$f \prec g :\Leftrightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \tag{2}$$

*In Worten: $f$ wächst asymptotisch langsamer als $g$. Diese Relation ist transitiv. Wir schreiben auch $g \succ f$ für $f \prec g$. In Worten: $g$ wächst asymptotisch schneller als $f$.*

**Definition 1.10 (O-Notation)**

$$O(g) = \{f |\ \exists n_0 \in \mathbb{N}_0, \exists c \in \Re^+, \forall n \geq n_0 : f(n) \leq c \cdot g(n). \tag{3}$$

*Aus historischen Gründen schreibt man $f = O(g)$ anstelle von $f \in O(g)$. In Worten: $f$ wächst asymptotisch höchstens so schnell wie $g$.*

**Definition 1.11 ($\Omega$-Notation)**

$$g = \Omega(f) :\Leftrightarrow f = O(g). \tag{4}$$

*In Worten: $g$ wächst asymptotisch mindestens so schnell wie $f$.*

**Definition 1.12 ($\Theta$-Notation)**

$$f = \Theta(g) :\Leftrightarrow f = O(g) \text{ und } f = \Omega(g) \tag{5}$$

*In Worten: $g$ wächst asymptotisch genau so schnell wie $f$.*

Man beachte, dass asymptotische Gleichheit von $f$ und $g$ impliziert, dass beide Funktionen asymptotisch genau so schnell wachsen, aber die Umkehrung gilt i.A. nicht. Zum Beispiel gilt $2n = \Theta(n)$, aber $2n$ und $n$ sind asymptotisch nicht gleich.

**Definition 1.13 ($o$- und $\omega$-Notation)**

$$o(g) = \{f |\ f \prec g\} \tag{6}$$
$$\omega(g) = \{f |\ f \succ g\} \tag{7}$$

*Die Schreibweise $f = o(g)$ bzw. $f = \omega(g)$ ist also zu $f \prec g$ bzw. $f \succ g$ äquivalent.*

Eine Funktion, die asymptotisch langsamer (bzw. schneller) als $n$ wächst heißt *sublinear* (bzw. *superlinear*). Eine Funktion, die asymptotisch schneller wächst als jedes Polynom, heißt *superpolynomiell*. Analog sind die folgenden Begriffe

*sublogarithmisch, superlogarithmisch, subexponentiell, ...*

zu verstehen.

Die folgenden Charakterisierung der asymptotischen Beziehung zwischen zwei Funktionen mit Hilfe des „limes superior“ (größter Häufungspunkt einer Folge) und „limes inferior“ (kleinster Häufungspunkt einer Folge) sind mitunter nützlich:

$$\begin{aligned}
f = O(g) &\Leftrightarrow \limsup_{n\to\infty} \frac{f(n)}{g(n)} < \infty \\
f = \Omega(g) &\Leftrightarrow \liminf_{n\to\infty} \frac{f(n)}{g(n)} > 0 \\
f = \Theta(g) &\Leftrightarrow 0 < \liminf_{n\to\infty} \frac{f(n)}{g(n)} \leq \limsup_{n\to\infty} \frac{f(n)}{g(n)} < \infty \\
f = o(g) &\Leftrightarrow \limsup_{n\to\infty} \frac{f(n)}{g(n)} = 0 \\
f = \omega(g) &\Leftrightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty
\end{aligned}$$

Falls hierbei $f(n)/g(n)$ konvergiert, so können wir natürlich von der Gleichung

$$\liminf_{n\to\infty} \frac{f(n)}{g(n)} = \limsup_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{f(n)}{g(n)}$$

Gebrauch machen.

### 1.3.2 Komplexitätsklassen

**Definition 1.14 (Platzschranken, Zeitschranken, Ressourcenschranken)** *Seien $S, T$ Funktionen von $\mathbb{N}_0$ nach $\mathbb{N}$ und $M$ eine DTM (mit evtl. mehreren Bändern).*

1. *$M$ heißt $S(n)$-platzbeschränkt, wenn eine Rechnung von $M$ auf einer Eingabe der Maximallänge $n$ maximal $O(S(n))$ Bandzellen verbraucht.*

2. *$M$ heißt $T(n)$-zeitbeschränkt, wenn eine Rechnung von $M$ auf einer Eingabe der Maximallänge $n$ maximal $O(T(n))$ Schritte dauert.*

*Den Begriff “Ressourcenschranke” verwenden wir als Oberbegriff von Platz- und Zeitschranke.*

Wir werden stets $T(n) \geq n$ voraus setzen, da $n$ Schritte allein schon zum Lesen einer Eingabe aus $\Sigma^n$ benötigt werden. Hingegen wird es durchaus von Interesse sein, sublineare Platzschranken zu studieren. Dazu muss freilich Definition 1.14 etwas modifiziert werden:

**Konvention** Bei Platzschranken mit $S(n) < n$ für alle (oder manche) Eingabelängen $n$ erweitern wir die Turing-Maschine um ein "read-only" Eingabeband (auf dem also nur gelesen, nicht aber geschrieben werden darf) und bezeichnen ihre anderen Bänder (im Unterschied dazu) als Arbeitsbänder. Nur der Platzverbrauch auf den Arbeitsbändern wird gezählt.

Ressourcenschranken induzieren Komplexitätsklassen im Sinne folgender Definition.

**Definition 1.15 (Deterministische Komplexitätsklassen)** *1. $DTime(T)$ ist die Klasse aller Sprachen $L$, die von einer $T(n)$-zeitbeschränkten DTM akzeptiert werden können.*

*2. $DSpace(S)$ ist die Klasse aller Sprachen $L$, die von einer $S(n)$-platzbeschränkten DTM akzeptiert werden können.*

*3. $DTimeSpace(T, S)$ ist die Klasse aller Sprachen $L$, die von DTM akzeptiert werden können, welche zugleich $T(n)$-zeitbeschränkt und $S(n)$-platzbeschränkt ist.*

Diese Definitionen lassen sich auf NTM's übertragen.

**Definition 1.16** *Eine NTM $M$ heißt $S(n)$-platzbeschränkt (bzw. $T(n)$-zeitbeschränkt), wenn zu jeder Eingabe $x \in L_M$ der Maximallänge $n$ eine akzeptierende Rechnung existiert, die maximal $O(S(n))$ Bandzellen (bzw. $O(T(n))$ Rechenschritte) benötigt.*

**Definition 1.17** *1. $NTime(T)$ ist die Klasse aller Sprachen, die von einer $T(n)$-zeitbeschränkten NTM akzeptiert werden können.*

*2. $NSpace(S)$ ist die Klasse aller Sprachen $L$, die von einer $S(n)$-platzbeschränkten NTM akzeptiert werden können.*

*3. $NTimeSpace(T, S)$ ist die Klasse aller Sprachen $L$, die von NTM akzeptiert werden können, welche zugleich $T(n)$-zeitbeschränkt und $S(n)$-platzbeschränkt ist.*

**Konventionen** Es sei $\mathcal{R}$ eine Klasse von Ressourcenschranken. Eine TM $M$ heißt dann $\mathcal{R}$-platz- bzw. zeitbeschränkt, wenn eine Ressourcenschranke $R \in \mathcal{R}$ existiert, so dass $M$ $R$-platz- bzw. $R$-zeitbeschränkt ist. Diese Konvention wird auch benutzt, um entsprechende Komplexitätsklassen zu bilden. Zum Beispiel gilt dann $DTime(\mathcal{R}) = \cup_{R \in \mathcal{R}} DTime(R)$.
Der Index $k$ in Verbindung mit einer Komplexitätsklasse soll bedeuten, dass wir nur $k$-Band Turing-Maschinen zulassen. Zum Beispiel ist dann $DTime_k(T)$ die Klasse aller Sprachen $L$, die von einer $T(n)$-zeitbeschränkten $k$-Band DTM akzeptiert werden können. Da wir mit einer DTM per Konvention eine Mehrband DTM meinen gilt natürlich $DTime(T) = \cup_{k \geq 1} DTime_k(T)$.

# 2 Robustheit des Maschinenmodells

*Kann der Flügelschlag eines Moskitos einen Taifun auslösen?*

In der Chaostheorie werden mitunter Bedingungen beschrieben, die ein dynamisches System in ein labiles Gleichgewicht versetzen, so dass unscheinbar daher kommende Ereignisse (wie der Flügelschlag eines Moskitos) dramatische Konsequenzen haben können (wie die Entstehung eines Taifuns). Ein solchermaßen chaotisches Verhalten ist sowohl bei Rechnern als auch bei der theoretischen Durchdringung derselben nicht wünschenswert. Anders ausgedrückt: die Komplexitätsklassen und die Hauptsätze der Komplexitätstheorie sollten sich nicht dramatisch verändern, wenn das Standard-Maschinenmodell „marginal" modifiziert wird. Die folgende These sollte in diesem Zusammenhang gesehen werden:

**Erweiterte Church'sche These** Alle „vernünftigen" Modelle von *deterministischen, sequentiellen Rechnern* sind „polynomiell miteinander verknüpft" in dem Sinne, dass wechselseitige Simulationen schlimmstenfalls einen „polynomiellen Blow-up" der betrachteten Ressourcenschranken erleiden.

Man beachte, dass die Aussage dieser These sich explizit nicht auf nicht-deterministische oder probabilistische Rechnermodelle erstreckt, ebensowenig auf Modelle von Parallelrechnern.[4]

Die erweiterte Church'sche These kann natürlich nur für konkrete Maschinenmodelle verifiziert werden. Es lässt sich zum Beispiel zeigen, dass deterministische Turing-Maschinen und sogenannte „Random Access Maschinen" (die einem realistischen Maschinenmodell moderner Computer recht nahe kommen) polynomiell miteinander verknüpft sind. In diesem Kapitel illustrieren wir die Robustheit der Komplexitätstheorie durch folgende (hier nur salopp formulierten) Resultate:

**Kompressionstheorem** Der Speicherplatzbedarf lässt sich um einen beliebig vorgegebenen konstanten Faktor reduzieren.

**Beschleunigungstheorem** Der Zeitbedarf lässt sich um einen beliebigen konstanten Faktor reduzieren (allerdings nicht unter Linearzeit).

**1. Bandreduktionstheorem** Beim Übergang von Mehrband- zu 1-Band Turing-Maschinen verändert sich die Platzschranke nicht und die Zeitschranke erleidet schlimmstenfalls einen „quadratischen Blow-up".

**2. Bandreduktionstheorem** Beim Übergang von Mehrband- zu 2-Band DTMs erleidet die Zeitschranke schlimmstenfalls einen „logarithmischen Blow-up". (Bei NTMs bleibt sie sogar unverändert.)

Kompressions- und Beschleunigungstheorem können als eine zusätzliche Rechtfertigung der $O$-Notation bei der Definition von $S(n)$-platzbeschränkten bzw. $T(n)$-platzbeschränkten Turing-Maschinen gesehen werden: die durch die $O$-Notation „versteckte Konstante" lässt sich durch Kompression bzw. Beschleunigung sowieso liquidieren.

[4] Die in jüngster heiß diskutierten sogenannten Quantenrechner (mit denen wir uns in dieser Vorlesung nicht beschäftigen werden) sind auch zu paralleler Verarbeitung von Information im Stande.

Die Lehre aus den Bandreduktionstheoremen ist, dass wir uns — sofern wir uns an einem schlimmstenfalls „kleinen“ (quadratischen bzw. logarithmischen) Blow-up der Zeitschranken nicht stören — auf 1-Band bzw. 2-Band Turing-Maschinen zurück ziehen dürfen.

In den folgenden beiden Abschnitten werden die genannten Theoreme exakt formuliert und (skizzenhaft) bewiesen. Unterwegs lernen Sie einige fundamentale Simulations- und Analysetechniken (Schritt-für-Schritt Simulation, Kodierungstricks, Technik des „Shiftens von Bandinschriften“ und der aufgelockerten Speicherung sowie amortisierte Analyse) kennen.

## 2.1 Kompression und Beschleunigung

In diesem (und nur diesem) Abschnitt wollen wir bei der Betrachtung von Platz- und Zeitschranken etwas „pingeliger“ sein. Eine DTM heiße *strikt* $T(n)$-zeitbeschränkt, wenn ihre Rechnung auf einer Eingabe der Länge $n$ maximal $T(n)$ — statt $O(T(n))$ — Schritte dauert. Sie heißt *strikt* $S(n)$-platzbeschränkt, wenn im Laufe der Rechnung auf Eingaben der Länge $n$ maximal $S(n)$ Zellen besucht werden. Die analoge Sprachregelung verwenden wir auch für NTMs.

**Satz 2.1 (Kompressionstheorem)** *Wenn die Sprache $L \subseteq \Sigma^*$ durch eine $S(n)$ strikt platzbeschränkte $k$-Band DTM akzeptiert werden kann, so kann sie für jede Konstante $0 < \epsilon < 1$ auch durch eine $\lceil \epsilon S(n) \rceil$ strikt platzbeschränkte $k$-Band DTM akzeptiert werden. Eine analoge Aussage gilt auch für NTMs.*

**Beweis** Wir skizzieren den Beweis für 1-Band DTMs (evtl. erweitert um ein read-only Eingabeband). Der Beweis für $k$-Band DTMs bzw. $k$-Band NTMs ergibt sich aber völlog analog. Sei $M$ eine $S(n)$ strikt platzbeschränkte 1-Band DTM mit $L = L_M$. Ziel ist die Konstruktion einer $\lceil \epsilon S(n) \rceil$ strikt platzbeschränkten 1-Band DTM $M'$ mit $L = L_{M'}$. Wir nehmen zunächst an, dass $M'$ über ein read-only Eingabeband verfügt (und rechtfertigen dies später). Setze $m := \lceil 1/\epsilon \rceil$. Die wesentliche Idee zur Konstruktion von $M'$ besteht darin, ein mächtigeres Bandalphabet zu verwenden, das die Kodierung von $m$ Symbolen durch ein einziges „Supersymbol“ erlaubt: wenn $M$ Bandalphabet $\Gamma$ verwendet, so spendieren wir $M'$ das Bandalphabet $\Gamma \cup \Gamma^m$. $M'$ überträgt zunächst die Eingabe $w$ in komprimierter Form auf ihr Arbeitsband. Danach simuliert sie $M$ Schritt für Schritt, wobei aber die Bandinschrift stets in komprimierter Form dargestellt wird. Die Details dieser Simulationsidee sind leicht einzufüllen (s. Übung).
Zur Vollständigkeit des Beweises fehlt nur noch die Rechtfertigung für die Verwendung des read-only Eingabebandes:

- Falls $\lceil \epsilon S(n) \rceil \geq n$ für alle $n$, dann kann die Kompression der Eingabe auch ohne das read-only Extraband erfolgen.
- Andernfalls liegt die Platzschranke für manche Eingabenlängen $n$ unterhalb von $n$ und $M'$ bekommt gemäß unserer Vereinbarung bei sublinearen Platzschranken ein read-only Eingabeband zugeteilt.

□

**Satz 2.2 (Beschleunigungstheorem)** *Wenn die Sprache $L \subseteq \Sigma^*$ durch eine $T(n)$ strikt zeitbeschränkte $k$-Band DTM $M$ akzeptiert werden kann, so kann sie für jede Konstante $0 < \epsilon < 1$ auch durch eine $n + \epsilon T(n)$ strikt zeitbeschränkte $\max\{2, k\}$-Band DTM $M'$ akzeptiert werden. Falls $T(n) = \omega(n)$, dann ist $M'$ sogar $\lceil \epsilon T(n) \rceil$ strikt zeitbeschränkt. Eine analoge Aussage gilt auch für NTMs.*

**Beweis** Wir skizzieren den Beweis für DTMs. (Für NTMs kann er analog geführt werden.) Wir spendieren für $M'$ zunächst die etwas komfortablere strikte Zeitschranke $n + \epsilon T(n) + 4$ sowie $k + 1$ Bänder (und rechtfertigen dies später). Die wesentliche Idee zur Konstruktion von $M'$ besteht wieder in der Verwendung des Bandalphabetes $\Gamma \cup \Gamma^m$, wobei die „Supersymbole“ aus $\Gamma^m$ die gleichzeitige Manipulation von $m$ einfachen Symbolen aus $\Gamma$ erlaubt. (Die Konstante $m$ wird später geeignet definiert.) $M'$ arbeitet wie folgt:

**Initialisierung** Die Eingabe der Länge $n$ wird in komprimierter Form auf Band $k + 1$ übertragen, wo sie dann nur $\lceil n/m \rceil$ Zellen beansprucht. Kopf $k + 1$ spaziert danach auf Zelle 1 (also zum Anfang der komprimierten Eingabe) zurück. Ab jetzt verwendet $M'$ Band $k + 1$ in der Rolle des Bandes 1 von $M$.

**Kommentar** Es ist hilfreich sich vorzustellen, dass das Arbeitsband von $M$ in Blöcke der Länge $m$ zerfällt, wobei die Beschriftung eines Blockes einem Supersymbol aus $\Gamma^m$ entspricht. $M'$ wird $m$ Schritte von $M$ (genannt „Phase“) in 3 Schritten simulieren, wobei zu Beginn einer Phase stets die folgenden Bedingungen gelten:

**Bedingung 1** Das Band von $M'$ enthält die Beschriftung des entsprechenden Bandes von $M$ in komprimierter Form. Wenn $B_i$ die Beschriftung des $i$-ten Blockes des Bandes von $M$ ist, dann bezeichne $S_i \in \Gamma^m$ das entsprechende Supersymbol auf dem Band von $M'$.

**Bedingung 2** Wenn der Kopf von $M$ auf einer Zelle in Block $i$ positioniert ist, dann ist der Kopf von $M'$ auf der Zelle mit Supersymbol $S_i$ positioniert und hat die Supersymbole $S_{i-1}, S_i, S_{i+1}$ (ebenso wie den aktuellen Zustand von $M$) in die endliche Kontrolle geladen.

Beachte, dass der Kopf von $M$ innerhalb einer Phase bestehend aus $m$ Schritten sich entweder nur in den Blöcken $B_i$ und (evtl.) $B_{i+1}$ oder in den Blöcken $B_i$ und (evtl.) $B_{i-1}$ aufhalten kann. In der folgenden Beschreibung der Simulation durch $M'$ setzen wir voraus, dass er sich in $B_i$ und (evtl.) $B_{i+1}$ aufhält. (Der andere Fall ist hierzu symmetrisch.)

**Simulation** Solange $M$ nicht stoppt, werden ihre nächsten $m$ Schritte (die aktuelle Phase) durch $M'$ simuliert wie folgt:

**Schritt 1** Ersetze $S_i$ durch das Supersymbol $S_i'$ für den Inhalt von Block $i$ am Ende der Phase, lade $S_i'$ an Stelle von $S_i$ in die endliche Kontrolle und bewege den Kopf auf $S_{i+1}$.

**Schritt 2** Ersetze $S_{i+1}$ durch das Supersymbol $S'_{i+1}$ für den Inhalt von Block $i+1$ am Ende der Phase und lade $S'_{i+1}$ an Stelle von $S_{i+1}$ in die endliche Kontrolle. Falls $M$ sich am Ende der Phase in Block $i$ aufhält, dann bewege den Kopf auf $S'_i$ zurück und beende die Simulation der Phase. Andernfalls ($M$ hält sich am Ende der Phase in Block $i+1$ auf) bewege den Kopf auf $S_{i+2}$ und vollziehe Schritt 3.

**Schritt 3** Lade $S_{i+2}$ an Stelle von $S_{i-1}$ in die endliche Kontrolle und bewege den Kopf wieder auf $S'_{i+1}$.

Induktiv lässt sich leicht zeigen, dass Bedingungen 1 und 2 zu Beginn einer jeden Phase gelten: sie gelten (bei geeigneter Implementierung) nach der Initialisierung (Induktionsanfang) und reproduzieren sich nach jeder simulierten Phase (Induktionsschritt). Hieraus ergibt sich die Korrektheit der Simulation. Für $M'$ ergibt sich (unter Ausnutzung von $T(n) \geq n$) die strikte Zeitschranke

$$\overbrace{n + \left\lceil \frac{n}{m} \right\rceil}^{\text{Initialisierung}} + \overbrace{3 \left\lceil \frac{T(n)}{m} \right\rceil}^{\text{Simulation}} \leq n + 4 \left\lceil \frac{T(n)}{m} \right\rceil \leq n + 4\frac{T(n)}{m} + 4 \ .$$

Setzen wir $m := m(\epsilon) := \lceil 4/\epsilon \rceil$, so ergibt sich die strikte Zeitschranke $n + \epsilon T(n) + 4$.
Den additiven Term 4 werden wir los, indem wir $m := m(\epsilon/2)$ setzen und ausnutzen, dass folgendes gilt:

$$\exists n_0, \forall n \geq n_0 : \frac{\epsilon}{2} T(n) + 4 \leq \epsilon T(n) \ .$$

Die endlich vielen Eingaben mit einer Länge $n \leq n_0$ kann $M'$ in strikter Linearzeit ($n$ Schritte) mit der endlichen Kontrolle bearbeiten. Somit ergibt sich die strikte Zeitschranke $n + \epsilon T(n)$.
Falls $k \geq 2$, so werden wir Band $k+1$ von $M'$ los, indem $M'$ zur Initialisierung Band 2 an Stelle von Band $k+1$ benutzt und bei der Simulation von $M$ die Bänder 1 und 2 in vertauschten Rollen verwendet.
Falls $T(n) = \omega(n)$, so ergibt sich die strikte Zeitschranke $\epsilon T(n)$, indem wir ausnutzen, dass folgendes gilt:

$$\exists n_0, \forall n \geq n_0 : n + \frac{\epsilon}{2} T(n) \leq \epsilon T(n) \ .$$

Damit ist das Beschleunigungstheorem vollständig bewiesen. □

## 2.2 Bandreduktion

In den Beweisen der folgenden Bandreduktionstheoreme ist es anschaulich, sich vorzustellen, dass ein „Supersymbol“ $(X_1, \ldots, X_k) \in \Gamma^k$ in einer Zelle mit $k$ „Spuren“ untergebracht ist (ein Symbol aus $\Gamma$ pro Spur). Formal gesehen handelt es sich aber weiterhin einfach um den uns schon bekannten Kniff, dass wir ein Bandalphabet $\Gamma$ zu einem mächtigeren Bandalphabet $\Gamma \cup \Gamma^k$ erweitern können. Wenn wir diese Technik einsetzen, sprechen wir im Folgenden von “Mehrspur DTMs“.

**Satz 2.3 (Erstes Bandreduktionstheorem)** *$DTimeSpace(T,S) \subseteq DTimeSpace_1(TS,S)$ und $NTimeSpace(T,S) \subseteq NTimeSpace_1(TS,S)$.*

**Beweis** Wir führen den Beweis für DTMs. (Für NTM's verläuft er aber völlig analog.)
Es bezeichne $M$ eine gegebene $T$-zeitbeschränkte und $S$-platzbeschränkte DTM. Wir konstruieren eine $TS$-zeitbeschränkte und $S$-platzbeschränkte DTM $M'$, welche $M$ simuliert (und somit im Stande ist, die gleiche Sprache wie $M$ zu akzeptieren). $M'$ besitzt für jeden Zustand $z$ von $M$ einen entsprechenden Zustand $z'$ (und weitere Zustände). $M'$ simuliert einen Schritt von $M$ mit Zustandswechsel von $z_1$ nach $z_2$ durch eine Folge von Schritten, welche im Zustand $z_1'$ startet und im Zustand $z_2'$ endet (Schritt für Schritt Simulation). Die wesentliche Schwierigkeit besteht darin, die Beschriftung der $k$-Band TM $M$ auf einem einzigen Band unterzubringen und die Arbeit der $k$ Köpfe von $M$ mit dem (einzigen) Kopf von $M'$ zu erledigen. Wir setzen zu diesem Zweck eine „Mehrspur DTM“ und die Technik des „Shiftens von Bandinschriften“ ein. Details folgen.
$M'$ benutzt ein Band mit $k$ Spuren. Dabei soll unmittelbar vor der Simulation eines Schrittes von $M$ durch $M'$ stets gelten:

(1) $M'$ befindet sich im Zustand $z'$, sofern $M$ sich im Zustand $z$ befindet.

(2) Spur $i$ des Bandes von $M'$ enthält die Beschriftung von Band $i$ von $M$ $(1 \leq i \leq k)$.

(3) Zelle 0 von $M'$ enthält genau die $k$ Symbole, auf denen die $k$ Köpfe von $M$ positioniert sind.

(4) Der Kopf von $M'$ befindet sich auf Zelle 0 („Normalposition“).

Bedingungen (3) und (4) sorgen dafür, daß $M'$ zu Beginn der Simulation eines Schrittes von $M$ die $k$ von $M$ gelesenen Symbole kennt. Um einen Schritt von $M$ zu simulieren, geht $M'$ vor wie folgt:

1. Wenn $M$ Symbole $X_1, \ldots, X_k$ durch $Y_1, \ldots, Y_k$ ersetzt, ersetzt $M'$ in Zelle 0 das Supersymbol $(X_1, \ldots, X_k)$ durch das Supersymbol $(Y_1, \ldots, Y_k)$.

2. Wenn $M$ Kopf $i$ nach rechts (bzw. links) bewegt, so verschiebt $M'$ die Inschrift von Spur $i$ um eine Position in die entgegengesetzte Richtung.

3. Der Kopf von $M'$ begibt sich wieder in Normalposition.

4. Schließlich geht $M'$ in den Zustand $z'$ über, sofern $M$ in den Zustand $z$ übergegangen ist.

Durch diese Vorgehensweise bleiben Bedingungen (1), (2) und (3) stets erhalten und die Simulation ist korrekt. Offensichtlich besucht $M'$ nicht mehr Zellen wie $M$ und ist daher $S(n)$-platzbeschränkt. Da 1 Schritt von $M$ durch $M'$ in $O(S(n))$ Schritten simuliert werden kann, ist $M'$ $O(TS)$-zeitbeschränkt. □

Aus dem ersten Bandreduktionstheorem können wir ein paar nahe liegende Schlussfolgerungen ziehen:

$$\begin{aligned} DSpace(S) &= DSpace_1(S) \quad \text{und} \quad NSpace(S) = NSpace_1(S) \\ DTime(T) &\subseteq DTime_1(T^2) \quad \text{und} \quad NTime(T) \subseteq NTime_1(T^2) \end{aligned}$$

Die Lehre hieraus ist die folgende: Bei der Analyse der Platzkomplexität spielt es keine Rolle, ob wir Ein- oder Mehrbandmaschinen betrachten; bei der Analyse der Zeitkomplexität hingegen könnte die Beschränkung auf 1 Band zu einem „quadratischen blow-up" der benötigten Rechenzeit führen.

**Einschub** Streng genommen haben wir nur gesehen, dass *unsere Simulation* einer Mehrband- durch eine 1-Band Maschine einen quadratischen blow-up der Zeitschranke erleidet. Gibt es zeiteffizientere Simulationen? Die Antwort lautet: Zumindest in Einzelfällen geht es nicht besser! Man kann nämlich mit informationstheoretischen Methoden (Technik der sogenannten „Crossing Sequences") zeigen, dass bestimmte Sprachen (wie zum Beispiel die Sprache der Palindrome) in $DTime_2(n) \setminus DTime_1(o(n^2))$ liegen. Eine solche Sprache kann also von einer 2-Band DTM in Linearzeit erkannt werden, aber von keiner 1-Band DTM in subquadratischer Zeit.

**Satz 2.4 (Zweites Bandreduktionstheorem für DTMs)**

$$DTimeSpace(T, S) \subseteq DTime_2(T \log S) \ .$$

**Beweis** Wir müssen im Wesentlichen zeigen, dass eine $k$-Band DTM $M$ mit Zeitschranke $T$ und Platzschranke $S$ simulierbar ist durch eine 2-Band DTM $M'$ mit Zeitschranke $T \log S$. $M'$ verwendet Band 2 als eine Art „Schmierblatt" und bringt die Inschriften der $k$ Bänder von $M$ alle auf ihrem Band 1 unter. Ähnlich wie bei der Simulation im Beweis zum 1. Bandreduktionstheorem wird Band 1 aus mehreren Spuren bestehen und es wird wieder die Technik des Shiftens von Bandinschriften eingesetzt werden. Um aber den quadratischen Blow-up der Zeitschranke zu vermeiden, wird die Bandinschrift diesmal in einer geeignet „aufgelockerten Form" abgespeichert, so dass nicht jedes Mal die gesamte Inschrift in einen Shift mit einbezogen werden muss. Die Simulationsidee mit dem Shiften von Bandinschriften setzen wir als bekannt voraus. Es folgen hauptsächlich Details zur „Technik der aufgelockerten Speicherung" und der resultierenden Zeitanalyse.
Band 1 von $M'$ besitzt insgesamt $2k$ „Spuren", d.h., das Bandalphabet von $M'$ enthält Supersymbole aus $\Gamma^{2m}$. Auf diese Weise stehen 2 Spuren pro Band von $M$ zur Verfügung.

**Einschub** Wir konzentrieren uns ab jetzt auf die Arbeit von $M'$ auf zwei (einem Band von $M$ entsprechenden) Spuren ihres ersten Bandes. Die Anzahl der Rechenschritte, die wir auf diese Weise ermitteln, muss im Prinzip mit $k$ multipliziert werden, da die gleiche Arbeit für jedes der $k$ Spurpaare zu leisten ist.

$M'$ unterteilt die Zellen von Band 1 in Blöcke $\ldots, B_{-2}, B_{-1}, B_0, B_1, B_2, \ldots$, wobei Block $B_0$ aus 1 Zelle und Blöcke $B_{-i}$ und $B_i$ für $i \geq 1$ aus $2^{i-1}$ Zellen bestehen. Die Inschrift

des betreffenden Bandes von $M$ erhalten wir, indem wir auf den beiden korrespondierenden Spuren des 1. Bandes von $M'$ die Blöcke von links nach rechts auslesen, wobei innerhalb eines Blockes die obere Spur zuerst gelesen wird. Zum Beispiel enthalten die beiden Spuren

| $\cdots$ | $-3$ | $-3$ | $-3$ | $-3$ | $-2$ | $-2$ | $-1$ | $0$ | $1$ | $2$ | $2$ | $3$ | $3$ | $3$ | $3$ | $\cdots$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\cdots$ | $A$ | $B$ | $C$ | $D$ | | | | | $F$ | $H$ | $I$ | | | | | $\cdots$ |
| $\cdots$ | | | | | | | | $E$ | $G$ | $J$ | $K$ | $L$ | $M$ | $N$ | $O$ | $\cdots$ |

(mit den Blocknummern in der obersten Zeile) die Inschrift $ABCDEFGHIJKLMNO$. Dass die Blöcke im obigen Beispiel entweder leer, halbvoll oder voll sind, ist kein Zufall, da $M'$ während der Simulation von $M$ folgende Invarianzbedingungen einhält:

1. Die (korrekt ausgelesene) Inschrift zweier Spuren von Band 1 von $M'$ ist stets gleich der Inschrift des betreffenden Bandes von $M$.

2. Für jedes Paar von Spuren (zur Darstellung eine Bandes von $M$) und jede Nummer $i \geq 1$ eines Blockes gilt genau eine der folgenden beiden Bedingungen[5]:

   (0) $B_i$ und $B_{-i}$ sind beide halbvoll.

   (1) $B_i$ ist voll und $B_{-i}$ ist leer (oder umgekehrt).

   $B_0$ ist stets halbvoll.

Es ist hilfreich den aktuellen Status der Blöcke durch einen Bitvektor $b$ anzuzeigen mit $b_i = 0$ bzw. $b_i = 1$, falls Bedingung (0) bzw. Bedingung (1) gilt.
Wieviel Arbeit kommt auf $M'$ zu, wenn die Inschrift zweier Spuren von Band 1 geshiftet werden muss? Wir betrachten einen Rechtsshift (wobei die Überlegungen für einen Linksshift völlig analog sind). $M'$ geht zur Implementierung eines Rechtsshifts vor wie folgt[6]:

1. Suche die minimale Nummer $i \geq 1$ eines nicht-vollen Blockes $B_i$.

2. Für $j = i-1, \ldots, 0$ transportiere die Inschrift des Blockes $B_j$ in die unterste freie Spur von $B_{j+1}$.

3. Transportiere die unterste besetzte Spur von $B_{-i}$ in die obere Spur von $B_{-(i-1)}, \ldots, B_{-1}$ und die untere Spur von $B_0$.

Wir wollen an dem obigen Beipiel die Wirkung eines Rechtsshiftes illustrieren. Der erste nicht-volle Block mit einer positiven Blocknummer ist $B_3$. Der Rechtsshift erstreckt sich demnach auf die Blöcke $B_{-3}, \ldots, B_3$ und führt zu folgenden Ergebnis:

| $\cdots$ | $-3$ | $-3$ | $-3$ | $-3$ | $-2$ | $-2$ | $-1$ | $0$ | $1$ | $2$ | $2$ | $3$ | $3$ | $3$ | $3$ | $\cdots$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\cdots$ | | | | | $A$ | $B$ | $C$ | | | | | $H$ | $I$ | $J$ | $K$ | $\cdots$ |
| $\cdots$ | | | | | | | | $D$ | $E$ | $F$ | $G$ | $L$ | $M$ | $N$ | $O$ | $\cdots$ |

[5] Hierbei betrachten wir (um unnötige Fallunterscheidungen zu vermeiden) die Blöcke $B_i$ und $B_{-i}$ als „halbvoll“, wenn sie noch unbenutzt sind (wenn also der Kopf 1 von $M'$ bei seiner Arbeit auf diesen beiden Spuren den Bereich der Blöcke $B_{-(i-1)}, \ldots, B_{i-1}$ noch nicht verlassen hat).

[6] Das folgende Manöver geht wegen der gewählten Blockgrößen und der geltenden Invarianzbedingungen immer gut!

Beachte, dass der Rechtsshift nur für einen Teil der Blöcke, nämlich für die Blöcke mit Nummern von $-i$ bis $i$, durchgeführt werden muss. Um dies deutlich zu machen, sprechen wir von einer $B_i$-Operation. Unter Verwendung eines Schmierblattes (Band 2) kann $M'$ eine $B_i$-Operation in $c2^i$ Schritten (für eine geeignet gewählte Konstante $c$) ausführen. Für die Zeitanalyse der geschilderten Simulation stellen sich zwei grundsätzliche Fragen:

**Frage 1** Wieviele Blöcke werden im Laufe der Simulation benutzt?

**Frage 2** Wieviele $B_i$-Operationen werden im Laufe der Simulation durchgeführt?

Die Antwort auf die erste Frage ist einfach: da $B_{-i}$ und $B_i$ für $i \geq 1$ zusammen (auf zwei Spuren) stets $2^i$ Symbole speichern, benutzt $M'$ höchstens $s \leq \log(S(n)+1)$ viele Blöcke. Zur Antwort auf die zweite Frage verwenden wir die Technik der *amortisierten Analyse*.[7] Zu diesem Zweck betrachten wir die „Potenzialfunktion“

$$\Phi_i := \sum_{j=1}^{i-1} b_j 2^j \ .$$

**Einschub** Die Grundidee zur amortisierten Analyse (geschildert am vorliegenden Fall) ist die folgende. Eine $B_i$-Operation ist um so teurer, je größer die Nummer $i$ ist. Wir werden für eine teure Operation dadurch „belohnt“ werden, dass das $\Phi_i$-Potenzial auf einen deutlich niedrigeren Level sinkt. Da die Potenzialfunktion niemals negative Werte annimmt und (wie wir noch zeigen werden) sie auch nur in begrenztem Umfang wachsen kann, kann die Anzahl der $B_i$-Operationen nicht „beliebig groß“ sein (was natürlich präziser ausgedrückt werden wird).

Wir wollen die Veränderung von $\Phi_i$ analysieren, die durch eine $B_i$-Operation ausgelöst wird. **Vor** der $B_i$-Operation gilt (oBdA im Falle eines Rechtsshiftes):

1. $b_1 = \cdots = b_{i-1} = 1$, d.h., $B_1, \ldots, B_{i-1}$ sind voll und $B_{-1}, \ldots, B_{-(i-1)}$ sind leer.
2. Entweder sind $B_i$ und $B_{-i}$ beide halbvoll (und somit $b_i = 0$) oder $B_i$ ist leer und $B_{-i}$ ist voll (und somit $b_i = 1$).

**Nach** der $B_i$-Operation gilt:

1. $b_1 = \cdots = b_{i-1} = 0$, d.h., $B_1, \ldots, B_{i-1}$ sowie $B_{-1}, \ldots, B_{-(i-1)}$ sind halbvoll.
2. Entweder sind $B_i$ und $B_{-i}$ beide halbvoll (und somit $b_i = 0$) oder $B_i$ ist leer und $B_{-i}$ ist voll (und somit $b_i = 1$). Wenn $B_i$ vorher halbvoll war, ist er nun voll ($b_i$ wechselt von 0 auf 1). Wenn $B_i$ vorher leer und $B_{-i}$ vorher voll war, so sind beide Blöcke nun halbvoll ($b_i$ wechselt von 1 auf 0.)

[7] Ein paar weiter führende Bemerkungen zu dieser wichtigen Analysetechnik sind im Begleitmaterial zur Vorlesung enthalten.

In jedem Fall ergibt sich, dass eine $B_i$-Operation das Potenzial $\Phi_i$ um $\sum_{j=1}^{i-1} 2^j = 2^i - 2$ verkleinert. Für $i \geq 2$ beträgt die Potenzialverkleinerung mindestens $2^{i-1}$. Darüberhinaus ergibt sich, dass eine $B_k$-Operation mit $k \neq i$ das Potenzial $\Phi_i$ maximal um 1 vergrößert (und evtl. sogar verkleinert). Beachte, dass das anfängliche Potenzial stets 0 ist. (Hier nutzen wir die Konvention, dass unbenutzte Blöcke als halbvoll betrachtet werden.) Es bezeichne $m_i$ die Gesamtzahl aller während der Simulation durchgeführten $B_i$-Operationen. Beachte, dass $\sum_{i=1}^{s} m_i = T(n)$, da jeder simulierte Schritt für ein $i \in \{1, \ldots, s\}$ eine $B_i$-Operation auslöst. Das $\Phi_i$-Potenzial kann daher höchstens $T(n)$-mal um 1 inkrementiert werden. Nach der Durchführung der $m_i$ vielen $B_i$-Operationen muss $\Phi_i$ (mit $i \geq 2$) demnach die Bedingung

$$0 \leq T(n) - m_i 2^{i-1}$$

erfüllen, woraus sich

$$m_i \leq \frac{T(n)}{2^{i-1}}$$

ergibt. Da trivialerweise $m_1 \leq T(n)$, ist diese Bedingung für $i = 1$ ebenfalls erfüllt. Die Laufzeit $T'(n)$ (bezogen auf eines der $k$ Spurpaare) ergibt sich aus den Gesamtkosten aller $B_i$-Operationen für $i = 1, \ldots, s$:

$$T'(n) \leq \sum_{i=1}^{s} m_i c 2^i \leq 2csT(n) \leq 2cT(n)\log(1 + S(n)) \ .$$

Für die Gesamtlaufzeit (bezogen auf alle $k$ Spurpaare und inklusive des Aufwandes für eine geeignete Initialisierung) ergibt sich eine Zeitschranke der Form $T \log S$. □

**Folgerung 2.5** $DTime(T) \subseteq DTime_2(T \log T)$.

Der Vollständigkeit halber geben wir noch das folgende Resultat an (dessen Beweis wir in den Übungen diskutieren):

**Satz 2.6 (Zweites Bandreduktionstheorem für NTMs)** $NTime(T) = NTime_2(T)$.

Bei der Zeitanalyse von nicht-deterministischen Turing-Maschinen könnte man sich also stets auf 2-Band NTMs zurück ziehen. 2-Band DTM hingegen erleiden im Vergleich zu Mehrband DTMs evtl. einen kleinen „Blow-up“ der Zeitschranke um einen logarithmischen Faktor.

# 3 Beziehungen zwischen den Komplexitätsklassen

## 3.1 Kontrolle der Ressourcenschranken

Eine Schlüsseltechnik zur Analyse der Beziehungen zwischen verschiedenen Komplexitätsklassen ist die Simulation einer TM $M$ durch eine andere TM $M'$. Dabei wird es von Bedeutung sein, dass $M'$ ihre Ressourcenschranken nicht überschreitet. Unter Umständen muss eine Simulation von $M$ durch $M'$ erfolglos abgebrochen werden, wenn sie zur Überschreitung der Ressourcenschranke führen würde. Wie aber kann $M'$ die Kontrolle darüber bewahren? Die Beantwortung dieser Frage beruht auf den Konzepten der Platz- und Zeitkonstruierbarkeit.

**Definition 3.1 (Platzkonstruierbarkeit)** *Eine Funktion $S : \mathbb{N} \to \mathbb{N}$ heißt* platzkonstruierbar*, wenn die Funktion $1^n \mapsto 1^{S(n)}$ von einer $S$-platzbeschränkten DTM berechnet werden kann.*

Die Hauptanwendung der Platzkonstruierbarkeit besteht darin, ein Bandsegment der Größe $S(n)$ abzustecken (zum Beispiel mit Endmarkierungen auf der Zelle am weitesten links bzw. rechts). Dies geschieht oft in Verbindung mit einer Simulation, welche abgebrochen wird, wenn sie zum Verlasssen des abgesteckten Bandsegmentes führen würde.

**Definition 3.2 (Zeitkonstruierbarkeit)** *Eine Funktion $T : \mathbb{N} \to \mathbb{N}$ heißt* zeitkonstruierbar*, wenn die Funktion $1^n \mapsto bin(n)$ von einer $T$-zeitbeschränkten 2-Band DTM $M$ berechnet werden kann.*

Die Hauptanwendung der Zeitkonstruierbarkeit besteht darin, einen Binärzähler mit $\mathrm{bin}(T(n))$ zu initialisieren und dann in Verbindung mit einer Simulation als „Uhr“ zu verwenden: nach jedem Schritt in der Simulation wird der Zähler um 1 dekrementiert; erreicht er den Wert 0 (Zeit abgelaufen), dann wird die Simulation abgebrochen.[8]

Die folgenden Beobachtungen (Beweise in den Übungen) zeigen, dass die Klasse der konstruierbaren Funktionen sehr reichhaltig ist:

1. Jede konstante Funktion $n \mapsto c$ mit $c \in \mathbb{N}$ ist platz- und zeitkonstruierbar.
2. Die Funktionen $n \mapsto |\mathrm{bin}(n)| = 1 + \lfloor \log n \rfloor$, $n \mapsto \lfloor \log n \rfloor$ und $n \mapsto \lceil \log n \rceil$ sind platzkonstruierbar.
3. Die Funktionen $n \mapsto n$ und $n \mapsto 2^n$ sind platz- und zeitkonstruierbar.
4. Platz- und zeitkonstruierbare Funktionen sind abgeschlossen unter Addition und Multiplikation.
5. Alle Polynome mit Koeffizienten aus $\mathbb{N}$ (aufgefasst als Funktionen von $\mathbb{N}$ nach $\mathbb{N}$) sind platz- und zeitkonstruierbar.

**Konvention** Mit einem „Polynom“ meinen wir im Folgenden stets ein Polynom mit Koeffizienten aus $\mathbb{N}$, das als Funktion von $\mathbb{N}$ nach $\mathbb{N}$ (oder manchmal auch als Funktion von $\mathbb{N}_0$ nach $\mathbb{N}_0$) aufgefasst wird.

[8] Mit einer amortisierten Analyse lässt sich zeigen, dass der „Overhead“ zum Zurückzählen des Binärzählers von $\mathrm{bin}(T(n))$ auf 0 durch $O(T(n))$ Rechenschritte beschränkt ist.

## 3.2 Das Rate-Verifikations-Prinzip

*DTMs* können wir im Prinzip *als NTMs mit genau einer Alternative pro Rechenschritt* auffassen. Wir werden in diesem Abschnitt *NTMs* so normalisieren, dass sie genau *zwei Alternativen pro Rechenschritt* haben und zudem nach dem sogenannten *Rate-Verifikations-Prinzip* vorgehen.

Jede NTM lässt sich ohne wesentlichen Effizienzverlust (unveränderte Zeit- und Platzschranken) simulieren durch eine NTM, die pro Rechenschritt genau zwei Alternativen zur Verfügung hat (s. Übung). Wenn es genau zwei Alternativen pro Schritt gibt, sagen wir „Alternative 0“ und “Alternative 1“, dann kann die NTM die „nicht-deterministischen Entscheidungen“ an den Anfang stellen und auf einer Eingabe $w \in \Sigma^n$ vorgehen wie folgt:

1. Rate einen Binärstring $u \in \{0,1\}^*$.[9]
2. Vollziehe die durch $u$ beschriebene deterministische (!) Rechnung der Maximallänge $|u|$, bei der in Schritt $i$ die Alternative $u_i$ gewählt wird. Falls nach $|u|$ Schritten keine Endkonfiguration erreicht wurde, dann stoppe verwerfend. Wurde eine Endkonfiguration erreicht, dann stoppe (und zwar akzeptierend gdw die Endkonfiguration alzeptierend ist).

Es bezeichne

$$\langle \cdot, \cdot \rangle : \Sigma^* \times \Sigma^* \to \Sigma^*$$

eine injektive in Linearzeit berechen- und invertierbare Abbildung, welche Wortpaaren $(u, w)$ ein Wort $\langle u, w \rangle$ über dem gleichen Alphabet $\Sigma$ zuordnet. Intuitiv können wir $\langle u, w \rangle$ als eine Kodierung des Paares $(u, w)$ anschauen, wobei Kodierung und Dekodierung in Linearzeit durchführbar sind. Eine denkbare Kodierung wäre zum Beispiel

$$(u_1 \cdots u_m, w_1 \cdots w_n) \mapsto u_1 u_1 \cdots u_m u_m 0 w_1 w_1 \cdots w_n w_n \quad .$$

Unsere Ausführungen zum Rate-Verifikations-Prinzip lassen sich in folgendem Resultat zusammen fassen:

**Satz 3.3 (Rate-Verifikations-Theorem)** *Wir betrachten Sprachen über einem festen Alphabet $\Sigma$. Es gilt $L \in NTime(T(n))$ gdw eine Sprache $L'$ und eine Konstante $c > 0$ existieren, so dass*

$$L = \{w \in \Sigma^* | \ \exists u \in \Sigma^{cT(|w|)} : \langle u, w \rangle \in L'\} = \{w \in \Sigma^* | \ \exists u \in \Sigma^* : \langle u, w \rangle \in L'\} \qquad (8)$$

*und die Mitgliedschaft von $\langle u, w \rangle$ in $L'$ von einer DTM in $O(|u| + T(|w|))$ vielen Schritten getestet werden kann.*

**Beweis** Sei zunächst $L \in NTime(T)$ voraus gesetzt. Es sei dann $M$ eine $T$-zeitbeschränkte nach dem Rate-Verifikations-Prinzip vorgehende NTM mit $L = L_M$. Wähle die Konstante

[9] Bei bekannter Zeitschranke $T$ würde es ausreichen, $u$ von der Länge $cT(n)$ (für eine geeignete Konstante $c$) zu wählen. Ohne Kenntnis einer Zeitschranke würde auch die Länge von $u$ von der NTM „geraten“.

$c > 0$ hinreichend groß, so dass zu jeder Eingabe $w \in L$ der Länge $n$ eine akzeptierende Rechnung von $M$ der Länge $cT(n)$ existiert. Wähle $L'$ als die Sprache aller Wörter der Form $\langle u, w\rangle$ mit der Eigenschaft, dass $M$, angesetzt auf die um Binärstring $u$ erweiterte Eingabe $w$, in der deterministischen Verifikationsphase eine akzeptierende Rechnung durchführt. Offensichtlich genügen $c$ und $L$ der Bedingung (8). Bleibt zu zeigen, dass die Mitgliedschaft von $\langle u, w\rangle$ in $L'$ von einer geeigneten DTM $M'$ in $O(T(|w|)$ vielen Schritten getestet werden kann. Wir erhalten eine solche DTM $M'$, indem wir aus Eingabe $\langle u, w\rangle$ zunächst die Wörter $u$ und $w$ extrahieren und anschließend das deterministische Programm für die Verifikationsphase von $M$ verwenden.
Sei nun umgekehrt voraus gesetzt, dass eine Konstante $c$ und eine Sprache $L' \in DTime(T)$ mit den im Satz beschriebenen Eigenschaften existieren. Es bezeichne $M'$ die DTM, welche die Mitgliedschaft von $\langle u, w\rangle$ in $L'$ in $O(|u| + T(|w|))$ Schritten testet. Wir erhalten eine $T$-zeitbeschränkte NTM $M$ mit $L = L_M$, indem wir zunächst einen String $u \in \Sigma^*$ raten, dann $\langle u, w\rangle$ der DTM $M'$ zum „Fraß vorwerfen“ und genau dann akzeptieren, wenn $M'$ akzeptiert. Gemäß (8) hat $M$ auf Eingabe $w \in \Sigma^n$ nur dann eine akzeptierende Rechnung, falls $w \in L$. Falls $w \in L$, dann gibt es sogar eine akzeptierende Rechnung der Länge $O(T(n))$. □

In der Logik bezeichnet ein *$k$-stelliges Prädikat* eine Funktion $Q(x_1, \ldots, x_k)$ mit dem Booleschen Wertebereich $\{0, 1\}$ (wobei 1 für „true“ und 0 für „false “ steht). Es bezeichne $\langle x_1, \ldots, x_k\rangle$ eine in Linearzeit berechen- und invertierbare injektive Abbildung, die $k$-Tupeln $(x_1, \ldots, x_k)$ von Strings einen String $\langle x_1, \ldots, x_k\rangle$ über dem gleichen Alphabet zuordnet. Wir sagen, dass $Q$ von einer $T$-zeitbeschränkten bzw. $S$-platzbeschränkten DTM berechnet werden kann, wenn eine $T$-zeitbeschränkte bzw. $S$-platzbeschränkte DTM existiert, die als Akzeptor von

$$L_Q := \{\langle x_1, \ldots, x_k\rangle |\ Q(x_1, \ldots, x_k)\}$$

auftritt. In der Sprache der Logiker liest sich das Rate-Verifikations-Theorem dann so:

**Folgerung 3.4** *Es gilt $L \in NTime(T)$ gdw für eine geeignete Konstante $c \in \mathbb{N}$ ein zweistelliges Prädikat $Q(u, w)$ existiert, welches von einer DTM in $O(|u| + T(|w|))$ Schritten berechnet werden kann und zu $L$ die Beziehung*

$$L = \{w \in \Sigma^* |\ \exists u \in \Sigma^{cT(|w|)} : Q(u, w)\} = \{w \in \Sigma^* |\ \exists u \in \Sigma^* : Q(u, w)\}$$

*unterhält.*

In der Sprache der Logiker werden NTMs gewissermaßen überflüssig. Der Nichtdeterminismus drückt sich in einem Existenz-Quantor aus.

## 3.3 Verhältnis von Determinismus und Nicht-Determinismus

$R, S, T$ seien Ressourcenschranken. Trivialerweise gilt

$$DTimeSpace(T, S) \subseteq NTimeSpace(T, S)$$

und somit auch

$$DTime(T) \subseteq NTime(T) \text{ und } DSpace(S) \subseteq NSpace(S) \ ,$$

da sich eine DTM als Spezialfall einer NTM auffassen lässt.

Wie steht es umgekehrt mit deterministischen Simulationen nicht-deterministischer Maschinen? Zu diesem Thema werden wir drei Simulationen kennen lernen:

1. Zunächst gehen wir von einer $T$-zeitbeschränkten NTM aus und simulieren diese mit einer $2^{O(T)}$-zeit- und $T$-platzbeschränkten DTM. Die Simulation beruht auf dem Rate-Verifikationsprinzip verbunden mit „Exhaustive Search".

2. Anschließend simulieren wir eine $S$-platzbeschränkte NTM durch eine $S^2$-platzbeschränkte DTM. Die Simulation benutzt Rekursion und die Technik des „Ariadne-Fadens". Das Resultat ist als „Satz von Savitch" bekannt.

3. Schließlich simulieren wir eine $S$-platzbeschränkte NTM durch eine $2^{O(S)}$-zeitbeschränkte DTM. Die Simulation benutzt das Konzept des Konfigurationsgraphen und Graphexplorationstechniken.

Der folgende Satz besagt, dass NTMs prinzipiell durch DTMs simuliert werden können, (vermutlich) aber nur unter wesentlichem Verlust an Zeiteffizienz:

**Satz 3.5** *Für jede platzkonstruierbare Zeitschranke $T$ gilt:*

$$NTime(T) \subseteq DTimeSpace\left(2^{O(T)}, T\right) \subseteq DSpace(T) \ .$$

**Beweis** Sei $L \in NTime(T)$ und $L' \in DTime(T)$ die im Rate-Verifikations-Theorem beschriebene Sprache mit

$$w \in L \Leftrightarrow \exists u \in \Sigma^{cT(|w|)} : \langle u, w \rangle \in L' \ .$$

Wir können eine $T$-zeit- (und somit auch $T$-platzbeschränkte) DTM $M'$ mit $L_{M'} = L'$ zu einem Akzeptor von $L$ machen, indem wir $M'$ die Eingaben $\langle u, w \rangle$ für

$$u = \epsilon, 0, 1, 00, 01, 10, 11, 000, \ldots$$

der Reihe nach „zum Fraß vorwerfen". Dabei werden zwei Abbruchbedingungen verwendet:

1. $M'$ akzeptiert $\langle u, w \rangle$.
   In diesem Falle akzeptieren wir $w$ und stoppen die Simulation.

2. Binärstring $u$ würde in der nächsten Iteration die Länge $cT(n)$ überschreiten.
   In diesem Falle verwerfen wir $w$ und stoppen die Simulation.

Beachte, dass die zweite Abbruchbedingung wegen der Platzkonstruierbarkeit von $T$ (und somit auch $cT$) leicht erkannt werden kann. Wir kommen nun zur Zeit- und Platzanalyse. Schlimmstenfalls müssen wir $M'$ für alle Binärstrings $u$ der Maximallänge $cT(n)$ auf $\langle u, w \rangle$ ansetzen. Dies ergibt die Zeitschranke $2^{O(T(n))}$. Da wir jede Simulation von $M'$ auf $\langle u, w \rangle$

auf dem gleichen Bandsegment ablaufen lassen können, brauchen wir nur den Platz, den $M'$ benötigt plus den Platz den der Binärzähler $u$ benötigt. Dies ergibt die Platzschranke $T(n)$. □

Bevor wir zur nächsten Simulation über gehen, schieben wir eine kleine Zwischenüberlegung ein. Bei einer platzbeschränkten Turing-Maschine ist auf Anhieb nicht klar, ob sie überhaupt auf jeder Eingabe nach endlich vielen Schritten anhält. Allerdings gibt es nur endlich viele Konfigurationen, in die sie auf Eingaben der Länge $n$ geraten kann. Für eine $S$-platzbeschränkte k-Band Turing-Maschine mit einem zusätzlichen Read-only Eingabeband zum Beispiel erhalten wir für die Menge $\mathcal{K}_n$ der möglichen Konfigurationen für Eingaben der Länge $n$ die folgende Ungleichung:

$$|\mathcal{K}_n| \leq |Z| \cdot n \cdot S^k(n) \cdot |\Gamma|^{kS(n)} \ .$$

Hierbei ist $|Z|$ ist die Anzahl der Zustände, $n$ die Anzahl der Positionen des Lesekopfes für die Eingabe, $S(n)$ die Anzahl der Positionen des Kopfes für ein Arbeitsband und $|\Gamma|^{S(n)}$ die Anzahl der Beschriftungen eines Arbeitsbandes. Da es insgesamt $k$ Arbeitsbänder gibt gehen die Faktoren $S(n)$ und $|\Gamma|^{S(n)}$ in die angegebene Schranke in der $k$-ten Potenz ein. Im Falle $S(n) \geq \log n$ gilt offensichtlich:

$$|\mathcal{K}_n| = 2^{O(S(n)} \ .$$

Der folgende Satz besagt, dass es relativ platzefiziente deterministische Simulationen nichtdeterministischer Maschinen gibt (lediglich quadratischer „blow-up“ der Platzschranke):

**Satz 3.6 (von Savitch)** *Es sei $S(n) \geq \log n$ eine platzkonstruierbare Platzschranke. Dann gilt*

$$NSpace(S) \subseteq DSpace^2(S) \ .$$

**Beweis** Es sei $L \in NSpace(S)$ und $M$ eine NTM mit Platzschranke $S$ und $L = L_M$. Wegen des 1. Bandreduktionstheorems dürfen wir voraus setzen, dass $M$ lediglich ein Arbeitsband (und evtl. ein weiteres Read-only Eingabeband) besitzt. Wegen $S(n) \geq \log n$ existiert eine Konstante $c$, so dass $M$ auf Eingaben der Länge $n$ maximal $2^{cS(n)}$ Konfigurationen annehmen kann. Es bezeichne $K_0(w)$ die Startkonfiguration von $M$ (bei Eingabe $w \in \Sigma^n$), $K_+$ eine oBdA eindeutige akzeptierende Endkonfiguration und $\mathcal{K}_n$ die Menge aller auf Eingaben der Länge $n$ denkbaren Konfigurationen (mit $|\mathcal{K}_n| \leq 2^{cS(n)}$). Die Platzkonstruierbarkeit von $S(n)$ wird im Folgenden benötigt, um alle Konfigurationen aus $\mathcal{K}_n$ systematisch (zum Beispiel in lexicographischer Ordnung) durchlaufen zu können.
Betrachte folgende Relation auf $\mathcal{K}_n$:

$$K \vdash_M^t K' :\Leftrightarrow M \text{ kann } K \text{ in maximal } t \text{ Rechenschritten nach } K' \text{ überführen.}$$

Es sei $T := 2^{cS(n)}$. Beachte, dass jede Rechnung von $M$ mit mindestens $T$ Schritten eine Konfiguration mehrfach durchläuft. „Zykelfreie“ Rechnungen von $M$ auf Eingabe $w$ dauern daher weniger als $T$ Schritte. Somit gilt:

$$w \in L_M \Leftrightarrow K_0(w) \vdash_M^T K_+ \ .$$

Wir wollen aufzeigen, wie die Beziehung $K_0(w) \vdash_M^T K_+$ *deterministisch* mit Platzschranke $S^2$ getestet werden kann. Der Schlüssel hierzu liegt in folgendem rekursiven Ansatz:

$$K \vdash_M^{2^s} K' \Leftrightarrow \exists K'' \in \mathcal{K}_n : K \vdash_M^{2^{s-1}} K'' \wedge K'' \vdash_M^{2^{s-1}} K' \ .$$

Es bezeichne TEST eine rekursive Boolesche Prozedur, die auf Parametern $K, K', s$ den Wert TRUE ausgibt gdw $K \vdash_M^{2^s} K'$. Für $s = 0$ kann diese Prozedur leicht ausgewertet werden, da $K \vdash_M^{2^0} K'$ bedeutet, dass $K'$ eine direkte Folgekonfiguration von $K$ bezüglich der NTM $M$ ist. Für $s \geq 1$ kann TEST rekursiv ausgewertet werden:

$$\text{TEST}(K, K', s) := \bigvee_{K'' \in \mathcal{K}_n} \text{TEST}(K, K'', s-1) \wedge \text{TEST}(K'', K', s-1) \ .$$

Um $w \in L$ zu testen, starten wir den Aufruf $\text{TEST}(K_0(w), K_+, cS(n))$. Es resultiert ein Rekursionsbaum der Tiefe $cS(n)$.
Wir haben abschließend die Frage zu klären, wie eine $S^2$-platzbeschränkte DTM $M'$ den Prozeduraufruf $\text{TEST}(K_0(w), K_+, cS(n))$ (und die dadurch ausgelösten rekursiven Aufrufe) implementieren kann. Die wesentliche Idee ist, den Rekursionsbaum niemals vollständig aufs Band zu schreiben, sondern immer nur den aktuellen „Ariadne-Faden“. Dies sind die Informationen, die einem Pfad im Rekursionsbaum von der Wurzel (erster Aufruf von TEST) bis zum aktuellen Knoten (aktueller Aufruf von TEST) entsprechen. Zur Speicherung des Ariadne-Fadens verwendet $M'$ eines ihrer Bänder als Kellerspeicher. Zu jedem neuen Knoten auf dem Ariadne-Faden (neuer rekursiver Aufruf $\text{TEST}(K, K', s)$) wird dabei ein neuer Informationsblock (genannt $\text{FRAME}(K, K', s)$) in den Keller gepusht. Wir werden weiter unten sehen, dass ein FRAME die Länge $O(S(n))$ besitzt. Da der Ariadne-Faden maximal aus $cS(n)$ Knoten besteht (dies ist die Tiefe des Rekursionsbaumes) ergibt sich somit die Platzschranke $S^2$.
Bleibt zu zeigen, dass eine FRAME-Größe von $O(S(n))$ Zellen ausreicht, um die Rekursion am Laufen zu halten. Beim Aufruf $\text{TEST}(K, K', s)$ werden die folgenden Informationen in den Keller gepusht:

- die aktuellen Eingabeparameter $K, K' \in \mathcal{K}_n$ und $s \in \{1, \ldots, cS(n)\}$
- den aktuellen Wert der lokalen „Laufvariable“ $K'' \in \mathcal{K}_n$ (von der wir annehmen, dass sie die Konfigurationen aus $\mathcal{K}_n$ in lexicographischer Ordnung durchläuft)
- die Boolesche Variable SUCCESS (initialisiert auf FALSE) zur Speicherung des Ergebniswertes des ersten rekursiven Aufrufs $\text{TEST}(K, K'', s-1)$

Wir bezeichnen das Bandsegment mit diesen Informationen mit $\text{FRAME}(K, K', s)$. Offensichtlich reichen $O(S(n))$ Zellen für einen FRAME aus.
Zu jeder festen Zwischenkonfiguration $K'' \in \mathcal{K}_n$ verfährt $M'$ wie folgt:

1. Der erste rekursive Aufruf $\text{TEST}(K, K'', s-1)$ wird getätigt (d.h., $\text{FRAME}(K, K'', s-1)$ wird angelegt).

2. Bei erfolgloser Rückkehr aus Aufruf 1 wird $K''$ lexicographisch inkrementiert (falls möglich) und zu Schritt 1 zurück gegangen. Falls jedoch $K''$ bereits die lexicographisch letzte Konfiguration in $\mathcal{K}_n$ war (keine weitere Inkrementierung möglich) wird der Aufruf TEST$(K, K', s)$ erfolglos terminiert, d.h., $M'$ löscht FRAME$(K, K', s)$ und kehrt erfolglos in den darunter liegenden FRAME (sofern vorhanden) zurück.

3. Bei erfolgreicher Rückkehr aus Aufruf 1 wird SUCCESS auf TRUE gesetzt und der zweite rekursive Aufruf TEST$(K'', K', s-1)$ getätigt.

4. Bei erfolgloser Rückkehr aus Aufruf 2 wird SUCCESS auf FALSE zurück gesetzt und ansonsten so vorgegangen wie bei der erfolglosen Rückkehr aus Aufruf 1.[10]

5. Bei erfolgreicher Rückkehr aus Aufruf 2 wird der Aufruf TEST$(K, K', s)$ erfolgreich terminiert, d.h., $M$' löscht FRAME$(K, K', s)$ und kehrt erfolreich in den darunter liegenden FRAME zurück.

Bei dieser Vorgehensweise merkt sich $M'$ in der endlichen Kontrolle, ob sie sich in einer PUSH- (neuer rekursiver Aufruf, neuer FRAME) oder POP-Phase (Rückkehr aus einem Aufruf, FRAME löschen) befindet. In einer POP-Phase merkt sie sich zudem, ob sie aus dem Aufruf des zuletzt gelöschten FRAME's erfolgreich oder erfolglos zurück kehrt.
Wenn $M'$ irgendwann ihren Keller vollständig geleert hat, wurde gerade der FRAME des initialen Aufrufes TEST$(K_0(w), K_+, cS(n))$ gelöscht. Zu diesem Zeitpunkt weiß $M'$ in der endlichen Kontrolle, ob dieser Aufruf erfolgreich war. Sie akzeptiert die Eingabe $w$ gdw dies der Fall ist.
Aus unserer Diskussion geht hervor, dass $w \in L$ von $M'$ korrekt getestet und dass die Platzschranke $S^2$ respektiert wird. □

**Satz 3.7** *Es sei $S(n) \geq \log n$ eine platzkonstruierbare Platzschranke. Dann gilt*

$$NSpace(S) \subseteq DTime(2^{O(S)}) \ .$$

**Beweis** Es sei $L \in NSpace(S)$ und $M$ eine $S$-platzbeschränkte NTM mit $L = L_M$. Konstante $c > 0$, $K_0(w)$, $K_+$ und $\mathcal{K}_n$ mit $|\mathcal{K}_n| \leq 2^{cS(n)}$ seien gewählt wie im Beweis zum Satz von Savitch. Wir betrachten diesmal $\mathcal{K}_n$ als die Knotenmenge eines Digraphen $G_n(M)$, wobei wir eine gerichtete Kante von $K$ nach $K'$ ziehen, wenn $K'$ eine direkte Folgekonfiguration von $K$ bezüglich der NTM $M$ ist. Offensichtlich gilt

$$w \in L_M \Leftrightarrow \text{Es gibt in } G_n(M) \text{ einen Pfad von } K_0(w) \text{ nach } K_+.$$

Der Digraph $G_n(M)$ hat maximal $2^{cS(n)}$ Knoten und maximal $(2^{cS(n)})^2 = 2^{2cS(n)}$ Kanten. Erreichbarkeitsprobleme, wo nach Pfadverbindungen zwischen zwei Knoten gefragt wird, sind mit einfachen Graphexplorationstechniken zeiteffizient lösbar. Eine DTM benötigt bei Digraphen der Größe $N$ hierfür maximal $O(N^k)$ Schritten (für eine geeignete Konstante $k$). Wegen $(2^{2cS(n)})^k = 2^{2kcS(n)}$ ergibt sich die Zeitschranke $2^{O(S)}$. □

Der im Beweis verwendete Digraph $G_n(M)$ heißt der *Konfigurationsgraph* von $M$.

[10] Wie merkt $M'$, ob sie aus dem 1. oder 2. rekursiven Aufruf für Zwischenkonfiguration $K''$ zurück kehrt? Antwort: am Booleschen Wert von SUCCESS. TRUE zeigt an, dass die Rückkehr aus die 2. Aufruf erfolgte.

## 3.4 Die Platz-Zeit-Hierarchie

Wir fügen den uns inzwischen bekannten Beziehungen zwischen Komplexitätsklassen noch die folgenden hinzu:

$$DTime(R) \subseteq DSpace(R) \text{ und } NTime(R) \subseteq NSpace(R) \ .$$

Diese ergeben sich einfach daraus, dass jeder Besuch einer noch unbesuchten Zelle mindestens einen Rechenschritt erfordert. Es ergibt sich nun die folgende Situation für eine platzkonstruierbare Ressourcenschranke $R$:

$$\begin{aligned} DSpace(R) \subseteq NSpace(R) \subseteq DTime\left(2^{O(R)}\right) &\subseteq NTime\left(2^{O(R)}\right) \\ &\subseteq NSpace\left(2^{O(R)}\right) = DSpace\left(2^{O(R)}\right) \ . \end{aligned}$$

Bei der letzten Gleichung wurde der Satz von Savitch benutzt.

# 4 Die universelle Turing-Maschine

Wir sind von modernen programmierbaren Computern gewohnt, dass es sich um „general purpose“ Rechner handelt, die beliebige Rechnungen ausführen können, solange diese durch Programme beschreibbar sind. Ein „general purpose“ Rechner hat streng genommen zwei Eingaben:

- das auszuführende Programm
- die eigentlichen Eingabedaten

Diese Grundvorstellung ist auf Turing-Maschinen übertragbar und führt zum Konzept der universellen Turing-Maschine. Um aber das Programm (sprich: die Turing-Tafel) einer speziellen Turing-Maschine $M$ zur Eingabe einer universellen Maschine machen zu können, brauchen wir ein Kodierungsschema, welches $M$ einen (binären) Kodierungsstring zuordnet.[11]

Wir besprechen als erstes ein für unsere Zwecke geeignetes Kodierungsschema. Bei einer 1-Band Turing-Maschine $M$ mit Zustandsmenge

$$Z = \{z_0, \ldots, z_{r-1}\}$$

und Bandalphabet

$$\Gamma = \{a_0, \ldots, a_{s-1}\}$$

können wir jeden Zustand $z_i$ mit der Nummer $i \in \{0, \ldots, r-1\}$ und jedes Symbol $a_j$ mit der Nummer $j \in \{0, \ldots, s-1\}$ identifizieren. Eine Richtungsangabe $d \in \{L, R, N\}$ für die Kopfbewegung können wir auch als Nummer auffassen, indem wir $L$ (Bewegung nach links) mit der Nummer 1, $R$ (Bewegung nach rechts) mit der Nummer 2 und $N$ (keine Bewegung) mit der Nummer 0 identifizieren. Auf diese Weise kann ein Eintrag

$$\delta(z_i, a_j) = (z_{i'}, a_{j'}, d)$$

der Turing-Tafel mit dem String

$$\#\#\text{bin}(i)\#\text{bin}(j)\#\text{bin}(i')\#\text{bin}(j')\#N(d) \in \{0, 1, 2, \#\}^*$$

identifiziert werden, wobei $N(d) \in \{0, 1, 2\}$ die Nummer der Richtungsangabe $d$ bezeichnet und $\text{bin}(n)$ die Binärdarstellung von $n$. Wir erhalten einen *(reduzierten) Kodierungsstring für* $M$, indem wir den String $\text{bin}(r)\#\text{bin}(s)$ (genannt die *Präambel von* $M$) mit allen Strings für die Einträge der Turing-Tafel von $M$ konkatenieren. Aus beweistechnischen Gründen wird es später vorteilhaft sein, für jede Turing-Maschine unendlich viele Kodierungsstrings zur Verfügung zu haben. Deshalb erlauben wir dem reduzierten Kodierungsstring für $M$ einen beliebigen String aus $\{\#\}^+$ voran zu stellen und sprechen dann von einem *(expandierten) Kodierungsstring* von $M$. Schließlich können wir mit Hilfe der Substitution

$$0 \mapsto 00, 1 \mapsto 01, 2 \mapsto 10, \# \mapsto 11$$

[11] In der Rekursionstheorie wird dieser Kodierungsstring *Gödelnummer von M* genannt und die Verwandlung von $M$ in eine Nummer bezeichnet man als *Gödelisierung*.

zu binären (reduzierten bzw. expandierten) Kodierungsstrings übergehen. Nicht jeder Binärstring ist ein syntaktisch korrekt gebildeter Kodierungsstring einer Turing-Maschine. Für illegale Strings legen wir aber eine (beliebig aber fest gewählte) „Default-Turing-Maschine“ fest, so dass ab jetzt folgendes gilt:

1. Jede Einband Turing-Maschine ist durch unendlich viele binäre Kodierungsstrings repräsentiert.

2. Jeder Binärstring repräsentiert eine Einband Turing-Maschine.

Wir können für $k$-Band Turing-Maschinen nach einem analogen Schema vorgehen. Im folgenden bezeichne $W_k(M)$ den reduzierten binären Kodierungsstring für eine $k$-Band Turing-Maschine $M$. Umgekehrt bezeichne (für eine vorher vereinbarte Anzahl $k$ von Bändern) $M_w$, die durch den Binärstring $w$ gegebene $k$-Band Turing-Maschine.

**Satz 4.1** *Für jedes $k \geq 2$ gibt es eine universelle $k$-Band DTM $U_k$, die auf einer Eingabe der Form $\langle w, x \rangle$ mit einem legalen[12] (evtl. expandierten) Kodierungsstring $w$ der Länge $p$ für eine $k$-Band DTM $M_w$ und einem eigentlichen Eingabestring $x$ der Länge $n$ das folgende leistet:*

- *Extraktion des reduzierten Kodierungsstrings $w_0$ (sagen wir der Länge $p_0$) und des eigentlichen Eingabestrings $x$ in $O(p+n)$ Schritten*

- *Simulation von $M_w$ auf Eingabe $x$*

*Weiterhin gilt:*

1. *Ein Schritt von $M_w$ wird von $U_k$ in $O(p_0)$ Schritten simuliert.*

2. *Wenn $M_w$ auf Eingabe $x$ maximal $T(n)$ Schritte rechnet, so dauert die Simulation maximal $O(p_0 T(n))$ Schritte.*

3. *Wenn $M_w$ im Laufe der Rechnung auf Eingabe $x$ maximal $S(n)$ Zellen besucht, dann besucht $U_k$ im Laufe der Simulation maximal $O(p_0 S(n))$ Zellen.*

*Für 1-Band DTMs gelten die analogen Aussagen, außer dass die Zeitschranke der Simulation einen „blow-up“ um den Faktor $p_0^2$ (statt $p_0$) erleidet.*

Der Beweis dieses Satzes wird in der Vorlesung vorgerechnet. Ein analoger Satz gilt auch für NTMs.

[12] Wenn wir verlangen, dass $U_k$ illegale Strings erkennt (und dann die „Default-DTM“ simuliert) verschlechtert sich die Laufzeit der Simulation etwas (s. Übung). Wir sind im Folgenden aber bereits zufrieden, wenn sie für legale Kodierungsstrings $w$ korrekt arbeitet.

# 5 Platz- und Zeithierarchien

## 5.1 Der allgemeine Hierarchiesatz

Seien $R_1, R_2 : \mathbb{N} \to \mathbb{N}$ zwei Ressourcenschranken und $C, D$ zwei Ressourcenarten (wie zum Beispiel *DTime* oder *NSpace*). Um einen Hierarchiesatz der Form

$$C(R_1) \subset D(R_2)$$

zu beweisen, genügen folgende Voraussetzungen:

1. Die Ressourcenschranke $R_2$ ist *kontrollierbar*, d.h., jede Turing-Maschine kann so modifiziert werden, dass sie ihre Rechnung frühzeitig abbricht, falls ansonsten die Ressourcenschranke durchbrochen würde.

2. Die Komplexitätsklasse $D(R_2)$ ist *abgeschlossen unter Komplement*, d.h, aus $L \in D(R_2)$ folgt stets $\bar{L} = \Sigma^* \setminus L \in D(R_2)$.

3. $D(R_2)$ ist *ist universell für* $C(R_1)$, d.h., es gibt Turing-Maschinen $U$ und $U'$ mit folgenden Eigenschaften:

   (a) Wenn $U$ auf Eingabe $\langle w, x\rangle$ gestartet wird, dann simuliert sie (unter Kontrolle der Ressourcenschranke $R_2$) die Turing-Maschine $M_w$ auf Eingabe $x$. Falls die Simulation frühzeitig abgebrochen wird, wird Eingabe $\langle w, x\rangle$ verworfen. Falls die Simulation unter Beachtung der Ressourcenschranke $R_2$ zu Ende geführt werden kann, dann wird $\langle w, x\rangle$ akzeptiert gdw $x$ von $M_w$ akzeptiert wird.

   (b) Für alle Turing-Maschinen $M$, die die Ressourcenschranke $R_1$ respektieren (was $L_M \in C(R_1)$ impliziert), für alle $w$ mit $M = M_w$ und alle hinreichend langen Strings $x$, kann $U$ die Simulation von $M_w$ auf $x$ (unter Beachtung der Ressourcenschranke $R_2$) zu Ende führen.

   (c) Wenn $U'$ auf Eingabe $w$ gestartet wird, dann berechnet sie zunächst $\langle w, w\rangle$ und arbeitet auf dieser Eingabe wie $U$. Falls $U'$ also nicht frühzeitig abbricht, dann wird Eingabe $w$ akzeptiert gdw $w$ von $M_w$ akzeptiert wird.

   (d) Für alle Turing-Maschinen $M$, die die Ressourcenschranke $R_1$ respektieren (was $L_M \in C(R_1)$ impliziert) und für alle hinreichend langen expandierten Kodierungsstrings $w$ mit $M = M_w$ kann $U'$ die Simulation von $M_w$ auf $w$ (unter Beachtung der Ressourcenschranke $R_2$) zu Ende führen.

**Satz 5.1 (Allgemeiner Hierarchiesatz)** *Unter den genannten Voraussetzungen an* $C(R_1)$ *und* $D(R_2)$ *gilt* $C(R_1) \subset D(R_2)$.

**Beweis** Wir beweisen zunächst $C(R_1) \subseteq D(R_2)$. Sei $L \in C(R_1)$, $M$ ein Akzeptor von $L$, der die Ressourcenschranke $R_1$ respektiert und $w$ ein Kodierungsstring für $M$. Gemäß unserer Voraussetzungen existiert ein $n_0 \in \mathbb{N}$, so dass $U$ für alle Eingaben $x$ mit $|x| \geq n_0$ die Simulation von $M_w$ auf $x$ unter Beachtung der Ressourcenschranke $R_2$ zu Ende führen kann. Betrachte folgende Variante $U[w]$ von $U$:

- Auf Eingabe $x$ mit $|x| \geq n_0$ berechnet $U[w]$ zunächst $\langle w, x \rangle$ und arbeitet auf dieser Eingabe dann wie $U$.

- Eingaben $x$ mit $|x| < n_0$ werden in Linearzeit mit Hilfe der endlichen Kontrolle bearbeitet und akzeptiert gdw $x \in L$.

Offensichtlich ist $U[w]$ ein Akzeptor von $L$, der die Ressourcenschranke $R_2$ respektiert, woraus sich $L \in D(R_2)$ ergibt.
Wir haben nachzuweisen, dass eine Sprache in $D(R_2) \setminus C(R_1)$ existiert. Zu diesem Zweck betrachten wir zunächst die Sprache $L_{U'}$. Da $U'$ ihre Ressourcenschranke kontrolliert, gilt $L_{U'} \in D(R_2)$. Da $D(R_2)$ unter Komplement abgeschlossen ist, gilt weiterhin $\overline{L_{U'}} \in D(R_2)$. Wir behaupten, dass $\overline{L_{U'}} \notin C(R_1)$ und machen (im Sinne eines Beweises durch Widerspruch) die

**Annahme** $\overline{L_{U'}} \in C(R_1)$.

**Widerspruch** Gemäß der Annahme existiert ein Akzeptor $M$ von $\overline{L_{U'}}$, der die Ressourcenschranke $R_1$ respektiert. Gemäß der im allgemeinen Hierarchiesatz geltenden Voraussetzungen kann $U'$ für hinreichend lange Kodierungsstrings $w$ von $M$ die Simulation von $M_w$ auf $w$ unter Beachtung der Ressourcenschranke $R_2$ zu Ende führen. Es ergibt sich der Widerspruch

$$w \in \overline{L_{U'}} \Leftrightarrow w \in L_M \Leftrightarrow w \in L_{U'} \quad .$$

Hierbei gilt die erste Äquivalenz, weil $M$ ein Akzeptor von $\overline{L_{U'}}$ ist; die zweite Äquivalenz gilt, weil $U'$ die Simulation von $M = M_w$ auf $w$ zu Ende führen kann.

□

## 5.2 Die Voraussetzungen des allgemeinen Hierarchiesatzes

Um die Kontrollierbarkeit von Ressourcenschranken zu gewährleisten, werden wir bei Zeitschranken die Zeitkonstruierbarkeit und bei Platzschranken die Platzkonstruierbarkeit fordern. Wie damit die Beachtung der Ressourcenschranke erzwungen werden kann, wurde an früherer Stelle bereits diskutiert.

Zu einer Komplexitätsklasse $\mathcal{C}$ bezeichne co-$\mathcal{C}$ die Klasse der zugehörigen Komplementärsprachen:

$$\text{co-}\mathcal{C} := \{\bar{L} |\ L \in \mathcal{C}\} \quad .$$

Der Abschluss unter Komplement ist für deterministische Komplexitätsklassen relativ leicht zu beweisen:

**Satz 5.2** *co-$DTime_k(T) = DTime_k(T)$.*

**Beweis** Sei $L \in DTime_k(T)$ und $M$ eine $T$-zeitbeschränkte $k$-Band DTM mit $L = L_M$. Wir erhalten eine $T$-zeitbeschränkte $k$-Band DTM $\bar{M}$ mit $L_{\bar{M}} = \overline{L_M}$, indem wir bei $M$ akzeptierende und nicht-akzeptierende Zustände vertauschen. □

**Folgerung 5.3** *co-DTime*$(T) = DTime(T)$

**Satz 5.4** *Für Platzschranken* $S(n) \geq \log n$ *gilt co-DSpace*$(S) = DSpace(S)$.

**Beweis** Wir führen den Beweis unter der Zusatzvoraussetzung, dass $S$ platzkonstruierbar ist. In der Übung diskutieren wir einen Beweis, der ohne diese Voraussetzung auskommt. Das Argument des Vertauschens von akzeptierenden und nicht-akzeptierenden Zuständen ist hier nicht ausreichend, da $M$ eine Eingabe $w$ auch dadurch verwerfen könnte, dass sie unendlich lange rechnet. Die wesentliche Schwierigkeit besteht also darin zu zeigen, dass $M$ so normalisiert werden kann, dass sie auf jeder Eingabe nach endlich vielen Schritten stoppt. Da die Anzahl der Konfigurationen auf Eingaben der Länge $n$ durch $2^{cS(n)}$ beschränkt ist, genügt es hierzu, mit einem Binärzähler die Anzahl der Rechenschritte mit zu zählen und bei Zählerstand $2^{cS(n)}$ (also eine Binärzahl mit einer Eins gefolgt von $cS(n)$ Nullen) verwerfend zu stoppen. (Beachte, dass dieser Zählerstand erkannt werden kann, da $S$ als platzkonstruierbar voraus gesetzt wurde.) □

Überraschenderweise kann man den Abschluss unter Komplement auch für nicht-deterministische Platzkomplexitätsklassen[13] nachweisen:

**Satz 5.5 (Immerman, Szelepscényi)** *Es sei* $S(n) \geq \log n$ *eine platzkonstruierbare Platzschranke. Dann gilt co-NSpace*$(S) = NSpace(S)$.

In der Vorlesung *Theoretische Informatik* wurde der Beweis für den Spezialfall $S(n) = n$ geführt (Abschluss der kontextsensitiven Sprachen unter Komplement). Da der Beweis des allgemeinen Satzes sehr ähnlich ist, lassen wir ihn aus (s. Übung).

Die Voraussetzung der Universalität wird (bei geeigneter Wahl der Ressourcenschranken $R_1, R_2$) mit Hilfe von Satz 4.1 garantiert werden können.

## 5.3 Spezielle Hierarchiesätze

In diesem Abschnitt können ein paar spezielle Hierarchiesätze „wie reife Früchte“ geerntet werden.

**Satz 5.6** *Es sei* $T_2$ *zeitkonstruierbar und* $T_1 = o(T_2)$. *Dann gilt* $DTime_k(T_1) \subset DTime_{k+1}(T_2)$.

Der Beweis ergibt sich unmittelbar aus dem allgemeinen Hierarchiesatz, dessen Voraussetzungen (wie eine einfache Überlegung zeigt) erfüllt sind. (Band $k+1$ dient dabei der „universellen DTM“ $U$ zur Kontrolle der Zeitschranke.)

[13] Für nicht-deterministische Zeitkomplexitätsklassen ist die Frage des Abschlusses unter Komplement ein offenes Problem.

**Folgerung 5.7** *Es sei $T_2$ zeitkonstruierbar und $T_1 \log T_1 = o(T_2)$. Dann gilt $DTime(T_1) \subset DTime(T_2)$.*

**Beweis** Die Voraussetzung der Universalität gilt nicht so ohne Weiteres, da die „universelle DTM" mit Zeitschranke $T_2$ irgend eine feste Anzahl $k$ von Bändern zugewiesen bekommt, dann aber evtl. nicht ohne Zeitverlust DTMs mit Zeitschranke $T_1$ und größerer Anzahl von Bändern simulieren kann. Der Beweis ergibt sich aber unter Einsatz des 2. Bandreduktionstheorems wie folgt:

$$DTime(T_1) \subseteq DTime_2(T_1 \log T_1) \subset DTime_3(T_2) \subseteq DTime(T_2) \ .$$

□

Es sei daran erinnert, dass für Platzschranken die Bandanzahl irrelevant ist (s. 1. Bandreduktionstheorem). Wir formulieren nun den Platzhierarchiesatz, dessen Beweis sich unmittelbar daraus ergibt, dass die Voraussetzungen des allgemeinen Hierarchiesatzes erfüllt sind:

**Satz 5.8** *Es sei $S_2(n) \geq \log n$ platzkonstruierbar und $S_1 = o(S_2)$. Dann gilt $DSpace(S_1) \subset DSpace(S_2)$ und $NSpace(S_1) \subset NSpace(S_2)$.*

Abschließend geben wir ein zwei weitere Hierarchiesätze (jeweils ohne Beweis) an:

**Satz 5.9** *Es sei $T_2$ zeitkonstruierbar und $T_1 = o(T_2)$. Dann gilt für alle $k \geq 2$: $DTime_k(T_1) \subset DTime_k(T_2)$.*

Im Vergleich zu Satz 5.6 benötigen die $T_2$-zeitbeschränkten DTMs hier kein Band $k+1$. Im Beweis wird gezeigt, dass der Zähler zur Kontrolle der Zeitschranke auf raffinierte Weise auf den Bändern $1, \ldots, k$ untergebracht werden kann, ohne die Zeiteffizienz der „universellen DTM" $U$ zu beeinträchtigen.

**Satz 5.10** *Es sei $T_2$ zeitkonstruierbar und $T_1(n) \leq T_1(n+1) = o(T_2(n))$. Dann gilt für alle $k \geq 1$: $NTime_k(T_1) \subset NTime_{k+1}(T_2)$.*

Der Beweis dieses Satzes kann nicht auf den allgemeinen Hierarchiesatz zurück greifen, da wir den Abschluss unter Komplement nicht garantieren können. Aus Satz 5.10 kann man (s. Übung) folgendes ableiten:

**Folgerung 5.11** *Es sei $T_2$ zeitkonstruierbar und $T_1 = o(T_2)$. Dann gilt $NTime(T_1) \subset NTime(T_2)$.*

# 6 P versus NP

Es bezeichne POL die Funktionsklasse der Polynome (mit Koeffizienten aus $\mathbb{N}$). Im weiteren Verlauf der Vorlesung sind wir besonders an den folgenden Komplexitätsklassen interessiert:

$$\begin{aligned} \mathcal{L} &:= DSpace(\log n) \\ \mathcal{NL} &:= NSpace(\log n) \\ P &:= DTime(POL) \\ NP &:= NTime(POL) \\ PSpace &:= DSpace(POL) \overset{Savitch}{=} NSpace(POL) \end{aligned}$$

Wir erinnern nochmals an die (sogenannte) Platz-Zeit-Hierarchie zu einer beliebig vorgegebenen platzkonstruierbaren Ressourcenschranke $S(n) \geq \log n$:

$$\begin{aligned} DSpace(R) \subseteq NSpace(R) \subseteq DTime\left(2^{O(R)}\right) &\subseteq NTime\left(2^{O(R)}\right) \\ &\subseteq NSpace\left(2^{O(R)}\right) = DSpace\left(2^{O(R)}\right) \ . \end{aligned}$$

Speziell für $R(n) = \log n$ ergibt sich

$$\mathcal{L} \subseteq \mathcal{NL} \subseteq P \subseteq NP \subseteq PSpace \ . \tag{9}$$

Dem Platzhierarchiesatz für NTMs entnehmen wir, dass die Inklusion

$$\mathcal{NL} \subset PSpace$$

echt ist. Die Inklusionskette (9) muss also an einer Stelle zwischen $\mathcal{NL}$ und *PSpace* zerreißen. Es wird vermutet, dass jede einzelne Inklusion in (9) echt ist. Dies konnte bis heute jedoch weder bewiesen noch widerlegt werden. Die berühmteste[14] dieser offenen Fragen betrifft die Klassen $P$ und $NP$:

*Ist die Inklusion $P \subseteq NP$ echt?*

In diesem Kapitel werden wir versuchen, den derzeitigen Stand der „P versus NP“ Frage anzudeuten. Insbesondere gehen wir auf die Theorie der NP-Härte bzw. NP-Vollständigkeit ein und skizzieren, wie man mit NP-harten Problemen (die sich voraussichtlich einer im „worstcase“ optimalen und zugleich praktikablen Behandlung widersetzen) umgeht.

Bevor wir die Komplexitätsklasse *NP* näher behandeln, leiten wir aus dem allgemeinen Rate-Verifikationstheorem die folgende Beziehung zwischen den Klassen $P$ und *NP* ab:

**Folgerung 6.1** *Es gilt $L \in NP$ gdw eine Sprache $L' \in P$ und ein Polynom $p$ existieren, so dass*

$$L = \{w \in \Sigma^* |\ \exists u \in \Sigma^{p(|w|)} : \langle u, w\rangle \in L'\} = \{w \in \Sigma^* |\ \exists u \in \Sigma^* : \langle u, w\rangle \in L'\} \ .$$

**Folgerung 6.2** *Es gilt $L \in NP$ gdw ein Polynom $p$ und ein zweistelliges in (deterministisch) Polynomialzeit berechenbares Prädikat $Q(u, w)$ existieren, so dass*

$$L = \{w \in \Sigma^* |\ \exists u \in \Sigma^{p(|w|)} : Q(u, w)\} = \{w \in \Sigma^* |\ \exists u \in \Sigma^* : Q(u, w)\} \ .$$

[14] Es handelt sich um das erste „Millenium Prize Problem“ auf der Liste des „Clay Mathematics Institute“ (www.claymath.org).

**Existenz kurzer effizient entscheidbarer Zertifikate** Man nennt in diesem Zusammenhang den String $u$ das „Zertifikat“ für die Mitgliedschaft von $w$ in $L$. Sprachen $L \in NP$ sind also dadurch charakterisiert, dass zum Nachweis von $w \in L$ ein Zertifikat $u$ existiert, dessen Länge polynomiell in $n = |w|$ beschränkt ist und das in Polynomialzeit überprüft werden kann.

## 6.1 Probleme in NP

Wir wollen in diesem Abschnitt demonstrieren, dass erstens die Klasse *NP* sehr reichhaltig ist und wichtige Grundlagenprobleme enthält, und dass zweitens die Mitgliedschaft zu *NP* meist verblüffend einfach zu beweisen ist.

Da Sprachen mit Entscheidungsproblemen identifiziert werden können, aber die Formulierung als Problem anregender klingt, sprechen wir oft von Problemen statt von Sprachen. Von folgenden Problemen lässt sich leicht zeigen, dass sie zur Klasse *NP* gehören:

**SAT** Satisfiability (Erfüllbarkeitsproblem der Booleschen Logik)

> **Eingabe** Kollektion $C_1, \ldots, C_m$ von Booleschen Klauseln in $n$ Booleschen Variablen $x_1, \ldots, x_n$. (Eine Boolesche Klausel ist eine Disjunktion von Booleschen Literalen. Ein Boolesches Literal ist eine negierte oder unnegierte Boolesche Variable.)
>
> **Frage** Existiert eine Belegung von $x_1, \ldots, x_n$ mit 0 oder 1, die alle Klauseln erfüllt, d.h., die dazu führt, dass $C_1, \ldots, C_m$ zu 1 ausgewertet werden?

**3-SAT** Einschränkung von SAT auf Eingaben, deren Boolesche Klauseln aus jeweils 3 (paarweise verschiedenen) Booleschen Literalen bestehen.

**CLIQUE** Cliquenproblem.

> **Eingabe** Ein ungerichteter Graph $G = (V, E)$ und eine Anzahl $k$.
>
> **Frage** Existiert in $G$ eine Clique der Größe $k$, d.h., eine Menge $C \subseteq V$ der Mächtigkeit $k$, deren Knoten paarweise in $G$ benachbart sind?

**IS** Independent Set (Unabhängige Menge)

> **Eingabe** Ein ungerichteter Graph $G = (V, E)$ und eine Anzahl $k$.
>
> **Frage** Existiert in $G$ eine „independent set“ (unabhängige Menge) der Größe $k$, d.h., eine Menge $U \subseteq V$ der Mächtigkeit $k$, deren Knoten paarweise in $G$ nicht benachbart sind?

**VC** Vertex Cover (Überdeckung mit Knoten)

> **Eingabe** Ein ungerichteter Graph $G = (V, E)$ und eine Anzahl $k$.
>
> **Frage** Existiert in $G$ ein „vertex cover“ (Überdeckung mit Knoten) der Größe $k$, d.h., eine Menge $C \subseteq V$ der Mächtigkeit $k$, die von jeder Kante $e \in E$ mindestens einen Randknoten enthält?

**HS** Hitting Set (Repräsentantensystem)

**Eingabe** Eine Kollektion von Teilmengen $S_1, \ldots, S_m$ aus $\{1, \ldots, n\}$

**Frage** Existiert in $G$ eine „hitting set“ (Repräsentantensystem) der Größe $k$, d.h., eine Menge $R \subseteq \{1, \ldots, n\}$ der Mächtigkeit $k$, die von jeder Menge $S_j \subseteq \{1, \ldots, n\}$ mindestens ein Element enthält?

**SC** Set Cover (Überdeckung mit Mengen)

**Eingabe** Eine Kollektion von Teilmengen $S_1, \ldots, S_m$ aus $\{1, \ldots, n\}$

**Frage** Können $k$ der gegebenen Teilmengen so ausgewählt werde, dass jedes Element der Grundmenge $\{1, \ldots, n\}$ überdeckt ist, d.h., existiert eine Menge $J \subseteq \{1, \ldots, m\}$ der Mächtigkeit $k$ so dass $\{1, \ldots, n\} = \cup_{j \in J} S_j$?

**Subset Sum** Summenproblem

**Eingabe** $n$ Zahlen $a_1, \ldots, a_n \in \mathbb{N}$ sowie die „Summenzahl“ $S \in \mathbb{N}$

**Frage** Kann man eine Auswahl (ohne Wiederholung) aus den Zahlen treffen, welche zur Summe $S$ führt, d.h., existiert eine Menge $I \subseteq \{1, \ldots, n\}$ so dass $\sum_{i \in I} a_i = S$?

**KP** Knapsack (Rucksackproblem)

**Eingabe** $n$ Objekte mit Gewichten $w_1, \ldots, w_n \in \mathbb{N}$ und Nutzen $p_1, \ldots, p_n \in \mathbb{N}$, eine Gewichtsschranke $W$ und eine Nutzenschranke $P$.

**Frage** Kann man einen Rucksack $R$ so packen, dass die Objekte in $R$ einen Gesamtnutzen von mindestens $P$ und ein Gesamtgewicht von höchstens $W$ besitzen, d.h., existiert eine Menge $I \subseteq \{1, \ldots, n\}$, so dass $\sum_{i \in I} p_i \geq P$ und $\sum_{i \in I} w_i \leq W$?

**PARTITION** Zerlegung in zwei gleich große Teilsummen

**Eingabe** $n$ Zahlen $a_1, \ldots, a_n \in \mathbb{N}$.

**Frage** Kann man diese Zahlen in zwei gleich große Teilsummen zerlegen, d.h., existiert eine Teilmenge $I \subseteq \{1, \ldots, n\}$, so dass $\sum_{i \in I} a_i = \sum_{j \notin I} a_j$?

**BP** Bin Packing Problem

**Eingabe** $n$ Objekte der Größen $a_1, \ldots, a_n \in \mathbb{N}$, $m$ Behälter (=bins) der Bingröße $b$.

**Frage** Kann man die $n$ Objekte unter Beachtung der Bingröße in die $m$ Behälter verpacken, d.h., existiert eine Zerlegung von $\{1, \ldots, n\}$ in $m$ paarweise disjunkte Teilmengen $I_1, \ldots, I_m$, so dass $\sum_{i \in I_j} a_i \leq b$ für alle $1 \leq j \leq m$ erfüllt ist?

**DHC** Directed Hamiltonian Circuit (Geichteter Hamiltonscher Kreis)

**Eingabe** Ein Digraph $G = (V, E)$

**Frage** Gibt es in $G$ einen gerichteten Hamiltonschen Kreis, d.h., können wir mit Kanten aus $E$ einen gerichteten Kreis formen, der jeden Knoten aus $V$ genau einmal durchläuft?

**HC** Hamiltonian Circuit (Hamiltonscher Kreis)
Dies ist das entsprechende Problem für (ungerichtete) Graphen.

**TSP** Travelling Salesman Problem (Problem des Handelsreisenden)

**Eingabe** Eine Kostenschranke $K$, $n$ Städte $C_0, \ldots, C_{n-1}$ und eine Distanzmatrix $D = (d_{ij})_{0 \leq i,j \leq n-1}$, wobei $d_{i,j} \in \mathbb{N}$ die Distanz zwischen $C_i$ und $C_j$ angibt

**Frage** Existiert eine Rundreise durch $C_0, \ldots, C_{n-1}$, deren Gesamtlänge $K$ nicht überschreitet, d.h., existiert eine Permutation $\sigma$ von $0, \ldots, n-1$, so dass

$$\sum_{i=0}^{n-1} d_{\sigma(i)\sigma(i+1 \bmod n)} \leq K \ ?$$

**Metrisches TSP** Einschränkung von TSP auf Eingaben, deren Distanzmatrix symmetrisch ist und die Dreiecksungleichung erfüllt.

**COMPOSITES** Menge der fakorisierbaren natürlichen Zahlen

**Eingabe** Eine Zahl $N \in \mathbb{N}$.

**Frage** Ist $N$ faktorisierbar, d.h., existieren zwei natürliche Zahlen $N_1, N_2 \geq 2$ mit $N = N_1 \cdot N_2$?

**GC** Graph Colorability (Graphenfärbungsproblem)

**Eingabe** Ein ungerichteter Graph $G = (V, E)$ und eine Anzahl $k$.

**Frage** Kann man die Knoten von $G$ mit $k$ Farben legal färben, d.h., existiert eine Abbildung $f : V \to \{1, \ldots, k\}$ so für alle Kanten $\{u, v\} \in E$ die Bedingung $f(u) \neq f(v)$ erfüllt ist (verschiedene Farben für benachbarte Knoten)?

**3-GC** Graph 3-Colorability (Graphenfärbungsproblem bei 3 Farben)
Einschränkung von GC auf den Fall $k = 3$.

**Lemma 6.3** *Alle vorgenannten Probleme gehören zur Klasse NP.*

**Beweis** Wir beschränken uns hier auf SAT und CLIQUE. Für COMPOSITES wurde der Beweis an früherer Stelle schon mal geführt. Für die anderen Probleme argumentiere analog.

**SAT** Rate (nichtdeterministisch) eine Belegung $a = (a_1, \ldots, a_n)$ der Booleschen Variablen $x_1, \ldots, x_n$ und verifiziere (deterministisch) durch Auswerten der Booleschen Klauseln $C_1, \ldots, C_m$, dass $C_1(a) = \cdots = C_m(a) = 1$.

**CLIQUE** Rate (nichtdeterministisch) $k$ Knoten $v_1, \ldots, v_k$ aus $V$ und verifiziere (deterministisch) durch Nachsehen in den jeweiligen Adjazenzlisten (oder den jeweiligen Zeilen der Adjazenzmatrix), dass sie paarweise in $G$ benachbart sind.

In jedem der genannten Fälle gibt es für die zu akzeptierenden Eingaben wenigstens eine akzeptierende Rechnung, wohingegen die zu verwerfenden Eingaben keine akzeptierende Rechnung besitzen. □

Diese trivialen Nachweise machen jeweils Gebrauch von der Existenz kurzer und effizient entscheidbarer Zertifikate.

## 6.2 NP-Vollständigkeit

Wenn auch das *P-NP* Problem bisher ungelöst blieb, so ist es dennoch gelungen innerhalb der Klasse *NP* härteste Probleme — sogenannte *NP*-vollständige Probleme — zu identifizieren, deren Zugehörigkeit zu $P$ die Gleichheit von $P$ und *NP* implizieren würde. Das Werkzeug zum Nachweis der *NP*-Vollständigkeit sind die polynomiellen Reduktionen.

**Definition 6.4** *Eine Beziehung $\leq_{POL}$ zwischen formalen Sprachen heißt* abstrakte polynomielle Reduktion, *wenn $\leq_{POL}$ transitiv ist und wenn $L \leq_{POL} L'$ und $L' \in P$ impliziert, dass $L \in P$.*

In Definition 6.4 wurde der Begriff der polynomiellen Reduktion durch abstrakte Bedingungen beschrieben. Die folgenden Definitionen liefern konkrete „Inkarnationen“ dieses abstrakten Begriffes:

**Definition 6.5** *Seien $L, L' \subseteq \Sigma^*$ formale Sprachen über einem Alphabet $\Sigma$. $L$ heißt* polynomiell reduzierbar *auf $L'$, in Zeichen $L \leq_{pol} L'$, wenn eine in Polynomialzeit berechenbare Abbildung $f : \Sigma^* \to \Sigma^*$ existiert mit der Eigenschaft*

$$\forall x \in \Sigma^* : \; x \in L \Leftrightarrow f(x) \in L'. \tag{10}$$

**Definition 6.6** *Eine Funktion $f : \Sigma^* \to \Sigma^*$ heißt „logspace-berechenbar“, wenn sie von einer DTM mit einem „read-only“ Eingabeband und einem „write-only“ Ausgabeband und logarithmisch (in Eingabelänge $n$) vielen besuchten Zellen auf den Arbeitsbändern berechnet werden kann.*
*Falls $L \leq_{pol} L'$ mit einer Reduktionsabbildung, die sogar logspace-berechenbar ist (was Berechenbarkeit in Polynomialzeit impliziert), dann nennen wor $L$* logspace-reduzierbar *auf $L'$ und schreiben $L \leq_{log} L'$.*

**Lemma 6.7** *Die Relationen „$\leq_{pol}$“ und „$\leq_{log}$“ sind abstrakte polynomielle Reduktionen.*

Der Beweis für „$\leq_{pol}$“ sollte aus der Vorlesung *Theoretische Informatik* bekannt sein. Den Beweis für „$\leq_{log}$“ behandeln wir in den Übungen.

**Definition 6.8** *Sei $\leq_{POL}$ eine abstrakte polynomielle Reduktion zwischen formalen Sprachen und $L' \subseteq \Sigma^*$.*

1. *$L'$ heißt NP-*hart *unter Reduktionen vom Typ $\leq_{POL}$, wenn $L \leq_{POL} L'$ für alle $L \in NP$.*
2. *Falls zusätzlich $L' \in NP$ gilt, dann heißt $L'$ NP-*vollständig *unter Reduktionen vom Typ $\leq_{POL}$.*
3. *NP-hart bzw. NP-vollständig ohne explizite Spezifikation des Reduktionstyps bedeute implizit NP-hart bzw. NP-vollständig unter der Reduktion „$\leq_{pol}$“.*
4. *NPC bezeichnet die Klasse der NP-vollständigen formalen Sprachen.*

Eine Hauptanwendung des Konzeptes der *NP*-Härte wird aus folgendem Resultat ersichtlich:

**Lemma 6.9** *Sei $\leq_{POL}$ eine abstrakte polynomielle Reduktion zwischen formalen Sprachen. Aus der Existenz einer (unter Reduktionen vom Typ $\leq_{POL}$) NP-harten Sprache $L'$ mit $L' \in P$ würde dann $P = NP$ folgen.*

Der Beweis ergibt sich unmittelbar aus den Definitionen der polynomiellen Reduktion und der *NP*-Härte.

Lemma 6.9 unterstreicht die Bedeutung von *NP*-harten (und somit auch von *NP*-vollständigen) Sprachen. Das folgende Lemma besagt, dass sich die *NP*-Härte entlang von Reduktionsketten vererbt.

**Lemma 6.10** *Sei $\leq_{POL}$ eine abstrakte polynomielle Reduktion zwischen formalen Sprachen. Falls $L' \leq_{POL} L''$ und $L'$ ist NP-hart unter Reduktionen vom Typ $\leq_{POL}$, dann ist auch $L''$ NP-hart unter Reduktionen vom Typ $\leq_{POL}$.*

Der Beweis ergibt sich unmittelbar aus der Transitivität abstrakter polynomieller Reduktionen und der Definition der *NP*-Härte.

Lemma 6.10 erlaubt einen "Stammbaum" *NP*-harter Probleme (Sprachen) zu kreieren nach folgendem Muster:

**Initialisierung des Stammbaums** Beweise die *NP*-Härte für ein erstes Problem (die Wurzel des Stammbaums).

**Erweiterung des Stammbaums** Reduziere eines der im Stammbaum schon aufgenommenen Probleme polynomiell auf ein neues Problem.

Dann ergibt sich aus der *NP*-Härte des Problems an der Wurzel des Stammbaums durch iterierte Anwendung von Lemma 6.10 die *NP*-Härte eines jeden im Stammbaum enthaltenen Problems. Lemma 6.9 besagt weiterhin, dass die abstrakte polynomielle Lösbarkeit eines der Probleme im Stammbaum die abstrakte polynomielle Lösbarkeit aller *NP*-Probleme impliziert. Abbildung 2 zeigt einen Ausschnitt aus dem Stammbaum *NP*-vollständiger Sprachen, der mit Hilfe von polynomiellen Reduktionen (vom Typ „$\leq_{pol}$“) kreiert werden kann.[15]

[15] Die betreffenden Reduktionen sowie das Theorem von Cook (NP-Vollständigkeit von SAT) wurden in der Vorlesung *Theoretische Informatik* behandelt und werden an dieser Stelle nicht wiederholt.

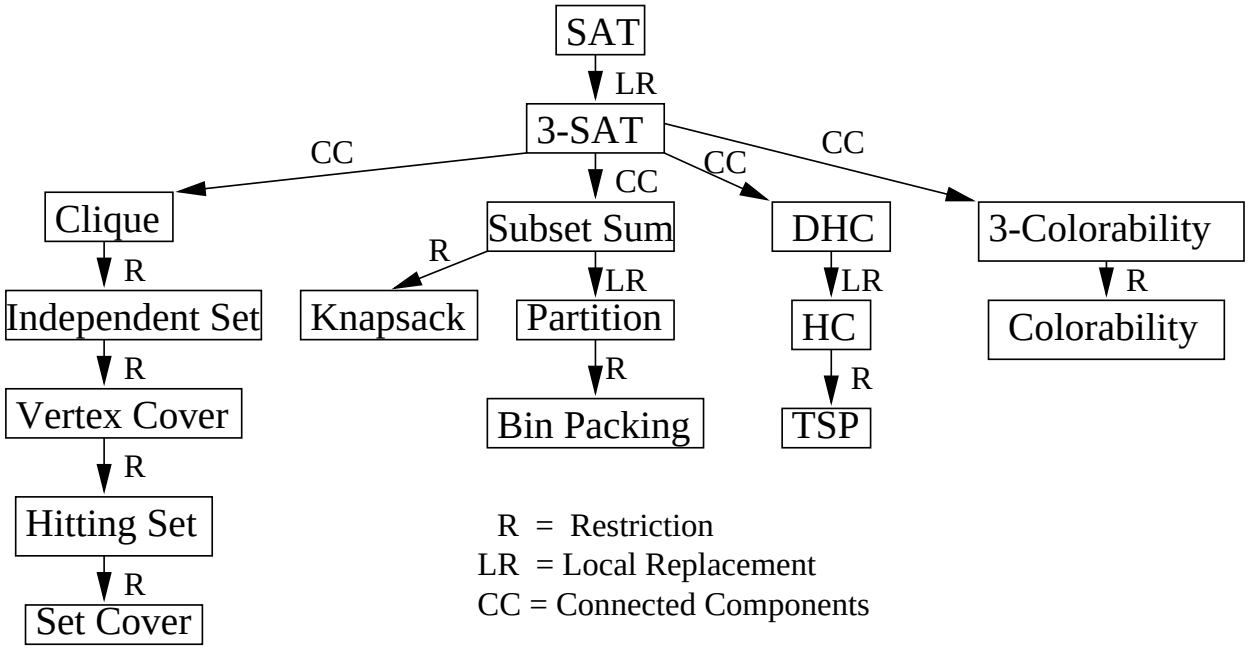



Abbildung 2: Stammbaum NP-vollständiger Probleme: Kantenmarkierungen „R,LR,CC“ stehen für die Reduktionstechniken „Restriction, Local Replacement, Connected Components“.

## 6.3 Entscheiden, Konstruieren und Optimieren

Komplexitätsklassen enthalten formale Sprachen über einem Alphabet. *Formale Sprachen* entsprechen *Entscheidungsproblemen* (also Problemen, die nur die Antworten JA oder NEIN zulassen):

- Zur Sprache $L \in \Sigma^*$ können wir das Problem assoziieren, zu einem gegebenen Wort $w \in \Sigma^*$ zu entscheiden, ob $w$ zu $L$ gehört (Mitgliedschaftsproblem der Sprache).

- Umgekehrt können wir zu einem Entscheidungsproblem mit Eingaben aus $\Sigma^*$ die formale Sprache aller Wörter $w \in \Sigma^*$ assoziieren, die mit der Antwort JA beschieden werden.

Praktische Rechenprobleme haben oft aber auch den Charakter von *Konstruktionsproblemen* oder *Optimierungsproblemen*. Zum Beispiel lassen sich die Probleme CLIQUE und IS auf natürliche Weise als Maximierungsprobleme auffassen: finde in einem ungerichteten Graphen eine Clique bzw. eine unabhängige Menge maximaler Größe. Die Probleme VC, HS, SC, BP, TSP und GC lassen sich analog als Minimierungsprobleme auffassen. KP lässt sich als Maximierungsproblem auffassen: packe einen Rucksack von maximalem Gesamtnutzen unter Beachtung der Gewichtsschranke $W$. Dual dazu kann man es aber auch als Minimierungsproblem sehen: packe einen Rucksack von minimalem Gesamtgewicht unter Beachtung der Nutzenschranke $P$. Die Probleme SAT (und Subset Sum, PARTITION, DHC bzw. HC) lassen sich auch als Konstruktionsprobleme auffassen: finde eine erfüllende Belegung (eine geeignete Auswahl von Zahlen, einen gerichten bzw. ungerichteten Hamiltonschen Kreis).

Auch jedes Optimierungsproblem kann zu einem Konstruktionsproblem gemacht werden: finde eine Lösung, die billiger ist als eine vorgegebene Kostenschranke (bzw. wertvoller ist als eine vorgegebene Nutzenschranke). Eine andere Modifikation eines Optimierungsproblems besteht darin, statt nach einer optimalen Lösung lediglich nach dem optimalen Kosten- oder Nutzenwert zu fragen. In diesem Abschnitt wollen wir zeigen, dass die Frage der polynomiellen Lösbarkeit aller dieser Problemtypen weitgehend an Hand von geeignet definierten formalen Sprachen diskutiert werden kann. Salopp gesprochen gilt für sequentielle Rechner [16]:

ENTSCHEIDEN = KONSTRUIEREN.

Wegen dieser Gleichheit kann man sich (von einem theoretischen Standpunkt aus gesehen) zumeist auf Entscheidungsprobleme bzw. formale Sprachen (also auf das Studium von Komplexitätsklassen) zurück ziehen.

Unsere Vorgehensweise ist wie folgt. Zunächst erinnern wir an die Diskussion des *Problems des Handelsreisenden* („Travelling Salesman Problem“ oder kurz „TSP“), dessen *NP*-Vollständigkeit aud der Vorlesung *Theoretische Informatik* bekannt sein sollte. Alle wesentlichen Ideen tauchen bereits in diesem Beispiel auf. Insbesondere werden wir Problemreduktionen kennen lernen, die nicht vom Typ „$\leq_{pol}$“ oder „$\leq_{log}$“ sind. Danach bringen wir die im TSP-Beispiel genannten Problemtypen in eine allgemeinere begrifflich gestraffte Form und führen zwei neue Reduktionstypen (die „Turing-“ und die „Levin-Reduktion“) formal ein.

### 6.3.1 Das Problem des Handelsreisenden

TSP liest sich als Optimierungsproblem wie folgt. Gegeben sei eine $(n \times n)$-Distanzmatrix $D = (d_{i,j})_{0 \leq i,j \leq n-1}$, wobei $d_{i,j}$ die Distanz von einer Stadt $i$ zu einer Stadt $j$ angibt. Gesucht ist die kürzeste Rundreise durch alle $n$ Städte, also eine Permutation $\sigma$ von $0, \ldots, n-1$, die die Kostenfunktion

$$|\sigma| := \sum_{i=0}^{n-1} d_{\sigma(i),\sigma(i+1 \bmod n)}$$

minimiert. $\sigma_*$ bezeichne im Folgenden eine Permutation, die eine kürzeste Rundreise repräsentiert.
Eine Abschwächung des TSP-Optimierungsproblems ist das TSP-Wertoptimierungsproblem, bei welchem nur nach der Länge einer kürzesten Rundreise gefragt ist.

TSP als Konstruktionsproblem hat als weiteren Eingabeparameter neben der Distanzmatrix $D$ eine Kostenschranke $K$. Gesucht ist eine Rundreise mit Maximalkosten $K$ (also eine Permutation $\sigma$ mit $|\sigma| \leq K$) bzw. die Meldung, dass eine solche Rundreise nicht existiert.
Eine Abschwächung des TSP-Konstruktionsproblems ist das TSP-Entscheidungsproblem, bei welchem nur gefragt ist, ob eine Rundreise mit Maximalkosten $K$ existiert. Rundreisen (Permutationen), die die Kostenschranke $K$ respektieren, nennen wir im Folgenden *zulässig*.

Wir wollen deutlich machen, dass alle vier Problemvarianten polynomiell verknüpft sind, d.h., ein deterministischer Polynomialzeitalgorithmus für eine Variante kann in einen eben solchen Algorithmus für alle anderen Varianten transformiert werden.

[16] nicht jedoch für Parallelrechner (worauf wir im Rahmen dieser Vorlesung allerdings nicht weiter eingehen können)

Die in Abbildung 3 dargestellte Hierarchie ist offensichtlich. Hierbei bedeutet $A \to B$, dass eine polynomiell zeitbeschränkte DTM $M'$, die Problem (oder Problemvariante) $B$ löst, in eine polynomiell zeitbeschränkte DTM $M$ für Problem (oder Problemvariante) $A$ transformiert werden kann. Auf eine griffige Formel gebracht: Entscheiden ist nicht schwerer

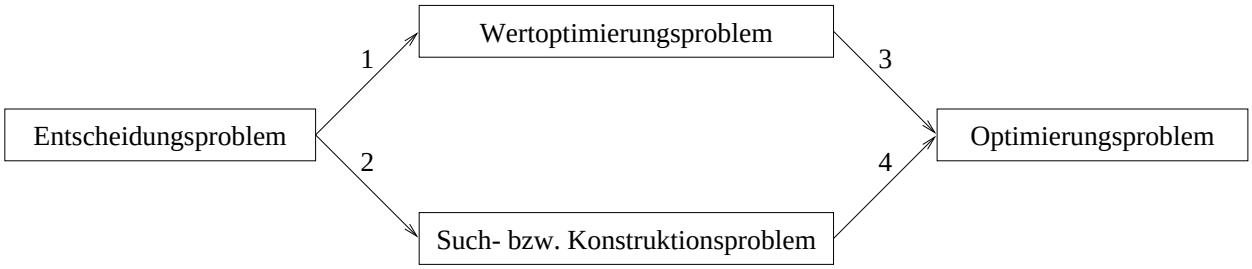



Abbildung 3: Reduktion des Entscheidungsproblems auf das Optimierungsproblem.

als Konstruieren, und Konstruieren ist nicht schwerer als Optimieren.

Überraschender ist, dass die Problemreduktionen in der obigen Hierarchie auch in der umgekehrten Richtung möglich sind, wie es in Abbildung 4 zu sehen ist. Griffig formuliert:

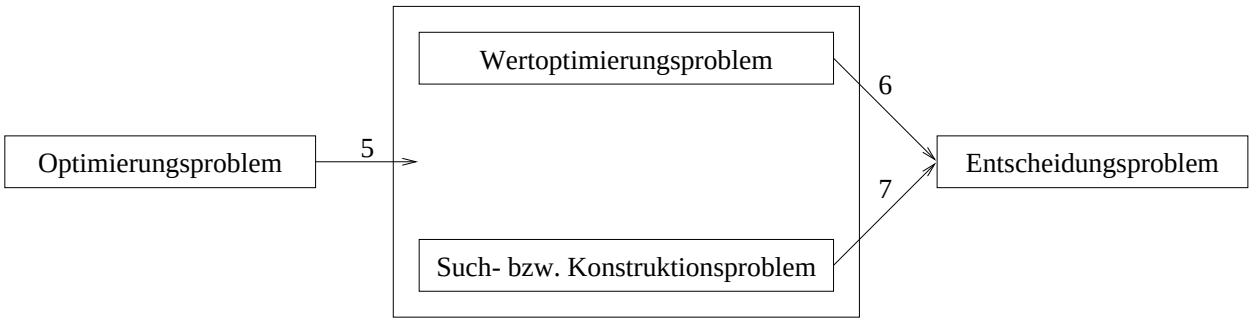



Abbildung 4: Reduktion des Optimierungproblems auf das Entscheidungsproblem.

Optimieren ist nicht schwerer als Konstruieren, und Konstruieren ist nicht schwerer als Entscheiden.

**Reduktion 5** Gegeben eine Prozedur zur Lösung des TSP-Wertoptimierungsproblems sowie eine Prozedur zur Lösung des TSP-Konstruktionsproblems. Das TSP-Optimierungsproblem kann dann wie folgt gelöst werden:

1. Bestimme die Länge $K_*$ einer kürzesten Rundreise.
2. Bestimme eine zulässige Rundreise $\sigma_*$ zu Distanzmatrix $D$ und Kostenschranke $K_*$.

**Reduktion 6** Gegeben eine Prozedur zur Lösung des TSP-Entscheidungsproblems, so erhalten wir die Länge $K_*$ einer kürzesten Rundreise mit Hilfe von Binärsuche (mehr Details dazu in den Übungen).

**Reduktion 7** Die Reduktion des Konstruktionsproblems auf das Entscheidungsproblem ist die interessanteste von allen. Sie macht sich die Eigenschaft der „Selbstreduzierbarkeit“ von TSP zunutze. Selbstreduzierbarkeit ist ein wichtiges Konzept der Komplexitätstheorie, das später noch formal sauber besprochen werden wird. Nehmen wir im Falle von TSP oBdA an, dass eine zulässige Rundreise mit Maximalkosten $K$ existiert. (Andernfalls gibt es nichts zu konstruieren.) Die entscheidende Idee ist, bei jeder einzelnen „Kante“ $(i, j)$ von Stadt $i$ nach Stadt $j$ zu testen, ob sie für eine zulässige Rundreise entbehrlich ist. Dazu werden ihre Kosten $d_{i,j}$ versuchsweise auf $K + 1$ gesetzt. Die Entscheidungsprozedur verrät uns dann, ob trotzdem noch eine zulässige Rundreise (logischerweise dann ohne Kante $(i, j)$) existiert. Falls ja, dann lassen wir $d_{i,j}$ auf seinem hohen Wert. Dies ist nicht weiter schlimm, da wir uns zuvor von der Entbehrlichkeit der Kante $(i, j)$ überzeugt haben. Falls aber die Entscheidungsprozedur signalisiert, dass keine zulässige Lösung mehr existiert, dann setzen wir $d_{i,j}$ auf seinen alten Wert zurück und nehmen Kante $(i, j)$ in die die zu konstruierende Rundreise $\sigma$ auf. Da wir nur unter Zugzwang Kanten in $\sigma$ aufnehmen, entsteht auf diese Weise eine zulässige Rundreise.[17]

### 6.3.2 Allgemeine Suchprobleme

Im vorigen Abschnitt haben wir das TSP-Entscheidungsproblem, das TSP-Konstruktionsproblem und das TSP-(Wert-)Optimierungsproblem diskutiert. In diesem Abschnitt werden wir den abstrakten Begriff des Suchproblems so allgemein fassen, dass alle Problemvarianten sich als Suchproblem darstellen lassen. Dies hat den Vorteil, dass Definitionen zu Suchproblemen den Fall von Entscheidungsproblemen (formalen Sprachen), sowie Konstruktions- und (Wert-)Optimierungsproblemen mit abdecken.

Es folgt die allgemeine Definition des Suchproblems:

**Definition 6.11** *Ein* Suchproblem *ist gegeben durch eine Relation $R \subseteq \Sigma^* \times \Sigma^*$. Zu einer Eingabe $x \in \Sigma^*$ ist eine $R$-*zulässige Lösung[18] *$y \in \Sigma^*$ gesucht, d.h., ein $y$ mit $(x, y) \in R$. Eine DTM löst das Suchproblem, wenn sie für jede Eingabe $x \in \Sigma^*$ ein zulässiges $y$ ausgibt (falls möglich). Im folgenden bezeichne $\mathcal{S}(R)$ das zu Relation $R$ assoziierte Suchproblem.*

Konstruktionsprobleme lassen sich in einer offensichtlichen Weise als Suchprobleme auffassen. Eine ähnliche Bemerkung gilt aber auch für Entscheidungsprobleme (formale Sprachen) und (Wert-)Optimierungsprobleme, wie die folgenden beiden Beispiele zeigen.

**Beispiel 6.12** *Eine formale Sprache $L \subseteq \Sigma^*$ kann als Suchproblem zur Relation*

$$R(L) = \{(x, \epsilon) |\ x \in L\} \tag{11}$$

*aufgefasst werden.*

[17] Die Ausfüllung der Details in dieser Beweisskizze erfolgt in den Übungen. Der formale Beweis benutzt das folgende Konzept: eine Kantenmenge heißt partielle Lösung, wenn sie durch Hinzufügen weiterer Kanten zu einer zulässigen Rundreise ergänzt werden kann. Der Schlüssel zum formalen Beweis liegt in der Einsicht, dass bei der oben beschriebenen Prozedur $\sigma$ zu jedem Zeitpunkt eine partielle Lösung ist.

[18] Wenn $R$ aus dem Kontext hervor geht, sprechen wir auch einfach von einer zulässigen Lösung

**Beispiel 6.13** *Das durch eine Relation $R$ und eine Bewertungsfunktion val gegebene Minimierungsproblem — zu gegebener Eingabe $x$ finde eine $R$-zulässige Lösung $y$ mit minimalem Wert $val(x, y)$ — kann als Suchproblem zur Relation*

$$R_{val} = \{(x, y_*) \in R |\ \forall y :\ (x, y) \in R \implies val(x, y) \geq val(x, y_*)\} \tag{12}$$

*aufgefasst werden. Eine analoge Bemerkung gilt für Maximierungsprobleme und für Wertoptimierungsprobleme.*

Ein Suchproblem könnte aus einem sehr banalen Grund schwer sein, zum Beispiel weil die zulässigen $y$ für eine gegebene Eingabe $x$ extrem lange Wörter sind. Evtl. braucht man dann superpolynomielle Zeit zum Lösen des Suchproblems, bloß weil das Aufschreiben der Ausgabe eine zeitraubende Angelegenheit ist. Die folgende Definition schließt diesen und weitere triviale Gründe für die Härte eines Suchproblems aus.

**Definition 6.14** *Eine Relation $R$ heißt* polynomiell beschränkt*, wenn ein Polynom $p$ existiert so dass für alle $(x, y) \in R$: $|y| \leq p(|x|)$.*
*$R$ heißt* polynomiell entscheidbar*, wenn eine polynomiell zeitbeschränkte DTM entscheiden kann, ob zwei vorgegebene Wörter $x, y$ in Relation $R$ zueinander stehen.*
*$R$ heißt* polynomiell verifizierbar*, wenn $R$ polynomiell beschränkt und polynomiell entscheidbar ist.*
*Die Wertefunktion $val(x, y)$ heißt* polynomiell auswertbar*, wenn zu gegebenem $(x, y) \in R$ die Binärdarstellung von $val(x, y)$ durch eine polynomiell zeitbeschränkte DTM berechenbar ist.*

**Beispiel 6.15** *Die Relation $R_{TSP}$, die dem TSP-Konstruktionsproblem entspricht, enthält genau die Paare $(x, y)$ mit $x = (D, K)$, $y = \sigma$ und Permutation $\sigma$ repräsentiert eine Rundreise mit Maximalkosten $K$ bezüglich der Distanzmatrix $D$.*[19] *$R_{TSP}$ ist offensichtlich polynomiell verifizierbar. Beim TSP-Optimierungsproblem verwenden wir die Länge einer Rundreise als Wertefunktion. Diese ist offensichtlich polynomiell auswertbar.*

Eine besondere Rolle spielen in der Komplexitätstheorie die sogenannten $NP$-Relationen:

**Definition 6.16** *Zu einer Relation $R \subseteq \Sigma^* \times \Sigma^*$ assoziieren wir die formale Sprache*

$$L(R) := \{x |\ \exists y :\ (x, y) \in R\}. \tag{13}$$

*$R$ heißt* $\mathcal{K}$-Relation *(für eine gegebene Komplexitätsklasse $\mathcal{K}$), falls $L(R) \in \mathcal{K}$. Insbesondere nennen wir $R$ eine* $NP$-Relation*, falls $L(R) \in NP$.*

Das folgende Resultat basiert wieder auf dem Rate-Verifikationsprinzip:

**Lemma 6.17** *$R$ ist eine $NP$-Relation genau dann, wenn $R$ polynomiell verifizierbar ist, d.h.,*

$$NP = \{L(R) |\ R \text{ ist polynomiell verifizierbar}\}. \tag{14}$$

Den offensichtlichen Beweis lassen wir aus.

[19] Streng genommen ist $x$ ein Wort, das auf eine „natürliche Weise“ $D$ und $K$ kodiert, und $y$ ein Wort, das „auf natürliche Weise“ $\sigma$ kodiert. Wir gehen davon aus, dass Sie sich vorstellen können, wie man mathematische Objekte als Wörter über einem Alphabet kodiert und ziehen die informellere (aber weniger gestelzte) Formulierung vor.

### 6.3.3 Abstrakte polynomielle Reduktionen zwischen Relationen

Die Definitionen im Zusammenhang mit abstrakten polynomiellen Reduktionen und die sich daraus ergebenden Lemmata lassen sich mühelos von formalen Sprachen auf Relationen (und damit auf Suchprobleme) erweitern. Der Vollständigkeit halber listen wir die wesentlichen Definitionen und Tatsachen hier kurz auf:

**Definition 6.18** *Eine Beziehung „$\leq_{POL}$“ zwischen Relationen heißt* abstrakte polynomielle Reduktion*, wenn sie transitiv ist und wenn $R \leq_{POL} R'$ und die Lösbarkeit von $\mathcal{S}_{R'}$ in Polynomialzeit impliziert, dass auch $\mathcal{S}_R$ in Polynomialzeit lösbar ist.*

**Definition 6.19** *Sei $\leq_{POL}$ eine abstrakte polynomielle Reduktion zwischen Relationen und $R' \subseteq \Sigma^* \times \Sigma^*$.*

1. *$R'$ heißt NP-*hart *unter Reduktionen vom Typ „$\leq_{POL}$“, wenn $R \leq_{POL} R'$ für alle NP-Relationen $R$.*
2. *Falls zusätzlich $R'$ selber eine NP-Relation ist, dann heißt $R'$ NP-*vollständig *unter Reduktionen vom Typ „$\leq_{POL}$“.*

**Lemma 6.20** *Sei „$\leq_{POL}$“ eine abstrakte polynomielle Reduktion zwischen Relationen. Aus der Existenz einer (unter Reduktionen vom Typ „$\leq_{POL}$“) NP-harten Relation $R'$ mit polynomiell lösbarem Suchproblem $\mathcal{S}_{R'}$ würde dann folgen, dass $\mathcal{S}_R$ für jede NP-Relation $R$ in Polynomialzeit lösbar ist.*

**Lemma 6.21** *Sei „$\leq_{POL}$“ eine abstrakte polynomielle Reduktion zwischen Relationen. Falls $R' \leq_{POL} R''$ und $R'$ ist NP-hart unter Reduktionen vom Typ „$\leq_{POL}$“, dann ist auch $R''$ NP-hart unter Reduktionen vom Typ „$\leq_{POL}$“.*

### 6.3.4 Turing- und Levin-Reduktionen

Die Problemreduktionen, die wir im Zusammenhang mit TSP skizziert haben, verlaufen zwischen allgemeinen Suchproblemen (und sind nicht vom Typ „$\leq_{pol}$“). Um den hierfür geeigneten Reduktionsbegriff einzuführen, benötigen wir zunächst die Definition der Orakel Turing Maschine.

**Definition 6.22** *Eine TM $M$ mit einer Relation $R$ als* Orakel*, genannt* Orakel Turing Maschine (OTM) *und notiert als $M^{[R]}$, verfügt über ein zusätzliches* Orakelband *und drei ausgezeichnete Zustände $q_?, q_+, q_-$. Auf das Orakelband kann $M$ ein beliebiges Wort $x \in \Sigma^*$ schreiben und dann den* Fragezustand *$q_?$ annehmen. Das Orakel ersetzt dann Wort $x$ durch ein Wort $y$ mit $(x, y) \in R$ (falls möglich) und $M$ wechselt in den Zustand $q_+$. Falls kein zulässiges $y$ existiert, wird $x$ nicht ersetzt und $M$ wechselt in den Zustand $q_-$. Ansonsten ändert sich die Konfiguration der Maschine bei diesem Übergang nicht. Wenn die Maschine ihre Frage auf dem Orakelband spezifiziert hat, kostet sie das Befragen des Orakels und das Erhalten der Antwort nur zwei Schritte.*[20]

[20] Das Orakel spendiert die Antwort sozusagen kostenlos wie eine gute Fee.

*Falls eine Relation der Form $R(L)$ für eine formale Sprache $L$ als Orakel verwendet wird, sprechen wir der Einfachheit halber von einem Orakel für $L$, das wir als $M^{[L]}$ notieren. Deterministische (bzw. nichtdeterministische) Orakel Turing Maschinen bezeichnen wir kurz als DOTMs (bzw. NOTMs).*

Mit Hilfe der Orakel-Maschinen definieren wir Turing-Reduktionen wie folgt:

**Definition 6.23** *$R$ heißt* Turing-reduzierbar *auf $R'$ , in Zeichen $R \leq_T R'$, falls eine polynomiell zeitbeschränkte DTM $M$ existiert, so dass die DOTM $M^{[R']}$ das Suchproblem $\mathcal{S}_R$ löst. Falls $R = R(L)$, dann schreiben wir der Einfachheit halber $L \leq_T R'$ statt $R(L) \leq_T R'$. Eine analoge Bemerkung gilt im Falle $R' = R(L')$.*

Offensichtlich ist eine polynomielle Reduktion $L \leq_{pol} L'$ eine sehr spezielle Turing-Reduktion, bei der zunächst $f(x)$ aus $x$ berechnet wird, um dann das Orakel für $L'$ nach $f(x)$ zu befragen. Insbesondere wird auch nur eine einzige Frage an das Orakel gestellt. Wie schon die Reduktion „$\leq_{pol}$“ gilt auch für „$\leq_T$“:

**Lemma 6.24** *Die Relation „$\leq_T$“ ist ist eine abstrakte polynomielle Reduktion zwischen Relationen.*

Mit Hilfe der Turing-Reduktionen können wir den Begriff der „Selbstreduzierbarkeit“ präzise definieren.

**Definition 6.25** *$R$ heißt* selbstreduzierbar, *falls $R \leq_T L(R)$.*

In den Übungen werden wir uns (auf elementare Weise) von der Selbstreduzierbarkeit einiger konkreter Relationen überzeugen. In der Vorlesung werden wir später nachweisen, dass **alle** $NPC$-Relationen selbstreduzierbar sind.

Alle am TSP-Beispiel vorgeführten Reduktionen waren Turing-Reduktionen. Man überlegt sich leicht, dass diese Reduktionen verallgemeinert werden können auf beliebige selbstreduzierbare Relationen mit polynomiell auswertbaren Bewertungsfunktionen. Da alle $NPC$-Relationen selbstreduzierbar sind, ergibt sich für eine breite Klasse von Problemen, dass Entscheiden, Konstruieren (Suchen) und Optimieren ungefähr gleich schwer sind. Dadurch erhalten wir die Berechtigung, uns beim Studium der Problemkomplexität auf Entscheidungsprobleme (formale Sprachen) zurückzuziehen.

Richard Karp (Universität Berkeley, Erfinder der polynomiellen Reduktion „$\leq_{pol}$“) und Stephen Cook (Universität Toronto, Erfinder der Turing-Reduktion „$\leq_T$“) und waren zwei der westlichen Protagonisten der $NP$-Vollständigkeitstheorie in den siebziger Jahren des vorigen Jahrhunderts. Zur gleichen Zeit wurde die Theorie im Osten von Levin vorangetrieben. Wir beschließen diesen Abschnitt mit dem von Levin vorgeschlagenen Reduktionstyp.

**Definition 6.26** *$R$ heißt* Levin-reduzierbar *auf $R'$ , in Zeichen $R \leq_L R'$, wenn drei in Polynomialzeit berechenbare Abbildungen $f, g, h : \Sigma^* \to \Sigma^*$ existieren, so dass für alle $x, y, z \in \Sigma^*$ folgendes gilt:*

$$x \in L(R) \quad \Leftrightarrow \quad f(x) \in L(R') \tag{15}$$

$$(x, y) \in R \quad \Leftrightarrow \quad (f(x), g(x, y)) \in R' \tag{16}$$

$$(f(x), y') \in R' \quad \Leftrightarrow \quad (x, h(x, y')) \in R \tag{17}$$

Wir können $f$ als eine *Eingabetransformation* auffassen, die gemäß (15) eine polynomielle Reduktion von $L(R)$ auf $L(R')$ repräsentiert. Abbildungen $g$ und $h$ sind *Lösungstransformationen*: zu gegebener Eingabe $x$ transformiert $g$ gemäß (16) eine $R$-zulässige Lösung für $x$ in eine $R'$-zulässige Lösung für $f(x)$; umgekehrt transformiert $h$ gemäß (17) eine $R'$-zulässige Lösung für $f(x)$ in eine $R$-zulässige Lösung für $x$.

Eine Levin-Reduktion von $R$ auf $R'$ impliziert nicht nur eine polynomielle Reduktion von $L(R)$ auf $L(R')$, sondern auch eine Turing-Reduktion von $R$ auf $R'$:

**Lemma 6.27** $R \leq_L R' \implies R \leq_T R'$.

**Beweis** Folgende DOTM mit einem Orakel für $R'$ löst das Suchproblem $\mathcal{S}_R$ in Polynomialzeit:

1. Transformiere $x$ in $f(x)$.
2. Befrage das Orakel für $R'$ nach einer $R'$-zulässigen Lösung $y'$ für $f(x)$. (Falls diese nicht existiert, so existiert wegen (15) auch keine $R$-zulässige Lösung für $x$.)
3. Transformiere $(x, y')$ in $h(x, y')$ und gib $h(x, y')$ aus. (Wegen (17) ist $h(x, y')$ eine $R$-zulässige Lösung für $x$.

□

Das folgende Resultat liegt auf der Hand:

**Lemma 6.28** *Die Relation „$\leq_L$“ ist eine abstrakte polynomielle Reduktion zwischen Relationen.*

### 6.3.5 Eine Verschärfung des Cook'schen Theorems

Das Erfüllbarkeitsproblem der Booleschen Logik, auch kurz SATISFIABILITY oder SAT genannt, war historisch gesehen das erste „natürliche" Problem, von welchem (durch Stephen Cook im Jahre 1971) die *NP*-Vollständigkeit gezeigt wurde. Es wurde dadurch zur Wurzel des Stammbaums der unter polynomiellen Reduktionen *NP*-vollständigen Probleme. Wir erinnern zunächst an den historischen Satz von Cook. Hernach beobachten wir, dass sein Beweis implizit ein etwas stärkeres Resultat enthält, nämlich die *NP*-Vollständigkeit der Relation $R_{SAT}$ unter Levin-Reduktionen. Diese (auf den ersten Blick womöglich unscheinbare) Erweiterung des Satzes von Cook wird uns im Abschnitt 6.3.6 ermöglichen zu beweisen, dass jede *NPC*-Relation selbstreduzierbar ist.

**Theorem 6.29 (Satz von Cook)** *SAT ist NP-vollständig.*

**Beweis** Da der Beweis dieses Satzes aus der Vorlesung *Theoretische Informatik* bekannt ist, rufen wir an dieser Stelle lediglich seine grobe Struktur in Erinnerung:

(1) Wir haben uns bereits in Abschnitt 6.1 davon überzeugt, dass SAT zur Klasse *NP* gehört. Somit genügt es die *NP*-Härte von SAT nachzuweisen, also zu zeigen, dass $L \leq_{pol} \text{SAT}$ für alle $L \in NP$.

(2) Ein nicht-deterministischer und polynomiell zeitbeschränkter Akzeptor $M$ einer (beliebigen aber festen) Sprache $L \in NP$ lässt sich so normalisieren, dass für geeignete Konstanten $c_1, c_2 \in \mathbb{N}$ und $R(n) = n^{c_1}$, $T(n) = n^{c_2}$ gilt:

**Raten** Bei Eingaben der Länge $n$ rät $M$ zunächst einen Binärstring $r \in \{0,1\}^{R(n)}$ und schreibt diesen in die Zellen $-1, \ldots, -R(n)$.

**Verifizieren** Auf Zelle 0 wird Trennsymbol \$ geschrieben. Zellen $1, \ldots, n$ enthalten weiterhin die Eingabe $w = w_1 \ldots w_n$. Gestartet auf Inschrift $r\$w$ (im Startzustand $q_0$ und Kopfposition 0) rechnet $M$ deterministisch höchstens $T(n)$ Schritte.

Wir erinnern daran, dass genau die Eingaben $w$ zu $L$ gehören, zu denen ein Ratestring (Zertifikat) $r$ existiert, so dass $M$ in der Verifikationsphase $r\$w$ akzeptiert.

(3) Aus $w$ kann in Polynomialzeit eine Boolesche Formel $F = F_w$ in konjunktiver Normalform berechnet werden, so dass gilt:

$$w \in L \Leftrightarrow \exists r \in \{0,1\}^{R(n)} : M \text{ akzeptiert } r\$w \text{ in der Verifikationsphase} \Leftrightarrow F_w \text{ erfüllbar} \ .$$

**Einschub:** Die Feststellungen in (1),(2),(3) implizieren bereits den Satz von Cook. Die im Folgenden fest gehaltenen Eigenschaften des ausführlichen Beweises aus der Vorlesung *Theoretische Informatik* sind wichtig, um die Verschärfung des Cook'schen Theorems zu erhalten.

(4) Die Klauseln von $F_w$ beschreiben im Wesentlichen die deterministische Rechnung von $M$ auf $r\$w$ in der Verifikationsphase. Zu diesem Zweck verwendet $F_w$ die folgenden Booleschen Variablen:

| Variable $X$ | Interpretation von $X = 1$ |
|---|---|
| $Z(i,z)$ | Zustand $z$ zum Zeitpunkt $i$ |
| $H(i,j)$ | Kopfposition $j$ zum Zeitpunkt $i$ |
| $S(i,j,c)$ | Bandsymbol $c$ in Zelle $j$ zum Zeitpunkt $i$ |

Hierbei ist $i = 0, \ldots, T(n)$ der Laufindex für die Zeit, $j = -T(n), \ldots, 0, \ldots, T(n)$ der Laufindex für die Zellnummern, $z \in Z_M$ eine Zustandsvariable und $c \in \Gamma_M$ die Variable für Symbole aus dem Bandalphabet.

(5) Die Klauseln in $F_w$ sind so konstruiert, dass sie höchstens dann erfüllt werden können, wenn die Belegung der Variablen folgenden Bedingungen genügt:

1. Für jedes $j \in \{1, \ldots, R(n)\}$ existiert genau ein $r_j \in \{0,1\}$ mit $S(0,-j,r_j) = 1$. Für alle $c \in \Gamma_M \setminus \{r_j\}$ muss $S(0,-j,c) = 0$ gelten. Somit gibt es eine „natürliche Bijektion“ zwischen den Ratestrings $r \in \{0,1\}^{R(n)}$ und den sinnvollen Belegungen der Variablen $S(0,-1,c), \ldots, S(0,-R(n),c)$.

2. Gegeben die zu $r \in \{0,1\}^{R(n)}$ korrespondierende Belegung von $S(0,-1,c), \ldots, S(0,-R(n),c)$, so ist erzwungen, dass die übrigen Variablen gemäß der oben angegebenen Interpretation belegt werden und somit die Rechung von $M$ auf $r\$w$ in der Verifikationsphase beschreiben.

**Bezeichnung:** von $r$ induzierte Variablenbelegung.
**Conclusio:** $F_w$ kann höchstens von Variablenbelegungen erfüllt werden, die von einem $r \in \{0,1\}^{R(n)}$ induziert werden.

(6) Eine von $r$ induzierte Variablenbelegung erfüllt $F_w$ genau dann, wenn $M$ auf $r\$w$ in der Verifikationsphase eine akzeptierende Rechnung vollzieht. □

Die Relation $R_{SAT}$ enthält alle Paare der Form $(F, a)$, so dass folgende Bedingungen gelten:

- $F$ repräsentiert eine Kollektion von Booleschen Klauseln, sagen wir über $n$ Booleschen Variablen.
- $a \in \{0,1\}^n$ ist eine Boolesche Belegung dieser Variablen.
- Belegung $a$ erfüllt alle in $F$ enthaltenen Klauseln.

Offensichtlich gilt $\mathrm{SAT} = L(R_{SAT})$.

**Folgerung 6.30** *$R_{SAT}$ ist eine unter Levin-Reduktionen NP-vollständige Relation.*

**Beweis** Sei $R$ eine *NP*-Relation. Somit gilt $L = L(R) \in NP$. Der Beweis des Satzes von Cook liefert eine in Polynomialzeit berechenbare Eingabetransformation $w \mapsto F_w$, welche implizit eine polynomielle Reduktion von $L = L(R)$ auf $\mathrm{SAT} = L(R_{SAT})$ enthält. Zum Nachweis von Folgerung 6.30 fehlt also nur die Angabe zweier geeigneter Lösungstransformationen:

1. Es muss in Polynomialzeit möglich sein, ein Paar $(w,r) \in R$ in eine $F_w$ erfüllende Variablenbelegung zu transformieren.
   Im Beweis des Satzes von Cook wurde gezeigt, dass die von $r$ induzierte Variablenbelegung die Formel $F_w$ erfüllt. Diese Belegung kann in der Tat aus $(w,r)$ in Polynomialzeit berechnet werden, indem wir die deterministische Rechnung von $M$ auf $r\$w$ in der Verifikationsphase laufen lassen und nebenbei die induzierte Variablenbelegung ausgeben.

2. Es muss in Polynomialzeit möglich sein, ein Paar $(F_w, a)$ mit einer $F_w$ erfüllenden Booleschen Belegung $a$ in ein Zertifikat $r$ mit $(w, r) \in R$ zu transformieren.
   Im Beweis des Satzes von Cook wurde demonstriert, dass eine $F_w$ erfüllende Variablenbelegung von einem Ratestring $r \in \{0,1\}^{R(n)}$ induziert sein muss. Zudem muss $r$ sogar ein Zertifikat für die Aussage $w \in L(R)$ sein, woraus sich $(w, r) \in R$ ergibt. Weiterhin lässt sich $r$ leicht aus der Belegung der Variablen $S(0,-1,c), \ldots, S(0,-R(n),c)$ ermitteln.

Damit hat sich eine Levin-Reduktion von $R$ auf $R_{SAT}$ ergeben. □

### 6.3.6 Selbstreduzierbarkeit aller NPC-Relationen

Das Hauptresultat dieses Abschnittes ist die folgende Aussage über *NPC*-Relationen:

**Theorem 6.31** *Jede NPC-Relation ist selbstreduzierbar.*

**Beweis** Wir beweisen zunächst die Selbstreduzierbarkeit von $R_{SAT}$:

**Behauptung** $R_{SAT} \leq_T$ SAT.

Sei $F = F(v_1, \ldots, v_n)$ eine Konjunktion von Klauseln über den Booleschen Variablen $v_1, \ldots, v_n$. Mit Hilfe eines SAT-Orakels können wir das Suchproblem zu $R_{SAT}$ lösen wie folgt:

**Vorabtest** Wir fragen das Orakel, ob $F$ erfüllbar ist. Falls nicht, können wir mit der Meldung, dass keine erfüllende Belegung existiert, abbrechen. Falls doch, dann weiter wie folgt.

**Konstruktion einer erfüllenden Belegung** Für $i \geq 0$ und eine partielle Belegung $(a_1, \ldots, a_i) \in \{0,1\}^i$ bezeichne $F_i = F(a_1, \ldots, a_i, v_{i+1}, \ldots, v_n)$ die vereinfachte Klauselkonjunktion, die entsteht, wenn wir für $j = 1, \ldots, i$ die Variable $v_j$ durch die Boolesche Konstante $a_j$ ersetzen. [21] Für $i = 0$ ist noch keine Variable belegt, d.h., $F_0 \equiv F$. Für $i = n$ sind alle Variablen belegt, d.h., $F_n \in \{\text{TRUE}, FALSE\}$. Falls $F_n \equiv$ TRUE, dann handelt es sich bei $a$ um eine erfüllende Belegung. Eine partielle Belegung heisse *gut*, wenn sie zu einer erfüllenden Belegung fortgesetzt werden kann. Da der Vorabtest garantiert, dass $F \equiv F_0$ erfüllbar ist, ist die leere Belegung gut. Unsere Strategie besteht darin, eine gute partielle Belegung von $a_1, \ldots, a_i$, $0 \leq i \leq n-1$, mit Hilfe des SAT-Orakels zu einer guten partiellen Belegung von $a_1, \ldots, a_i, a_{i+1}$ zu erweitern. Falls dies gelingt, können wir iterativ die gute leere Belegung zu einer guten vollständigen (und somit $F$ erfüllenden) Belegung fortsetzen. Nehmen wir also an, dass $a_1, \ldots, a_i$ eine gute partielle Belegung ist. Dann ist $F_i$ erfüllbar. Wir fragen das SAT-Orakel, ob $F(a_1, \ldots, a_i, 0, v_{i+2}, \ldots, v_n)$ erfüllbar ist. Falls ja, dann setzen wir $a_{i+1} = 0$. Falls nein, dann muss erzwungenermaßen $F(a_1, \ldots, a_i, 1, v_{i+2}, \ldots, v_n)$ erfüllbar sein, und wir setzen $a_{i+1} = 1$. Auf diese Weise gelangen wir nach $n$ Iterationen zu einer $F$ erfüllenden Belegung $a = (a_1, \ldots, a_n)$.

[21] Vereinfachungen: durch die partielle Belegung bereits erfüllte Klauseln können eliminiert bzw. durch TRUE ersetzt werden; in den verbleibenden Klauseln können alle Literale $X_j, \bar{X}_j$ für $j = 1, \ldots, i$ entfernt bzw. durch FALSE ersetzt werden.

Um Theorem 6.31 zu beweisen haben wir zu zeigen:

**Behauptung** Sei $R$ eine $NPC$-Relation. Dann gilt $R \leq_T L(R)$.

Sei $x$ eine Eingabe für das Suchproblem zu $R$. Eine Anfrage an das $L(R)$-Orakel klärt wieder vorab, ob überhaupt eine $R$-zulässige Lösung für $x$ existiert. OBdA sei dies der Fall.

Wir stehen nun vor dem Problem, mit Hilfe des $L(R)$-Orakels eine $R$-zulässige Lösung $y$ mit $(x, y) \in R$ zu konstruieren. Dazu gehen wir in zwei Phasen vor. Wir argumentieren zunächst, dass die Konstruktion eines geeigneten $y$ mit Hilfe des SAT-Orakels anstelle des $L(R)$-Orakels leicht durchzuführen ist. Danach zeigen wir, dass ein $L(R)$-Orakel es auf einfache Weise erlaubt, ein SAT-Orakel zu simulieren.

Für **Phase 1** bedienen wir uns der polynomiellen Levin-Reduktion von $R$ auf $R_{SAT}$, die gemäß Folgerung 6.30 existieren muss. Sei $f$ die zur Reduktion gehörende Eingabetransformation und $h$ die zweite Lösungstransformation. (Vgl. Definition 6.26.) Wir gehen dann vor wie folgt:

- Transformiere $x$ vermöge $f$ in eine Konjunktion $F_x$ Boolescher Klauseln. Da $x$ eine zulässige Lösung besitzt, muss $F_x$ erfüllbar sein.
- Nutze die Selbstreduzierbarkeit von $R_{SAT}$ aus, um mit Hilfe eines SAT-Orakels eine $F_x$ erfüllende Belegung $a$ zu konstruieren.
- Transformiere $(F_x, a)$ vermöge $h$ in eine $R$-zulässige Lösung $y$ für $x$.

Dieses Verfahren ist offensichtlich polynomiell zeitbeschränkt. Einziger Schönheitsfehler: wir haben real kein SAT- sondern nur ein $L(R)$-Orakel.

In **Phase 2** demonstrieren wir, dass man mit Hilfe eines $L(R)$-Orakels ein virtuelles SAT-Orakel bereitstellen kann. Dazu bedienen wir uns der polynomiellen polynomielle Reduktion von SAT nach $L(R)$, die wegen $L(R) \in NPC$ existieren muss. Sei $f'$ die dazu gehörende Eingabetransformation. Eine Anfrage an das SAT-Orakel nach der Erfüllbarkeit von $F$ beantworten wir wie folgt:

- Transformiere $F$ vermöge $f'$ in eine Eingabe $x'$ für das Mitgliedschaftsproblem zu $L(R)$.
- Frage das $L(R)$-Orakel, ob $x \in L(R)$ und gib die Antwort aus.

Da die polynomielle Reduktion garantiert, dass $F$ genau dann erfüllbar ist, wenn $x' \in L(R)$, ist die von uns ausgegebene Antwort korrekt. □

Wie aus der Vorlesung *Theoretische Informatik* bekannt sein sollte, ist die Klasse $NPC$ sehr reichhaltig. Die Eigenschaft der Selbstreduzierbarkeit gilt daher für eine umfangreiche Klasse von Relationen. Für alle diese Relationen ist also das Konstruieren einer zulässigen Lösung nicht wesentlich aufwendiger als das Entscheiden, ob eine solche Lösung existiert.

Nichtsdestotrotz gibt es einige natürliche Relationen, deren Suchproblem vermutlich nicht in Polynomialzeit lösbar ist und die vermutlich dennoch keine $NPC$-Relationen sind:

**COMPOSITES** COMPOSITES liegt vermutlich in $NPC \setminus P$:

Das zugehörige Suchproblem ist das *Faktorisierungsproblem*: zu gegebener Zahl $N$ finde (falls möglich) eine Faktorisierung $N = N_1 \cdot N_2$ mit $N_1, N_2 \geq 2$. Wäre COMPOSITES

*NP*-vollständig, dann wäre gemäß Satz 6.31 die zugehörige Relation selbstreduzierbar und somit würde ein Primzahlorakel erlauben, das Faktorisierungsproblem zu lösen. Da kürzlich von Agrawal, Kayal und Saxena PRIMES $\in P$ bewiesen wurde, wäre damit das Faktorisierungsproblem in Polynomialzeit lösbar. Damit würden alle auf diesem Problem basierenden Kryptosysteme (wie zum Beispiel das populäre RSA-Cryptosystem) zusammenbrechen. Beim gegenwärtigen Stand der Forschung müssen die genannten Konsequenzen als unwahrscheinlich[22] gelten. Von daher ist zu vermuten, dass COMPOSITES in $NPC \setminus P$ liegt.

**GRAPHENISOMORPHIE** Das *Graphenisomorphieproblem* ist die Frage, ob zwei vorgegebene Graphen $G = (V, E), G' = (V', E')$ mit $|V| = |V'|$ *isomorph* sind, d.h., sind beide Graphen bis auf Umbenennung der Knoten gleich? Das zugehörige Suchproblem verlangt nach Angabe einer *Isomorphie*, also einer bijektiven Abbildung $h : V \to V'$, so dass $E' = \{(h(v), h(w))|\ (v, w) \in E\}$. Wir werden zu einem späteren Zeitpunkt der Vorlesung (nach dem Abschnitt über die polynomielle Hierarchie von Stockmeyer) deutlich machen, warum vermutlich GRAPHENISOMORPHIE in $NPC \setminus P$ liegt.

Im Unterschied zu COMPOSITES kann man bei GRAPHENISOMORPHIE auf einfache Weise eine Turing-Reduktion vom Suchproblem auf das Entscheidungsproblem angeben. Die zentralen Ideen zum Nachweis der Selbstreduzierbarkeit sind wie folgt:

- Eine partielle Lösung des Grpahenisomorphieproblems ist eine injektive Abbildung $h$ einer Teilmenge $U$ von $V$ nach $V'$, die zu einer Isomorphieabbildung fortsetzbar ist. Nehmen wir oBdA an, dass $G$ und $G'$ isomorph sind. Dann ist die *leere Abbildung* mit Definitionsbereich $U = \emptyset$ eine partielle Lösung.

- Die Idee ist, den Definitionsbereich einer partiellen Lösung mit Hilfe des GRAPHENISOMORPHIE-Orakels iterativ zu erweitern. Wenn wir testen könnten, ob eine Erweiterung der Form $h(w) = z$ mit $w \in V \setminus U$ immer noch eine partielle Lösung ist, dann könnten wir durch Ausprobieren aller Kandidaten $z$ schließlich eine Fortsetzung der partiellen Lösung ausfindig machen. Schönheitsfehler: Wir verfügen nicht über einen solchen „Fortsetzungstest“ sondern nur über ein GRAPHENISOMORPHIE-Orakel.

- Es bleibt zu zeigen, dass man mit einem GRAPHENISOMORPHIE-Orakel einen virtuellen Fortsetzungstest bereitstellen kann. Sei $h$ bisher auf $U = \{u_1, \ldots, u_i\}$ definiert und $u'_j = h(u_j)$ für $j = 1, \ldots, i$. Wir wollen testen, ob $h(u_{i+1}) = u'_{i+1}$ eine geeignete Fortsetzung ist. Zu diesem Zweck erzeugen wir zwei Hilfsgraphen $H$ und $H'$. $H$ geht aus $G$ hervor, indem wir für $j = 1, \ldots, i+1$ dem Knoten $u_j$ $jn$-viele neue Knoten als Nachbarn geben. Analog geht $H'$ geht aus $G'$ hervor, indem wir für $j = 1, \ldots, i+1$ dem Knoten $u'_j$ $jn$-viele neue Knoten als Nachbarn geben. (Mehr oder weniger) offensichtlich (s. Übung) ist $h$ mit erweitertem Definitionsbereich $\{u_1, \ldots, u_i, u_{i+1}\}$ genau dann eine partielle Lösung, wenn $H$ und $H'$ isomorph sind. Der Fortsetzungstest wird also implementiert, indem wir das GRAPHENISOMORPHIE-Orakel nach der Isomorphie von $H$ und $H'$ befragen.

[22] eine psychologische (keine mathematische) Aussage

Wir fassen die letzte Diskussion zusammen:

**Theorem 6.32** *Die Relation zum Graphenisomorphieproblem ist selbstreduzierbar.*

## 6.4 Die Grenze zwischen P und NPC

Eine alte Binsenweisheit besagt, dass Glück und Leid dicht beieinander liegen. Wenn wir unter Leid die *NP*-Härte eines zu lösenden Berechnungsproblems und unter Glück seine Lösbarkeit in Polynomialzeit verstehen, dann trifft die obige Feststellung auch auf den Algorithmenentwurf zu. Der ausgeteilte Handzettel 2 (entnommen aus *Garey & Johnson*) illustriert, wie dicht Probleme aus $P$ und $NPC$ beieinander liegen können:

- Der **kürzeste** Pfad zwischen zwei Knoten in einem Graphen mit positiven Kantengewichten kann (zum Beispiel mit dem Algorithmus von Dijkstra) effizient gefunden werden. Der **längste** Pfad hingegen vermutlich nicht. Das korrespondierende Entscheidungsproblem, LONGEST PATH BETWEEN TWO VERTICES, ist nämlich (sogar eingeschränkt auf Einheitsgewichte für Kanten) *NP*-vollständig.[23]

- VERTEX COVER ist, wie wir bereits wissen, *NP*-vollständig. Es gibt also vermutlich keinen effizienten Algorithmus, der zu einem gegebenen Graphen $G = (V, E)$ die kleinste Knotenmenge $U \subseteq V$ findet, die von jeder Kante mindestens einen Randknoten enthält. Das duale EDGE COVER Problem, das nach einer kleinsten Kantenmenge fragt, die jeden Knoten überdeckt, ist hingegen mit Hilfe von MAXIMUM MATCHING effizient lösbar.

- Das Problem MINIUMUM EQUIVALENT DIGRAPH fragt nach einer kleinsten Teilmenge $A' \subseteq A$ von Kanten eines gegebenen Digraphen $G = (V, A)$, so dass zwei beliebige $u, v \in V$ genau dann in $G$ durch einen Pfad (von $u$ nach $v$) verbunden werden können, wenn dies in $G' = (V, A')$ möglich ist. TRANSITIVE REDUCTION unterscheidet sich von MINIUMUM EQUIVALENT DIGRAPH nur dadurch, dass $A' \subseteq V \times V$, d.h., die $A'$ muss keine Teilmenge von $A$ sein. TRANSITIVE REDUCTION kann (mit Hilfe von TRANSITIVE CLOSURE und Berechnung von starken Zusammenhangskomponenten) effizient gelöst werden. Das Entscheidungsproblem zu MINIMUM EQUIVALENT DIGRAPH hingegen enthält DIRECTED HAMILTONIAN CIRCUIT[24] als Teilproblem und ist daher *NP*-vollständig.

- Eine Eingabeinstanz zum Problem SCHEDULING WITH INDIVIDUAL DEADLINES besteht aus einer Menge $T$ von Aufgaben (tasks), einer partiellen Ordnung $\prec$ auf $T$, einem individuellen Schlusstermin (deadline) $d(t)$ für jede Aufgabe $t \in T$ und einer Anzahl $m$ von Prozessoren. Jede Aufgabe $t$ kann in einer Zeiteinheit auf einem Prozessor ausgeführt werden. $t \prec t'$ bedeutet, dass $t$ ausgeführt sein muss, bevor die

[23] Dies kann (relativ leicht) mit einer Kette von Karp-Reduktionen nachgewiesen werden, die von HAMILTONIAN CIRCUIT über HAMILTONIAN PATH (auf die offensichtiche Weise definiert) nach LONGEST PATH BETWEEN TWO VERTICES führt.

[24] genauer: die Einschränkung von DIRECTED HAMILTONIAN CIRCUIT auf stark zusammenhängende Digraphen, die auch *NP*-vollständig ist (gleiche Reduktion wie in der Vorlesung)

Ausführung von $t'$ begonnen wird. Die Frage ist, ob man die Aufgaben aus $T$ den $m$ Prozessoren so zuordnen kann, dass alle Schlusstermine eingehalten werden. Eine partielle Ordnung auf $T$ heisst INTREE oder auch *Montagebaum*, wenn jedes $t \in T$ maximal einen unmittelbaren Nachfolger hat. Analog spricht man von einem OUTTREE oder auch *Sortierbaum*, wenn jedes $t \in T$ maximal einen unmittelbaren Vorgänger hat. INTREE SCHEDULING (bzw. OUTTREE SCHEDULING) ist die Spezialisierung von SCHEDULING WITH INDIVIDUAL DEADLINES auf partielle Ordnungen der Form INTREE (bzw. OUTTREE). Wie Brucker, Garey und Johnson 1977 gezeigt haben, kann INTREE SCHEDULING effizient gelöst werden, wohingegen OUTTREE Scheduling *NP*-vollständig ist (Reduktion von VERTEX COVER).

Es gehört zur Grundbildung der Informatik, das Grenzgebiet zwischen $P$ und $NPC$ gut zu kennen. Die obere Abbildung auf dem ausgeteilten Handzettel Nummer 3 illustiert den Grenzverlauf. Die untere Grenzlinie von $NPC$ wird von den am weitesten eingeschränkten *NP*-vollständigen Teilproblemen gebildet. Jede weitere Spezialisierung eines dieser Probleme führt aus $NPC$ heraus. Die obere Grenzlinie von $P$ wird von den allgemeinsten Teilproblemen gebildet, die noch in Polynomialzeit gelöst werden können. Jede weitere Generalisierung eines dieser Probleme führt aus $P$ heraus. Je dichter diese beiden Grenzlinien aneinanderliegen, desto kleiner ist das unbekannte Terrain der Probleme mit einem offenen Status. Die voranschreitende Forschung schiebt die Grenzlinien tendenziell aufeinander zu, da sowohl der Fundus der *NP*-vollständigen Probleme als auch der Fundus der effizienten Algorithmen ständig erweitert wird. Anschaulich gesprochen gibt es *kleine Engelchen*, die das bekannte Territorium von $P$ auf immer allgemeinere Berechnungsprobleme ausdehnen, und *kleine Teufelchen*, die von immer spezielleren Problemen die *NP*-Vollständigkeit nachweisen. Es ist aus der strukturellen Komplexitätstheorie bekannt, dass (unter der Vorausetzung $P \neq NP$) die Klasse $NPC \setminus P$ nicht leer ist. So sehr also Engelchen und Teufelchen sich mühen, sie werden sich niemals auf einer scharfen Grenzlinie begegnen.

Wenden wir doch dieses allgemeine Schema einmal auf eine konkrete Problemhierachie an. PRECEDENCE CONSTRAINED SCHEDULING ist der Spezialfall von SCHEDULING WITH INDIVIDUAL DEADLINES bei dem die individuellen Schlusstermine alle identisch sind zu einem gemeinsamen Schlusstermin $D$. Mit einer von CLIQUE ausgehenden Karp-Reduktion kann man die *NP*-Vollständigkeit von PRECEDENCE CONSTRAINED SCHEDULING nachweisen. Aus der unteren Abbildung auf Handzettel 3 geht hervor, dass die Spezialfälle, bei denen die partielle Ordnung auf $T$ leer oder ein Baum (Montage- oder Sortierbaum)[25] ist, in Polynomialzeit gelöst werden kann. Ebenso für allgemeine partielle Ordnung auf $T$ und eine feste Anzahl 1, 2 oder 3 von Prozessoren (fleissige Engelchen). Für eine feste Anzahl von 4 oder mehr Prozessoren ist der Status des Problems offen.

[25] Für Montagebäume folgt dies aus dem oben zitierten allgemeineren Resultat für INTREE SCHEDULING (individuelle Schlusstermine). Für Sortierbäume kann man im Falle eines einheitlichen Schlusstermines $D$ den Algorithmus für Montagebäume zusammen mit einem Symmetrieargument verwenden: Erstens, kehre im Sortierbaum SB die Orientierung aller Kanten um und erhalte einen Montagebaum MB. Zweitens, wende das Verfahren für Montagebäume an und erhalte einen optimalen Plan, der alle Aufgaben auf $m$ Prozessoren in $T \leq D$ Zeiteinheiten $1, \ldots, T$ unter Einhaltung der (spiegelverkehrten) partiellen Ordnung $MB$ ausführt. Drittens, spiegele den Plan, d.h., durchlaufe ihn in der zeitlichen Reihenfolge $T, \ldots, 1$. Dann wird die partielle Ordnung SB respektiert.

Die Exploration des Grenzgebietes zwischen *NPC* und *P* geschieht in der Praxis meist für eine konkrete Problemhierarchie (wie zum Beispiel PRECEDENCE CONSTRAINED SCHEDULING).

Bei Graphenproblemen ergibt sich eine Problemhierarchie auf natürliche Weise, indem man von der Klasse aller (endlicher) Graphen zu speziellen Graphklassen übergeht. Zum Beispiel:

- Bäume
- planare Graphen
- Graphen mit beschränktem Knotengrad

Eine weitere Spezialisierung ergibt sich, indem man Variablen der Eingabe zu Konstanten "gefriert". Wir werden in Abschnitt 6.5 exemplarisch die Problemhierarchie zum Graphenfärbungsproblem diskutieren.

Für ein Zahlenproblem erhalten wir ein natürliches Teilproblem, indem wir verlangen, dass der Maximalbetrag eines Zahlparameters der Eingabe polynomiell durch die Eingabelänge beschränkt ist. Dies führt zu der Frage, ob ein Zahlenproblem nur darum nicht zu *P* gehört, weil man mit wenigen Bits "riesige Zahlen" spezifizieren kann, oder ob es auch schon für "moderate Zahlen" *NP*-vollständig ist. Probleme der ersten Kategorie heißen *pseudopolynomiell lösbar*, Probleme der letzteren Kategorie heißen *stark NP-vollständig*. Wir werden später sehen, dass zum Beispiel PARTITION pseudopolynomiell lösbar, aber eine Verallgemeinerung davon (3-PARTITION) stark *NP*-vollständig ist.

## 6.5 Analyse von eingeschränkten Graphproblemen

Eine *k-Färbung* eines Graphen $G = (V, E)$ ist eine Färbung der Knoten, die monochromatische Kanten vermeidet, d.h., eine Abbildung $f : V \to \{1, \ldots, k\}$ mit $f(v) \neq f(w)$ für alle $\{v, w\} \in E$. Zu gegebener Färbung heisst die Menge aller Knoten der gleichen Farbe $i$ die *Farbklasse* zu $i$. Die *chromatische Zahl* $\chi(G)$ ist die kleinste Anzahl $k$ von Farben, die eine legale Färbung von $G$ ermöglicht. Das *Graphenfärbungsproblem* (als Optimierungsproblem) besteht darin, zu einem gegebenen Graphen $G$ eine $\chi(G)$-Färbung anzugeben. COLORABILITY sei das Problem zu gegebenem Graphen $G$ und gegebener Kostenschranke $k$ zu entscheiden, ob $G$ $k$-färbbar ist. $k$-COLORABILITY sei das Teilproblem, bei dem (die Konstante) $k$ nicht Teil der Eingabe ist (Gefrieren einer Variable zu einer Konstanten).

Ein natürliches Anwendungsszenario für Graphenfärbung ist die Durchführung von Prozessen mit gemeinsamen Ressourcen. Zwei Prozesse stehen im Konflikt zueinander, wenn sie die gleiche Ressource benötigen. Zwei solche Prozesse können nicht im gleichen Zeitabschnitt ausgeführt werden. Prozesse und ihre Ressourcenkonflikte können durch einen Konfliktgraphen modelliert werden, dessen Knoten Prozesse und dessen Kanten Ressourcenkonflikte repräsentieren. Die chromatische Zahl des Konfliktgraphen gibt die minimale Anzahl von Zeitabschnitten an, die eine konfliktfreie Ausführung zulässt.

Es ist bekannt, dass 2-COLORABILTY zu *P* gehört. Wir skizzieren kurz die Grundidee. Eine Teilmenge $U$ der Knotenmenge $V$ eines Graphen $G = (V, E)$ heisst *unabhängig (Independent Set)*, wenn die Knoten aus $U$ paarweise in $G$ nicht benachbart sind. Man beachte,

dass jede Farbklasse eine unabhängige Menge ist. $G$ heisst *bipartit*, wenn $V$ sich in zwei unabhängige Mengen zerlegen lässt. Das folgende Resultat ist nicht schwer zu zeigen:

**Lemma 6.33** *Die folgenden Aussagen sind äquivalent:*

1. *$G$ ist 2-färbbar.*
2. *$G$ ist bipartit.*
3. *$G$ enthält keine Kreise ungerader Länge.*

Da man im Rahmen einer systematischen Graphexploration (wie DFS oder BFS) die Existenz von Kreisen ungerader Länge leicht testen kann, erhalten wir die

**Folgerung 6.34** *2-COLORABILITY$\in P$.*

Es bezeichne $\Delta(G)$ den maximalen Grad eines Knotens in $G$. Ein naives Verfahren zum Färben von $G$ liefert der folgende gierige Algorithmus:
Sei $\Delta := \Delta(G)$. Färbe einen Knoten nach dem anderen mit Farben aus $C = \{1, \ldots, \Delta + 1\}$ gemäß folgender Strategie: der aktuelle Knoten $v$ erhält die kleinste Farbe $i$ aus $C$, die noch nicht für einen seiner Nachbarn verwendet wurde.
Dieses naive Verfahren ist effizient durchführbar und kommt mit $1+\Delta(G)$ Farben aus. Somit gilt $\chi(G) \leq \Delta(G) + 1$. Für die $(\Delta + 1)$-Clique ist diese Schranke scharf. Darüberhinaus gilt:

**Lemma 6.35 (Satz von Brooks)** — *ohne Beweis* —
*Es sei $G$ ein zusammenhängender Graph und $\Delta = \Delta(G)$. Dann ist $\chi(G) \leq \Delta$ sofern $G$ nicht eine Clique der Größe $\Delta + 1$ bildet.*

**Folgerung 6.36** *COLORABILITY eingeschränkt auf Graphen vom Maximalgrad 3 gehört zu $P$.*

**Beweis** Wir zeigen, dass $\chi(G)$ effizient berechenbar ist. Da man die Zusammenhangskomponenten eines Graphen unabhängig voneinander färben kann, dürfen wir oBdA annehmen, dass der gegebene Graph zusammenhängend ist. Falls $G$ nicht aus einem einzigen Knoten besteht, benötigen wir mindestens zwei Farben. Wie oben erwähnt können wir in Polynomialzeit testen, ob $G$ 2-färbbar ist. Nehmen wir zu unseren Ungunsten an dass wir mindestens 3 Farben brauchen. Da $\Delta(G) \leq 3$, brauchen wir maximal 4 Farben. Falls $G$ eine 4-Clique ist, benötigen wir genau 4 Farben. Andernfalls garantiert der Satz von Brooks, dass drei Farben ausreichen. □

Insgesamt hat sich also gezeigt, dass COLORABILTY eingeschränkt auf zwei Farben oder auf maximalen Knotengrad 3 zu $P$ gehört. Soweit die Nachrichten aus den himmlischen Sphären. Hören wir uns an, was die bocksbeinigen Kerlchen aus der unteren Etage zu dem Thema zu sagen haben.

**Theorem 6.37** *3-COLORABILITY ist NP-vollständig. Dies gilt sogar dann, wenn wir die zulässigen Eingaben auf planare Graphen vom maximalen Knotengrad 4 einschränken.*

**Beweis** Mitgliedschaft zu *NP* ist offensichtlich. Wir haben die *NP*-Härte nachzuweisen. Der Beweis vollzieht sich in drei Stufen.

**Stufe 1** 3-SAT$\leq_{pol}$ 3-COLORABILTY.

Die Beweisskizze hierzu (die auch aus der Vorlesung *Theoretische Informatik* bekannt sein sollte) ist auf dem ausgeteilten Handzettel 4 zu sehen. Das Herzstück der Reduktion ist die Klauselkomponente. Sie ist so kunstvoll entworfen, dass legale 3-Färbungen der Klauselkomponente implizit die korrespondierende Klausel "korrekt auswerten". Zu weiteren Details s. den Handzettel. Ergebnis von Stufe 1: 3-COLORABILITY ist *NP*-hart.

**Stufe 2** Entwurf des "Crossing-Over-Gadget"

Wir erinnern daran, dass ein Graph planar ist, wenn man die Knoten und Kanten so in die Ebene einbetten kann, dass Kanten sich nur an gemeinsamen Randpunkten berühren. Die in Stufe 1 benutzte Reduktion erzeugt nicht notwendig einen planaren Graphen. Wir können ihn aber in einen planaren Graphen transformieren, indem wir bei sich kreuzenden Kanten $\{x, x'\}$ und $\{y, y'\}$ ein "Crossing-Over-Gadget" (s. Handzettel 5) einbauen. Die Kreuzung wird dabei durch einen kreuzungsfreien Untergraphen ersetzt. Das Gadget ist so kunstvoll konstruiert, dass es beim Färben des Ursprungsgraphen keine neuen Freiheitsgrade einbaut und keine alten Freiheitsgrade vernichtet. Zu weiteren Details s. den Handzettel. Ergebnis von Stufe 2: 3-COLORABILITY eingeschränkt auf planare Graphen ist *NP*-hart.

**Stufe 3** Entwurf des "Portal-Gadgets"

Die in Stufen 1 und 2 skizzierte Reduktion erzeugt einen Graphen mit evtl. hohem maximalen Knotengrad. Wir setzen die Reduktion so fort, dass sie den maximalen Knotengrad auf 4 reduziert. Dies kann mit Hilfe einer lokalen Ersetzung geschehen (s. Handzettel 6). Dabei wird ein Knoten $v$ mit $k$ Nachbarn durch ein Portal-Gadget $H_k$ ersetzt. $H_k$ ist ein Untergraph mit inneren Knoten (die nicht durch Kanten mit Knoten ausserhalb $H_k$ verbunden sind) und $k$ Portalen. Jedes der Portale ist über eine Kante mit genau einem Nachbarn von $v$ verbunden. Die Portale haben innerhalb $H_k$ den Grad 3, also insgesamt den Grad 4. Die inneren Knoten von $H_k$ haben Grad 4. $H_k$ ist so kunstvoll konstruiert, dass die Portale alle die gleiche Farbe haben können und müssen. Man hat also beim Färben des neuen Graphen die gleichen Freiheitsgrade wie zuvor. Zu weiteren Details s. den Handzettel.

Stufe 3 schließt den Beweis des Theorems ab. □

Die Grenze zwischen *P* und *NP* hat sich beim Graphenfärbungsproblem somit (einigermaßen) präzise bestimmen lassen.

## 6.6 Starke NP-Vollständigkeit

### 6.6.1 Beispiele für pseudopolynomielle Algorithmen

Anna und Bert lassen sich durch die neuesten Nachrichten zur NP-Vollständigkeit nicht entmutigen. Anna hat eine Idee, um das PARTITION-Problem zu lösen. Seien $w_1, \ldots, w_n \geq 0$ die gegebenen Zahlen und $S = w_1 + \cdots + w_n$ ihre Summe. Nehmen wir oBdA an, dass $S$ eine gerade Zahl ist. Anna will eine $n \times (1 + S/2)$-Tabelle $T = (t_{r,s})_{1 \leq r \leq n, 0 \leq s \leq S/2}$ mit den Booleschen Einträgen

$$t_{r,s} = \left( \exists I \subseteq \{1, \ldots, r\} : \sum_{i \in I} w_i = s \right) \tag{18}$$

berechnen. In Worten: die Indikatorvariable $t_{r,s}$ hat genau dann den Wahrheitswert 1, wenn sich aus den Zahlen $w_1, \ldots, w_r$ eine Auswahl mit Summenwert $s$ treffen lässt. Die erste Zeile der Tabelle ist leicht auszurechnen:

$$t_{1,s} = ((s = 0) \vee (w_1 = s)) \,. \tag{19}$$

Gegeben Zeile $r-1$, dann lässt sich auch Zeile $r$ mit Hilfe einer einfachen Vorschrift ausfüllen:

$$t_{r,s} = (t_{r-1,s} \vee ((s \geq w_r) \wedge t_{r-1,s-w_r})) \,.$$

Denn Summenwert $s$ lässt sich genau dann mit einer Auswahl aus $w_1, \ldots, w_r$ erhalten, wenn er sich durch eine solche Auswahl **ohne** Teilnahme von $w_r$ oder durch eine solche Auswahl **mit** Teilnahme von $w_r$ erhalten lässt. Der erste Fall liegt genau dann vor, wenn man Summenwert $S$ bereits durch eine Auswahl der Zahlen $w_1, \ldots, w_{r-1}$ erhalten kann. Der zweite Fall liegt genau dann vor, wenn $s \geq w_r$ und man Summenwert $s - w_r$ durch eine Auswahl der Zahlen $w_1, \ldots, w_{r-1}$ erhalten kann.
Anna kann die Rechenvorschriften (18) und (19) benutzen, um die Tabelle $T$ zeilenweise auszufüllen. Offensichtlich gehört die Eingabe $w_1, \ldots, w_n$ genau dann zur Sprache PARTITION, wenn $t_{n,S/2}$ den Wahrheitswert 1 hat. Der Eintrag in der rechten unteren Ecke von Tabelle $T$ verrät uns also die Lösung.
*Das ist nicht übel*, sagt Bert. *Ich habe übrigens eine ähnliche Methode zum Lösen von KNAPSACK (KP). Meine Tabelle ist auch 2-dimensional und hat Einträge $t_{r,s} \in \mathbb{N}_0$, die den maximalen Nutzen angeben, der sich mit den Objekten $1, \ldots, r$ unter Beachtung von Gewichtsschranke $s$ erzielen lässt. Ich kann diese Tabelle mit einer ähnlich einfachen Rechenvorschrift systematisch ausfüllen wie Du Deine. Schon gut*, sagt Anna. *Ich kann mir vorstellen wie Du das machst.*[26] *Wir haben also beide ein NP-vollständiges Problem gelöst.*

Um die vorangehenden Überlegungen zu klaren Aussagen zusammenzufassen, benötigen wir die folgende

**Definition 6.38** *Bei einem Zahlenproblem (formale Sprache) $L$ assoziieren wir zu einer Eingabeinstanz $I$ neben der Eingabelänge $n(I)$ die maximale Größe $M(I)$ einer der in $I$ gegebenen Zahlparameter. Zu einem festen Polynom $p$ bezeichne $L_p$ die Einschränkung von*

[26] die Hörer und Hörerinnen der Vorlesung hoffentlich ebenfalls (s. Übung)

*$L$ auf Eingaben $I$ mit $M(I) \leq p(n(I))$. $L$ heißt* pseudopolynomiell entscheidbar*, wenn $L_p \in P$ für jedes Polynom*[27] *$p$. Ein* pseudopolynomieller *Algorithmus für $L$ ist eine DTM, die das Mitgliedschaftsproblem für $L$ löst und deren Laufzeit polynomiell in $n(I)$ und $M(I)$ beschränkt ist.*

Aus der Existenz eines pseudopolynomiellen Algorithmus für $L$ folgt offensichtlich, dass $L$ pseudopolynomiell entscheidbar ist. Anna's und Bert's Verfahren ergeben also folgendes

**Theorem 6.39** *PARTITION und KP sind mit pseudopolynomiellen Algorithmen lösbar.*

Da SUBSET SUM ein Teilproblem von KP ist, überträgt sich dieses Resultat auf SUBSET SUM.

### 6.6.2 Erste Beispiele für stark NP-vollständige Probleme

Kann man vielleicht alle Zahlenprobleme pseudopolynomiell lösen? Zum Beispiel auch BP? Oder gibt es Zahlenprobleme, bei denen alle pseudopolynomiellen Lösungsversuche an unüberwindliche Barrieren stoßen?

Die folgende Definition liefert ein wichtiges Werkzeug zur Beantwortung dieser Frage:

**Definition 6.40** *Ein Zahlenproblem (formale Sprache) $L$ heißt* stark *NP*-hart*, wenn es ein Polynom $p$ gibt, so dass $L_p$ NP-hart ist. Gilt zusätzlich $L \in NP$, dann heißt $L$* stark *NP*-vollständig.

Wir merken an, dass rein kombinatorische Probleme (wie zum Beispiel SAT, 3-SAT, CLIQUE, DHC, HC) ohne (echte) Zahlparameter automatisch stark *NP*-vollständig sind, wenn sie *NP*-vollständig sind. Der neue Begriff macht nur Sinn bei echten Zahlenproblemen, deren Eingaben potenziell superpolynomiell in $n(I)$ große Zahlen enthalten können. Das folgende Resultat ist offensichtlich:

**Theorem 6.41** *Falls $P \neq NP$, dann kann es zu einem stark NP-vollständigen Problem keinen pseudopolynomiellen Algorithmus geben.*

Ein erstes stark *NP*-vollständiges Zahlenproblem kennen wir schon:

**Theorem 6.42** *TSP ist stark NP-vollständig.*

**Beweis** Die Reduktion von HC auf TSP, die wir zum Nachweis der *NP*-Vollständigkeit von TSP verwendet haben, benutzt in der Distanzmatrix nur Zahlenparameter mit den Werten 1 oder 2. Die Kostenschranke hatte den Wert $n$ (Anzahl der Städte). Es treten also keine Zahlparameter auf, die superpolynomiell in der Länge der Eingabe für TSP sind. □

Mit TSP kennen wir ein stark *NP*-vollständiges Anordnungsproblem (mit Zahlparametern). Wir begeben uns nun auf die Jagd nach einem stark *NP*-vollständigen Zerlegungsproblem (mit Zahlparametern). Objekt unserer Begierde sind die folgenden Probleme:

[27] wie üblich mit Koeffizienten aus $\mathbb{N}$

**3-DM** Perfektes 3-Dimensional Matching

**Eingabe** Drei disjunkte gleichmächtige Mengen $X = \{x_1, \ldots, x_q\}$, $Y = \{y_1, \ldots, y_q\}$, $Z = \{z_1, \ldots, z_q\}$ sowie eine Menge $M \subseteq X \times Y \times Z$ von Tripeln. Hierbei reicht es, $M$ in der Eingabe zu spezifizieren, da $X, Y, Z$ implizit durch die Komponenten der Tripel aus $M$ gegeben sind.

**Frage** Gibt es eine Auswahl $T \subseteq M$ von $q$ Tripeln aus $M$, so dass jedes Element aus $X \cup Y \cup Z$ in genau einem Tripel vorkommt?

**3-PARTITION** Partition in $m$ Tripel mit gleichen Teilsummen

**Eingabe** $n = 3m$ Zahlen $a_1, \ldots, a_n \in \mathbb{N}$ mit einer Gesamtsumme $B = a_1 + \cdots + a_n$ der Form $B = mb$, $b \in \mathbb{N}$. Für $i = 1, \ldots, n$ gelte $b/4 < a_i < b/2$.

**Frage** Kann man diese Zahlen in $m$ gleich große Teilsummen zerlegen, d.h., existiert eine Zerlegung von $\{1, \ldots, n\}$ in Mengen $I_1, \ldots, I_m$ mit

$$\forall j = 1, \ldots, m : \sum_{i \in I_j} a_i = b? \tag{20}$$

Beachte, dass wegen $b/4 < a_i < b/2$ die Bedingung (20) höchstens dann erfüllbar ist, wenn jedes $I_j$ dreielementig ist.

**4-Partition** Partition in $m$ Quadrupel mit gleichen Teilsummen

**Eingabe** $n = 4m$ Zahlen $a_1, \ldots, a_n \in \mathbb{N}$ mit einer Gesamtsumme $B = a_1 + \cdots + a_n$ der Form $B = mb$, $b \in \mathbb{N}$. Für $i = 1, \ldots, n$ gelte $b/5 < a_i < b/3$.

**Frage** Kann man diese Zahlen in $m$ gleich große Teilsummen zerlegen, d.h., existiert eine Zerlegung von $\{1, \ldots, n\}$ in Mengen $I_1, \ldots, I_m$ mit

$$\forall j = 1, \ldots, m : \sum_{i \in I_j} a_i = b? \tag{21}$$

Beachte, dass wegen $b/5 < a_i < b/3$ die Bedingung (21) höchstens dann erfüllbar ist, wenn jedes $I_j$ vierelementig ist.

Wir merken kurz an, dass 2-DM (Perfektes 2-Dimensionales Matching) das zu 3-DM analoge Problem mit Paaren anstelle von Tripeln ist. Bei 2-DM geht es dann um die Frage, ob eine Auswahl $T \subseteq M$ von $q$ Paaren aus $M$ existiert, so dass jedes Element in $X \cup Y$ in genau einem Paar vorkommt. Da man sich die Elemente aus $X$ als *Frauen*, die Elemente aus $Y$ als *Männer*, die Paare aus $M$ als *verträgliche Paarbildungen* und die Auswahl $T \subseteq M$ als ein *perfektes Heiratssystem* vorstellen kann, ist 2-DM auch unter dem Namen *Heiratsproblem* bekannt.[28] Mit Maximum Matching Techniken kann man zeigen (s. Vorlesung *Effiziente Algorithmen*):

[28] 3-DM ist das Heiratsproblem für eine Gesellschaft, in welcher sich neben *männlich* und *weiblich* ein dritter Sextypus etabliert hat. Es wird Traditionalisten zutiefst befriedigen, dass (wie sich im Verlaufe dieses Abschnittes noch zeigen wird) 3-DM *NP*-vollständig ist.

**Lemma 6.43** *2-DM* $\in P$.

Wir beabsichtigen, die folgende Kette

$$\text{3-SAT} \leq_{pol} \text{3-DM} \leq_{pol} \text{4-PARTITION}_p \tag{22}$$

von Reduktionen zu entwerfen, wobei $p$ ein Polynom ist. Damit hätten wir 4-PARTITION als stark *NP*-vollständiges Zahlenproblem etabliert.

**Lemma 6.44** *3-SAT*$\leq_{pol}$*3-DM.*

**Beweis** Sei $C = (C_0, \ldots, C_{m-1})$ mit $C_i = z_{i1} \vee z_{i2} \vee z_{i3}$ und $z_{ij} \in \{x_1, \bar{x}_1, \ldots, x_n, \bar{x}_n\}$ eine Eingabe für 3-SAT. Wir geben eine Eingabetransformation $C \mapsto M$ an, die $C$ eine Tripelmenge $M$ zuordnet. $C$ soll genau dann erfüllbar sein, wenn wir für $M$ ein perfektes 3-dimensionales Matching $T \subseteq M$ finden. Die Tripelmenge $M$ besteht aus der *Belegungskomponente* $M^B$, der *Klauselerfüllungskomponente* $M^K$ und der *Ergänzungskomponente* $M^E$:

**Belegungskomponente** Zu jeder Variable $x_k$, $k = 1, \ldots, n$, assoziieren wir die folgenden Tripel:

$$\begin{aligned} T_k &= \{(\bar{x}_{k,i}, a_{k,i}, b_{k,i}) |\ i = 0, \ldots, m-1\} \\ F_k &= \{(x_{k,i}, a_{k,i+1 \bmod m}, b_{k,i}) |\ i = 0, \ldots, m-1\} \end{aligned}$$

Wir setzen $M_k^B = T_k \cup F_k$ und $M^B = \cup_{k=1}^n M_k^B$. Abbildung 5 illustriert diese Konstruktion. Die Elemente $x_{k,i}, \bar{x}_{k,i}$ nennen wir *Literalelemente*. Wir werden sehen, dass sie auch noch in anderen Komponenten vorkommen. Die Elemente $a_{k,i}$ und $b_{k,i}$ sind *interne Elemente* von $M_k^B$, d.h., sie kommen in keinen Tripeln außerhalb $M_k^B$ vor. Man überlegt sich leicht, dass es zu einer Zerlegung der internen Elemente von $M_k^B$ in Tripel nur zwei Möglichkeiten gibt: **entweder** alle Tripel aus $T_k$ **oder** alle Tripel aus $F_k$. Intuitiv verknüpfen wir mit der Entscheidung für $T_k$ die Vorstellung, $x_k$ mit 1 zu belegen, und mit der Entscheidung für $F_k$ die Vorstellung, $x_k$ mit 0 zu belegen.

**Klauselerfüllungskomponente** Zu jeder Klausel $C_i$, $i = 0, \ldots, m-1$, assoziieren wir die folgenden Tripel:

$$\begin{aligned} S_i^+ &= \{(x_{k,i}, c_i, d_i) |\ 1 \leq k \leq n, x_k \in C_i\} \\ S_i^- &= \{(\bar{x}_{k,i}, c_i, d_i) |\ 1 \leq k \leq n, \bar{x}_k \in C_i\} \end{aligned}$$

Wir setzen $M_i^K = S_i^+ \cup S_i^-$ und $M^K = \cup_{i=0}^{m-1} M_i^K$. Die Elemente $c_i$ und $d_i$ sind *interne Elemente* von $M_i^K$, d.h., sie kommen in keinen Tripeln außerhalb $M_i^K$ vor. Um diese Elemente in die Tripelzerlegung mit einzubeziehen, sind wir gezwungen, ein Tripel aus $M_i^K$ auszuwählen. Intuitiv verbinden wir damit die Auswahl eines $C_i$ erfüllenden Literals.

**Ergänzungskomponente** Die Ergänzungskomponente wird dazu dienen, noch unüberdeckte Literalelemente in die Tripelzerlegung einzubeziehen. Die internen Elemente von $M^B$ und $M^K$ werden durch $nm + m$ Tripel exakt überdeckt werden. Damit sind

auch bereits $nm + m$ Literalelemente überdeckt. Von den insgesamt $2nm$ Literalelementen wären dann noch $(n-1)m$ unüberdeckt. Dies motiviert die folgende Definition der Ergänzungskomponente:

$$M^E = \{(x_{k,i}, e_l, f_l), (\bar{x}_{k,i}, e_l, f_l | \ 1 \leq k \leq n, 0 \leq i \leq m-1, 1 \leq l \leq (n-1)m\} \ .$$

$e_l, f_l$ sind die *internen Elemente* von $M^E$.

Abbildung 5: Ein Ausschnitt aus der Belegungskomponente im Falle $m = 3$: die Tripel aus $T_k$ sind schraffiert, die Tripel aus $F_k$ unschraffiert.

Die Transformation $C \mapsto M$ mit $M = M^B \cup M^K \cup M^E$ ist offensichtlich in Polynomialzeit berechenbar. Der Beweis wird abgeschlossen durch die
**Behauptung** $C$ ist genau dann erfüllbar, wenn für $M$ ein perfektes 3-dimensionales Matching existiert.

Nehmen wir zunächst an, dass eine $C$ erfüllende Belegung gegeben ist. Wir bilden ein perfektes 3-dimensionales Matching $T$ für $M$ nach folgender Strategie:

1. Für $k = 1, \ldots, n$ nimm alle Tripel aus $T_k$ in $T$ auf, falls $x_k = 1$, und nimm alle Tripel aus $F_k$ in $T$ auf, falls $x_k = 0$.
   **Effekt** Alle internen Elemente von $M^B$ sind exakt überdeckt; ebenso alle Literalelemente $x_{k,i}$ mit $x_k = 0$ bzw. alle Literalelemente $\bar{x}_{k,i}$ mit $x_k = 1$. Die anderen Literalelemente (die zu erfüllten Literalen korrespondieren) sind noch unüberdeckt.

2. Fixiere zu jeder Klausel $C_i$, $0 \leq i \leq m-1$, ein erfülltes Literal $z_{i,j}$. Es existiert ein $k$, so dass $z_{i,j} \in \{x_k, \bar{x}_k\}$. Falls $z_{i,j} = x_k$, dann nimm das Tripel $(x_{k,i}, c_i, d_i) \in S_i^+$ in $T$ auf. Falls $z_{i,j} = \bar{x}_k$, dann nimm das Tripel $(\bar{x}_{k,i}, c_i, d_i) \in S_i^-$ in $T$ auf. Beachte, dass in

beiden Fällen ein noch unüberdecktes Literalelement überdeckt wurde.
**Effekt** Alle internen Elemente von $M^K$ sind exakt überdeckt. Insgesamt $nm + m$ Literalelemente sind nun exakt überdeckt. Somit sind $(n-1)m$ Literalelemente noch unüberdeckt.

3. Benutze — in der offensichtlichen Weise — $(n-1)m$ Tripel aus $M^E$, um die noch unüberdeckten Literalelemente sowie die internen Elemente von $M^E$ exakt zu überdecken.

Nehmen wir nun umgekehrt an, dass ein perfektes 3-dimensionales Matching $T$ für $M$ gegeben ist. Wie bereits oben angemerkt, muss $T$ für jedes $k$ entweder alle Tripel aus $T_k$ oder alle Tripel aus $F_k$ enthalten. Im ersten Fall setzen wir $x_k = 1$, im zweiten Fall $x_k = 0$. Wir zeigen, dass die so erhaltene Belegung in jeder Klausel mindestens 1 Literal erfüllt. Die Tripel in $T \cap M^B$ lassen genau die Literalelemente unüberdeckt, die zu erfüllten Literalen korrespondieren. Da die Tripel aus $T \cap M^K$ alle internen Elemente von $M^K$ überdecken, muss es zu jeder Klausel ein erfülltes Literal geben: die erste Komponente des $c_i, d_i$ überdeckenden Tripels ist nämlich ein von $T \cap M^B$ noch unüberdecktes Literalelement, das zu einem $C_i$ erfüllenden Literal korrespondiert. □

**Lemma 6.45** *Es existiert ein Polynom $p$ mit 3-DM$\leq_{pol}$4-PARTITION$_p$.*

**Beweis** Sei $M \subseteq X \times Y \times Z$ mit $X = \{x_1, \ldots, x_q\}$, $Y = \{y_1, \ldots, y_q\}$ und $Z = \{z_1, \ldots, z_q\}$ eine gegebene Tripelmenge. Zu jedem $w \in W = X \cup Y \cup Z$ bezeichne $L(w)$ die Anzahl der Tripel, in denen $w$ vorkommt. Sei weiter $m = |M|$ und $r = 32q$. Wir können oBdA $m \geq q$ voraussetzen, da kein perfektes 3-dimensionale Matching für $M$ mit weniger als $q$ Tripeln existiert. Die zu $M$ korrespondierende Eingabeinstanz für 4-PARTITION bestehe aus den folgenden Zahlen:

**Elementzahlen** Zu jedem $w \in W$ assoziieren wir die *Hauptzahl* $a_1(w)$ und die *Nebenzahlen* $a_l(w)$ gemäß folgender Definition:

$$\begin{aligned}
a_1(x_i) &= 10r^4 + ir + 1,\ 1 \leq i \leq q \\
a_l(x_i) &= 11r^4 + ir + 1,\ 1 \leq i \leq q,\ 2 \leq l \leq L(x_i) \\
a_1(y_j) &= 10r^4 + jr^2 + 2,\ 1 \leq j \leq q \\
a_l(y_j) &= 11r^4 + jr^2 + 2,\ 1 \leq j \leq q,\ 2 \leq l \leq L(y_j) \\
a_1(z_k) &= 10r^4 + kr^3 + 4,\ 1 \leq k \leq q \\
a_l(z_k) &= 8r^4 + kr^3 + 4,\ 1 \leq k \leq q,\ 2 \leq l \leq L(z_k)
\end{aligned}$$

**Tripelzahlen** Zu jedem $(x_i, y_j, z_k) \in M$ assoziieren wir die Zahl

$$a(x_i, y_j, z_k) = 10r^4 - kr^3 - jr^2 - ir + 8.$$

Wir erhalten insgesamt eine Menge $\mathcal{A}$ bestehend aus $\sum_{w \in W} L(w) = 3m$ Elementzahlen und $m$ Tripelzahlen. Es wird sich weiter unten implizit ergeben, dass die Gesamtsumme aller $4m$ Zahlen genau $mb$ beträgt, wobei

$$b = 40r^4 + 15.$$

Man verifiziert leicht, dass jede Zahl aus $\mathcal{A}$ größer als $b/5$ und kleiner als $b/3$ ist. (Daher können nur Zahlquadrupel aus $\mathcal{A}$ die Teilsumme $b$ ergeben.) Die Transformation $M \mapsto \mathcal{A}$ ist offensichtlich in Polynomialzeit berechenbar. Da alle Zahlen in $\mathcal{A}$ kleiner als $12r^4 = 12 \cdot (32q)^4 \leq 12 \cdot (32m)^4 \leq 12 \cdot (8|\mathcal{A}|)^4$ sind, ist der maximale Zahlparameter in $\mathcal{A}$ polynomiell in Eingabelänge $n(\mathcal{A}) \geq |\mathcal{A}|$ beschränkt. Der Beweis wird daher abgeschlossen durch folgende **Behauptung** Es existiert genau dann ein perfektes 3-dimensionales Matching für $M$, wenn $\mathcal{A}$ sich in $m$ Quadrupel (vierelementige Teilmengen) zerlegen lässt, so dass jedes Quadrupel die Teilsumme $b$ liefert.

Nehmen wir zunächst an, dass ein perfektes 3-dimensionales Matching $T$ für $M$ vorliegt. Wir bilden für jedes der $m$ Tripel aus $M$ ein Zahlenquadrupel nach der folgenden Strategie:

**Fall 1** Sei $(x_i, y_j, z_k) \in T$. Dann bildet $a(x_i, y_j, z_k)$ zusammen mit $a_1(x_i), a_1(y_j), a_1(z_k)$ ein Quadrupel. Die Summe dieser vier Zahlen ist offensichtlich $b$.

**Fall 2** Sei $(x_i, y_j, z_k) \in M \setminus T$. Dann bildet $a(x_i, y_j, z_k)$ zusammen mit $a_{l_1}(x_i), a_{l_2}(y_j), a_{l_3}(z_k)$ ein Quadrupel. Hierbei werden $l_1, l_2, l_3 \geq 2$ so gewählt, dass keine Nebenzahl mehrfach verwendet wird. (Der Vorrat an Nebenzahlen ist genau passend bemessen.) Die Summe dieser vier Zahlen ist wiederum $b$.

Nehmen wir jetzt umgekehrt an, dass eine erfolgreiche Zerlegung von $\mathcal{A}$ in Zahlenquadrupel vorgegeben ist. Sei $(A_1, A_2, A_3, A)$ eines dieser Quadrupel. Aus $A_1 + A_2 + A_3 + A = b$ folgt

$$(A_1 + A_2 + A_3 + A \bmod r) = (b \bmod r) = 15.$$

Diese Gleichung kann nur eingehalten werden, wenn Indizes $i, j, k, i', j', k', l_1, l_2, l_3$ existieren, so dass die Komponenten des Quadrupels die Form

$$A = a(x_i, y_j, z_k),\ A_1 = a_{l_1}(x_{i'}),\ A_2 = a_{l_2}(y_{j'}),\ A_3 = a_{l_3}(z_{k'})$$

haben. Die Rechnung modulo $r^2$ ergibt dann

$$(i'r + 1) + 2 + 4 + (-ir + 8) = (A_1 + A_2 + A_3 + A \bmod r^2) = (b \bmod r^2) = 15.$$

Hieraus können wir $i' = i$ ableiten. Die Rechnung modulo $r^3$ liefert

$$(ir + 1) + (j'r^2 + 2) + 4 + (-jr^2 - ir + 8) = (A_1 + A_2 + A_3 + A \bmod r^3) = (b \bmod r^3) = 15.$$

Hieraus können wir $j' = j$ ableiten. Die Rechnung modulo $r^4$ ergibt

$$\begin{aligned}(ir + 1) + (jr^2 + 2) + (k'r^3 + 4) + (-kr^3 - jr^2 - ir + 8) \quad &= \\ (A_1 + A_2 + A_3 + A \bmod r^3) = (b \bmod r^3) &= 15.\end{aligned}$$

Es folgt $k' = k$. Der Koeffizient von $r^4$ bei der Summenzahl $b = 40r^4 + 15$ hat den Wert 40. Die Summe von $A_1, A_2, A_3, A$ muss daher auch Koeffizient 40 vor dem Term $r^4$ liefern. Eine Inspektion der Elementzahlen ergibt, dass $A_1, A_2, A_3$ **entweder** drei Hauptzahlen **oder** drei Nebenzahlen sein müssen. (Im ersten Fall ergibt sich Koeffizient 40 als eine Summe der Form $10 + 10 + 10 + 10$, im zweiten als Summe der Form $11 + 11 + 8 + 10$.) Man überzeugt sich nun leicht, dass die Hauptzahl-Quadrupel ein perfektes 3-dimensionales Matching für $M$ repräsentieren. □

**Folgerung 6.46** *4-PARTITION ist stark NP-vollständig.*

### 6.6.3 Weitere stark NP-vollständige Probleme

*NP*-Härte vererbt sich entlang von Karp-Reduktionsketten. Reduktionen, die starke *NP*-Härte erhalten, sind der Gegenstand der folgenden

**Definition 6.47** *Seien $L, L'$ Zahlenprobleme. Eine Karp-Reduktion von $L$ nach $L'$ mit Reduktionsabbildung $f$ heißt* pseudopolynomiell*, falls ein Polynom $q$ existiert mit $M(f(I)) \leq q(M(I))$. In Worten: wenn das Maximum der Zahlparameter in Eingabe $I$ durch $M(I)$ beschränkt ist, dann ist das Maximum der Zahlparameter in der transformierten Eingabe $I' = f(I)$ durch $q(M(I))$ beschränkt.*

Das folgende Resultat ist offensichtlich:

**Theorem 6.48** *Seien $L$ und $L'$ Zahlenprobleme und sei $L$ pseudopolynomiell Karp-reduzierbar auf $L'$. Falls ein pseudopolynomieller Algorithmus für $L'$ existiert, dann existiert auch ein pseudopolynomieller Algorithmus für $L$. Falls andererseits $L$ stark NP-hart ist, dann ist auch $L'$ stark NP-hart.*

Mit Hilfe pseudopolynomieller Reduktionen können wir ausgehend von 4-PARTITION weitere stark *NP*-vollständige Zahlenprobleme ausfindig machen:

**Lemma 6.49** *(ohne Beweis)*
*4-PARTITION ist pseudopolynomiell Karp-reduzierbar auf 3-PARTITION.*

**Lemma 6.50** *4-PARTITION und 3-PARTITION sind pseudopolynomiell Karp-reduzierbar auf BP.*

**Beweis** Wir führen den Beweis für 3-PARTITION. (Der Beweis für 4-PARTITION ergibt sich mit einem analogen Argument.) Es handelt sich um eine Transformation mit Spezialisierung: BP ist das Teilproblem von 3-PARTITION, bei welchem die Binkapazität auf $b$ und die Kostenschranke (Schranke für die erlaubte Anzahl von Bins) auf $m$ gesetzt wird. $m$ Bins der Kapazität $b$ reichen nämlich genau dann aus, wenn die gegebene Zahlenkollektion (mit Gesamtsumme $mb$) sich in $m$ Tripel mit Teilsumme $b$ zerlegen lässt. □

**Folgerung 6.51** *3-PARTITION und BP sind stark NP-vollständig.*

Wir bemerken abschließend, dass 3-PARTITION für pseudopolynomielle Karp-Reduktionen ein ähnlich beliebtes Quellproblem ist wie 3-SAT für (gewöhnliche) Karp-Reduktionen.

## 6.7 Approximationsalgorithmen und NP-harte Approximation

Ein *Approximationsalgorithmus* zu einem Optimierungsproblem $\Pi$ ist ein polynomiell zeitbeschränkter Algorithmus $A$, der zu einer Eingabeinstanz $I$ eine "legale Lösung" $\sigma$ ausgibt, die eine "mathematische Gütegarantie" besitzt. Dabei sagen wir $A$ hat *Güte* $k$, wenn $A$ zu jeder Eingabeinstanz $I$ eine Lösung $\sigma$ konstruiert, deren Wert vom Wert der optimalen Lösung multiplikativ maximal um Faktor $k$ abweicht. Ein Approximationsalgorithmus der Güte 4/5 für ein Maximierungsproblem würde also stets mindestens 80 Prozent des optimalen Profites erbeuten. Ein Approximationsalgorithmus der Güte 1.5 für ein Minimierungsproblem würde höchstens anderthalb mal soviel Kosten wie nötig verursachen. Es ist naheliegend zu versuchen, ein *NP*-hartes Optimierungsproblem fast-optimal mit einem Approximationsalgorithmus zu lösen, dessen Güte möglichst nahe bei 1 liegen sollte. I.A. wird es eine Barriere $k_0$ geben, so dass Approximationsalgorithmen $A_k$ mit Güte $k > k_0$ existieren, aber nicht mit Güte $k < k_0$ (unter der $P \neq NP$ Voraussetzung). Grenzfälle sind (formuliert für Minimierungsprobleme):

**Keine konstante Gütegarantie** $k_0 = \infty$.
Jeder Approximationsalgorithmus produziert Lösungen, deren Kosten die minimalen Kosten um einen beliebig großen Faktor überschreiten können.

**Approximationsschema** $k_0 = 1$.
Für jedes $\epsilon > 0$ existiert eine Approximationsalgorithmus $A_\epsilon$ mit Güte $1 + \epsilon$.

Wir behandeln das Thema der Approximationsalgorithmen in diesem Abschnitt nur am Beispiel des Problems des Handelsreisenden und am Beispiel des Rucksackproblems. Es wird sich folgendes Bild ergeben:

- TSP besitzt keine konstante Gütegarantie (sofern $P \neq NP$).
- Für „Metrisches TSP" (ein Teilproblem von TSP) hingegen gibt es einen Approximationsalgorithmus der Güte 1.5.
- Für KNAPSACK gibt es ein Approximationsschema.

Am Ende des Abschnittes diskutieren wir kurz grundsätzliche Barrieren beim Design von Approximationsalgorithmen in Form von Resultaten zur NP-harten Approximation.

### 6.7.1 Approximierbarkeit von TSP

Für TSP (in seiner allgemeinen Form) kann man sich klar machen, dass keine konstante Gütegarantie existiert (außer wenn $P = NP$). Wir reduzieren zu diesem Zweck das Problem des Hamiltonschen Kreises (HC) in geeigneter Weise auf das Problem des Handelsreisenden (TSP). Sei $G = (V, E)$ der Eingabegraph zu HC. Knotenmenge $V$ bestehe aus $n \geq 2$ Knoten, die wir mit den Nummern von 1 bis $n$ identifizieren. Will man lediglich HC $\leq_{pol}$ TSP nachweisen, genügt die uns bereits bekannte Reduktion:
Setze in der $(n \times n)$-Distanzmatrix $D$ den Eintrag $d_{i,j}$ auf 1 falls $\{i, j\} \in E$, und andernfalls auf 2.

Hieraus ergibt sich zwar die $NP$-Härte von TSP, aber es ist noch nicht ausgeschlossen, dass vernünftige Approximationsalgorithmen existieren. Betrachten wir aber nun die folgende leichte Modifikation der alten Reduktion:
Setze in der $(n \times n)$-Distanzmatrix $D$ den Eintrag $d_{i,j}$ auf 1 falls $\{i, j\} \in E$, und andernfalls auf $kn$.
Es folgt, dass bez. $D$ genau dann eine Rundreise der Länge $n$ existiert, wenn $G$ einen Hamiltonschen Kreis enthält. Falls jedoch in $G$ kein Hamiltonscher Kreis existiert, muss die Rundreise mindestens eine Kante außerhalb von $E$ verwenden. In diesem Fall hat sie mindestens die Länge $n - 1 + kn$. Gäbe es einen Approximationsalgorithmus für TSP der Güte $k$, so würde im Falle der Existenz eines Hamiltonschen Kreises in $G$ eine Rundreise der Länge höchstens $kn$ produziert, andernfalls jedoch eine Rundreise der Länge mindestens $kn + n - 1 > kn$. Mit anderen Worten: mit Hilfe der approximativen Lösung des TSP könnten wir HC exakt lösen. Falls $P \neq NP$, ist dies jedoch nicht möglich. Somit gibt es keine garantierte Güte $k$ für Approximationsalgorithmen zu TSP (außer wenn $P = NP$).

Die Theorie der NP-Vollständigkeit hat uns signalisiert, dass es vermutlich Zeitverschwendung ist, nach einem Approximationsalgorithmus für TSP zu suchen. Betrachten wir nun aber die Einschränkung von TSP auf Distanzmatrizen $D$, die symmetrisch sind, d.h.,

$$\forall 1 \leq i < j \leq n : \; d_{ij} = d_{ji},$$

und die Dreiecksungleichung erfüllen, d.h.,

$$\forall 1 \leq i, j, k \leq n : \; d_{ik} \leq d_{ij} + d_{jk}.$$

Diese Einschränkungen sind für praktische Anwendungen durchaus vernünftig, denn sie besagen in salopper Formulierung:

**Symmetrie** Von A nach B ist es soweit wie von B nach A.

**Dreiecksungleichung** Von A nach C kann es nicht weiter sein als von A über B nach C.

Das durch Symmetriebedingung und Dreiecksungleichung eingeschränkte TSP-Problem wird als *Metrisches TSP* bezeichnet.

Eine Eingabe des metrischen TSP lässt sich auf die offensichtliche Weise als vollständiger, ungerichteter Graph mit Kantengewichten auffassen. Die Knoten repräsentieren die Städte und ein Kantengewicht die Distanz zwischen zwei Städten. Wegen obiger Symmetriebedingung können wir den Graphen als ungerichtet auffassen.

Wir stellen zur Bequemlichkeit ein paar (teilweise schon bekannte) graphentheoretische Konzepte bereit, die beim Entwurf eines Approximationsalgorithmus für das metrische TSP eine Rolle spielen. Sei $G$ ein ungerichteter Graph. Ein *Weg* in $G$ ist eine Folge $v_1, \ldots, v_r$ von $r \geq 1$ Knoten, wobei für alle $i = 1, \ldots, r-1$ Knoten $v_i$ und $v_{i+1}$ durch eine Kante verbunden sein müssen. (Ein Grenzfall ist der aus nur einem Knoten bestehende "Punktweg".) Falls $r \geq 2$ und $v_1 = v_r$, dann heisst der Weg auch *geschlossener Weg* oder *Kreis*. Eine *Euler-Tour* in $G$ ist ein Kreis, der jede Kante in $G$ genau einmal durchläuft. $G$ heisst *zusammenhängend*, wenn zwei Knoten sich stets durch einen Pfad miteinander verbinden lassen. Ein *ungerichteter Baum* ist ein zusammenhängender, kreisloser ungerichteter Graph. Wenn man aus einem

Baum eine Kante entfernt (ohne die Randknoten der Kante dabei mitzuentfernen), zerfällt er in zwei Teile. Ein Baum ist gewissermaßen die ökonomischste Art, alle Knoten durch Pfade miteinander zu verbinden. Ein *Untergraph* von $G = (V, E)$ ist gegeben durch eine Knotenmenge $V' \subseteq V$ und alle Kanten aus $E$, die Knoten aus $V'$ miteinander verbinden. Man spricht auch von dem *durch $V'$ in $G$ induzierten Untergraphen*. Ein *Teilgraph* von $G = (V, E)$ ist ein Graph $G' = (V', E')$ mit $V' \subseteq V$ und $E' \subseteq E$. Ein *Spannbaum (spanning tree)* von $G$ ist ein Teilgraph der Form $T = (V, E')$, der ein Baum ist. Er enthält also alle Knoten von $G$ und verbindet diese baumartig. Ein solcher Spannbaum kann natürlich nur dann existieren, wenn $G$ zusammenhängend ist. Im Falle von Kantengewichten können wir $T = (V, E')$ die Kosten

$$c(T) = \sum_{e \in E'} w(e)$$

zuordnen. Ein *minimaler Spannbaum (minimum spanning tree)* ist ein Spannbaum minimaler Kosten. Es ist bekannt, dass minimale Spannbäume effizient berechnet werden können (zum Beispiel durch den Algorithmus von Kruskal).

Mit diesem Wissen ausgestattet ist es nun leicht, einen Approximationsalgorithmus der Güte 2 für das metrische TSP zu skizzieren. Es sei $G = (V, E)$ der vollständige ungerichtete Graph mit $n$ Knoten $1, \ldots, n$. Knoten $i$ repräsentiert dabei die $i$-te Stadt. Kante $\{i, j\}$ erhält als Gewicht die Distanz $d_{i,j}$ zwischen den Städten $i$ und $j$. Wir berechnen einen minimalen Spannbaum $T = (V, E')$ von $G$ (zum Beispiel mit dem Algorithmus von Kruskal). Seien $c$ die Gesamtkosten von $T$. Wir erhalten eine Rundreise

$$R(T) = 3, 5, 8, 5, 1, 5, 7, 10, 7, 5, 3, 2, 9, 4, 9, 6, 9, 2, 3,$$

wenn wir (wie in Abbildung 6 angedeutet) einmal um $T$ herumlaufen und dabei die angetroffenen Knoten (=Städte) der Reihe nach auflisten. Da Rundreise $R(T)$ jede Kante von $T$ zweimal durchläuft, hat sie Länge $2c$. $R(T)$ hat noch einen Schönheitsfehler: die Städte werden i.A. mehrfach besucht. Wir erhalten eine legale Lösung von TSP — also eine Permutation $P(T)$ von $1, \ldots, n$ —, wenn wir in $R(T)$ alle Vorkommen von Knoten außer dem ersten (also alle Duplikate) streichen. In unserem Beispiel führt dies zu

$$P(T) = 3, 5, 8, 1, 7, 10, 2, 9, 4, 6.$$

Wegen der Dreiecksungleichung kann $P(T)$ nicht länger als $R(T)$ sein. Die Kosten von $P(T)$ sind also maximal $2c$.

Wie verhält es sich nun mit einer optimalen Rundreise? Im Graphen $G$ formt diese einen Hamiltonschen Kreis $C$. Entfernen wir aus $C$ (irgend-) eine Kante, entsteht ein (sehr spezieller) Spannbaum $T(C)$ (s. Abbildung 7). Die Länge von Rundreise $C$ ist nicht kleiner als die Gesamtkosten von $T(C)$. Diese wiederum betragen mindestens $c$. Also hat $C$ mindestens die Länge $c$ und der skizzierte Approximationsalgorithmus die Güte 2.

Vom bisherigen Erfolg berauscht stecken wir uns jetzt ein ehrgeizigeres Ziel: Entwurf eines Approximationsalgorithmus mit Güte 1.5. Zu diesem Zweck muss unser Approximationsalgorithmus noch etwas "aufgepeppt" werden. Der Faktor 2, der uns vom Optimum trennt, resultiert daraus, dass die konstruierte Rundtour jede Kante des minimalen Spannbaumes $T$

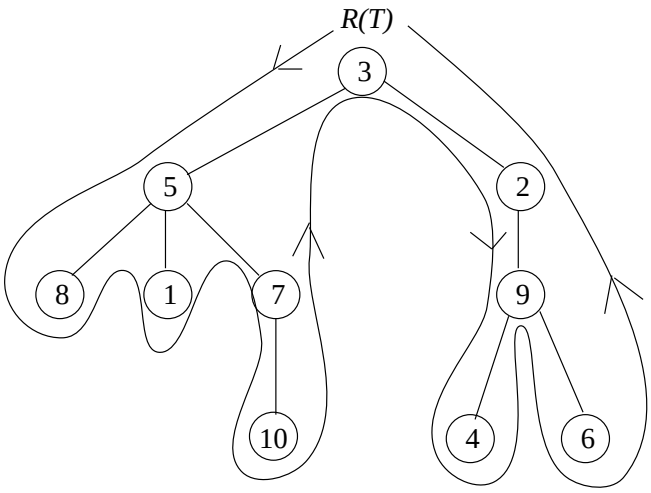



Abbildung 6: Die aus einem Minimum Spanning Tree abgeleitete Rundreise.

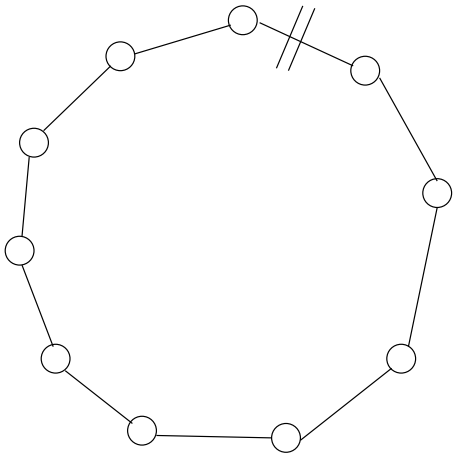


Abbildung 7: Der aus einer Rundreise abgeleitete Spanning Tree.

implizit 2-mal durchläuft. Wenn wir eine Euler-Tour durch $T$ zur Verfügung hätten, würden wir jede Kante nur 1-mal durchlaufen.

Dies wirft die Frage auf, für welche Graphen Euler-Touren existieren. Die Antwort liefert folgendes

**Lemma 6.52** *Ein zusammenhängender Graph besitzt genau dann eine Euler-Tour, wenn jeder Knoten einen geraden Grad (also geradzahlig viele Nachbarn) besitzt.*[29]

**Beweis** Wenn ein Knoten $v$ mit ungeradem Grad, sagen wir Grad $2d+1$ existiert, dann kann es keine Euler-Tour geben: wenn der Kreis den Knoten $v$ zum $d+1$-mal betritt, kann er ihn nicht mehr verlassen (Sackgassenargument).
Wenn ein zusammenhängender Graph $G$ nur Knoten mit geradem Grad besitzt, dann kann eine Euler-Tour (effizient) konstruiert werden wie folgt:

[29] In diesem Fall spricht man auch von einem *Euler'schen Graphen.*

1. Konstruiere einen ersten Kreis $C$, indem Du solange auf $G$ spazieren gehst (ohne eine Kante doppelt zu laufen), bis Du zum Ausgangspunkt zurückkehrst.[30]

2. Wenn $C$ bereits eine Euler-Tour darstellt, dann gib $C$ aus. Andernfalls mache weiter mit Schritt 3.

3. Wähle einen Knoten $v$ auf $C$, der noch unbenutzte Ausgangskanten hat.[31] Konstruiere von $v$ aus einen zweiten Kreis $C'$ (wieder durch „spazieren gehen"). Verschmelze[32] $C$ und $C'$ zu einem neuen Kreis und nenne diesen wieder $C$. Gehe zurück zu Schritt 2.

□

Aus dem Beweis ergibt sich die

**Folgerung 6.53** *Es kann in Polynomialzeit getestet werden, ob ein Graph $G$ eine Euler-Tour enthält. Gegebenenfalls kann diese auch in Polynomialzeit*[33] *konstruiert werden.*

Zurück zur Frage, ob es eine Euler-Tour durch den minimalen Spannbaum $T$ gibt? Leider nein! In seiner Eigenschaft als Baum muss $T$ Knoten ungeraden Gerades besitzen (zum Beispiel alle Blätter). Es gilt aber immerhin das folgende

**Lemma 6.54** *Jeder Graph $G$ hat geradzahlig viele Knoten ungeraden Grades.*

**Beweis** Wenn wir die Knotengrade $d_1, \ldots, d_n$ addieren zählen wir jede der $m$ Kanten zweimal (da jede Kante zwei Randknoten besitzt):

$$\sum_{i=1}^{n} d_i = 2m \quad .$$

Da die rechte Seite der Gleichung eine gerade Zahl ist, muss es auf der linken seite geradzahlig viele ungeradzahlige Terme geben. □

**Idee** Es seien $u_1, \ldots, u_{2k}$ die Knoten ungeraden Grades in $T$. Ergänze $T$ zu einem Euler'schen Graphen, indem Du $k$ „Heiratskanten" hinzufügst, die die Knoten $u_i$ „perfekt verheiraten". Wähle hierzu $k$ Heiratskanten mit minimalem Gesamtgewicht (ein sogenanntes „Minimum Weight Perfect Matching", welches in Polynomialzeit berechenbar ist).

Hieraus ergibt sich der folgende von Christofides vorgeschlagene Approximationsalgorithmus:

[30] Da jeder Knoten geraden Grad hat, kann man den Spaziergang stets fortsetzen, solange man noch nicht den Ausgangspunkt erreicht hat.

[31] Da $G$ zusammenhängend ist, muss es einen solchen Knoten $v$ auf $C$ geben.

[32] Laufe durch $C$ bis zum Erreichen von $v$, dann durchlaufe $C$' bis zum erneuten Erreichen von $v$ und schließlich durchlaufe den Rest des temporär unterbrochenen Kreises $C$.

[33] Auf eine „Random Access Maschine" können wir „Polynomialzeit" durch „Linearzeit" präzisieren.

1. Gegeben die $n$-Clique (bestehend aus den $n$ Städten) mit Kantengewichten $d_{i,j}$, berechne einen minimalen Spannbaum $T$.

2. Ergänze $T$ (wie soeben beschrieben) durch ein „Minimum Weight Perfect Matching“ für seine ungeraden Knoten zu einem Euler'schen Graphen $T'$.

3. Konstruiere eine Euler-Tour durch $T'$ und gib diese als Rundreise aus.

**Satz 6.55** *Die vom Algorithmus von Christofides konstruierte Approximationsalgorithmus für TSP hat die Güte* 1.5.

**Beweis** Es sei $R_*$ eine optimale Rundreise mit Kosten $c(R_*)$. Wir wissen bereits, dass $c(R_*) \geq c(T)$. Der Algorithmus von Christofides konstruiert eine Rundreise $R$ mit Kosten $c(T)+c(M)$, wobei $M$ das „Minimum Weight Perfect Matching“ bezeichnet und $c(M)$ das Gesamtgewicht der dabei beteiligten Kanten. Die Güte 1.5 ergibt sich, wenn wir $c(R_*) \geq c(M)/2$ nachweisen können. Zu diesem Zweck sei $R'_*$ sie Subtour von $R$, die resultiert, wenn wir Knoten geraden Grades in $R_*$ auslassen. $R_*$ enthält $2k$ Kanten, sagen wir $e_1, \ldots, e_{2k}$. Offensichtlich bildet sowohl $\{e_1, e_3, \ldots, e_{2k-1}\}$ als auch $\{e_2, e_4, \ldots, e_{2k}\}$ ein perfektes Heiratssystem für die Knoten $u_1, u_2, \ldots, u_{2k}$. Somit gilt

$$c(R_*) \geq c(R'_*) \geq 2c(M) \ ,$$

was den Beweis abschließt. □

### 6.7.2 Approximierbarkeit von KNAPSACK

Sahni hat 1975 folgende Approximationsalgorithmen $A_k$ für KNAPSACK (Eingabe: Gewichte $w_1, \ldots, w_n$, Profite $p_1, \ldots, p_n$ und Gewichtsschranke $W$) vorgeschlagen:

1. Sortiere die $n$ Objekte nach ihrer „Profitrate“, so dass $p_1/w_1 \geq \cdots \geq p_n/w_n$.

2. Zu jeder Teilmenge $I \subseteq \{1, \ldots, n\}$ mit $|I| \leq k$ bestimme ihr Gesamtgewicht $W(I) = \sum_{i \in I} w_i$ und den durch sie realisierten Profit $P(I) = \sum_{i \in I} p_i$.

3. Bestimme eine „Kandidatenrucksack“ $R(I)$, in den zunächst die durch $I$ bestimmten Objekte aufgenommen werde, um ihn danach (unter Beachtung der Gewichtsschranke) mit anderen Objekten aufzufüllen. Beim Auffüllen erhalten Objekte mit höherer Profitrate höhere Priorität.

4. Wähle schließlich den profitabelsten Kandidatenrucksack.

**Beispiel 6.56** *Wir betrachten eine Eingabe für KNAPSACK mit* $n = 8$ *Objekten und Gewichtsschranke* $W = 110$. *Sowohl die weiteren Eingabeparameter als auch der Beispiellauf von* $A_0$ *sind aus folgender Tabelle ersichtlich:*

| $i$ | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* |
|---|---|---|---|---|---|---|---|---|
| $p_i$ | *11* | *21* | *31* | *33* | *43* | *53* | *55* | *65* |
| $w_i$ | *1* | *11* | *21* | *23* | *33* | *43* | *45* | *55* |
| $b_i$ | *1* | *1* | *1* | *1* | *1* | *0* | *0* | *0* |
| $W_\Sigma$ | *1* | *12* | *33* | *56* | *89* | *89* | *89* | *89* |
| $P_\Sigma$ | | | | | | | | *139* |

*Hierbei ist folgendes zu beachten:*

- *Die Eingabeparameter sind bereits nach absteigender Profitrate sortiert.*
- *$b_i$ bezeichnet ein Indikatorbit, welches mit dem Wert 1 anzeigt, dass Objekt $i$ in den Rucksack gesteckt wurde.*
- *$W_\Sigma$ ist eine dynamische Variable, die on-line das bisher akkumulierte Gewicht mitzählt. Mit Hilfe von $W_\Sigma$ kann entschieden werden, ob das nächste inspizierte Objekt ohne Überschreitung der Gewichtsschranke $W = 110$ in den Rucksack gesteckt werden kann.*
- *$P_\Sigma$ bezeichnet in entsprechender Weise den akkumulierten Profit. Da uns hier nur der endgültige Wert interessiert, haben wir die Zwischenergebnisse nicht angegeben.*

*Wir halten als Ergebnis fest, dass $A_0$ die Lösung $I_0 = \{1, 2, 3, 4, 5\}$ mit Profit 139 (und Gewicht 89) berechnet.*
*Betrachten wir nun einen Beispiellauf von $A_1$. Es sei daran erinnert, dass $I$ mit $|I| = k$ die Menge der Objekte (bestehend aus 1 Objekt im Falle $k = 1$) bezeichnet, die vorab in den Rucksack gesteckt werden. Da $A_0$ die Objekte $1, 2, 3, 4, 5$ ausgewählt hat, würden die Beispielläufe mit $I = \{1\}, \{2\}, \{3\}, \{4\}, \{5\}$ nur das Ergebnis $I_0 = \{1, 2, 3, 4, 5\}$ reproduzieren. Wir können uns also auf die Kandatenrucksäcke $R(I)$ mit $I = \{6\}, \{7\}, \{8\}$ beschränken. Diese drei Beispielläufe sind in den folgenden Tabellen zu sehen (wobei die Eintragungen für das vorab in $R(I)$ aufgenommene Objekt zur besseren Kenntlichkeit* **fett** *gedruckt sind):*

| $i$ | **6** | *1* | *2* | *3* | *4* | *5* | *7* | *8* |
|---|---|---|---|---|---|---|---|---|
| $p_i$ | **53** | *11* | *21* | *31* | *33* | *43* | *55* | *65* |
| $w_i$ | **43** | *1* | *11* | *21* | *23* | *33* | *45* | *55* |
| $b_i$ | **1** | *1* | *1* | *1* | *1* | *0* | *0* | *0* |
| $W_\Sigma$ | **43** | *44* | *55* | *76* | *99* | *99* | *99* | *99* |
| $P_\Sigma$ | | | | | | | | *149* |

| $i$ | **7** | *1* | *2* | *3* | *4* | *5* | *6* | *8* |
|---|---|---|---|---|---|---|---|---|
| $p_i$ | **55** | *11* | *21* | *31* | *33* | *43* | *53* | *55* |
| $w_i$ | **45** | *1* | *11* | *21* | *23* | *33* | *43* | *55* |
| $b_i$ | **1** | *1* | *1* | *1* | *1* | *0* | *0* | *0* |
| $W_\Sigma$ | **45** | *46* | *57* | *78* | *101* | *101* | *101* | *101* |
| $P_\Sigma$ | | | | | | | | *151* |

| $i$ | **8** | *1* | *2* | *3* | *4* | *5* | *6* | *8* |
|---|---|---|---|---|---|---|---|---|
| $p_i$ | **65** | *11* | *21* | *31* | *33* | *43* | *53* | *55* |
| $w_i$ | **55** | *1* | *11* | *21* | *23* | *33* | *43* | *45* |
| $b_i$ | **1** | *1* | *1* | *1* | *0* | *0* | *0* | *0* |
| $W_\Sigma$ | **55** | *66* | *67* | *88* | *88* | *88* | *88* | *88* |
| $P_\Sigma$ | | | | | | | | *118* |

*Der beste Kandidatenrucksack (und somit die Ausgabe von $A_1$) ist $R(\{7\})$. Er enthält die Objekte $1, 2, 3, 4, 7$ und erzielt einen Profit von $151$ (bei einem Gewicht von $101$). Man überlegt sich leicht, dass die optimale Lösung die Objekte $1, 2, 3, 5, 6$ in den Rucksack steckt. Sie erzielt einen Profit von $159$ (bei einem Gewicht von $109$). Algorithmus $A_2$ hätte die optimale Bepackung des Rucksackes beim Beispiellauf mit $I = \{5, 6\}$ aufgespürt.*

Der Algorithmus $A_0$ geht im Prinzip nur Profitraten-basiert vor (da er vorab in den Rucksack nur „die leere Menge packt“ und ihn danach Profitraten-basiert auffüllt). Anhand von „teuflisch“ ausgewählten Eingaben lässt sich zeigen (s. Übung), dass $A_0$ keine konstante Güte $c$ mit $c < \infty$ besitzt. Wenn allerdings „kleine Engelchen“ die Eingabe auswählen, dann ist $A_0$ gar nicht so übel:

**Lemma 6.57** *Die Objekte $1, \ldots, n$ seien absteigend nach Profitrate sortiert. Falls die Eingabe die Bedingung*

$$\exists j \in \{1, \ldots, n\} : \sum_{i=1}^{j} w_i = W \tag{23}$$

*erfüllt, dann packt $A_0$ einen optimalen Rucksack.*

**Beweis** Da $A_0$ Profitraten-basiert vorgeht, werden die Objekte $1, \ldots, j$ in den Rucksack gepackt (und sonst nichts, da dann die Gewichtsschranke $W$ erreicht ist). Intuitiv sollte klar sein, dass die Gewichtsschranke optimal ausgenutzt wurde, da die Objekte mit den $j$ höchsten Profitraten ausgewählt wurden. Folglich ist $I_0 := \{1, \ldots, j\}$ eine optimale Lösung. Ein etwas formalerer Beweis könnte von einer optimalen Lösung $I_*$ ausgehen und folgendes ausnutzen:

- Wir können oBdA $I_0 \cap I_* = \emptyset$ annehmen. Falls diese Bedingung nicht erfüllt ist, können wir nämlich die Objekte aus $I_0 \cap I_*$ aus der Eingabe entfernen und so zu einer neuen Eingabe $E_R$ mit den restlichen Objekten $R := \{1, \ldots, n\} \setminus (I_0 \cap I_*)$ und der Gewichtsschranke $W_R = W - \sum_{i \in I_0 \cap I_*} w_i$ übergehen. Offensichtlich ist $I_* \cap R$ eine optimale Lösung für $E_R$ und $I_0 \cap R$ ist die Lösung, die $A_0$ auf Eingabe $E_R$ produzieren würde. Weiterhin erfüllt $E_R$ immer noch die Bedingung (23). Wenn nun $A_0$ auf $E_R$ optimal ist, dann gilt $\sum_{i \in I_0 \cap R} p_i = \sum_{i \in I_* \cap R} p_i$ und somit wäre $A_0$ dann auch auf der vollständigen Eingabe $E$ optimal.
- Für jede Auswahl $I \subseteq \{1, \ldots, n\}$ gilt (s. Übung):

$$\min_{i \in I} \frac{p_i}{w_i} \leq \frac{\sum_{i \in I} p_i}{\sum_{i \in I} w_i} \leq \max_{i \in I} \frac{p_i}{w_i} \quad .$$

Wenn $I_0 = \{1, \ldots, j\}$ und $I_*$ disjunkt sind, dann gilt $I_* \subseteq \{j+1, \ldots, n\}$ und es folgt

$$\frac{\sum_{i \in I_0} p_i}{\sum_{i \in I_0} w_i} \geq \min_{i \in I_0} \frac{p_i}{w_i} = \frac{p_j}{w_j} \geq \max_{i \in I_*} \frac{p_i}{w_i} \geq \frac{\sum_{i \in I_*} p_i}{\sum_{i \in I_0} w_i} \ .$$

Wegen

$$\sum_{i \in I_*} w_i \leq W = \sum_{i \in I_0} w_i$$

ergibt sich hieraus $\sum_{i \in I_0} p_i \geq \sum_{i \in I_*} p_i$. □

Approximationsalgorithmus $A_k$ mit $k \geq 1$ kommt auch mit Eingaben ganz gut zurecht, die sich ein „Teufel“ ausgedacht hat:

**Satz 6.58** *$A_k$ ist ein Approximationsalgorithmus für KNAPSACK mit Güte $1 + 1/k$.*

**Beweis** Es bezeichne $I_*$ die Indexmenge eines profitabelsten Rucksackes $R_*$ mit Profit $P_* = \sum_{i \in I_*} p_i$. Falls $|I_*| \leq k$, dann wäre $R_*$ einer der Kandidatenrucksäcke von $A_k$. In diesem Fall würde $A_k$ den maximalen Profit erzielen. Für die weitere Diskussion können wir uns also auf den Fall $|I_*| \geq k+1$ konzentrieren. Wir indizieren die Objekte so um, dass $I_* = \{1, \ldots, k, k+1, \ldots, k+l\}$, wobei die Objekte $1, \ldots, k$ die $k$ profitabelsten Objekte in $R_*$ seien. Weiterhin seien die Objekte $k+1, \ldots, k+l$ absteigend nach ihrer Profitrate geordnet. Da $1, \ldots, k$ die $k$ profitabelsten Objekte in $R_*$ sind und da auch $k + \lambda$ zu $R_*$ gehört, folgt

$$p_{k+\lambda} \leq \frac{1}{k+1} \left( p_{k+\lambda} + \sum_{i=1}^{k} p_i \right) \leq \frac{P_*}{k+1} \ . \tag{24}$$

Wir betrachten nun den Kandidatenrucksack $R(I)$ mit $I = \{1, \ldots, k\}$. Dieser Rucksack enthält von vorne herein die Objekte $1, \ldots, k$ mit Gesamtgewicht $W_0 := \sum_{i=1}^{k} w_i$ und Gesamtprofit $P_0 = \sum_{i=1}^{k} p_i$. Beachte, dass $A_k$ auf den restlichen Objekten $R := \{k+1, \ldots, n\}$ bezüglich Gewichtsschranke $W_R = W - W_0$ Profitraten-basiert vorgeht so wie Algorithmus $A_0$. Es bezeichne $\lambda \in \{1, \ldots, l\}$ den kleinsten Index eines nicht in $R(I)$ aufgenommenen Objektes (den es geben muss, wenn $R(I)$ und $R_*$ nicht übereinstimmen, was wir oBdA annehmen). Es bezeichne $I' \supseteq \{k+1, \ldots, k+\lambda-1\}$ die Menge der Objekte aus $R$, die sich zum Zeitpunkt der Inspektion von Objekt $k + \lambda$ in $R(I)$ befinden und es sei

$$R' := I' \cup \{k+\lambda, \ldots, k+l\} \supseteq I_* \cap R \ .$$

Da Objekt $k + \lambda$ nicht in $R(I)$ aufgenommen wurde, muss

$$W' := w_{k+\lambda} + \sum_{i \in I'} > W_R$$

gelten. Bezüglich Gewichtsschranke $W'$ an Stelle von $W_R$ wäre folgendes geschehen:

- $A_k$ hätte Objekt $k+\lambda$ ebenfalls aufgenommen (und damit Gewichtsschranke $W'$ genau erreicht).

- Die Objekte aus $R'$ hätten zusammen mit der Gewichtsschranke $W'$ Bedingung (23) erfüllt.

- Gemäß Lemma 6.57 wäre $I' \cup \{k+\lambda\}$ eine optimale Lösung auf der Objektmenge $R'$ mit Gewichtsschranke $W'$ gewesen.

Hieraus folgt nun, dass

$$P_* \leq P_0 + \left(\sum_{i \in I'} p_i\right) + p_{k+\lambda} = P' + p_{k+\lambda} \overset{(24)}{\leq} P' + \frac{P_*}{k+1} \ ,$$

woraus sich mit einer leichten Rechnung $P_*/P' \leq 1 + 1/k$ ergibt. Da der von $R(I)$ erzielte Profit mindestens $P'$ beträgt, ist der Beweis jetzt abgeschlossen. □

Anhand von „teuflisch“ ausgewählten Eingaben für KNAPSACK lässt sich zeigen, dass $A_k$ mit $k \geq 1$ keine konstante Güte $c$ mit $c < 1 + 1/k$ besitzt (s. Übung).

Der Algorithmus $A_k$ ist für jede Konstante $k$ polynomiell zeitbeschränkt. Wie aber ist die Abhängigkeit der Zeitschranke von dem Güteparameter $k$?[34] Die Antwort ist etwas frustrierend: allein schon die Anzahl der Kandidatenrucksäcke (sprich: die Anzahl aller Teilmengen $I \subseteq \{1, \ldots, n\}$ mit $|I| \leq k$) ist proportional zu $n^k$. Die Laufzeit wächst also exponentiell mit $k$. Das Approximationsschema von Sahni ist daher nur für kleine Werte von $k$ praktikabel.

**Definition 6.59** *Ein Algorithmus $A$ für ein Optimierungsproblem $\Pi$, der neben der eigentlichen Eingabe einen zusätzlichen Eingabeparameter $k$ erhält und für festes $k$ einen Approximationsalgorithmus $A_k$ der Güte $1 + 1/k$ für $\Pi$ repräsentiert, heißt* volles Approximationsschema*, wenn seine Laufzeit sich nach oben durch ein bivariates Polynom in der Eingabelänge $N$ und in $k$ beschränken lässt.*

Ibarra und Kim haben 1975 mit einer Technik namens „Rounding and Scaling“ ein volles Approximationsschema für KNAPSACK entworfen. Wir skizzieren im Folgenden die Grundidee hiervon. Es bezeichne $P_n = \sum_{i=1}^{n} p_i$ die Summe aller Einzelprofite, $P_*$ den vom optimalen Rucksack erzielten Profit, $M := \max_{i=1,\ldots,n} p_i$ den maximalen Profit-Zahlparameter und $N$ die Kodierungslänge der Eingabe $E$. Wir nehmen im Folgenden oBdA an, dass kein Objekt ein $W$ überschreitendes Gewicht hat. Folglich gilt $P_* \geq M$. Zweifellos gibt es einen pseudopolynomiellen Algorithmus $A$, der eine zweidimensionale Tabelle $T = (T[i,j])_{1 \leq i \leq n, 1 \leq j \leq P_n}$ ausfüllt, so dass $T[i,j]$ das minimale Gewicht ist, mit welchem sich ein Profit von mindestens $j$ durch eine geeignete Auswahl aus den Objekten $1, \ldots, i$ erzielen lässt (bzw. $T[i,j] = \infty$, falls $p_1 + \cdots p_i < j$). Eine geeignete Implementierung dieses Algorithmus (dynamisches Programmieren; s. Übung) hat eine Laufzeit, welche polynomiell von $N$ und in $M$ abhängt. Aus der $n$-ten Zeile von $T$ kann man leicht den maximal möglichen Profit $P_*$ ablesen. Algorithmus $A$ ist leider nur pseudopolynomiell, weil $M$ exponentiell groß in Abhängigkeit von $N$ sein kann. Nun ist der Moment gekommen, in dem „Rounding and Scaling“ eine Rolle spielt. Wir

[34] Genau genommen sollten wir hier einen uniformen Algorithmus $A$ betrachten, der $k$ als zusätzlichen Eingabeparameter erhält und dann vorgeht wie $A_k$.

skalieren die Eingabe $E$ (mit eventuell riesenhaftem $M$) herunter, indem wir die Paramter $p_i$ durch

$$p'_i := \left\lfloor \frac{p_i}{K} \right\rfloor \tag{25}$$

ersetzen, wobei $K$ gemäß

$$K := \frac{M}{(k+1)n} \leq \frac{P_*}{(k+1)n} \tag{26}$$

gewählt wird (aus Gründen, die sich erst später erhellen).[35] Wir bezeichnen die neue Eingabe mit $E'$. Der Approximationsalgorithmus von Ibarra und Kim, im folgenden mit $A$ bezeichnet, geht vor wie folgt:

1. Berechne aus Eingabe $E$ mit den Parametern $p_1, \ldots, p_n; w_1, \ldots, w_n; W$ und dem zusätzlichen Eingabeparameter $k$ die Eingabe $E'$ mit den gemäß (25),(26) berechneten Parametern $p'_1, \ldots, p'_n$ anstelle von $p_1, \ldots, p_n$.

2. Wende den pseudopolynomiellen Algorithmus $A'$ auf $E'$ an und lies aus der resultierenden Tabelle $T$ eine optimale Lösung $I \subseteq \{1, \ldots, n\}$ von $E'$ ab.

3. Gib $I$ als Lösung für die ursprüngliche Eingabe $E$ aus.

**Satz 6.60** *Der Algorithmus A von Ibarra und Kim ist ein volles Approximationsschema für KNAPSACK.*

**Beweis** Wir beginnen mit der Zeitanalyse. Die Laufzeit wird dominiert durch die Berechnung der optimalen Lösung $I$ für die Eingabe $E'$ unter Anwendung des pseudopolynomiellen Algorithmus' $A'$. Wegen $M \leq P_n$ kostet dies größenordnungsmäßig $\text{poly}(N, M/K) = \text{poly}(N, (k+1)n)$ viele Schritte, woraus sich wegen $n \leq N$ eine in $N$ und $k$ polynomiell beschränkte Rechenzeit ergibt.
Abschließend berechnen wir den von $I$ erzielten Profit und vergleichen ihn mit $P_*$. Bezogen auf Eingabe $E$ bzw. $E'$ erzielt $I$ den Profit

$$P_E = \sum_{i \in I} p_i \text{ bzw. } P_{E'} = \sum_{i \in I} p'_i = \sum_{i \in I} \left\lfloor \frac{p_i}{K} \right\rfloor$$

Offensichtlich unterhalten $P_E, P_*$ und $P_{E'}$ die Beziehung

$$KP_{E'} \leq P_E \leq P_* < KP_{E'} + K|I| \leq KP_{E'} + Kn \leq P_E + Kn \ , \tag{27}$$

wobei die drittletzte Ungleichung ausnutzt, dass $I_*$ (also die Lösung mit Profit $P_*$) bezogen auf $E'$ höchstens den (best möglichen) Profit $P_{E'}$ erzielen kann. Nun ergibt sich die Güte $1 + 1/k$ durch eine einfache Rechnung aus

$$P_E \geq P_* - Kn = P_* - \frac{M}{k+1} \geq P_* - \frac{P_*}{k+1} \ .$$

□

[35] Grundsätzlich gilt, dass hohe Werte von $K$ die Laufzeit verbessern, aber die Güte verschlechtern (und umgekehrt für niedrige Werte von $K$). Die Wahl von $K$ ist somit ein Balance-Akt.

### 6.7.3 NP-harte Approximation

Einige Optimierungsprobleme (wie zum Beispiel KNAPSACK) erlauben (volle) Approximationsschemata, andere (wie zum Beispiel TSP) besitzen nicht einmal konstante Gütegarantien (sofern $P \neq NP$). Wo liegen die theoretischen Grenzen für die Güte von Approximationsalgorithmen für ein gegebenes Optimierungsproblem? In diesem Abschnitt gehen wir dieser Frage nach und lernen ein paar Techniken zum Nachweis der Nichtapproximierbarkeit kennen.

Als erste Basistechnik zum Nachweis der Nichtapproximierbarkeit eine Optimierungsproblems besprechen wir die Herstellung einer „mupltiplikativen Lücke (multiplicative gap)“ in Bezug auf den zu optimierenden Parameter. In Verbindung mit TSP hatten wir diese Technik implizit bereits kennen gelernt: falls $P \neq NP$, dann besitzt TSP keine konstante Gütegarantie. Wie hatten wir dieses Resultat erzielt? Wesentlich bei der verwendeten Reduktion von HC auf TSP war die folgende Eigenschaft:
*Wenn der gegebene Graph $G = (V, E)$ einen Hamilton'schen Kreis besitzt, dann erlaubt die von $G$ induzierte Distanzmatrix eine Rundreise der Länge $n = |V|$. Gibt es hingegen keinen Hamilton'schen Kreis, dann hat die kürzeste Rundreise mindestens die Länge $kn + 1$.*
Die Lücke zwischen den Kostenparametern $n$ und $kn + 1$ erlaubt den Schluss:
*Wenn ein Algorithmus für TSP die Güte $k$ besitzt, dann kann er (ohne wesentlichen Effizienzverlust) dazu benutzt werden, HC exakt zu lösen.*
Die meisten Nichtapproximierbarkeitsresultate beruhen auf einer polynomiellen Reduktion, welche eine multiplikative Lücke (im eben beschriebenen Sinn) herstellt. Einfache Beispiele hierfür ergeben sich durch Optimierungsprobleme, die bereits für eine konstante Kosten- bzw. Profitschranke NP-hart sind. Diese erlauben die Herstellung einer multiplikativen Lücke auf triviale Weise. Wir demonstrieren dies anhand von COLORABILITY:

**Bemerkung 6.61** *Falls $P \neq NP$, dann kann kein Approximationsalgorithmus für COLORABILITY eine konstante Güte unterhalb von $4/3$ besitzen.*

**Beweis** Das Graphenfärbungsproblem ist bereits für Kostenschranke 3 (3-COLORABILITY) NP-vollständig. Ein Approximationsalgorithmus mit Güte $c < 4/3$ kann benutzt werden, um die Frage der 3-Färbbarkeit eines gegebenen Graphen exakt zu lösen. Falls ein solcher Algorithmus existierte, wäre $P = NP$. □

Völlig analog ergibt sich der

**Satz 6.62** *Unter der Voraussetzung $P \neq NP$ gilt folgendes. Wenn ein Minimierungspoblem (bzw. Maximierungsproblem) $\Pi$ bereits mit konstanter Kostenschranke $k$ NP-hart ist, dann kann kein Approximationsalgorithmus für $\Pi$ eine konstante Güte unterhalb von $(k + 1)/k$ (bzw. $k/(k - 1)$) besitzen.*

Als zweite Basistechnik verwenden wir den Nachweis der starken NP-Härte. Es gelten nämlich folgende Resultate:

**Satz 6.63** *Es sei $\Pi$ ein Optimierungsproblem mit einem natürlich-zahligen Optimierungsparameter und der Eigenschaft, dass der Wert $C_*(E)$ einer optimalen Lösung für Eingabe $E$ polynomiell in der Kodierungslänge $N(E)$ und in dem maximalen in $E$ vorkommenden Zahlparameter $M(E)$ nach oben beschränkt sind. Dann kann ein volles Approximationsschema für $\Pi$ in einen pseudopolynomiellen Algorithmus für $\Pi$ transformiert werden.*

**Beweis** Wir beschränken uns auf die Betrachtung von einem Minimierungsproblem $\Pi$. Der Beweis für Maximierungsprobleme lässt sich analog führen.
Nach Voraussetzung gibt es ein Polynom $q$ mit der Eigenschaft

$$C_*(E) < q(N(E), M(E))$$

für alle Eingabeinstanzen $E$ von $\Pi$ und ein volles Approximationsschema $A$ für $\Pi$. Der korrespondierende pseudopolynomielle Algorithmus $A'$ (zur exakten Lösung von $\Pi$) geht auf Eingabe $E$ vor wie folgt:

1. $k := q(N(E), M(E))$.
2. Wende $A$ auf Eingabe $E$ und Genauigkeitsparameter $k$ an und erhalte eine Lösung mit Profit $C(E) \leq (1 + 1/k)C_*(E)$.

Da die Laufzeit von $A$ polynomiell in $N(E)$ und $k$ beschränkt ist, ist die Laufzeit von $A'$ polynomiell in $N(E)$ und $M(E)$ beschränkt (pseudopolynomielle Laufzeit). Offensichtlich gilt

$$C(E) - C_*(E) \leq \frac{C_*(E)}{k} = \frac{C_*(E)}{q(N(E), M(E))} < 1 \ .$$

Aus $C(E), C_*(E) \in \mathbb{N}_0$ folgt $C(E) = C_*(E)$. □

**Folgerung 6.64** *Falls $P \neq NP$, dann kann ein stark NP-hartes Optimierungsproblem (mit polynomiell in $N(E)$ und $M(E)$ beschränktem Wert einer optimalen Lösung) kein volles Approximationsschema besitzen.*

Gemäß Satz 6.63 lässt sich ein volles Approximationsschema in einen pseudopolynomiellen Algorithmus transformieren. Zumindest für den Spezialfall von KNAPSACK haben Ibarra und Kim die Umkehrung dieses Satzes demonstriert, indem sie (mit der Technik des „Rounding and Scaling“) einen pseudopolynomiellen Algorithmus in ein volles Approximationsschema transformiert haben. Obschon es kein „Metatheorem“ gibt, welches die Konstruktion von Ibarra und Kim auf beliebige pseudopolynomiell lösbare Optimierungsprobleme verallgemeinert, ist die Technik des „Rounding and Scaling“ in vielen Fällen (ähnlich wie bei KNAPSACK) erfolgreich anwendbar.

**Approximation bis auf eine „additive Konstante“?** Wir haben Approximatiosnalgorithmen kennen gelernt, die das Optimum bis auf einen konstanten Faktor treffen. Stärker wären freilich mathematische Gütegarantien, das Optimum bis auf eine additive Konstante zu treffen. Ist so etwas denkbar? Die Antwort lautet für fast alle natürlichen Optimierungsprobleme NEIN. Wir demonstrieren dies an zwei Beispielen:

**Bemerkung 6.65** *Falls $P \neq NP$, dann kann es keinen Approximationsalgorithmus $A$ für KNAPSACK mit einer Gütegarantie von der Form*

$$\exists k \in \mathbb{N}, \forall E: \ C_*(E) - C_A(E) \leq k$$

*geben, wobei $C_*(E)$ den auf Eingabe $E$ maximal erzielbaren Profit bezeichnet und $C_A(E)$ den vom Algorithmus $A$ auf $E$ erzielten Profit.*

**Beweis** Angenommen es gäbe einen Approximationsalgorithmus $A$ mit einer solchen Gütegarantie. Dann würde der folgende Algorithmus $A'$ KNAPSACK in Polynomialzeit optimal lösen:

1. Es sei $E'$ die aus $E$ resultierende Eingabe, wenn wir alle Zahlparameter um den Faktor $k+1$ hochskalieren (also mit $k+1$ multiplizieren).
2. Wende $A$ auf $E'$ an und erhalte eine Auswahl $I$ der in den Rucksack gepackten Objekte, deren Gesamtprofit vom maximal erzielbaren Profit additiv nur um maximal $k$ abweicht.
3. Gib $I$ als Lösung für die ursprüngliche Eingabe $E$ aus.

Beachte, dass die Probleme $E$ und $E'$ „isomorph“ sind: Korrespondierende Eingabeparameter und Gesamtprofite von korrespondierenden Lösungen unterscheiden sich immer nur um den Faktor $k+1$. Eine optimale Lösung für $E'$ ist demnach auch eine optimale Lösung für $E$. Da in $E'$ nur Zahlen auftauchen, die Vielfache von $k+1$ sind, ist „vom Profit einer optimalen Lösung additiv um maximal $k$ abweichen" gleichbedeutend mit „einen optimalen Profit erzielen“. Somit ist $A'$ ein Optimierungsalgorithmus für KNAPSACK (der offensichtlich in Polynomialzeit arbeitet) und es folgt $P = NP$. □

Der Beweis war darum so leicht, weil ein Zahlenproblem vorlag und wir alle Zahlparameter einfach mit einer geeigneten Konstante multiplizieren konnten. Wie sieht es aber bei rein kombinatorischen Problemen aus? Die Antwort lautet GENAUSO:

**Bemerkung 6.66** *Falls $P \neq NP$, dann kann es keinen Approximationsalgorithmus $A$ für Independent Set mit einer Gütegarantie von der Form*

$$\exists k \in \mathbb{N}, \forall E: \; P_*(G) - P_A(G) \leq k$$

*geben, wobei $P_*(G)$ die maximale Anzahl paarweise unabhängiger Knoten im Eingabegraphen $G$ bezeichnet und $P_A(G)$ die Mächtigkeit der von Algorithmus $A$ konstruierten unabhängigen Menge in $G$.*

**Beweis** Gehe vor wie bei KNAPSACK, außer dass die Transformation von $G$ in $G'$ die Technik der „kombinatorischen Multiplikation“ verwendet: $G'$ besteht aus $k+1$ disjunkten Kopien von $G$. Die weitere Argumentation kann aufgebaut werden wie bei dem entsprechenden Nachweis für KNAPSACK. □

Da fast alle natürlichen Optimierungsprobleme die „kombinatorische Multiplikation“ der Eingabeinstanz mit einer Konstanten (sprich: die Vervielfältigung der Eingabeinstanz) erlauben, sind „additive Gütegarantien“ im Prinzip nicht erreichbar.

### 6.7.4 Komplexitätsklassen für Optimierungsprobleme

Es bezeichne NPO die Klasse aller Optimierungsprobleme, die durch eine polynomiell verifizierbare Relation und eine polynomiell auswertbare Wertefunktion gegeben sind. APX bezeichne die Probleme aus NPO, die einen Approximationsalgorithmus mit konstanter Güte besitzen (die sogenannten „approximierbaren“ Probleme). PTAS bezeichne die Probleme aus NPO, die ein Approximationsschema besitzen. Schließlich bezeichne FPTAS die Probleme aus NPO, die ein volles Approximationsschema besitzen (wie zum Beispiel KNAPSACK). Es ergibt sich die Hierarchie

$$FPTAS \subseteq PTAS \subseteq APX \subseteq NPO \ ,$$

die unter der Voraussetzung $P \neq NP$ echt ist:

1. „Independent Set“ eingeschränkt auf planare Graphen gehört zu PTAS (ohne Beweis). Da es sich hier nicht um ein Zahlenproblem und daher automatisch um ein stark NP-vollständiges Problem handelt, kann es aber gemäß Folgerung 6.64 kein volles Approximationsschema geben.

2. Probleme wie „Bin Packing“, „Vertex Cover“, und „Metrisches TSP“ besitzen Approximationsalgorithmen mit konstanter Güte, aber kein Approximationsschema (was mit Hilfe des berühmten PCP-Theorems gezeigt werden kann).

3. Probleme wie „Set Cover“, „Graphenfärbung“ und CLIQUE besitzen keine Approximationsalgorithmen mit einer konstanten Güte (was ebenfalls mit dem PCP-Theorem gezeigt werden kann).

Das angesprochene PCP-Theorem und seine Bedeutung für die Theorie der Approximationsalgorithmen werden wir evtl. zu einem späteren Zeitpunkt der Vorlesung besprechen.

Für die Klassen NPO und APX lassen sich mit Hilfe eines geeigneten Reduktionsbegriffes vollständige Probleme (also schwerste Vertreter ihrer Klasse vom Standpunkt der Approximierbarkeit) ausfindig machen. Die uns bekannten Reduktionsbegriffe sind hierfür nicht geeignet (aus Gründen, die wir in den Übungen kurz diskutieren). Man braucht vielmehr eine Art „approximationserhaltende“ Reduktion. Populäre approximationserhaltende Reduktionen sind die sogenannten AP- bzw. L-Reduktionen. Wir werden in den Übungen kurz darauf eingehen.

Ansonsten sei auf die Webseite

`www.nada.kth.se/theory/compendium/`

verwiesen, die von Viggo Kann gepflegt wird. Sie enthält zu einer großen Liste von grundlegenden Optimierungsproblemen die neuesten „guten“ und „schlechten“ Nachrichten (sprich: verbesserte Approximationsalgorithmen bzw. verbesserte Nachweise der inhärenten Nichtapproximierbarkeit). In einigen Fällen konnte die magische Schwelle $k_0$ für die bestmögliche Güte exakt bestimmt werden. Ein paar Informationen, die von dieser Webseite gezogen wurden (und die uns bekannten Optimierungsprobleme betreffen) finden sich in dem heute ausgeteilten Begleitmaterial.

## 6.8 Parametrisierte Komplexität

Es sei daran erinnert, dass ein „vertex cover“ eines Graphen $G = (V, E)$ eine Teilmenge $C \subseteq V$ der Knotenmenge ist, die von jeder Kante $e \in E$ mindestens einen Randknoten enthält. Mit Hilfe der Knoten aus $C$ kann man gewissermaäsen alle Kanten des Graphen kontrollieren. Es sei weiter daran erinnert, dass eine „independent set“ eines Graphen $G = (V, E)$ eine Teilmenge $U \subseteq V$ der Knotenmenge ist, die aus paarweise nicht benachbarten Knoten besteht. Es gilt, dass $C$ ein „vertex cover“ ist gdw $U := V \setminus C$ eine „independent set“ ist. Dies hatten wir in der Vorlesung „Theoretische Informatik“ ausgenutzt, um „Independent Set (IS)“ mit der Reduktionsabbildung

$$(G, k) \mapsto (G, |V| - k)$$

polynomiell auf „Vertex Cover (VC)“ zu reduzieren: es gibt nämlich in $G$ eine „independent set“ der Größe $k$ gdw es in $G$ ein „vertex cover“ der Größe $|V| - k$ gibt. In Bezug auf Entscheidungs- oder Optimierungsalgorithmen sind „Vertex Cover“ und „Independent Set“ zwei „isomorphe“ Probleme. Wir wissen bereits, dass dies in Bezug auf Approximationsalgorithmen falsch ist, da „Vertex Cover“ zur Klasse APX der (mit Güte 2) approximierbaren Probleme gehört, wohingegen „Indpendent Set“ (unter der $P \neq NP$ Voraussetzung) keine Approximationsalgorithmen mit konstanter Güte zulässt. In diesem Abschnitt wird sich ein ähnlicher Gegensatz ergeben: „Vertex Cover“ gehört zur Klasse FPT der sogenannten „Fest-Parameter behandelbaren“ Probleme, wohingegen „Independent Set“ (unter der $P \neq NP$ Voraussetzung) nicht zu FPT gehört. Die entsprechende Aussage gilt auch für das folgende Problem:

**Dominating Set (DS)** Knotenüberwachung durch möglichst wenige Knoten

**Eingabe** Ein Graph $G = (V, E)$ und eine Kostenschranke $k$

**Frage** Kann man mit $k$ Knoten alle Knoten überwachen, d.h., existiert eine Teilmenge $D \subseteq V$ (genannt „dominating set“) der Maximalgröße $k$, so dass jeder Knoten $v \in V$ mindestens einen Nachbarn in $D$ besitzt.

**Kommentar** Dieses Problem findet Anwendungen bei der Überwachung von Rechnern in einem Rechnernetz.

Oberflächlich betrachtet sind die Probleme „Vertex Cover“ und „Dominating Set“ enge Verwandte, wie durch die folgende polynomielle Reduktion unterstrichen wird:

**Lemma 6.67** *$VC \leq_{pol} DS$.*

**Beweis** Es sei $(G, k)$ mit $G = (V, E)$ eine gegebene Eingabeinstanz von VC. Wir verwenden die (effizient berechenbare) Eingabetransformation $(G, k) \mapsto (G', k + n)$, wobei $G'$ aus $G$ durch die in Abbildung 8 dargestellte lokale Ersetzung hervorgeht.

Wie aus der Abbildung ersichtlich hat $G'$ die Knotenmenge $V_0 \cup V_1 \cup V_2 \cup E_0$ (mit der offensichtlichen Bedeutung von $V_0, V_1, V_2, E_0$). Die Kantenmenge von $G'$ ist ebenfalls leicht aus der Abbildung abzulesen. Der Beweis wird abgeschlossen durch die

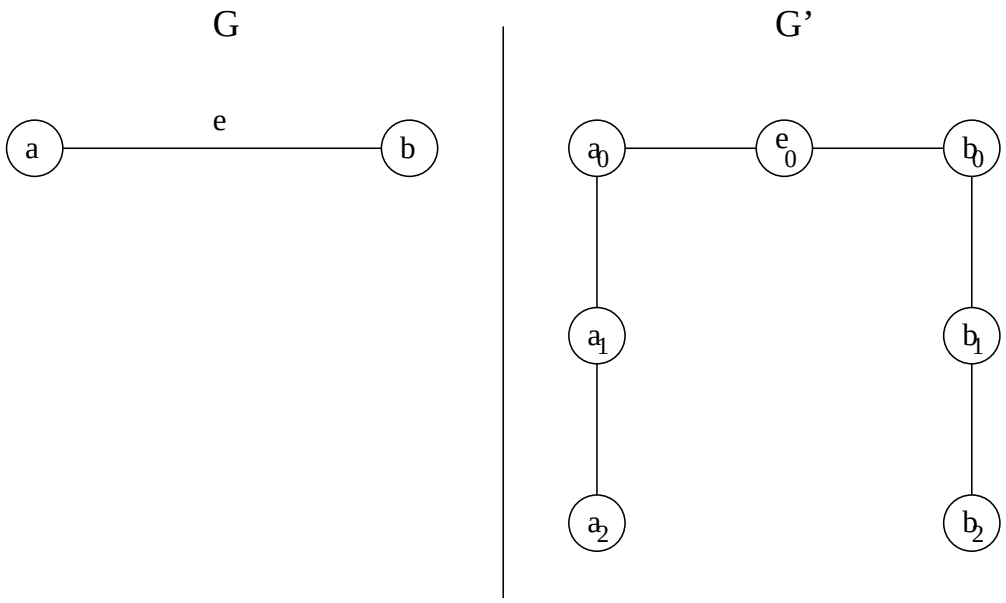



Abbildung 8: Lokale Ersetzung bei der polynomiellen Reduktion von VC auf DS.

**Behauptung** Es gibt genau dann ein Vertex Cover der Maximalgröße $k$ in $G$, wenn es eine Dominating Set der Maximalgröße $k+n$ in $G'$ gibt.

Nehmen wir an, dass wir in $G$ jede Kante mit Knoten aus $U \subseteq V$ überwachen können und $|U| \leq k$. Dann können wir in $G'$ jeden Knoten mit den Knoten aus $U \cup V_1$ überwachen. Es gilt $|U \cup V_1| = |U| + n \leq k + n$.

Nehmen wir an, dass wir in $G'$ jeden Knoten mit Knoten aus $U' \subseteq V_0 \cup V_1 \cup V_2 \cup E_0$ überwachen können und $|U'| \leq k+n$. Wir können oBdA annehmen, dass in $U'$ keine Knoten aufgenommen werden, die in ihrer „Überwachungspotenz“ von anderen Knoten dominiert werden. Zum Beispiel ist $v_1$ „potenter“ als $v_2$, da $v_1$ die Knoten $v_0, v_1, v_2$ überwacht und $v_2$ lediglich die Knoten $v_1, v_2$. Somit gilt oBdA $U' \cap V_2 = \emptyset$. Da andererseits alle Knoten aus $V_2$ überwacht werden müssen und $v_2$ nur von sich selbst oder von $v_1$ aus überwacht werden kann, gilt $V_1 \subseteq U'$. Durch $V_1$ werden bereits alle Knoten aus $V_0$ überwacht, aber noch keine Knoten aus $E_0$. Das Restproblem ist also auf die Überwachung der Knoten aus $E_0$ zusammengeschrumpft. Unter diesen Bedingungen gilt für jede Kante $e = \{a, b\}$, dass $a_0$ (bzw. $b_0$) „mindestens so potent“ ist wie $e_0$. Knoten $a_0$ (bzw. $b_0$) könnte nämlich potentiell neben $e_0$ noch weitere Knoten aus $E_0$ überwachen. Wir können also oBdA annehmen, dass $U'$ die Form $U' = V_1 \cup U_0$ mit $U_0 \subseteq V_0$ hat, wobei die maximal $k$ Knoten aus $U_0$ die Überwachung von $E_0$ gewährleisten. Es folgt, dass $U = \{v |\ v_0 \in U_0\}$ ein Vertex Cover der Maximalgröße $k$ in $G$ ist. □

**Folgerung 6.68** *„Dominating Set“ ist NP-vollständig.*

In Abschnitt 6.8.1 werden wir die Fest-Parameter Behandelbarkeit von „Vertex Cover“ herausarbeiten. In Abschnitt 6.8.2 präsentieren wir eine formale Definition der Klasse FPT

und gehen auf den dazu passenden Reduktionsbegriff ein, mit Hilfe dessen die Nichtmitgliedschaft von „Independent Set“ und „Dominating Set“ in FPT (unter der $P \neq NP$ Voraussetzung) bewiesen werden könnte.

### 6.8.1 Ein Fest-Parameter behandelbares Beispielproblem

Der Star dieses Unterabschnittes ist das Vertex Cover Problem. Wir wollen im folgenden herausarbeiten, dass VC gegenüber CLIQUE, IS oder DS eine Sonderrolle einnimmt.

Beginnen wir mit einer Gemeinsamkeit der genannten Probleme. Wenn wir die Kostenschranke $k$ zu einer Konstanten einfrieren, dann werden Probleme von der Art CLIQUE, IS, VC oder DS trivialisiert. Wir können nämlich mit *Exhaustive Search* alle $\binom{n}{k}$ Teilmengen $U \subseteq V$ der Größe $k$ der Reihe nach durchmustern und jeweils auf die entsprechende Eigenschaft testen. Falls $k$ eine von $n = |V|$ unabhängige Konstante ist, dann gilt $\binom{n}{k} = O(n^k)$ und wir erhalten eine polynomielle Zeitschranke.

Nun ist $n^k$ zwar ein Polynom, aber $\Omega(n^k)$ Rechenschritte sind für große Konstanten $k$ (selbst bei moderater Eingabelänge $n$) nicht tolerierbar. Der Ansatz der parametrisierten Komplexität ist zu überprüfen, ob nicht auch eine Zeitschranke der Form $f(k)n^c$ eingehalten werden kann. Hierbei ist $f$ eine beliebige, nur von $k$ (aber nicht von $n$) abhängige Funktion und $c \geq 1$ ist eine von $n$ und $k$ unabhängige Konstante. Für konstantes $k$ und $c = 1$ hätten wir zum Beispiel eine lineare Laufzeitschranke vorliegen, im Vergleich zu $n^k$ eine radikale Verbesserung.

Für VC geht dieser Plan voll auf:

**Satz 6.69 (Downey und Fellows, 1992)** *VC kann in $O(2^k n)$ Schritten (auf einer Random Access Maschine mit uniformem Kostenmaß) gelöst werden.*

**Beweisskizze** Wir skizzieren nur die Idee, die den Faktor $2^k$ ins Spiel bringt. Von jeder Kante $e = \{v, w\}$ enthält ein legales Vertex Cover den Knoten $v$ oder den Knoten $w$. Wir können daher einen binären Entscheidungsbaum $T$ aufbauen. Jeder Knoten von $T$ repräsentiert ein partielles Vertex Cover, das anfangs (im Wurzelknoten von $T$) noch leer ist. Jede neue binäre Entscheidung führt dazu, dass $T$ sich an einem bisherigen Blattknoten $z$ wieder binär verzweigt und das von $z$ repräsentierte partielle Vertex Cover um einen Knoten erweitert wird. Um das Wachstum von $T$ levelweise voranzutreiben, protokollieren wir an jedem Knoten $z$ die korrespondierende partielle Lösung $U_z$ und die Menge der noch unüberwachten Kanten $E_z$. Wir brechen den Wachstumsprozess von $T$ an einem Blatt $z$ ab, wenn (der Misserfolgsfall) $|U_z| = k, E_z \neq \emptyset$ oder wenn (der Erfolgsfall) $E_z = \emptyset$. $T$ braucht (wegen der Kostenschranke $k$) nicht über Tiefe $k$ hinaus wachsen und enthält daher $O(2^k)$ Knoten. Falls $T$ zu wachsen aufhört, ohne dass der Erfolgsfall eingetreten ist, dann existiert kein Vertex Cover der Größe $k$. Im Erfolgsfall hingegen haben wir ein Vertex Cover der Maximalgröße $k$ aufgespürt. Der Overhead, der an jedem Knoten von $T$ zu leisten ist, ist polynomiell in $n$ beschränkt. Eine verfeinerte Implementierung und eine genauere Analyse zeigen, dass eine Laufzeitschranke der Form $O(2^k n)$ eingehalten werden kann. □

Die in diesem Beweis benutzte Daternstruktur trägt den Namen „beschränkter Suchbaum (bounded search tree)“. Eine weitere Schlüsseltechnik ist die sogenannte „Reduktion auf einen Problemkern“, deren Anwendung auf Vertex Cover zu folgendem Resultat führt:

**Satz 6.70 (Buss, 1989)** *VC kann in $O(k^k + n)$ Schritten (auf einer Random Access Maschine mit uniformem Kostenmaß) gelöst werden.*

**Beweisskizze** Wir skizzieren nur die Idee, die zu der „additiven" Form $f(k) + g(n)$ der Zeitschranke führt. Die Reduktion auf den Problemkern von Vertex Cover mit fester Kostenschranke $k$ nutzt die folgenden offensichtlichen Tatsachen aus:

- Jeder Knoten von einem $k$ überschreitenden Grad muss in das „vertex cover" $C$ aufgenommen werden.
- Bei einem Graphen ohne isolierte Knoten ist jedes „vertex cover" auch eine „dominating set". Wenn jeder Knoten maximal $k$ Nachbarn hat und es gibt ein „vertex cover" (und somit eine „dominating set") der Größe $k'$, dann kann es maximal $k'(k+1)$ Knoten geben.

Dies legt folgenden Algorithmus nahe:

1. Teste, ob es mehr als $k$ Knoten mit jeweils mehr als $k$ Nachbarn gibt. Falls JA, dann kann es kein „vertex cover" der Größe $k$ geben. Falls NEIN, dann nimm die $p \leq k$ Knoten mit jeweils mehr als $k$ Nachbarn in das (anfangs leere) „vertex cover" $C$ auf, reduziere die Kostenschranke auf $k' := k - p$, entferne aus $G$ alle Knoten aus $C$ und die zu ihnen inzidenten Kanten sowie alle resultierenden isolierten Knoten (die zu einer Überwachung der Kanten des Restgraphen nichts beitragen können) und erhalte so den „Restgraphen" $G'$.
2. Teste, ob $G'$ mehr als $k'(k+1)$ Knoten enthält. Falls JA, dann kann es in $G'$ kein „vertex cover" der Größe $k'$ (und somit in $G$ kein „vertex cover" der Größe $k$) geben. Falls NEIN, dann mache weiter wie folgt.
3. Teste, ob $G'$ (ein Graph mit maximal $k'(k+1)$ Knoten) ein „vertex cover" $C'$ der Größe $k'$ besitzt. Falls JA, dann gib $C \cup C'$ als „vertex cover" der Größe $k$ von $G$ aus. Falls NEIN, dann existiert kein „vertex cover" der geforderten Größe.

Bei geschickter Implementierung kann die Reduktion auf den Problemkern (hier: die Berechnung des Restgraphen $G'$) in Linearzeit geschehen. Die Berechnung eines „vertec cover" einer Größe von maximal $k'$ (sofern vorhanden) kann bei geschickter Implementierung in $O(k^k)$ Schritten geschehen. Dies führt zu der Zeitschranke $O(k^k + n)$. □

**Folgerung 6.71 (Balasubramanian, Downey, and Fellows, 1992)** *VC kann in $O(2^k k^2 + n)$ Schritten (auf einer Random Access Maschine mit uniformem Kostenmaß) gelöst werden.*

**Beweis** Reduziere zuerst auf den Problemkern und wende auf diesen die Methode mit den beschränkten Suchbäumen an. □

Der folgende Satz markiert den aktuellen Weltrekord[36], was die Laufzeit für Vertex Cover in Abhängigkeit von $n$ und $k$ angeht:

[36] resultierend aus einer geschickten Kombination verschiedener Vertex Cover Algorithmen

**Satz 6.72** *— ohne Beweis —*
*VC kann in*

$$O\left(kn + \max\left\{(1.25542)^k k^2, (1.2906)^k 2.5k\right\}\right)$$

*Schritten (auf einer Random Access Maschine mit uniformem Kostenmaß) gelöst werden.*

Experimente zeigen, dass dieser Algorithmus Eingaben $(G, k)$ bis zur Schwelle $k \approx 200$ (Parameter $n$ ist weniger kritisch und darf viel größer sein) mit einem noch realistischen Verbrauch an Rechenzeit abarbeiten kann. Eingaben mit derart hohen Werten von $k$ könnten bei einem naiven Verfahren mit Laufzeit $\Omega(n^k)$ in Jahrmilliarden nicht vollständig bearbeitet werden.

Für welche Probleme außerhalb von $P$ lässt sich ein Eingabeparameter $k$ dingfest machen, so dass Lösungsalgorithmen mit einer Zeitschranke der Form $f(k)n^c$ existieren? Für welche Probleme existieren solche Eingabeparameter $k$ nicht? Um diese Frage korrekt zu beantworten bedarf es einiger abstrakter Definitionen, die den Kern der Fest-Parameter Behandelbarkeit herausarbeiten.

### 6.8.2 Probleme innerhalb und außerhalb FPT

**Definition 6.73** *Ein* parametrisiertes Problem *(sprich: eine* parametrisierte formale Sprache*) liegt vor, wenn die Eingabeparameter in einen* Variablen-Parameter Teil *und einen* Fest-Parameter Teil *zerfallen. Syntaktisch korrekte Eingaben sind also Paare $(x, p)$ — genauer: Kodierungen solcher Paare — wobei $x$ die Kollektion der Variablen-Parameter und $p$ die Kollektion der Fest-Parameter repräsentiert.*

**Definition 6.74** *Eine parametrisierte Sprache heisst* Fest-Parameter behandelbar (fixed-parameter tractable)*, wenn ihr Mitgliedschaftsproblem in $f(k)n^c$ Schritten lösbar ist. Hierbei ist $n$ die Länge von $x$, $k$ die Länge von $p$, $f$ eine nur von $k$ (aber nicht von $n$) abhängige Funktion und $c \geq 1$ eine von $n$ und $k$ unabhängige Konstante. Die Klasse der Fest-Parameter behandelbaren parametrisierten Sprachen bezeichnen wir als FPT.*

Aus der Definition von FPT folgt sofort:

$$P \subseteq \text{FPT} \tag{28}$$

**Bemerkung 6.75** *Die Definition von FTP ist robust gegen einige Modifikationen. Zum Beispiel ändert sich die Klasse FPT nicht, wenn wir $p$ und $k$ identifizieren (also etwa voraussetzen, dass $p$ die unäre Kodierung einer Zahl $k$ ist). Von dieser Vereinfachung machen wir meistens Gebrauch und sprechen von dem Fest-Parameter $k$ der Eingabe. Weiterhin ändert sich FPT nicht, selbst wenn wir Laufzeitschranken der Form $f(k) + n^c$ (anstelle $f(k) \cdot n^c$) fordern (wie 1997 von Cai, Chen, Downey und Fellows gezeigt wurde).*

Mit diesen Definitionen können wir das Resultat des vorigen Abschnitts zusammenfassen wie folgt:

**Satz 6.76** *$VC \in FPT$.*

Obwohl wir im Rahmen dieser Vorlesung nicht näher darauf eingehen können, sei bemerkt, dass FPT eine sehr reichhaltige Klasse ist, und dass die Nachweise zur Mitgliedschaft in FPT einige grundlegende Methoden und Tricks zum Algorithmenentwurf hervorgebracht haben. Andererseits gibt es auch eine breite Palette von Problemen, die (vermutlich) nicht zu FPT gehören. In diese Kategorie der *Fest-Parameter harten* Probleme gehören zum Beispiel CLIQUE, IS und DS. Die Vermutung der Fest-Parameter Härte eines Problems wird wieder mit einem geeigneten Reduktionsbegriff gestützt. Anders als bei der Klasse *NPC* haben sich aber mehrere harte Klassen gebildet, die oberhalb von FPT eine Hierarchie bilden. Auch hierauf können wir im Rahmen dieser Vorlesung nicht näher eingehen. Wir definieren lediglich den Reduktionstypus, den man in der Fest-Parameter Komplexitätstheorie zugrunde legen sollte:

**Definition 6.77** *Seien $L, L'$ parametrisierte formale Sprachen. $L$ heisst* Fest-Parameter transformierbar *in $L'$, falls eine in $f(k)n^c$ Schritten berechenbare* Fest-Parameter Eingabetransformation $(x,k) \mapsto (x',k')$ *mit $k' = g(k)$ und $(x,k) \in L \Leftrightarrow (x',k') \in L'$ existiert. Hierbei ist $n$ die Länge von $x$, $f, g$ sind nur von $k$ (aber nicht von $n$) abhängige Funktionen und $c \geq 1$ ist eine von $n$ und $k$ unabhängige Konstante.*

Zum Beispiel ist die (aus der Vorlesung „Theoretische Informatik“ bekannte) polynomielle Reduktion von CLIQUE nach IS (Übergang zum Graphen mit der gleichen Knoten- aber komplementären Kantenmenge) eine Fest-Parameter Transformation (mit $g(k) = k' = k$). Die polynomiellen Reduktionen von IS nach VC (mit $k' = n - k$) und von VC nach DS (mit $k' = n + k$) hingegen sind keine Fest-Parameter Transformationen, da Fest-Parameter $k'$ jeweils eine (verbotene) Abhängigkeit von $n$ aufweist.

Offensichtlich gilt:

**Lemma 6.78** *Falls $L$ Fest-Parameter transformierbar in $L'$ ist und $L' \in FPT$, dann folgt $L \in FPT$.*

Umgekehrt bedeutet dies natürlich, dass sich eine etwaige Fest-Parameter Härte von $L$ auf $L'$ vererben würde.

Wir bemerken abschließend, dass die Hierarchie die von FPT und den diversen Klassen von Fest-Parameter harten Problemen gebildet wird „windschief“ zur polynomiellen Platz-Zeit Hierarchie liegt. Probleme aus *NP* können bereits Fest-Parameter hart sein. Andererseits gehören sogar einige PSPACE-vollständige Probleme zur Klasse FPT.

## 6.9 Zwischenbilanz zum Thema „P versus NP“

Bei großen Softwareprojekten entstehen nach einer Strukturierungs- und Modellierungsphase meist klar umrissene Teilprobleme. Ein Lernziel beim Studium der *Effizienten Algorithmen* und der *Komplexitätstheorie* ist

- zu erkennen, wenn ein Teilproblem mit einem effizienten Algorithmus lösbar ist,
- zu erkennen, wenn dies nicht der Fall ist (zum Beispiel aufgrund der *NP*-Härte des Problems).

*NP*-harte Probleme verschwinden nicht, wenn wir sie als *NP*-hart erkannt haben. Es stellt sich also weiterhin die Frage, wie mit harten Problemen umzugehen ist. In den voran gegangenen Abschnitten haben wir ein paar Methoden zum Umgang mit NP-harten Problemen (sowie die theoretischen Grenzen dieser Methoden) kennen gelernt:

**Einschränkung des Problems** Suche nach (praktisch relevanten) und effizient lösbaren Teilproblemen eines NP-vollständigen Problems.
Zum Beispiel sind 2-SAT (im Gegensatz zu 3-SAT) und 2-dimensionales Matching (im Gegensatz zum 3-dimensionalen Matching) effizient lösbar. PRECEDENCE CONSTRAINED SCHEDULING wird effizient lösbar, wenn wir nur Bäume (Sortier- oder Montagebäume) als partielle Relation auf der Menge der Aufgaben zulassen. Manche Zahlenprobleme (wie PARTITION oder KNAPSACK) sind pseudopolynomiell lösbar und somit polynomiell lösbar im Falle „kleiner" Zahlparameter in der Eingabe. Die Grenzen dieses Ansatzes ergeben sich durch extrem eingeschränkte aber immer noch NP-harte Teilprobleme bzw., im Falle der Zahlprobleme, durch das Phänomen der starken NP-Vollständigkeit.

**Aufgabe des Anspruches der Optimalität** Versuche ein NP-hartes Optimierungsproblem mit einem Approximationsalgorithmus zu lösen, welcher in Polynomialzeit eine „gute" (aber i.A. nicht optimale) Lösung liefert.
Die Grenzen dieses Ansatzes (den wir am Beipsiel der Probleme TSP und KNAPSACK diskutiert haben) ergeben sich durch das Phänomen der NP-harten Approximation.

**Fest-Parameter Behandelbarkeit** Versuche aus den Eingabeparametern einen sogenannten „Fest-Parameter" $k$ zu isolieren, der für die NP-Härte des Problems verantwortlich ist und suche einen Algorithmus mit einer Zeitschranke der Form $f(k)n^c$.
Diesen Ansatz haben wir am Beispiel von Vertex Cover diskutiert. Seine Grenzen findet er im Phänomen der Fest-Parameter Härte.

Ein anderer (bisher von uns nicht diskutierter) Ansatz besteht darin ein mächtigeres Maschinenmodell als die Deterministische Turing Maschine (DTM) zu verwenden. Wir diskutieren ein paar konkrete Vorschläge:

**Probabilistisches Maschinenmodell** Statte $M$ mit einer *perfekten Münze* aus. $M$ hat pro Schritt zwei Handlungsalternativen und wählt durch Münzwurf eine davon zufällig aus. Dieser Ansatz führt zum Modell der *Probabilistischen Turing Maschine (PTM).* Um das Mitgliedschaftsproblem für Sprache $L$ zu lösen, muss eine PTM $M$ Eingaben $x \in L$ mit "hinreichend hoher" Wahrscheinlichkeit akzeptieren und Eingaben $x \notin L$ mit "hinreichend hoher" Wahrscheinlichkeit verwerfen. Wir gehen in einem späteren Stadium der Vorlesung noch genauer auf PTMs und die entsprechenden Komplexitätsklassen ein.

**Parallelrechnermodell** Verwende $p$ parallel arbeitende Prozessoren, die miteinander kommunizieren können, um kooperativ ein gemeinsames Problem zu lösen. In diese Rubrik fallen neben der *Parallel Random Access Machine (PRAM)* und den diversen *Rechnernetzwerken* auch die sogenannten (in der letzten Dekade in Mode gekommenen) *DNA-Rechner*. Ein *NP*-hartes Problem lässt sich allerdings auch von einem Parallelrechner

nicht zufriedenstellend lösen, da jeder Faktor an Zeitgewinn mit einem mindestens ebenso großen Faktor an zusätzlichem Hardwareaufwand bezahlt wird. Wir können im Rahmen dieser Vorlesung auf Parallelrechner nicht näher eingehen.

**Qauntenrechner** Die DTM und dazu verwandte Modelle sind mathematische Abstraktionen von physikalischen Rechnern, die den Gesetzen der klassischen Mechanik gehorchen. In der letzten Dekade wurde eine neue Generation von Rechnern vorgeschlagen, die den Gesetzen der Quantenmechanik gehorchen. Ein Quantenrechner ist zu einem bestimmten Zeitpunkt nicht in **einer** Konfiguration, sondern in einer **Überlagerung von vielen** Konfigurationen. Erst durch externe Beobachtung kollabiert die Überlagerung zu einer einzigen Konfiguration (gemäß einer zur Überlagerung assoziierten Wahrscheinlichkeitsverteilung). Quantenrechner sind vermutlich wesentlich mächtiger als DTMs. Zum Beispiel konnte Peter Shor nachweisen, dass das Faktorisierungsproblem (Berechnung der Primfaktorzerlegung einer gegebenen ganzen Zahl) und das Diskrete-Logarithmus-Problem auf Quantenrechnern effizient lösbar sind. Auf der vermeintlichen Härte dieser Probleme beruhen die meisten aktuell verwendeten asymmetrischen Kryptosysteme. Es ist zur Zeit noch völlig unklar, ob Quantenrechner physikalisch realisiert werden können. Im Rahmen dieser Vorlesung werden wir auf Quantenrechner nicht weiter eingehen können.

Ein weiterer Ansatz, den wir im Rahmen dieser Vorlesung nicht behandeln können ist die

**Average-Case Analyse** Average-case Analyse steht im Gegensatz zur worst-case Analyse. Hier wird die Forderung aufgegeben, auf allen Eingaben erfolgreich sein zu müssen. Man verwendet ein Wahrscheinlichkeitsmaß auf den Eingabeinstanzen und ist zum Beispiel zufrieden, wenn die mittlere Laufzeit polynomiell ist. Ein Argument **für average-case Analyse** (oder zumindest gegen worst-case Analyse) ist, dass die von polynomiellen Reduktionen produzierten Eingabeinstanzen i.A. kunstvoll und bizarr anmuten. Es könnte also sein, dass genau der Eingabetypus, welcher die *NP*-Härte bezeugt, in Anwendungen nicht (oder selten) anzutreffen ist. Ein Argument **gegen average-case Analyse** ist, dass die Auswahl eines analysierbaren Wahrscheinlichkeitsmaßes meist willkürlich und durch die Anwendung nicht zu rechtfertigen ist.

Schließlich gibt es den sogenannten „heuristischen Ansatz“ zum Lösen von NP-harten Optimierungsproblemen.

**Heuristiken und Metaheuristiken** Eine *Heuristik* ist ein Algorithmus, der in der Praxis ganz gut zu funktionieren scheint, ohne dass dies durch eine mathematisch beweisbare Qualitätsgarantie abgesichert wäre. *Metaheuristiken* sind vielseitig verwendbare Heuristiken, die durch geeignete Adjustierung einiger programmierbarer Parameter auf die Lösung von konkreten Problemen spezialisiert werden können. Daneben gibt es ad-hoc Heuristiken, die für ein konkretes Problem zusammengeschustert wurden. Beispiele für Metaheuristiken sind:

- Branch and Bound

- Lokale Optimierung
- Simulated Annealing
- Tabu Search
- Neuronale Netze oder Hopfield Netze
- Genetische Algorithmen
- Great Deluge (große Überschwemmung)
- Roaming Ants (umherstreifende Ameisen)

Auch diesen Ansatz werden wir aus Zeitgründen nicht weiter verfolgen können.

## 6.10 Die Struktur von NP

Falls $P = NP$, dann gibt es zur Struktur von $NP$ nicht viel zu berichten. Alle Betrachtungen dieses Abschnittes basieren auf der

**Vermutung 1** $P \neq NP$.

Wir verwenden im Folgenden Vermutung 1 als Voraussetzung, ohne dies jedesmal explizt kennntlich zu machen. Wir machen also öfter eine Aussage $A$, die streng genommen die Form: $P \neq NP \Rightarrow A$ haben müsste.

Es sei daran erinnert, dass $NPC$ die Klasse der $NP$-vollständigen Probleme bezeichnet. Die Klasse

$$NPI := NP \setminus (P \cup NPC)$$

bezeichne die Klasse der Sprachen aus $NP$, die einerseits nicht zu $P$ gehören, andererseits aber auch nicht $NP$-vollständig sind.

Aus der $NP$-Vollständigkeitstheorie folgt direkt, dass die Existenz einer $NP$-vollständigen Sprache in $P$ die Gleichheit von $P$ und $NP$ zur Folge hätte. Vermutung 1 zugrunde gelegt, müssen also $NPC$ und $P$ disjunkte Teilmengen von $NP$ sein. Mit der Definition von $NPI$ ergibt sich eine erste Strukturaussage:

- $NP$ zerfällt in die paarweise disjunkten Klassen $P$, $NPC$ und $NPI$.

Diese Einsicht ist in Abbildung 9 visualisiert.

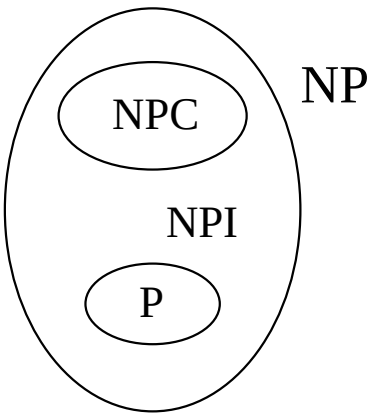



Abbildung 9: Die Struktur von $NP$.

Wir kennen bereits zahlreiche Sprachen (Probleme) in $P$ bzw. $NPC$. Wie schaut es aber mit $NPI$ aus: könnte es sein, dass $NPI$ leer ist? Dies würde bedeuten, dass jede Sprache aus $NP \setminus P$ bereits NP-vollständig sein müsste. Der folgende Satz schließt diese Möglichkeit aus:

**Satz 6.79 (Ladner, 1975)** $NPI \neq \emptyset$.

**Beweis** Es bezeichne

$$M_1, M_2, \ldots$$

eine Aufzählung aller DTMs, die mit einer polynomiellen Zeitschranke als Sprachakzeptor arbeiten, so dass

$$P = \bigcup_{i \geq 1} L_{M_i} \ .$$

Weiter bezeichne

$$R_1, R_2, \ldots$$

eine Aufzählung aller DTMs, die mit einer polynomiellen Zeitschranke als Funktionsberechner arbeiten. Weiter bezeichne $S$ eine DTM mit $L_S = SAT$ (und einer exponentiellen Zeitschranke). Beachte, dass es zu jeder Sprache $L \in NPC$ eine polynomielle Reduktion von SAT auf $L$ gibt. Somit gibt es einen Index $i$ mit

$$\forall z \in \Sigma^* : z \in SAT \Leftrightarrow R_i(z) \in L \ .$$

Unser Ziel ist es, eine TM $K$ anzugeben mit

$$L_K \in NP \setminus (P \cup NPC) \ .$$

Wir sagen $z \in \Sigma^*$ *trennt* $K$ *von* $M_i$, wenn $K(z) \neq M_i(z)$; $z \in \Sigma^*$ *trennt* $K$ *von* $R_i$, wenn $S(z) \neq K(R_i(z))$. Wenn $K$ (mit Hilfe geeigneter Eingaben) von jeder Maschine $M_i$ und jeder Maschine $R_i$ getrennt werden kann, dann gilt offensichtlich $L_K \in NP \setminus (P \cup NPC)$ und wir sind am Ziel. Im Folgenden bezeichne

$$z_1, z_2, \ldots$$

eine Aufzählung (in natürlicher Reihenfolge) aller Wörter aus $\Sigma^*$.
Auf einer Eingabe $x \in \Sigma^n$ wird $K$ arbeiten wie folgt:

**if** $S(x) = 1$ und $f(n)$ ist gerade **then** K(x):=1 **else** K(x):=0 **fi**

Hierbei haben wir noch nicht geklärt, welche Funktion sich hinter

$$f : \mathbb{N} \to \mathbb{N}$$

verbirgt und, in der Tat, die Definition von $f$ ist das „Herzstück“ des Beweises. Die Grundidee zur Konstruktion einer DTM $F$ für $f$ ist wie folgt:

- $F$ wird auf einer Eingabe $1^n$ (unäre Kodierung von $n$) exakt $2n$ Schritte arbeiten. Die (zugegebenermaßen streng limitierte) Arbeitszeit wird $F$ nutzen, um $K$ Eingaben zu verschaffen, die sie von den Maschinen $R_i$ und $M_i$ trennen (eine Art „doppelte Diagonalisierung“). Dabei wird der Funktionswert $f(n)$ anzeigen, wie weit die doppelte Diagonalisierung bereits voran geschritten ist.

- $f(n) = 2i - 1$ wird bedeuten, dass $F$ bereits gewährleisten kann, dass Eingaben existieren, welche $R_1, \ldots, R_{i-1}$ von $K$ separieren (wohingegen die Separation von $R_i$ noch nicht geklärt ist).
- $f(n) = 2i$ wird bedeuten, dass $F$ bereits gewährleisten kann, dass Eingaben existieren, welche $M_1, \ldots, M_{i-1}$ von $K$ separieren (wohingegen die Separation von $M_i$ noch nicht geklärt ist).

Es folgt eine detailliertere Beschreibung der Maschine $F$, die über ein Eingabe-, ein Ausgabe- und ein Arbeitsband verfügt. Auf Eingabe $1^n$ arbeitet $F$ in zwei Phasen zu je $n$ Schritten. In Phase 1 wandert der Eingabekopf von links nach rechts über die Eingabe hinweg (bis zum Lesen eines Leerzeichens) und in Phase 2 bewegt er sich wieder in die Ausgangsposition (Zelle 1) zurück. Der Hauptzweck dieser Übung ist den Taktgeber für die zwei Phasen zu spielen. Die Hauptarbeit beider Phasen findet (wie könnte es anders sein) auf dem Arbeitsband statt:

**Phase 1** $F$ berechnet (solange die Zeit reicht) $f(1), f(2), \ldots$. Es bezeichne $f(j) = k$ den zuletzt berechneten Wert.[37] Es wird sich zeigen, dass der Ausgabewert $f(n)$ entweder auf $k$ oder auf $k + 1$ gesetzt wird. Die Entscheidung darüber fällt in Phase 2.

**Phase 2** In dieser Phase haben wir zwei Fälle zu unterscheiden.

**Fall 1** $k = 2i - 1$ (eine ungerade Zahl).
$F$ versucht $K$ von $R_i$ zu trennen. Zu diesem Zweck berechnet sie (höchstens solange die Zeit reicht)

$$R_i(z_1), S(R_i(z_1)), f(|R_i(z_1)|), S(z_1), R_i(z_2), S(R_i(z_2)), f(|R_i(z_2)|), S(z_2), \ldots \ .$$

Beachte, dass mit den Größen $R_i(z), S(R_i(z)), f(|R_i(z)|), S(z)$ leicht gecheckt werden kann (s. Übung), ob $S(z) \neq K(R_i(z))$. Wenn unter den durchlaufenen $z_i$ ein $z$ mit $S(z) \neq K(R_i(z))$ gefunden wurde (welches also $K$ von $R_i$ trennt), dann wird $f(n)$ auf $k + 1 = 2i$ gesetzt; andernfalls wird $f(n) := k = 2i - 1$ ausgegeben.

**Fall 2** $k = 2i$ (eine gerade Zahl).
$F$ versucht $K$ von $M_i$ zu trennen. Zu diesem Zweck berechnet sie (höchstens solange die Zeit reicht)

$$M_i(z_1), S(z_1), f(|z_1|), M_i(z_2), S(z_2), f(|z_2|), \ldots \ .$$

Beachte, dass mit den Größen $M_i(z), S(z), f(|z|)$ leicht gecheckt werden kann (s. Übung), ob $K(z) \neq M_i(z)$. Wenn unter den durchlaufenen $z_i$ ein $z$ mit $K(z) \neq M_i(z)$ gefunden wurde (welches also $K$ von $M_i$ trennt), dann wird $f(n)$ auf $k+1 = 2i + 1$ gesetzt; andernfalls wird $f(n)$ auf $k = 2i$ ausgegeben.

Wir behaupten nun als erstes, dass $L_K \in NP$. Dies ist leicht einzusehen, da die Bedingung $S(x) = 1$ (also $x \in SAT$) durch Raten einer erfüllenden Belegung der durch $x$ kodierten CNF-Formel in Polynomialzeit verifiziert werden kann. Die Zusatzbedingung der Geradzahligkeit

[37] Wenn die Zeit von $n$ Schritten noch nicht einmal zur Berechnung von $f(1)$ reicht (was zum Beispiel bei $n = 1$ passiert), dann wird der Funktionswert „per default“ auf 1 gesetzt.

von $f(n)$ kann mit Hilfe von $F$ in $2n$ Schritten getestet werden.
Zweitens behaupten wir, dass $L_K \notin P$. Wäre nämlich $L_K \in P$, dann wähle einen mimimalen Index $i$ mit $L_K = L_{M_i}$. Aus der Arbeitsweise von $F$ ergibt sich dann die Folgerung

$$\exists n_0 \in \mathbb{N}, \forall n \geq n_0 : f(n) = 2i \ .$$

Dann aber würde $L_K$ bis auf endlich viele Ausnahmen mit SAT übereinstimmen, was im Widerspruch zur Annahme $L_K \in P$ steht.
Schließlich behaupten wir, dass $L_K \notin NPC$. Wäre nämlich $L_K \in NPC$, dann wähle einen minimalen Index $i$ so dass $S(z) = K(R_i(z))$ für alle $z \in \Sigma_*$. Aus der Arbeitsweise von $F$ ergibt sich dann die Folgerung

$$\exists n_0 \in \mathbb{N}, \forall n \geq n_0 : f(n) = 2i - 1 \ .$$

Dann aber wäre $L_K$ eine endliche Sprache, was im Widerspruch zu $L_K \in NPC$ steht. □

Wir wollen nun die Stuktur von $NP$ differenzierter untersuchen.

**Definition 6.80** *Wir sagen zwei Sprachen $L_1$ und $L_2$ sind* polynomiell äquivalent*, wenn sie wechselseitig aufeinander polynomiell reduzierbar sind. In Zeichen:*

$$L_1 \overset{pol}{\sim} L_2 :\Leftrightarrow L_1 \leq_{pol} L_2 \text{ und } L_2 \leq_{pol} L_1 \ .$$

*Diese Relation ist (offensichtlich) eine Äquivalenzrelation. Eine Äquivalenzklasse bezüglich „$\overset{pol}{\sim}$“ heiße* Schwierigkeitsgrad *bezüglich „$\overset{pol}{\sim}$“ oder kurz* p-Grad.

Wenn wir polynomielle Unterschiede zwischen Laufzeiten ignorieren, können wir zwei polynomiell äquivalente Sprachen als (im Wesentlichen) gleich schwer betrachten. Die folgende Definition liefert eine partielle Ordnung auf den p-Graden;

**Definition 6.81** *Wir sagen $L_1$ ist* leichter *als $L_2$ bezüglich polynomieller Reduktion, wenn zwar $L_1$ polynomiell reduzierbar auf $L_2$ ist, aber nicht umgekehrt. In Zeichen (mit „$\neg$“ für „logische Negation“):*

$$L_1 <_{pol} L_2 :\Leftrightarrow L_1 \leq_{pol} L_2 \text{ und } \neg(L_2 \leq_{pol} L_1) \ .$$

*Wir sagen der p-Grad von $L_1$ ist* leichter *als der p-Grad von $L_2$, wenn $L_1$ leichter als $L_2$ bezüglich polynomieller Reduktion ist.*

Es ist leicht einzusehen, dass $<_{pol}$ auf Sprachen eine irreflexive, asymmetrische partielle Ordnung bildet, die auch auf p-Graden wohldefiniert (also nicht abhängig von der Auswahl des Repräsentanten eines p-Grades) ist. In diesem Lichte besteht $NP$ aus einer Hierarchie von p-Graden. Eine Teilhierarchie (s. Übung) ist in Abbildung 10 zu sehen:

- Die leere Menge und $\Sigma^*$ bilden zwei triviale p-Grade, angesiedelt auf der untersten Stufe der Hierarchie.
- $P \setminus \{\emptyset, \Sigma^*\}$, also die restlichen Sprachen aus $P$, bilden den nächst-einfachsten p-Grad, den wir einfach als $P$-Grad bezeichnen.

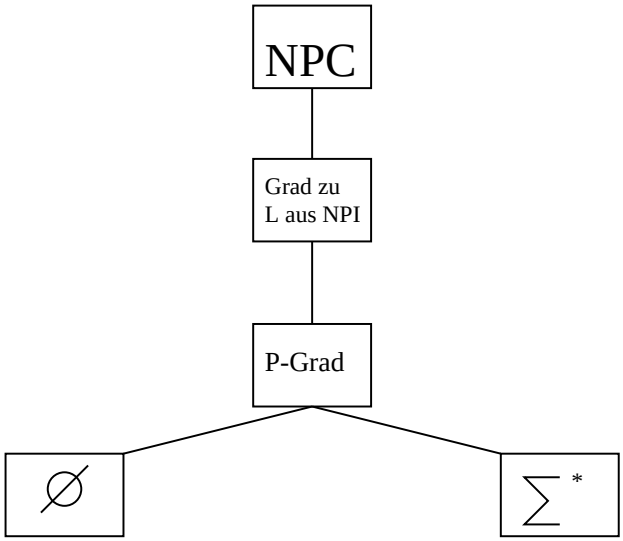



Abbildung 10: Eine Teilhierarchie der p-Grade.

- *NPC*, also die *NP*-vollständigen Probleme, bilden den schwersten p-Grad in *NP*.
- Da *NPI* nicht leer ist, können wir willkürlich eine Sprache $L$ aus *NPI* rausgreifen. Ihr p-Grad muss echt zwischen dem *P*-Grad und *NPC* liegen.

Es stellt sich nun die Frage, ob die Struktur der p-Grade zwischen *P* und *NPC* nicht wesentlich komplexer ist, als dies durch Abbildung 10 suggeriert wird. Dies ist in der Tat der Fall, wie folgende Sätze belegen:

**Satz 6.82 (Ladner, 1975)** *1. Es gibt unendlich lange Ketten von p-Graden zwischen P und NPC.*

*2. Es gibt unvergleichbare p-Grade zwischen P und NPC.*

Die erste Aussage bedeutet, dass es Sprachen $L_0, L_1, L_2, \ldots, L_* \in NP$ gibt, so dass $L_0 \in P$, $L_* \in NPC$ und

$$L_0 <_{pol} L_1 <_{pol} L_2 <_{pol} \cdots <_{pol} L_* \ .$$

Die Sprachen $L_1, L_2, \ldots$ spannen eine unendliche Kette zwischen den p-Graden *P* und *NPC* auf. Die zweite Ausage macht deutlich, dass $<_{pol}$ nur eine partielle, nicht aber totale, Ordnung auf den p-Graden liefert. Der Beweis dieses Satzes, den wir in unserer Vorlesung auslassen, basiert (wie schon der Beweis von Satz 6.79) auf der Diagonalisierungstechnik.

Wir wollen im Folgenden das Bild durch die Betrachtung von co-*NP* abrunden. Es sei an die generelle Definition der *Komplementärklasse* zu einer Sprachklasse $\mathcal{C}$ erinnert:

$$\text{co-}\mathcal{C} := \{\bar{L} : L \in \mathcal{C}\} \ .$$

$\mathcal{C} = \text{co-}\mathcal{C}$ würde gerade bedeuten, dass $\mathcal{C}$ abgeschlossen unter Komplementbildung ist. Wir formulieren jetzt die Vermutung, dass *NP* diese Abschlusseigenschaft nicht besitzt:

**Vermutung 2** $NP \neq \text{co-}NP$.

Die folgende Überlegung zeigt, dass Vermutung 2 stärker ist als Vermutung 1:

**Bemerkung 6.83** *$NP \neq$ co-$NP \Rightarrow P \neq NP$.*

**Beweis** Wir führen einen indirekten Beweis, d.h., wir beweisen (unter Ausnutzung von $P = \text{co-}P$) die Aussage $P = NP \Rightarrow NP = \text{co-}NP$:

$$P = NP \Rightarrow \text{co-}NP = \text{co-}P = P = NP \ .$$

□

co-$NP$ ist eine zu $NP$ duale Klasse. Strukturen in $NP$ haben ihr duales Pendant in co-$NP$. Eine erste Illustration dieses Sachverhaltes liefert

**Lemma 6.84** *co-$NPC$, also die Komplementärklasse der $NP$-vollständigen Sprachen, stimmt überein mit der Klasse co-$NP$-vollständigen Sprachen.*

**Beweis** Wir nutzen die Implikation $L_1 \leq_{pol} L_2 \Rightarrow \bar{L}_1 \leq_{pol} \bar{L}_2$ aus. Somit sind die folgenden Aussagen äquivalent:

$$\begin{aligned}
L_0 \in \text{co-}NPC &\Leftrightarrow \bar{L}_0 \in NPC \\
&\Leftrightarrow (\bar{L}_0 \in NP) \wedge (\forall L \in NP : L \leq_{pol} \bar{L}_0) \\
&\Leftrightarrow (L_0 \in \text{co-}NP) \wedge (\forall L \in NP : \bar{L} \leq_{pol} L_0) \\
&\Leftrightarrow (L_0 \in \text{co-}NP) \wedge (\forall L \in \text{co-}NP : L \leq_{pol} L_0) \\
&\Leftrightarrow L_0 \text{ ist co-}NP\text{-vollständig.}
\end{aligned}$$

□

Wir merken kurz an, dass Lemma 6.84 allgemeiner hätte formuliert werden können: an der Stelle der polynomiellen Reduuktion „$\leq_{pol}$“ könnte eine beliebige abstrakte polynomielle Reduktion „$\leq_{POL}$“ stehen, welche zusätzlich die Bedingung

$$L_1 \leq_{POL} L_2 \Rightarrow \bar{L}_1 \leq_{POL} \bar{L}_2$$

erfüllt. Cook- oder Levin-Reduktionen bezitzen diese Eigenschaft ebenfalls (s. Übungen). Somit ergibt sich die

**Folgerung 6.85** *Die Komplementärklasse der unter Cook- bzw. Levin-Reduktionen $NP$-vollständigen Sprachen stimmt überein mit der Klasse unter Cook- bzw. Levin-Reduktionen co-$NPC$-vollständigen Sprachen.*

Das folgende Lemma gibt ein paar zur $NP \neq$ co-$NP$-Voraussetzung äquivalente Bedingungen an.

**Lemma 6.86** *Folgende Aussagen sind äquivalent:*

1. $NP \neq co\text{-}NP$.
2. $NPC \cap co\text{-}NP = \emptyset$.
3. $co\text{-}NPC \cap NP = \emptyset$.
4. $NP \cap co\text{-}NP \subseteq NPI \cup P$.

**Beweis** Zur Äquivalenz der ersten beiden Aussagen genügt es freilich

$$NP = \text{co-}NP \Leftrightarrow NPC \cap \text{co-}NP \neq \emptyset$$

zu zeigen. $NP = \text{co-}NP \Rightarrow NPC \cap \text{co-}NP \neq \emptyset$ erhalten wir wie folgt:

$$NP = \text{co-}NP \Rightarrow NPC = \text{co-}NPC \Rightarrow NPC \cap \text{co-}NP = NPC \neq \emptyset \ .$$

Nehmen wir umgekehrt an, dass $NPC \cap \text{co-}NP \neq \emptyset$. Dann gibt es eine $NP$-vollständige Sprache $L_0$, die auch zu co-$NP$ gehört. Da jede Sprache $L \in NP$ auf $L_0$ polynomiell reduzierbar ist, ergibt sich hieraus $NP \subseteq \text{co-}NP$. Dies wiederum impliziert $\text{co-}NP \subseteq \text{co-co-}NP = NP$ und wir erhalten $NP = \text{co-}NP$.
Zur Äquivalenz von Aussage 2 und 3 genügt es freilich

$$NPC \cap \text{co-}NP \neq \emptyset \Leftrightarrow \text{co-}NPC \cap NP \neq \emptyset$$

zu zeigen. Hierzu nutze aus, dass für alle Sprachen $L$ die Aussagen $L \in NPC \cap \text{co-}NP$ und $\bar{L} \in \text{co-}NPC \cap NP$ äquivalent sind.
Die Äquivalenz der Aussagen 2 und 4 ist offensichtlich. □

Die durch Lemma 6.86 beschriebene Situation ist in Abbildung 11 noch einmal zusammengefasst.

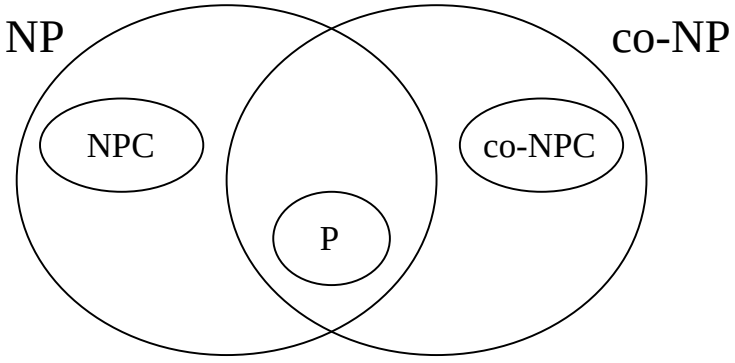



Abbildung 11: Struktur von $NP \cup \text{co-}NP$ unter der $NP \neq \text{co-}NP$-Voraussetzung.

## 6.11 Isomorphe, dünne und dichte Sprachen

In diesem Abschnitt legen wir die Vermutung nahe, dass alle NP-vollständigen Sprachen zueinander „isomorph“ sind. Dies würde ausschließen, dass es „dünne“ NP-vollständige Sprachen (mit poly($n$) Wörtern einer Maximallänge $n$) gibt.

Die Theorie der isomorphen, dünnen und dichten Sprachen wurde 1977 von Berman und Hartmanis ins Leben gerufen. Da sie die „P versus NP“ Frage aus einer neuen Perspektive beleuchtet, präsentieren wir im Folgenden ihre zentralen Konzepte und Resultate.

**Definition 6.87** *Zwei Sprachen $L_1, L_2 \in \Sigma^*$ heißen* polynomiell isomorph*, wenn eine bijektive Abbildung $h : \Sigma^* \to \Sigma^*$ mit Umkehrabbildung $h^{-1} : \Sigma^* \to \Sigma^*$ existiert, die folgende Bedingungen erfüllt:*

1. *Sowohl $h$ als auch $h^{-1}$ sind in Polynomialzeit berechenbar.*
2. *Für alle $x \in \Sigma^*$ gilt $x \in L_1$ gdw $h(x) \in L_2$.*

*In Zeichen: $L_1 \overset{pol}{\simeq} L_2$.*

Offensichtlich gilt

$$L_1 \overset{pol}{\simeq} L_2 \Rightarrow L_1 \overset{pol}{\sim} L_2 \ ,$$

d.h., zwei polynomiell isomorphe Sprachen sind erst recht polynomiell äquivalent.

Wie wir wissen sind alle NP-vollständigen Sprachen wechselseitig aufeinander polynomiell reduzierbar und somit polynomiell äquivalent. Die polynomiellen Reduktionen, die wir dabei üblicherweise benutzen, sind aber weit davon entfernt, Bijektionen zu sein. Insbesondere die Surjektivität ist selten erfüllt, da wir bei einer Reduktion eines Quellproblems auf ein Zielproblem in der Regel extrem spezielle Eingabeinstanzen des Zielproblems konstruieren. Eine der seltenen Ausnahmen bildet die Reduktion $(G, k) \mapsto (\bar{G}, k)$ von CLIQUE auf IS bzw. die Reduktion $(G, k) \mapsto (G, n-k)$ von IS auf VC, wobei $\bar{G}$ den Komplementärgraphen[38] zu $G$ bezeichnet und $n$ die Anzahl der Knoten in $G$.

Wir skizzieren im Folgenden eine Technik zur Verwandlung einer polynomiellen Reduktion in eine polynomielle Isomorphie. Diese Technik ist immer dann anwendbar, wenn die Zielsprache der polynomiellen Reduktion über eine sogenannte „Polsterfunktion (padding function“) verfügt:

**Definition 6.88** *Eine Funktion $pad : \Sigma^* \times \Sigma^* \to \Sigma^*$ heißt* Polsterfunktion *für $L \subseteq \Sigma^*$, wenn die folgenden Bedingungen erfüllt sind:*

1. *Die Funktion $pad$ ist in Polynomailzeit berechenbar.*
2. *Für alle $x, y \in \Sigma^*$ gilt $x \in L$ gdw $pad(x, y) \in L$.*
3. *Für alle $x, y \in \Sigma^*$ gilt $|pad(x, y)| > |x| + |y|$.*

[38] Zwei Knoten sind in $\bar{G}$ benachbart gdw sie in $G$ nicht benachbart sind.

4. *Aus $pad(x, y)$ kann $y$ in Polynomialzeit extrahiert werden.*

Eine Polsterfunktion für $L$ ist also im Wesentlichen eine Längen-expandierende polynomielle Reduktion von $L$ auf sich selbst, die (auf leicht dekodierbare Weise) einen zusätzlichen String $y$ in den Eingabestring $x$ hinein kodiert. Wir verwenden im Folgenden oBdA das Binäralphabet $\Sigma = \{0, 1\}$.

**Beispiel 6.89** *Wir skizzieren den Entwurf einer Polsterfunktion für SAT. Es bezeichne $x \in \Sigma^*$ einen String zur natürlichen Kodierung[39] einer Eingabeinstanz von SAT, d.h., $x$ repräsentiert eine Boolesche Formel in konjunktiver Normalform mit $m$ Klauseln $C_0, \ldots, C_{m-1}$, in denen $n$ Variable, sagen wir $v_1, \ldots, v_n$, verwendet werden. Für $p = |y|$ bezeichne dann $pad(x, y)$ den Kodierungsstring für die Klauseln $C_0, \ldots, C_{m-1}$ (alte Klauseln) sowie $C_m, \ldots, C_{m+p+1}$ (neue Klauseln). Die neuen Klauseln verwenden neue Variable $v_{n+1}, \ldots, v_{n+1+p}$ und haben die folgende Form:*

- *$C_m$ bestehe nur aus dem Literal $v_{n+1}$ und $C_{m+1}$ sei ein Duplikat von $C_m$.*
- *Für $i = 1, \ldots, p$ bestehe $C_{m+1+i}$ nur aus dem Literal $v_{n+1+i}$ sofern $y_i = 1$ bzw. nur aus dem Literal $\bar{v}_{n+1+i}$ sofern $y_i = 0$.*

*Die hierdurch beschriebene Funktion pad ist in Polynomialzeit berechenbar. Offensichtlich sind die Klauseln $C_0, \ldots, C_{m-1}$ erfüllbar gdw $C_0, \ldots, C_{m-1}, C_m, \ldots, C_{m+1+p}$ erfüllbar sind (da die neu hinzu gefügten Klauseln erstens erfüllbar sind und zweitens nur neue Variable verwenden). Die neue Instanz $pad(x, y)$ hat offensichtlich eine $|x|+|y|$ überschreitende Länge. Weiterhin lässt sich $y$ aus $pad(x, y)$ leicht ablesen: durchmustere $pad(x, y)$ von rechts nach links bis zum ersten Auffinden von zwei identischen Klauseln ($C_m$ und $C_{m+1}$); lies aus den davon rechts befindlichen Klauseln den Binärstring $y$ (auf die offensichtliche Weise) ab. Somit sind alle an eine Polsterfunktion gerichteten Bedingungen erfüllt.*

Polsterfunktionen lassen sich für NP-vollständige Sprachen in der Regel ähnlich leicht angeben wie für SAT. Wir diskutieren in den Übungen weitere Beispiele.

Das folgende Resultat ergibt sich unmittelbar aus den Eigenschaften von Polsterfunktionen und Reduktionsabbildungen.

**Lemma 6.90** *Es sei pad eine Polsterfunktion für $L_2$ und $L_1 \leq_{pol} L_2$ mit Reduktionsabbildung $R : \Sigma^* \to \Sigma^*$. Dann ist*

$$x \mapsto pad(R(x), x)$$

*eine Längen-expandierende, injektive und in Polynomialzeit invertierbare Reduktionsabbildung.*

Wechselseitige Längen-expandierende, injektive und in Polynomialzeit invertierbare Reduktionsabbildungen können in Isomorphismen transformiert werden:

[39] Hier und im Folgenden legen wir die Details einer „natürlichen Kodierung“ nicht fest. Es ist aber prinzipiell leicht (wenngleich mühselig), solche Kodierungen über einem Alphabet mit mindestens zwei Zeichen fsetzulegen.

**Lemma 6.91 (Berman und Hartmanis, 1977)** *Es seien $L_1$ und $L_2$ polynomiell äquivalente Sprachen, $R$ eine Reduktionsabbildung für $L_1 \leq_{pol} L_2$ und $S$ eine Reduktionsabbildung für $L_2 \leq_{pol} L_1$. Sowohl $R$ als auch $S$ seien Längen-expandierend, injektiv und in Polynomialzeit invertierbar. Dann sind $L_1$ und $L_2$ polynomiell isomorph.*

**Beweis** Da $R, S : \Sigma^* \to \Sigma^*$ Längen-expandierend, injektiv und in Polynomialzeit invertierbare Abbildungen sind, sind die Inversen $R^{-1}, S^{-1} : \Sigma^* \to \Sigma^*$ Längen-reduzierende, partiell definierte und in Polynomialzeit berechenbare Abbildungen. Der Beweis beruht auf dem Konzept der $R$- und $S$-Ketten. Eine *$S$-Kette* für $x \in \Sigma^*$ hat die Form

$$\cdots R^{-1}(S^{-1}(R^{-1}(S^{-1}(x)))) \ .$$

Dies ist so zu verstehen, dass wir die Abbildungen $S^{-1}$ und $R^{-1}$ solange alternierend anwenden wie das jeweilige Argument im Definitionsbereich liegt. Da $R^{-1}$ und $S^{-1}$ beide Längen-reduzierend sind, muss die Kette nach maximal $n = |x|$ Schritten abbrechen. Wir unterscheiden die folgenden beiden Fälle:

**Fall 1** Die $S$-Kette für $x$ stoppt mit undefiniertem $R^{-1}$.
In diesem Fall hat die $S$-Kette die Form $S^{-1}(R^{-1}(\cdots S^{-1}(x) \cdots))$.

**Fall 2** Die $S$-Kette für $x$ stoppt mit undefiniertem $S^{-1}$.
In diesem Fall hat die $S$-Kette die Form $R^{-1}(S^{-1}(\cdots S^{-1}(x) \cdots))$.

Beachte, dass der Grenzfall, dass bereits $S^{-1}(x)$ undefiniert ist, zur $S$-Kette $x$ führt und einen Unterfall zu Fall 2 darstellt.
Eine *$R$-Kette* für $x \in \Sigma^*$ ist symmetrisch definiert mit vertauschten Rollen von $R$ und $S$.
Wir definieren jetzt eine Abbildung $h : \Sigma^* \to \Sigma^*$, von der wir anschließend nachweisen, dass sie einen Isomorphismus von $L_1$ nach $L_2$ darstellt:

1. Falls die $S$-Kette für $x$ mit undefiniertem $R^{-1}$ stoppt, dann setzen wir $h(x) := S^{-1}(x)$.
2. Falls die $S$-Kette für $x$ mit undefiniertem $S^{-1}$ stoppt, dann setzen wir $h(x) := R(x)$.

Wir weisen zunächst die Injektivität von $h$ nach, indem wir die Annahme es existierten $x, y \in \Sigma^*$ mit $x \neq y$ und $h(x) = h(y)$ zum Widerspruch führen. Wegen der Injektivität von $R$ und $S$ könnte die angenommene Situation höchstens dann eintreten, wenn die $S$-Ketten für $x$ und $y$ sich auf die beiden Fälle verteilen, sagen wir, die $S$-Kette für $x$ stoppt mit undefiniertem $R^{-1}$ und die $S$-Kette für $y$ stoppt mit undefiniertem $S^{-1}$. Hieraus ergibt sich

$$S^{-1}(x) = h(x) = h(y) = R(y)$$

und somit

$$y = R^{-1}(S^{-1}(x)) \ .$$

Dann wäre aber die $S$-Kette für $y$ ein Präfix der $S$-Kette für $x$ und die Ketten müssten entweder beide mit undefiniertem $R^{-1}$ oder mit undefiniertem $S^{-1}$ stoppen (Widerspruch). Demnach muss $h$ eine injektive Abbildung sein.
Als nächstes weisen wir die Surjektivität von $h$ nach, indem wir für jedes $y \in \Sigma^*$ ein Urbild $x$ mit $h(x) = y$ dingfest machen:

**Fall 1** Die $R$-Kette für $y$ stoppt mit undefiniertem $S^{-1}$.
In diesem Fall hat die $R$-Kette die Form

$$R^{-1}(S^{-1}(\cdots R^{-1}(y)\cdots)) \ .$$

Wir definieren dann $x = R^{-1}(y)$. Somit stoppt die $S$-Kette für $x$ (welche ein Präfix der $R$-Kette für $y$ ist) mit undefiniertem $S^{-1}$ und es gilt (gemäß der Definition von $h$)

$$h(x) = R(x) = R(R^{-1}(y)) = y \ .$$

**Fall 2** Die $R$-Kette für $y$ stoppt mit undefiniertem $R^{-1}$.
In diesem Fall hat die $R$-Kette die Form

$$S^{-1}(R^{-1}(\cdots R^{-1}(y)\cdots)) \ .$$

Wir definieren dann $x = S(y)$, was $y = S^{-1}(x)$ zur Folge hat. Somit stoppt die $S$-Kette für $x$ (welche die $R$-Kette für $y$ zum Präfix hat) mit undefiniertem $R^{-1}$ und es gilt (gemäß der Definition von $h$)

$$h(x) = S^{-1}(x) = S^{-1}(S(y)) = y \ .$$

Es hat sich somit die Surjektivität von $h$ ergeben.
Wir diskutieren nun die Berechnungskomplexität der Funktionen $h$ und $h^{-1}$. Zur Berechnung von $h(x)$ müssen wir zunächst die vollständige $S$-Kette für $x$ berechnen, um festzustellen, ob sie mit undefiniertem $R^{-1}$ oder mit undefiniertem $S^{-1}$ stoppt. Dies erfordert lediglich $n$ Auswertungen der Funktionen $R^{-1}$ bzw. $S^{-1}$, was in Polynomialzeit geleistet werden kann. Nachdem klar ist, welcher Fall vorliegt, muss dann entweder $h(x) = S^{-1}(x)$ oder $h(x) = R(x)$ berechnet werden. Auch dies ist in Polynomialzeit möglich. Die Berechnung von $h^{-1}$ ist aus Symmetriegründen genau so aufwendig wie die Berechnung von $h$. Wenn wir nämlich die obige Diskussion zur Surjektivität von $h$ noch einmal aufmerksam durchlesen, stellen wir fest, dass die Definition von $h^{-1}$ symmetrisch zur Definition von $h$ (mit vertauschten Rollen von $R$ und $S$) ist. Also sind sowohl $h$ als auch $h^{-1}$ in Polynomialzeit berechenbar.
Um den Nachweis der Isomorphie-Bedingungen für $h$ zu vervollständigen, müssen wir lediglich noch zeigen, dass jeder String $x \in \Sigma^*$ zu $L_1$ gehört gdw $h(x)$ zu $L_2$ gehört. Dies ist aber klar, da entweder $h(x) = S^{-1}(x)$ oder $h(x) = R(x)$ und sowohl $S^{-1}$ als auch $R$ (in ihrer Funktion als Reduktionsabbildungen) die geforderte Eigenschaft besitzen. □

**Folgerung 6.92 (Berman und Hartmanis, 1977)** *Alle NP-vollständigen Sprachen, die mit einer Polsterfunktion versehen werden können, sind zueinander polynomiell isomorph.*

Die Tatsache, dass alle bekannten NP-vollständigen Probleme eine Polsterfunktion besitzen, führte zu der

**Isomorphie-Vermutung (Berman und Hartmanis, 1977)** Alle NP-vollständigen Sprachen sind zueinander isomorph.

Es ist bis heute nicht geklärt, ob diese Vermutung zutrifft.

**Definition 6.93** *Die* Dichte *bzw.* Dichtefunktion *einer Sprache $L \subseteq \Sigma^*$ ist definiert als die Folgende Funktion in der Eingabelänge $n$:*

$$dens_L(n) := |\{x \in L : \ |x| \leq n\}| \ .$$

*$L$ heißt* dünn *(oder auch* spärlich*), wenn ihre Dichte polynomiell in $n$ beschränkt ist; $L$ heißt* dicht*, wenn ihre Dichte superpolynomiell mit $n$ wächst.*

Alle uns bisher bekannten NP-vollständigen Sprachen sind dicht!

**Beobachtung 6.94** *Wenn $L_1$ und $L_2$ polynomiell isomorph sind, dann gibt es Polynome $q_1, q_2$, so dass*

$$dens_{L_1}(n) \leq dens_{L_2}(q_1(n)) \text{ und } dens_{L_2}(n) \leq dens_{L_1}(q_2(n)) \ .$$

*Hieraus folgt, dass eine dünne und eine dichte Sprache nicht zueinander polynomiell isomorph sein können.*

Wenn die Isomorphie-Vermutung zutrifft, dann würde aus dieser Beobachtung folgen, dass eine dünne Sprache nicht NP-vollständig sein kann. Diese Folgerung kann jedoch bereits unter der schwächeren $P \neq NP$-Voraussetzung bewiesen werden:

**Satz 6.95 (Mahaney, 1982)** *Falls $P \neq NP$, dann gibt keine dünne und zugleich NP-vollständige Sprache.*

Ein Spezialfall von dünnen Sprachen stellen die *unären Sprachen $L \subseteq \{0\}^*$* dar. Anstelle des Satzes von Mahaney beweisen wir den folgenden (etwas leichteren)

**Satz 6.96 (Berman, 1978)** *Falls $P \neq NP$, dann gibt keine unäre und zugleich NP-vollständige Sprache.*

**Beweis** Wir führen den Beweis indirekt und zeigen, dass die Existenz einer $NP$-vollständigen unären Sprache $L_0 \subseteq \{0\}^*$ die Gleichheit von $P$ und $NP$ zur Folge hat. Dazu genügt es, mit Hilfe einer Reduktionsabbildung $R : \Sigma^* \to \Sigma^*$ für SAT $\leq_{pol} L_0$ einen deterministischen polynomiell zeitbeschränkten Algorithmus für SAT zu entwerfen. Sei also $x \in \Sigma^N$ der Kodierungsstring für eine CNF-Formel $F$ über den Booleschen Variablen $v_1, \ldots, v_n$. Wir skizzieren im Folgenden ein effizientes Verfahren, um die Erfüllbarkeit von $F$ zu testen. Dabei identifizieren wir einen String $a \in \{0,1\}^j$ mit der partiellen Belegung

$$v_1 = a_1, \ldots, v_j = a_j \ .$$

Es bezeichne $F[a]$ die vereinfachte CNF-Formel, die aus $F$ durch die von $a$ repräsentierte partielle Belegung hervor geht.[40] Der Kodierungsstring für $F[a]$ sei mit $x[a]$ bezeichnet. Für

[40] Elimination aller durch die partielle Belegung bereits erfüllten Klauseln und, in den verbleibenden Klauseln, Elimination der durch die partielle Belegung bereits falsifizierten Literale

$a = \epsilon$ (leeres Wort bzw. leere partielle Belegung) identifizieren wir $F[a]$ mit $F$ und $x[a]$ mit $x$. Offensichtlich gilt für $j = 0, \ldots, n-1$ und alle $a \in \{0,1\}^j$ die Beziehung

$$F[a] \text{ ist erfüllbar} \Leftrightarrow F[a0] \text{ oder } F[a1] \text{ ist erfüllbar.}$$

Für alle $a \in \{0,1\}^n$ ist $F[a]$ eine Boolesche Konstante und somit erfüllbar gdw es sich um die Konstante 1 handelt. Ein exponentiell zeitbeschränkter Algorithmus könnte vorgehen wie folgt:

**Top-down Pass** Bilde einen vollständig binären Baum $T$ der Tiefe $n$. Assoziiere mit einer Kante zum linken Kind das Bit 0 und mit einer Kante zum rechten Kind das Bit 1. Auf diese Weise ist zu jedem Knoten $z$ der Tiefe $j$ des Baumes ein eindeutiger Binärstring $a(z) \in \{0,1\}^j$ assoziiert. Insbesondere entsprechen die $2^n$ Blätter (Knoten der Tiefe $n$) 1-zu-1 den Strings aus $\{0,1\}^n$.

**Bottom-up Pass** Markiere jedes Blatt $z$ mit der Booleschen Konstante $F[a(z)]$. Markiere anschließend jeden inneren Knoten $z$ mit bereits markierten Kindern $z_0, z_1$ mit der Disjunktion (logisches „Oder“) der Booleschen Markierungen seiner Kinder.

Nach unseren obigen Ausführungen sollte klar sei, dass eine Formel $F[a(z)]$ mit 1 markiert wird gdw $F[a(z)]$ erfüllbar ist. Die Markierung an der Wurzel verrät uns, ob $F$ erfüllbar ist. Der „Schönheitsfehler“ dieses Verfahrens ist die exponentielle Größe der verwendeten Datenstruktur $T$. Jetzt kommen zwei Ideen ins Spiel, die darauf hinaus laufen, dass nur ein polynomiell kleines Anfangsstück $T'$ von $T$ wirklich gebraucht wird:

**Lazy Evaluation** Wenn $F[a0]$ erfüllbar ist, dann ist (unabhängig von der Erfüllbarkeit von $F[a1]$) auch die Formel $F[a]$ erfüllbar. Im Baum $T$ bedeutet dies für einen Knoten $z$ mit linkem Kind $z_0$ und rechtem Kind $z_1$: wenn $z_0$ beim „bottom-up pass“ die Markierung 1 erhalten hat, brauchen wir uns für den Unterbaum mit Wurzel $z_1$ nicht mehr zu interessieren und können $z$ direkt mit 1 markieren. Nach welcher Strategie muss der Baum $T$ gebildet werden, um diesen Effekt voll auszunutzen? Die Antwort lautet: „Durchlauf in Präordnung (preorder traversal)“.[41] Der Durchlauf in Präordnung stellt sicher, dass wir den „bottom-up pass“ des Unterbaumes von $z_0$ bereits abgeschlossen haben, bevor wir in den „top-down pass“ des Unterbaumes von $z_1$ einsteigen. Abbildung 12 illustriert diesen Gedankengang.

**Hashing** Wir verwenden die Reduktionsabbildung $R$ als eine Art „Hashfunktion“. Dabei nutzen wir aus, dass zwei CNF-Formeln $F, F'$ mit Kodierungsstrings $x, x'$ und $R(x) = R(x')$ entweder beide erfüllbar sind (nämlich, wenn $R(x) = R(x') \in L_0$) oder beide nicht erfüllbar sind (nämlich, wenn $R(x) = R(x') \notin L_0$). Im Baum $T$ bedeutet dies für einen Knoten $z'$: falls zum Zeitpunkt des Kreierens von $z'$ bereits ein Knoten $z$ in $T$ existiert, welcher bereits mit einem Bit markiert ist und die Bedingung $R(x[a(z)]) = R(x[a(z')]$ erfüllt, dann können wir $z'$ direkt mit dem gleichen Bit markieren. (Da wir für $z'$ keine Kinder kreieren müssen, wird $z'$ dann zu einem Blatt des Baumes $T'$.)

[41] Durchlauf eines Baumes mit Wurzel $r$, Wurzelkindern $r_0, r_1$ und den Unterbäumen $T_0, T_1$ in Präordnung ist rekursiv definiert wie folgt: durchlaufe zuerst $r$ dann (rekursiv) alle Knoten von $T_0$ und schließlich (rekursiv) alle Knoten von $T_1$.

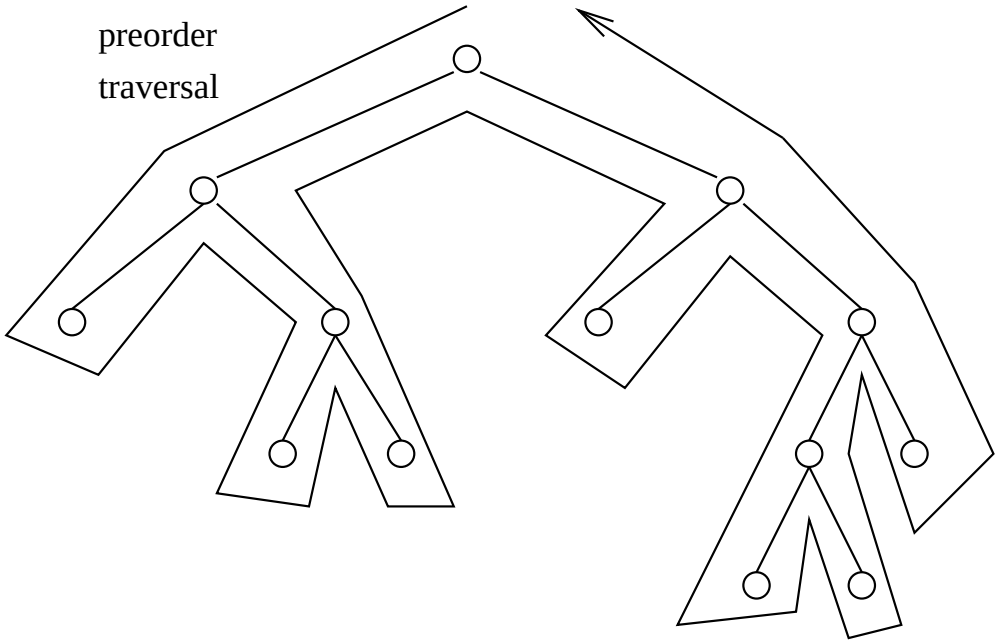



Abbildung 12: Ein Durchlauf der Knoten eines Baumes in Präordnung: bei zwei unabhängigen Knoten $z, z'$ mit $z$ links von $z'$ ist der „bottom-up pass“ von $z$ abgeschlossen, bevor der „top-down pass“ von $z'$ beginnt.

Beide Ideen verfolgen im Kern eine „pruning“-Technik: wir können von dem exponentiell großen Baum $T$ Teilbäume „abschneiden“, wenn sie für unsere Markierungsstrategie nur redundante Information liefern. Es sollte klar sein, dass der komplette Algorithmus für SAT (unter Einsatz von „pruning“) in Polynomialzeit implementiert werden kann, wenn das Anfangsstück $T'$ von $T$ ($T$ ohne die abgeschnittenen bzw. gar nicht erst kreierten Teilbäume) eine polynomiell beschränkte Größe hat. Wir argumentieren nun, dass dies der Fall ist. Zunächst setzen wir oBdA die Bedingung

$$\forall x \in \Sigma^* : R(x) \in \{0\}^*$$

voraus. Wir können nämlich jeden String $R(x)$, der neben 0 noch weitere Buchstaben verwendet, durch einen beliebigen (fest ausgewählten) String aus $\{0\}^* \setminus L_0$ ersetzen.[42] Da $R$ in Polynomialzeit berechenbar ist, muss es ein Polynom $q$ geben, so dass für alle $x$ gilt

$$|x| \leq N \Rightarrow |R(x)| < q(N) \ .$$

Da von $R$ nur unäre Bilder produziert werden, induzieren die Strings der Maximallänge $N$ maximal $q(N)$ paarweise verschiedene Bilder unter $R$. Das bedeutet, dass die Anzahl der bei unserem SAT-Algorithmus auftretenden Hashwerte durch $q(N)$ beschränkt ist. Wie ist nun die Beziehung zwischen der Anzahl $m$ der Knoten in $T'$ und der maximalen Anzahl $q(N)$ von Hashwerten? Die entscheidende Beobachtung ist die folgende: zwei unabhängige[43] innere

[42] Wegen der (angenommenen) *NP*-Vollständigkeit von $L_0$ muss $L_0$ eine echte Teilmenge von $\{0\}^*$ sein!

[43] „unabhängig“ bedeutet „kein Knoten ist Vorfahr des anderen“.

Knoten $z, z'$ in $T'$ haben verschiedene Hashwerte. Der Grund hierfür ist einfach: einer von beiden, sagen wir $z$, wurde zuerst durchlaufen und mit einem Bit markiert, bevor der andere, also $z'$, besucht wird; wäre $R(z) = R(z')$ dann hätten wir zu $z'$ keine Kinder kreiert und $z'$ müsste ein Blatt sein. Nun bestimmen wir die Knotenanzahl $m$ wie folgt:

- Es bezeichne $m'$ die Anzahl der Blätter in $T'$, $T''$ den Baum $T'$ abzüglich der $m'$ Blätter und $m''$ die Anzahl der Blätter in $T''$.
- Da die Blätter in $T''$ paarweise unabhängige, innere Knoten von $T'$ sind, müssen sie paarweise verschiedene Hashwerte haben und es gilt $m'' \leq q(N)$.
- Da die Tiefe von $T''$ durch $n-1$ beschränkt ist, enthält $T''$ maximal $nm''$ Knoten: jeder Knoten in $T''$ liegt nämlich auf einem Pfad der Länge höchstens $n-1$ von einem Blatt in $T''$ zur Wurzel.
- Da jeder Knoten in $T'$ maximal zwei Kinder hat, gibt es in $T'$ maximal ein Blatt mehr als innere Knoten (d.h., als Knoten in $T''$). Somit hat $T'$ maximal $nm+1$ Blätter.

Insgesamt besitzt $T'$ also maximal $(nm+1)+(nm) = 2nm+1 \leq 2nq(N)+1$ Knoten. Da $n \leq N$ und $N = |x|$ die Eingabelänge bezeichnet, ist der Beweis abgeschlossen. □

Die Sätze 6.95 und 6.96 bedeuten im Umkehrschluss: wenn es gelingt eine NP-vollständige unäre bzw. dünne Sprache ausfindig zu machen, dann ist $P = NP$.

## 6.12 Relativierung der P versus NP Frage

Baker, Gill und Solovay sind in einer Aufsehen erregenden Publikation aus dem Jahre 1975 der Frage nachgegangen, ob die Frage der (Un-)gleichheit von P und NP in einer durch die Anwesenheit von Orakeln „relativierten Welt“ entschieden werden kann. In einer solchen Welt würde die Rechenkraft von DTMs oder NTMs künstlich erhöht, indem bestimmte (einer Relation oder formalen Sprache entsprechende) Anfragen an ein Orakel weiter geleitet werden können und von diesem dann jeweils in einem Schritt beantwortet werden (Konzept der Orakel-Turing-Maschinen oder kurz OTMs, DOTMs, NOTMs). Warum sollte die Anwesenheit eines Orakel die Beantwortung der P versus NP Frage erleichtern? Hier sind zwei Gründe:

- Um den Unterschied zwischen P und NP zu nivellieren, könnte man ein Orakel benutzen, welches deterministische Maschinen so mächtig macht, dass durch Nichtdeterminismus keine zusätzlichen Fähigkeiten erwachsen.
- Um den Unterschied zwischen P und NP zu verstärken (und dadurch mathematisch beweisbar zu machen), könnte man Orakel benutzen, die die Fähigkeiten nichtdeterministischer Maschinen signifikant stärker potenzieren als die von deterministischen.

Es wird sich zeigen, dass in der Tat beide Grundideen durchführbar sind. In diesem Sinne ergibt sich durch Relativierung der P versus NP Frage gewissermaßen ein „unentschieden“: für manche Relativierungen gilt $P = NP$, für andere gilt $P \neq NP$.

## *Also was (so what)?*

Diese Untersuchungen zeigen in erster Linie auf, dass klassische Techniken zum Nachweis der (Un-)gleichheit zweier Komplexitätsklassen in Bezug auf die P versus NP Frage zum Scheitern verdammt sind:

- Eine klassische Technik zum Nachweis von $\mathcal{C} = \mathcal{C}'$ (Gleichheit zweier Komplexitätsklassen) ist *(wechselseitige) Schritt-für-Schritt Simulation*: zeige, dass jeder Rechenschritt einer zu $\mathcal{C}$ passenden Maschine $M$ durch (evtl. mehrere) Rechenschritte einer zu $\mathcal{C}'$ passenden Maschine $M'$ simuliert werden kann (und umgekehrt). Diese Beweistechnik lässt sich relativieren: wenn beide Maschinen zusätzlich mit einem Orakel versehen sind, dann kann die Simulation aufrecht erhalten werden, indem herkömmliche Rechenschritte simuliert werden wie zuvor und eine Orakelanfrage von $M$ „simuliert wird" durch dieselbe Orakelanfrage von $M'$. Wenn sich also $\mathcal{C} = \mathcal{C}'$ mit Hilfe von Schritt-für-Schritt Simulation nachweisen lässt, dann sollte die Gleichheit in **jeder** relativierten Welt gelten.

- Eine klassische Technik zum Nachweis von $\mathcal{C} \subset \mathcal{C}'$ (echte Inklusion) ist *(einseitige) Schritt-für-Schritt Simulation* verbunden mit *Diagonalisierung*: wenn $M_1, M_2, \ldots$ eine Aufzählung aller zu $\mathcal{C}$ passenden Maschinen ist und $z_1, z_2, \ldots$ ist eine Aufzählung aller Eingabestrings, dann versuche eine zu $\mathcal{C}'$ passende Maschine $M'$ anzugeben, die auf Eingabe $z_i$ die Maschine $M_i$ Schritt-für-Schritt simuliert, aber am Ende der Rechnung die Antwort JA/NEIN zu NEIN/JA rumdreht. $M'$ ist dann Akzeptor einer Sprache aus $\mathcal{C}' \setminus \mathcal{C}$.[44] Auch diese Beweistechnik lässt sich (in der offensichtlichen Weise) relativieren. Wenn sich also $\mathcal{C} \subset \mathcal{C}'$ mit Hilfe von (auf Schritt-für-Schritt Simulation basierender) Diagonalisierung nachweisen lässt, dann sollte die echte Inklusionsbeziehung in **jeder** relativierten Welt gelten.

Diese Überlegungen zeigen im Umkehrschluss: da die Antwort auf die P versus NP Frage in den relativierten Welten zweideutig ausfällt, kann weder die Gleicheit noch die Ungleichheit von $P$ und $NP$ mit einer relativierenden Technik (wie Schritt-für-Schritt Simulation, Diagonalisierung,...) geführt werden.[45]

Nach diesen philosophischen Anmerkungen kommen wir nun zur eigentlichen Arbeit. Es sei daran erinnert, dass $M^R$ bzw. $M^L$ eine Orakel-Turing-Maschine (OTM) mit einem Orakel für die Relation $R$ bzw. für die Sprache $L$ bezeichnet. Wenn $M$ einen Fragestring $x$ auf das spezielle Orakelband schreibt und in den Fragezustand $q_?$ übergeht, dann ersetzt das $R$-Orakel (in **einem** Schritt) den String $x$ durch einen String $y$ mit $(x, y) \in R$ (sofern möglich) und „beamt" $M$ in den Zustand $q_+$. Wenn es keinen passenden String $y$ gibt, dann wird $x$ nicht ersetzt und in den Zustand $q_-$ „gebeamt". Im Falle eines $L$-Orakels wird im Falle $x \in L$ der String $x$ gelöscht und $M$ in Zustand $q_+$ „gebeamt"; falls $x \notin L$, dann wird $x$ nicht gelöscht und $M$ in Zustand $q_-$ „gebeamt". Wenn $M$ deterministisch ist, sprechen wir von einer DOTM $M^R$ bzw. $M^L$; ist $M$ nichtdeterministisch sprechen wir von einer NOTM. Wenn $\mathcal{C}$

[44] Diese Technik hatten wir beim Nachweis des allgemeinen Hierachiesatzes eingesetzt.

[45] Viele „Beweise" für $P = NP$ oder $P \neq NP$, die (mehr oder weniger qualifizierte) Forscher der staunenden Fachwelt präsentiert haben, scheitern bereits an dieser Hürde: es wurde eine relativierende Beweistechnik verwendet. Ein solcher Beweis kann von vorne herein nicht korrekt sein.

eine (deterministische oder nicht-deterministische) Zeitkomplexitätsklasse bezeichnet, dann sei $\mathcal{C}^R$ bzw. $\mathcal{C}^L$ die Klasse aller Sprachen, die sich mit einer OTM $M^R$ bzw. $M^L$ erkennen lassen, wobei $M$ eine zu $\mathcal{C}$ passende Maschine sein muss.

Welches Orakel könnte geeignet sein, den Unterschied zwischen $P$ und $NP$ (beweisbar) zu nivellieren? Der Satz von Savitch suggeriert, ein $A$-Orakel für eine *PSpace*-vollständige Sprache $A$ zu verwenden. Diese Idee kommt im Beweis des folgenden Resultates zum Tragen:

**Satz 6.97 (Baker, Gill, and Solovay, 1975)** *Es gibt eine Sprache $A$ mit $P^A = NP^A$.*

**Beweis** Es sei $A$ eine *PSpace*-vollständige Sprache.[46] Dann folgt $P^A = NP^A$ aus

$$PSpace \subseteq P^A \subseteq NP^A \subseteq NSpace(POL) = DSpace(POL) = PSpace \ . \tag{29}$$

Die erste Inklusion gilt wegen der *PSpace*-Vollständigkeit von $A$. Sei nämlich $L \in PSpace$, $R$ eine Reduktionsabbildung für $L \leq_{pol} A$ und $x \in \Sigma^n$ ein Eingabestring. Um die Frage der Mitgliedschaft von $x$ in $L$ zu entscheiden, können wir zunächst in poly$(n)$ Schritten $R(x)$ berechnen und dann das $A$-Orakel nach der Mitgliedschaft von $R(x)$ in $A$ befragen.
Die zweite Inklusion in (29) ist trivial.
Die dritte Inklusion in (29) ergibt sich daraus, dass eine NOTM $M^A$ für $L \in NP^A$ durch eine NTM $M'$ mit polynomieller Platzschranke simuliert werden kann: wann immer $M^A$ das $A$-Orakel nach der Mitgliedschaft eines Strings $z$ zur Sprache $A$ befragt, kann sich $M'$ (wegen $A \in PSpace$) die Antwort selber herleiten.
Die Gleichheit von $NSpace(POL)$ und $DSpace(POL) = PSpace$ ergibt sich aus dem Satz von Savitch. □

Ein Orakel, dass $P$ von $NP$ trennt, ist etwas schwieriger zu konstruieren, existiert aber ebenfalls:

**Satz 6.98 (Baker, Gill, and Solovay, 1975)** *Es gibt eine Sprache $B$ mit $P^B \neq NP^B$.*

**Beweis** Wir konstruieren eine Sprache $B$ und eine (von $B$ abhängige) Sprache $L_* \in NP^B \setminus P^B$ und beginnen mit der Definition von $L_*$:

$$L_* := \{0^n | \ B \cap \Sigma^n \neq \emptyset\} \ .$$

Bereits ohne Kenntnis von $B$ ist sonnenklar, dass $L_*$ zur Klasse $NP^B$ gehört, da die Wörter $0^n \in L_*$ mit Hilfe des $B$-Orakels nichtdeterministisch in Polynomialzeit erkannt werden können: rate $x \in \Sigma^n$ und verifiziere $x \in B$ durch Anfrage an das $B$-Orakel. Die Kampfaufgabe im restlichen Beweis wird darin bestehen, $B$ so zu definieren, dass $L_* \notin P^B$. Zu diesem Zweck greifen wir auf die Technik der Diagonalisierung zurück. Es bezeichne

$$M_1^B, M_2^B, \ldots$$

[46] Beispiele solcher Sprachen werden wir in einem späteren Abschnitt kennen lernen.

eine Aufzählung[47] aller polynomiell zeitbeschränkter DOTMs mit Orakel $B$ (und einer Uhr, sie poly($n$) viele Rechenschritte zulässt), so dass

$$P^B = \bigcup_{i \geq 1} L_{M_i^B} \quad .$$

Wir setzen zusätzlich voraus, dass jede Sprache in $P^B$, die in Zeit $n^c$ erkannt werden kann in der Aufzählung mit unendlich vielen Akzeptoren (deren Uhr mindestens $n^c$ Rechenschritte zulässt) vertreten ist (was durch „redundante Kodierung“ von Turing-Maschinen-Programmen leicht zu erreichen ist). Wir definieren nun $B$ stufenweise, wobei

$$B_i = B \cap \Sigma^i$$

die *$i$-the Stufe* von $B$ bezeichnet. Nebenbei protokollieren wir eine Menge $X$ von „Ausnahmestrings“, die wir auf keinen Fall in $B$ aufzunehmen wünschen. Anfangs gilt

$$B_0 = X = \emptyset \quad .$$

Die Menge $X$ ist aber eine dynamische Größe, die sich im Laufe der Konstruktion von $B$ verändert. In Stufe $i$ der Konstruktion wollen wir im Prinzip erreichen, dass der String $0^i$ zwischen $L_*$ und $L_{M_i^B}$ trennt:

**Stufe i** „Simuliere“ $M_i^B$ auf Eingabestring $0^i$ für

$$q(i) := i^{\lfloor i \rfloor}$$

viele Schritte.[48] Bei der Simulation kann $M_i^B$ das $B$-Orakel nach der Mitgliedschaft von Strings $z$ zur Sprache $B$ befragen. Falls $|z| \leq i-1$, antworten wir in der Simulation mit JA, falls $z \in B_{|z|}$, und mit NEIN, falls $z \notin B_{|z|}$. Beachte dabei, dass die $|z|$-te Stufe von $B$ bereits vorher konstruiert wurde. Falls aber $|z| \geq i$, dann antworten wir mit NEIN und nehmen $z$ in $X$ auf (was die Antwort NEIN im Nachhinein rechtfertigen wird, da wir Strings aus $X$ niemals in $B$ aufnehmen werden). Auf diese Weise kann die Simulation am Laufen gehalten werden. Beachte, dass in Stufe $i$ maximal $q(i)$ Strings in $X$ aufgenommen werden. Eine leichte Rechnung ergibt, dass

$$\sum_{j=1}^{i} q(j) < 2^i \quad ,$$

d.h., nach Stufe $i$ muss $X \cap \Sigma^i$ eine echte Teilmenge von $\Sigma^i$ sein. Nun sind drei Fälle denkbar:

[47] Diese Aufzählung hängt **nicht** von $B$ ab, da im Programm einer DOTM $M_i$ lediglich festgelegt ist, wie nach Erreichen der Zustände $q_-$ bzw. $q_+$ weiter gearbeitet wird. Die Abhängigkeit von $B$ ergibt sich so zu sagen erst zur Laufzeit.

[48] Die Funktion $q(i)$ ist so gewählt, dass sie asymptotisch schneller wächst als jedes Polynom, aber langsamer als $2^i$.

**Fall 1** $M_i^B$ verwirft $0^i$ in maximal $q(i)$ Schritten.
Dann setzen wir

$$B_i := \Sigma^i \setminus X \neq \emptyset \ .$$

Nun gilt $0^i \notin L_{M_i^B}$ und (gemäß der Definition von $L_*$) $0^i \in L_*$.

**Fall 2** $M_i^B$ akzeptiert $0^i$ in maximal $q(i)$ Schritten.
Dann setzen wir

$$B_i := \emptyset$$

mit der Konsequenz, dass $0^i \in L_{M_i^B}$ und (wieder gemäß der Definition von $L_*$) $0^i \notin L_*$.

**Fall 3** $M_i^B$ kommt im Laufe von $q(i)$ Schritten zu keiner Entscheidung.
In diesem Fall setzen wir auch $B_i := \emptyset$, gelangen aber (im Unterschied zu den ersten beiden Fällen) zu keiner Separation von $L_*$ und $L_{M_i^B}$.

Ohne den Fall 3 wäre der Diagonalisierungsbeweis nun abgeschlossen. Erinnern wir uns daran, dass wir listigerweise verlangt hatten, dass jede Sprache $L$, die mit Zeitschranke $n^c$ erkennbar ist, in der Aufzählung der $M_i$ unendlich oft vertreten ist. Da $q(i)$ schneller wächst als jedes Polynom, wird für Akzeptoren von $L$ mit hinreichend großem Index $i$ der Fall 3 nicht eintreten. Somit kann jede Sprache aus $P^B$ von $L_*$ getrennt werden. □

# 7 Die polynomielle Hierarchie von Larry Stockmeyer

## 7.1 Einleitende Betrachtungen

Wir gehen der Frage nach, ob die Klasse der Sprachen, welche polynomiell reduzierbar auf eine Sprache aus *NP* sind, mit *NP* zusammenfällt oder *NP* echt enthält. Im zweiten Fall würde die polynomielle Reduktionstechnik gewissermaßen für eine „neue Qualität“ an Rechenkraft sorgen. Es wird sich zeigen, dass die Antwort auf diese Frage vom gewählten Reduktionsbegriff abhängt. Im Falle von Turing-Reduktionen entsteht eine Komplexitätsklasse die (voraussichtlich) *NP* echt enthält, wohingegen polynomielle Reduktionen nicht aus *NP* hinausführen.

### 7.1.1 Polynomielle Reduktionen auf Sprachen aus NP

Falls $L \leq_{pol} L'$ und $L' \in NP$, dann folgt bekanntlich $L \in NP$. Wir erhalten nämlich eine polynomielle NTM $M$ für $L$ mit Hilfe der polynomiellen NTM $M'$ für $L'$ und der polynomiellen Reduktion $f : \Sigma^* \to \Sigma^*$ wie folgt:

1. Transformiere die Eingabe $x$ zum Mitgliedschaftsproblem für die Sprache $L$ in die korrespondierende Eingabe $x' = f(x)$ zum Mitgliedschaftsproblem für die Sprache $L'$.
2. Sezte $M'$ auf $x'$ an und akzeptiere genau dann, wenn $M'$ akzeptiert.

Die Klasse der auf Sprachen aus *NP* polynomiell reduzierbaren Sprachen führt also nicht aus *NP* heraus.

### 7.1.2 Turing-Reduktionen auf Sprachen aus NP

Bei Turing-Reduktionen ergibt sich ein völlig anderes Bild. Wir wollen dies durch ein paar Beobachtungen und Beispiele illustrieren.

Eine erste Beobachtung wäre, dass jede Sprache auf ihr Komplement Turing-reduzierbar ist:

**Beobachtung 7.1** $\forall L \subseteq \Sigma^* : L \leq_T \bar{L}$.

**Beweis** Zu Eingabe $x$ können wir das $\bar{L}$-Orakel befragen und seine Antwort umkehren. □

Aus Beobachtung 7.1 ergibt sich unmittelbar, dass jede Sprache aus $NP \cup$ co-$NP$ auf das Erfüllbarkeitsproblem der Booleschen Logik Turing-reduzierbar ist:

**Beobachtung 7.2** $\forall L \in NP \cup co\text{-}NP : L \leq_T SAT$.

**Beweis** Für $L \in NP$ ergibt sich $L \leq_{pol}$ SAT und somit auch $L \leq_T$ SAT aus der *NP*-Vollständigkeit von SAT. Für $L \in$ co-$NP$ folgt zunächst $\bar{L} \in NP$. Zusammen mit Beobachtung 7.1 erhalten wir die Reduktionskette $L \leq_T \bar{L} \leq_T$ SAT. Nun folgt $L \leq_T$ SAT aus der Transitivität von Turing-Reduktionen. □

Die Klasse der auf SAT Turing-reduzierbaren Sprachen führt also (unter der $NP \neq$ co-$NP$-Voraussetzung) aus der Klasse $NP$ heraus. Eine DOTM mit „Rechenkraft“ $P$ und ein Orakel der „Rechenkraft“ $NP$ können demzufolge in Kooperation eine „Rechenkraft“ oberhalb von $NP$ erlangen.

Ein Orakel für eine $NP$-vollständige Sprache nennen wir im folgenden einfach $NP$-Orakel. Wir wollen illustrieren, wie mächtig eine polynomielle NOTM wird, wenn sie mit einem $NP$-Orakel ausgestattet ist. Dabei spielen die in den folgenden Beispielen beschriebenen Sprachen eine wichtige Rolle:

**Beispiel 7.3** *MINIMUM EQUIVALENT CIRCUIT (kurz: MEC) sei die Sprache aller Paare (genauer: Kodierungen von Paaren) der Form $(S, k)$, wobei gilt:*

- *$S$ ist ein Boolescher Schaltkreis.*
- *$k$ ist eine nicht-negative ganze Zahl.*
- *Es gibt einen Booleschen Schaltkreis $S'$ mit höchstens $k$ Bausteinen (aus der Basis AND, OR, NOT), der die gleiche Boolesche Funktion wie $S$ berechnet.*

*Das zu MEC korrespondierende Minimierungsproblem gehört in den Bereich der Hardwareminimierung: suche die billigste Realisierung eines gegebenen Schaltkreises.*

**Beispiel 7.4** *UNSAT (ausgeschrieben: UNSATISFIABILITY) sei die Sprache der nicht erfüllbaren Booleschen CNF-Formeln $C$. Da UNSAT im Wesentlichen das Komplement von SAT ist, ist UNSAT co-NP-vollständig.*

**Beispiel 7.5** *SAT$^*$ sei die Sprache aller Schaltkreise (genauer: Kodierungen von Schaltkreisen), die nicht die konstante Nullfunktion berechnen. Da SAT offensichtlich ein Teilproblem von SAT$^*$ ist und andererseits SAT$^*$ zu NP gehört, ist SAT$^*$ NP-vollständig.*

Wir werden im Folgenden nachweisen, dass MEC ein sehr hartes Problem ist: es ist co-$NP$-hart unter polynomiellen Reduktionen und $NP$-hart unter Turing-Reduktionen. Vermutlich gehört es nicht zu $NP \cup$ co-$NP$, aber diese Frage ist offen. Unter der $NP \neq$ co-$NP$-Voraussetzung kann man zumindest ausschließen, dass MEC zu $NP$ gehört. Der Härte von MEC zum Trotz werden wir eine polynomielle NOTM angeben, die mit Hilfe eines $NP$-Orakels als Akzeptor von MEC auftritt. Die kombinierte Rechenkraft zweier Agenten (NOTM und Orakel) mit „Rechenkraft“ $NP$ reicht also aus, um die „harte Nuss“ namens MEC zu knacken.

Wir starten unsere Analyse von MEC mit der folgenden

**Beobachtung 7.6** *UNSAT $\leq_{pol}$ MEC.*

**Beweis** Wir verwenden im Prinzip die Technik der Spezialisierung. Sei $C$ eine Boolesche CNF-Formel, $f_C$ die von $C$ berechnete Boolesche Funktion und $S_C$ der zweistufige Schaltkreis, der $C$ auf die offensichtliche Weise berechnet (Berechnung der Klauseln mit OR-Gattern auf der ersten Stufe und Berechnung der Konjunktion aller Klauseln mit einem

AND-Gatter auf der zweiten Stufe). OBdA enthalte $C$ mindestens eine nicht-tautologische Klausel.[49] Dann gilt erst recht $f_C \not\equiv 1$ und die folgenden Äquivalenzen sind offensichtlich:

$$\begin{aligned} C \in \text{UNSAT} &\Leftrightarrow f_C \equiv 0 \\ &\Leftrightarrow f_C \text{ ist eine konstante Funktion} \\ &\Leftrightarrow f_C \text{ ist von einem Schaltkreis mit 0 Bausteinen berechenbar} \\ &\Leftrightarrow (S_C, 0) \in \text{MCE} \end{aligned}$$

Da $(S_C, 0)$ leicht aus $C$ berechenbar ist, erhalten wir eine polynomielle Reduktion von UNSAT auf MCE. □

**Folgerung 7.7** *1. SAT $\leq_T$ MEC.*

*2. MEC ist co-NP-hart unter polynomiekllen oder Turing-Reduktionen.*

*3. MEC ist NP-hart unter Turing-Reduktionen.*

*4. MEC $\notin$ NP unter der NP $\neq$ co-NP-Voraussetzung.*

**Beweis**

1. Aus UNSAT $\leq_{pol}$ MEC (gemäß Beobachtung 7.6) folgt UNSAT $\leq_T$ MEC. Wegen SAT $\leq_T$ UNSAT (gemäß Beobachtung 7.1) ergibt sich dann SAT $\leq_T$ MEC aus der Transitivität der Turing-Reduktionen.

2. Die co-*NP*-Härte von UNSAT vererbt sich wegen UNSAT $\leq_{pol}$ MEC auf MEC.

3. Die *NP*-Härte von SAT (unter polynomiellen Reduktionen) impliziert die *NP*-Härte von SAT unter Turing-Reduktionen. Diese vererbt sich wegen SAT $\leq_T$ MEC auf MEC.

4. Würde MEC als ein (unter polynomiellen Reduktionen) co-*NP*-vollständiges Problem zu *NP* gehören, dann würde jede Sprache $L \in$ co-*NP* wegen $L \leq_{pol}$ MEC und MEC $\in$ *NP* ebenfalls zu *NP* gehören. (Vgl. Abschnitt 7.1.1.) Dies hätte co-*NP* $\subseteq$ *NP* und daher auch co-*NP* = *NP* zur Folge. Im Umkehrschluss ergibt sich MEC $\notin$ *NP* aus der *NP* $\neq$ co-*NP*-Voraussetzung.

□

Wie angekündigt hat sich ergeben, dass MEC ein sehr hartes Problem ist. Wir komplettieren das Bild jetzt durch folgende

**Beobachtung 7.8** *Es gibt eine polynomielle NOTM, die mit Hilfe eines SAT*-Orakels als Akzeptor von MEC auftritt.*

[49] Eine Klausel heiße *tautologisch*, wenn sie nicht falsifiziert werden kann (wie zum Beispiel die leere Klausel oder die Klausel $v_1 \vee \bar{v}_1$).

**Beweis** Die NOTM gehe auf einer Eingabe $(S, k)$ vor wie folgt:

1. Rate einen Schaltkreis $S'$ mit maximal $k$ Bausteinen.
2. Seien $f$ und $f'$ die von $S$ und $S'$ berechneten Booleschen Funktionen. Synthetisiere auf die offensichtliche Weise die Schaltkreise $S$ und $S'$ zu einem neuen Schaltkreis $S^{\oplus}$, welcher die Funktion

   $$f \oplus f' := (f \wedge \bar{f}') \vee (\bar{f} \wedge f')$$

   berechnet.
3. Befrage das SAT$^*$-Orakel nach der Erfüllbarkeit von $S^{\oplus}$.
4. Akzeptiere genau dann, wenn $S^{\oplus}$ nicht erfüllbar ist.

Diese Vorgehensweise ist wegen

$$\begin{aligned} f \equiv f' &\Leftrightarrow f \oplus f' \equiv 0 \\ &\Leftrightarrow S^{\oplus} \text{ ist nicht erfüllbar} \end{aligned}$$

korrekt. Offensichtlich existiert genau dann, eine akzeptierende Rechnung zur Eingabe $(S, k)$, wenn $(S, k)$ zur Sprache MEC gehört. Die Laufzeit der skizzierten NOTM ist sicherlich polynomiell beschränkt. □

## 7.2 Definition und elementare Eigenschaften der Hierarchie

ge-
plant
für
06.07.

**Definition 7.9** *1. Sei $L \subseteq \Sigma^*$ eine Sprache.*

*(a) $P[L]$ bezeichne die Klasse aller Sprachen, die von einer polynomiellen DOTM mit Orakel $L$ akzeptiert werden können.*

*(b) $NP[L]$ bezeichne die Klasse aller Sprachen, die von einer polynomiellen NOTM mit Orakel $L$ akzeptiert werden können.*

*2. Sei $\mathcal{C}$ eine Klasse von Sprachen über Alphabet $\Sigma$.*

$$P[\mathcal{C}] := \bigcup_{L \in \mathcal{C}} P[L]$$
$$NP[\mathcal{C}] := \bigcup_{L \in \mathcal{C}} NP[L]$$

*(Notationen $P^L, NP^L, P^{\mathcal{C}}, NP^{\mathcal{C}}$ anstelle von $P[L], NP[L], P[\mathcal{C}], NP[\mathcal{C}]$ sind ebenso gebräuchlich.)*

Zum Aufwärmen machen wir ein paar einfache Beobachtungen:

**Lemma 7.10** *1. $\mathcal{C} \subseteq P[\mathcal{C}] \subseteq NP[\mathcal{C}]$.*

*2. $P[P] = P$ und $NP[P] = NP$.*

*3. Falls $L_0$ $\mathcal{C}$-vollständig (unter polynomiellen Reduktionen) ist, dann gilt $P[L_0] = P[\mathcal{C}]$ und $NP[L_0] = NP[\mathcal{C}]$.*

**Beweis**

1. $P[\mathcal{C}] \subseteq NP[\mathcal{C}]$ ist klar, da eine DOTM als eine spezielle NOTM angesehen werden kann. Um die Inklusion $\mathcal{C} \subseteq P[\mathcal{C}]$ nachzuweisen, genügt es, $L \in P[L]$ für eine beliebige Sprache $L \subseteq \Sigma^*$ zu zeigen. Dies ist aber klar, da eine mit einem $L$-Orakel ausgestattete DOTM die Mitgliedschaft von $x$ in $L$ durch Befragung des $L$-Orakels nach $x$ unmittelbar testen kann.

2. Eine polynomielle DTM (oder gar NTM) kann die Arbeit eines $P$-Orakels auch gleich selber machen.

3. Wir beschränken uns auf die Aussage $P[L_0] = P[\mathcal{C}]$. Der Nachweis der Aussage $NP[L_0] = NP[\mathcal{C}]$ erfolgt analog.
   $P[L_0] \subseteq P[\mathcal{C}]$ ist die triviale Beweisrichtung. Zum Nachweis von $P[\mathcal{C}] \subseteq P[L_0]$ haben wir für eine beliebige Sprache $L \in \mathcal{C}$ die Inklusion $P[L] \subseteq P[L_0]$ zu zeigen. Hierzu nutzen wir die $\mathcal{C}$-Vollständigkeit von $L_0$ aus. Sei $f$ die polynomiell berechenbare Abbildung bei der polynomiellen Reduktion von $L$ nach $L_0$. Die an ein $L$-Orakel gerichtete Nachfrage nach einem Wort $x \in \Sigma^*$ kann dann durch eine an ein $L_0$-Orakel gerichtete Nachfrage nach dem Wort $f(x)$ ersetzt werden.

□

Aussage 1 von Lemma 7.10 besagt gewissermaßen, dass eine Sprache $\mathcal{C}$ durch Anwendung des „$P$-Operators“ zu einer Oberklasse $P[\mathcal{C}]$ „aufgeblasen“ wird. Die Anwendung des mächtigeren „$NP$-Operators“ bläst $\mathcal{C}$ zu einer (evtl.) noch größeren Klasse auf. In diesem Lichte bedeutet Aussage 2, dass der $P$-Operator auf $P$ stationär ist und der $NP$-Operator $P$ zu $NP$ aufbläst. Aussage 3 bedeutet, dass es keinen Unterschied macht, ob wir einen der Operatoren auf $\mathcal{C}$ oder eine $\mathcal{C}$-vollständige Sprache anwenden. Die folgende Definition der polynomiellen Hierarchie, welche sich zwischen $P$ bzw. $NP$ und $PSpace$ aufspannt, beruht auf iterativer Anwendung des $P$- und des $NP$-Operators.

**Definition 7.11** *Die Sprachklassen $\Sigma_k$ und $\Delta_k$ sind für $k \geq 0$ induktiv definiert wie folgt:*

$$\begin{aligned} \Sigma_0 &:= \Delta_0 := P \\ \Sigma_k &:= NP[\Sigma_{k-1}] \\ \Delta_k &:= P[\Sigma_{k-1}] \end{aligned}$$

*Die Sprachlassen $\Pi_k$ ergeben sich als Komplementärklassen zu den Sprachklassen $\Sigma_k$:*

$$\Pi_k := co\text{-}\Sigma_k \ .$$

*Die Klassen $\Sigma_k, \Delta_k, \Pi_k$ bilden (zusammen mit den zwischen ihnen geltenden Inklusionsbeziehungen) die sogenannte* polynomielle Hierarchie.

Die untersten drei Stufen dieser Hierarchie ergeben sich aus dieser Definition wie folgt:

**Stufe 0** $\Sigma_0 = \Delta_0 = P$ und $\Pi_0 = \text{co-}P = P$.

**Stufe 1** $\Sigma_1 = NP[P] = NP$, $\Delta_1 = P[P] = P$ und $\Pi_1 = \text{co-}NP$.

**Stufe 2** $\Sigma_2 = NP[NP]$, $\Delta_2 = P[NP]$ und $\Pi_2 = \text{co-}NP[NP]$.

Wir sind jetzt darauf vorbereitet, die Beobachtung 7.8 etwas gelehrsamer zu formulieren:

**Lemma 7.12** *$MEC \in \Sigma_2$.*

**Beweis** Da SAT$^*$ $NP$-vollständig ist, gilt $\Sigma_2 = NP[NP] = NP[\text{SAT}^*]$. Beobachtung 7.8 ist äquivalent zu MEC $\in NP[\text{SAT}^*]$. □

Die folgenden Lemmas formulieren elementare Eigenschaften der polynomiellem Hierarchie und sind darüberhinaus eine geeignete Aufwärmübung, die die Vertrautheit mit Definition 7.11 erhöhen sollte. Wir geben bei jedem Beweis nur die entscheidende Idee an.

Das folgende Lemma macht deutlich, dass die polynomielle Hierarchie sich nicht verändert, wenn wir den $P$- oder $NP$-Operator auf $\Pi_{k-1}$ anstelle von $\Sigma_{k-1}$ anwenden.

**Lemma 7.13** *$\Sigma_k = NP[\Pi_{k-1}]$ und $\Delta_k = P[\Pi_{k-1}]$ für alle $k \geq 1$.*

**Beweis** Nutze aus, dass ein $L$-Orakel gleichwertig zu einem $\bar{L}$-Orakel ist: statt nach $w \in L$ zu fragen, frage nach $w \in \bar{L}$ und negiere die Antwort. □

Das nächste Lemma besagt, dass die Klassen $\Delta_k$ abgeschlossen unter Komplementbildung sind.

**Lemma 7.14** $\Delta_k = co\text{-}\Delta_k$ *für alle* $k \geq 0$.

**Beweis** Nutze aus, dass eine $L$ akzeptierende DOTM durch Vertauschen von akzeptierenden und nicht-akzeptierenden Zuständen in eine $\bar{L}$ akzeptierende DOTM transformiert wird. □

Das nun folgende Lemma deckt die Inklusionsbeziehungen auf:

**Lemma 7.15** $\Delta_k \subseteq \Sigma_k \cap \Pi_k$ *und* $\Sigma_k \cup \Pi_k \subseteq \Delta_{k+1}$ *für alle* $k \geq 0$.

**Beweis** Der Beweis erfolgt induktiv. Für $k = 0$ sind die Aussagen gültig. Für $k \geq 1$ ergibt sich zunächst

$$\Delta_k = P[\Sigma_{k-1}] \subseteq NP[\Sigma_{k-1}] = \Sigma_k \ .$$

Dies impliziert

$$\Delta_k = \text{co-}\Delta_k \subseteq \text{co-}\Sigma_k = \Pi_k$$

und daher auch $\Delta_k \subseteq \Sigma_k \cap \Pi_k$. Der Rest ergibt sich mit Hilfe der Lemmas 7.10 und 7.13:

$$\Sigma_k \subseteq P[\Sigma_k] = \Delta_{k+1} \text{ und } \Pi_k \subseteq P[\Pi_k] = \Delta_{k+1} \ .$$

□

Die Inklusionsbeziehungen in Lemma 7.15 sind in Abbildung 13 visualisiert. Es ergibt sich ausgehend von $P$ und $NP$ (auf den unteren Stufen) eine (voraussichtlich) unendliche Hierarchie von immer mächtigeren Klassen. Wir erhalten eine gemeinsame Oberklasse $PH$ durch die Definition

$$PH := \bigcup_{k \geq 0} \Sigma_k \ . \tag{30}$$

Wir werden später zeigen, dass (wie in Abbildung 13 bereits zu sehen) $PH$ in PSpace enthalten ist.

**Offene Probleme** Es ist nicht wirklich geklärt, ob $PH$ eine unendliche Hierarchie ist oder (das andere Extrem) vielleicht zu $NP$ oder gar $P$ kollabiert. Wir werden später sehen, dass $P = NP$ die scheinbar stärkere Aussage $P = PH$ impliziert. Auf ähnliche Weise würde $NP =$ co-$NP$, also Abgeschlossenheit von $NP$ unter Komplementbildung, $NP = PH$ implizieren. In Verbindung mit der $P \neq NP$-Vermutung bzw. der $NP \neq$ co-$NP$-Vermutung, gibt es den naheliegenden Verdacht, dass $PH$ nicht auf eine feste Stufe kollabiert, sondern dass jede neue Stufe der Hierarchie eine echt erweiterte Sprachklasse repräsentiert. In diesem Fall ergäbe sich eine unendliche Hierarchie. Eine vertiefte Debatte dieser Fragen werden wir führen können, wenn wir in einem späteren Abschnitt den Satz von Celia Wrathall kennengelernt haben, der die polynomielle Hierarchie zur prädikatenlogischen Beschreibungskomplexität in Beziehung setzt.

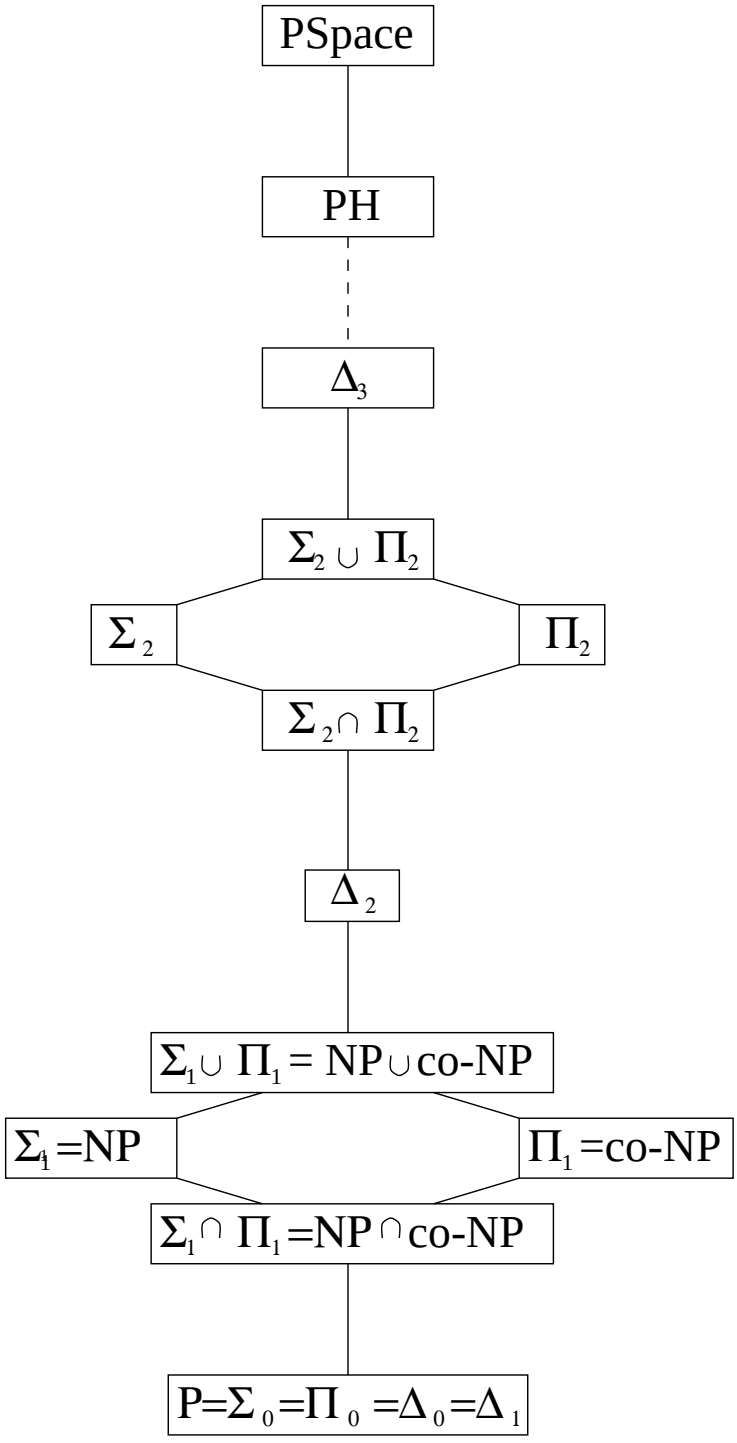



Abbildung 13: Die polynomielle Hierarchie von Stockmeyer.

# 8 Der Satz von Celia Wrathall

Im Jahre 1977 fand Celia Wrathall eine prädikatenlogische Charaktersisierung der Klassen $\Sigma_k$ und $\Pi_k$, welche wir auf der $k$-ten Stufe der polynomiellen Hierarchie vorfinden. Wir erinnern uns, dass $\Sigma_k$ aus $\Sigma_{k-1}$ durch Anwendung des $NP$-Operators hervorgeht: $\Sigma_k = NP[\Sigma_{k-1}]$. Durch anschließenden Übergang zur Komplementärklasse erhalten wir $\Pi_k = \text{co-}NP[\Sigma_{k-1}]$. Anschaulich können wir die Hintereinanderausführung des $NP$-Operators und des Übergangs zur Komplementärklasse als co-$NP$-Operator auffassen. Die Operatoren $NP$ und co-$NP$ sind berechnungstheoretischer Natur: die aktuelle Komplexitätsklasse wird erweitert durch Hinzufügen von „Rechenpower“. Celia Wrathall setzt implizit[50] an die Stelle der Operatoren $NP$ und co-$NP$ Operatoren prädikatenlogischer Natur. Wir notieren diese Operatoren im Folgenden durch $(\exists)_{pol}$ und $(\forall)_{pol}$. Es wird sich zeigen, dass die iterative Anwendung der neuen Operatoren die gleiche Hierarchie aufbaut wie die iterative Anwendung der ursprünglichen Operatoren. Der Aufbau der polynomiellen Hierarchie mit den Operatoren $(\exists)_{pol}$ uns $(\forall)_{pol}$ führt zu einer prädikatenlogischen Charakterisierung von $\Sigma_k$ und $\Pi_k$ vermöge alternierender Quantorenketten der Länge $k$. Der Satz von Celia Wrathall zeigt gewissermaßen wie sich die Berechnungskomplexität einer Sprache der polynomiellen Hierarchie in prädikatenlogische „Beschreibungskomplexität“ übersetzt (und umgekehrt).

Aus Gründen der Bequemlichkeit treffen wir nun noch ein paar notationelle Vereinbarungen:

1. Das Kürzel „pol“ steht im Folgenden stets für ein (geeignet gewähltes) Polynom $p$. Wir setzen oBdA stets voraus, dass $p$ platz- und zeitkonstruierbar ist.

2. Das Kürzel $(\exists y)_{pol}$ bzw. $(\forall y)_{pol}$ stehe für Quantifizierung (vom Typ „$\exists$“ bzw. „$\forall$“ über alle Wörter (gebildet über einem festen Alpabet) einer Maximallänge $\text{pol}(|x|)$, wobei der Bezugsstring $x$ stets aus dem Kontext hervorgeht.

3. Die Notation $\langle z_1, z_2\rangle$ bezeichne eine feste „natürliche“ Kodierung des Wortpaares $(z_1, z_2)$ durch *ein* Wort. Induktiv definieren wir

$$\langle z_1, z_2, \ldots, z_k\rangle := \langle z_1, \langle z_2, \ldots, z_k\rangle\rangle \ .$$

   $\langle z_1, z_2, \ldots, z_k\rangle$ ist somit ein Kodierungsstring für das Tupel $(z_1, \ldots, z_k)$.

Wir merken an dieser Stelle kurz an, dass die induktive Definition von $\langle z_1, z_2, \ldots, z_k\rangle$ im Folgenden zwar bequem ist, aber evtl. dazu führt, dass die Länge der Kodierung eines $k$-Tupels exponentiell mit $k$ wächst.[51] Dies wird uns bei der Behandlung der polynomiellen Hierarchie nicht stören, da $k$ auf jeder Stufe der Hierarchie eine Konstante ist. Wann immer jedoch $k$ von $n$ abhängt, wollen wir annehmen, dass $\langle z_1, z_2, \ldots, z_k\rangle$ eine Kodierung ist, deren Länge polynomiell in der Gesamtlänge von $z_1, z_2, \ldots, z_k$ beschränkt ist.

[50] Wir weichen bei der Darstellung des Satzes von Celia Wrathall und seines Beweises von der Originalarbeit ab.

[51] zum Beispiel, wenn $\langle z_1, z_2\rangle$ die Kodierung ist, welche die Buchstaben von $z_1$ und $z_2$ verdoppelt und dazwischen eine einzelne „0“ als Trennzeichen einschiebt

## 8.1 Die prädikatenlogischen Operatoren

Nach der wortreichen Einleitung wird es nun Zeit konkret zu werden. Voilà, hier ist die genaue Definition der geheimnisvollen neuen Operatoren:

**Definition 8.1** *Sei $\mathcal{C}$ eine Klasse von Sprachen (über einem festen Alphabet).*

1. *$(\exists)_{pol}[\mathcal{C}]$ sei die Klasse aller Sprachen $L$, die sich für ein $L_0 \in \mathcal{C}$ in der Form*

$$L = \{x : (\exists y)_{pol}\langle y, x\rangle \in L_0\} \tag{31}$$

   *schreiben lassen.*

2. *$(\forall)_{pol}[\mathcal{C}]$ sei die Klasse aller Sprachen $L$, die sich für ein $L_0 \in \mathcal{C}$ in der Form*

$$L = \{x : (\forall y)_{pol}\langle y, x\rangle \in L_0\} \tag{32}$$

   *schreiben lassen.*

Da unsere Schreibkonventionen zwar bequem aber zugegebenermaßen auch etwas schlampig sind, geben wir hier (zum letzten Mal) zusätzlich die ausführliche Schreibweise an (um Missverständnissen vorzubeugen). Gleichung (31) bedeutet: Es existiert ein Polynom $p$, so dass die Sprache $L$ genau diejenigen Wörter $x$ enthält, zu denen ein Wort $y$ der Länge höchstens $p(|x|)$ existiert, so dass der Kodierungsstring für das Paar $(y, x)$ in $L_0$ zu liegen kommt. Gleichung (32) bedeutet: Es existiert ein Polynom $p$, so dass die Sprache $L$ genau diejenigen Wörter $x$ enthält, welche die Eigenschaft haben, dass für alle Wörter $y$ der Länge höchstens $p(|x|)$ der Kodierunsstring für das das Paar $(y, x)$ in $L_0$ zu liegen kommt. (Uff, es lebe die Schlampigkeit!)

Für einen Quantor $Q \in \{\exists, \forall\}$ bezeichne seine Negation $\bar{Q}$ den jeweils anderen „dualen“ Quantor:

$$\bar{Q} := \begin{cases} \forall, & \text{falls } Q = \exists, \\ \exists, & \text{falls } Q = \forall. \end{cases} \tag{33}$$

Wir werden im Folgenden häufig mit alternierenden Quantorenketten arbeiten. Um diese bequem zu notieren, definieren wir

$$Q_k := \begin{cases} \exists, & \text{falls } k \text{ ungerade ist}, \\ \forall, & \text{falls } k \text{ gerade ist}. \end{cases} \tag{34}$$

Trivialerweise gilt dann

$$\bar{Q}_k = \begin{cases} \forall, & \text{falls } k \text{ ungerade ist}, \\ \exists, & \text{falls } k \text{ gerade ist}. \end{cases} \tag{35}$$

Eine alternierende Quantorenkette der Länge $k$ hat dann die Form

$$(\exists)_{pol}(\forall)_{pol}(\exists)_{pol}\cdots(Q_k)_{pol} \ ,$$

wenn sie mit $(\exists)_{pol}$ beginnt. Beginnt sie mit $(\forall)_{pol}$ hat sie die Form

$$(\forall)_{pol}(\exists)_{pol}(\forall)_{pol}\cdots(\bar{Q}_k)_{pol} \ .$$

Die analogen Schreibweisen benutzen wir auch für $(\exists y)_{pol}$ und $(\forall y)_{pol}$.

Es folgen ein paar Aufwärmübungen zu den neuen Definitionen und Schreibweisen.

**Lemma 8.2** *Beim Übergang zur Komplementärklasse gelten die folgenden Regeln:*

$$co\text{-}(\exists)_{pol}(\forall)_{pol}(\exists)_{pol}\cdots(Q_k)_{pol}[\mathcal{C}] = (\forall)_{pol}(\exists)_{pol}(\forall)_{pol}\cdots(\bar{Q}_k)_{pol}[co\text{-}\mathcal{C}] \ . \quad (36)$$

$$co\text{-}(\forall)_{pol}(\exists)_{pol}(\forall)_{pol}\cdots(\bar{Q}_k)_{pol}[\mathcal{C}] = (\exists)_{pol}(\forall)_{pol}(\exists)_{pol}\cdots(Q_k)_{pol}[co\text{-}\mathcal{C}] \ . \quad (37)$$

**Beweis** Wir beschränken uns auf den Nachweis von (36). Der Nachweis von (37) lässt sich völlig analog führen.
$k$-fache Anwendung von Definition 8.1 führt zu folgender Einsicht. Zu einer Sprache $L \in (\exists)_{pol}(\forall)_{pol}(\exists)_{pol}\cdots(Q_k)_{pol}[\mathcal{C}]$ gibt es eine Sprache $L_0 \in \mathcal{C}$, so dass $L$ gegeben ist durch

$$L = \{x : (\exists y_1)_{pol}(\forall y_2)_{pol}(\exists y_3)_{pol}\cdots(Q_k y_k)_{pol}\langle y_1, \ldots, y_k, x\rangle \in L_0\} \ . \quad (38)$$

Die Komplementärklasse co-$\mathcal{C}$ enthält anstelle von $L$ die Sprache $\bar{L}$, deren Definitionsbedingung sich durch logische Negation der Definitionsbedingung von $L$ ergibt:

$$\bar{L} = \{x : (\forall y_1)_{pol}(\exists y_2)_{pol}(\forall y_3)_{pol}\cdots(\bar{Q} y_k)_{pol}\langle y_1, \ldots, y_k, x\rangle \in \bar{L}_0 \ . \quad (39)$$

Hierbei haben wir die de Morgan'schen Regeln angewendet, welche dazu führen, dass jeder Quantor durch sein duales Gegenstück ersetzt und die atomore Bedingung $\langle y_1, \ldots, y_k, x\rangle \in L_0$ am Ende der Quantorenkette zu $\langle y_1, \ldots, y_k, x\rangle \in \bar{L}_0$ logisch negiert wird. Durch $k$-fache Anwendung von Definition 8.1 erkennen wir jetzt, dass die Bedingung (39) gerade Sprachen aus $(\forall)_{pol}(\exists)_{pol}(\forall)_{pol}\cdots(\bar{Q}_k)_{pol}[\text{co-}\mathcal{C}]$ charakterisiert. □

Wegen co-$P = P$ ergibt sich unmittelbar die

**Folgerung 8.3**

$$co\text{-}(\exists)_{pol}(\forall)_{pol}(\exists)_{pol}\cdots(Q_k)_{pol}[P] = (\forall)_{pol}(\exists)_{pol}(\forall)_{pol}\cdots(\bar{Q}_k)_{pol}[P] \ .$$

$$co\text{-}(\forall)_{pol}(\exists)_{pol}(\forall)_{pol}\cdots(\bar{Q}_k)_{pol}[P] = (\exists)_{pol}(\forall)_{pol}(\exists)_{pol}\cdots(Q_k)_{pol}[P] \ .$$

Ähnlich leicht erhalten wir die

**Folgerung 8.4** *$NP = (\exists)_{pol}[P]$ und co-$NP = (\forall)_{pol}[P]$.*

**Beweis** Die Aussage $NP = (\exists)_{pol}[P]$ entpuppt sich mit etwas Nachdenken als unsere gute alte Charakterisierung von $NP$ mit Hilfe von polynomiell verifizierbaren Relationen (Stichwort: Rate- und Verifikationsprinzip). Die zweite Aussage ergibt sich dann mit Hilfe von Folgerung 8.3: co-$NP$ = co-$(\exists)_{pol}[P] = (\forall)_{pol}[P]$. □

Wir stellen uns als nächstes die Frage, ob die Operatoren $(\exists)_{pol}$ und $(\forall)_{pol}$ zu einer (evtl. unechten) Erweiterung einer Sprachklasse führen. Vorsicht ist insofern geboten, als man künstliche Sprachklassen $\mathcal{C}$ angeben kann, für welche die Inklusionsbeziehung $\mathcal{C} \subseteq (\exists)_{pol}[\mathcal{C}]$ bzw. $\mathcal{C} \subseteq (\forall)_{pol}[\mathcal{C}]$ *nicht* gilt (s. Übungen). Andererseits beweisen wir folgendes

**Lemma 8.5** *$\mathcal{C}$ besitze die die folgende Abschluss-Eigenschaft. Falls $L \in \mathcal{C}$, dann gehöre auch die Sprache*

$$L_\epsilon := \{\langle \epsilon, x\rangle : x \in L\}$$

*zu $\mathcal{C}$. (Hierbei bezeichne $\epsilon$ wie üblich das leere Wort.) Unter dieser Voraussetzung gilt $\mathcal{C} \subseteq (\exists)_{pol}[\mathcal{C}]$ und $\mathcal{C} \subseteq (\forall)_{pol}[\mathcal{C}]$.*

**Beweis** Sei $L \in \mathcal{C}$ und somit auch $L_\epsilon \in \mathcal{C}$. Um $\mathcal{C} \subseteq (\exists)_{pol}[\mathcal{C}]$ einzusehen, genügt es $L \in (\exists)_{pol}[\mathcal{C}]$ einzusehen. Dies ist aber klar, da wir $L$ in folgender Form schreiben können

$$L = \{x : (\exists y)_{pol} \langle y, x \rangle \in L_\epsilon\} \ .$$

Das geeignete Polynom, das sich hinter dem Index „pol" von $(\exists y)_{pol}$ versteckt, kann in diesem Fall als das Nullpolynom $p \equiv 0$ gewählt werden. In der Tat existiert ja für die Wörter $x$ von $L$ (und nur für diese) ein Wort $y$ der Maximallänge 0 (nämlich das leere Wort $y = \epsilon$) mit $\langle y, x \rangle \in L_\epsilon$.
Der Nachweis von $\mathcal{C} \subseteq (\forall)_{pol}[\mathcal{C}]$ geschieht völlig analog (oder durch Dualisierung). □

Es ist leicht einzusehen, dass alle Sprachklassen der polynomiellen Hierarchie (wie überhaupt alle „vernünftig definierten" Komplexitätsklassen) die in Lemma 8.5 geforderte Abschluss-Eigenschaft besitzen (s. Übungen). Für $\Sigma_k$ und $\Pi_k$ zeigen wir aber darüberhinaus das

**Lemma 8.6** *Für alle $k \geq 1$ gilt $\Sigma_k = (\exists)_{pol}[\Sigma_k]$ und $\Pi_k = (\forall)_{pol}[\Pi_k]$.*

**Beweis** Wir zeigen zunächst $\Sigma_k = (\exists)_{pol}[\Sigma_k]$. $\Sigma_k \subseteq (\exists)_{pol}[\Sigma_k]$ ergibt sich aus Lemma 8.5. Zum Nachweis von $(\exists)_{pol}[\Sigma_k] \subseteq \Sigma_k$ sei $L \in (\exists)_{pol}[\Sigma_k]$ für ein $L_0 \in \Sigma_k$ gegeben durch

$$L = \{x : (\exists y)_{pol} \langle y, x \rangle \in L_0\} \ .$$

Sei $M_0$ ein Akzeptor von $L_0 \in \Sigma_k$. Wegen $k \geq 1$ ist $M_0$ eine polynomielle NOTM mit einem Orakel für eine Sprache $L_0' \in \Sigma_{k-1}$. (Dies schließt in einer etwas verklausulierten Form auch den Fall $k = 1$ ein.) Folgende NOTM $M$, versehen mit einem Orakel für $L_0' \in \Sigma_{k-1}$, ist ein polynomieller Akzeptor von $L$:

1. Rate ein Wort $y$ der Länge $\text{pol}(|x|)$.
2. Benutze die NOTM $M_0$ mit dem $L_0'$-Orakel, um $\langle y, x \rangle \in L_0$ zu verifizieren (falls möglich).

Man überlegt sich leicht, dass $M$ auf einer Eingabe $x$ genau dann eine akzeptierende Rechnung besitzt, wenn $x \in L$.
Mit unserem Hintergrundwissen ergibt sich nun $\Pi_k = (\forall)_{pol}[\Pi_k]$ leicht durch Dualisierung:

$$\Pi_k = \text{co-}\Sigma_k = \text{co-}(\exists)_{pol}[\Sigma_k] = (\forall)_{pol}[\text{co-}\Sigma_k] = (\forall)_{pol}[\Pi_k] \ .$$

□

Im Beweis von $\Sigma_k = (\exists)_{pol}[\Sigma_k]$ haben wir (salopp gesprochen) ausgenutzt, dass eine NOTM ihren Nichtdeterminismus dazu verwenden kann, den $(\exists)_{pol}$-Operator (durch Raten eines geeigneten Wortes $y$) überflüssig zu machen.

Aus beweistechnischen Gründen definieren wir jetzt die polynomielle Hierarchie ein zweites Mal (wie sich später herausstellen wird), wobei diesmal die neuen prädikatenlogischen Operatoren zum Einsatz kommen:

geplant für 08.07

**Definition 8.7** *Die Sprachklassen $\Sigma'_k$ und $\Pi'_k$ sind für $k \geq 0$ induktiv definiert wie folgt:*

$$\begin{aligned} \Sigma'_0 &:= \Pi'_0 := P \ . \\ \Sigma'_k &:= (\exists)_{pol}[\Pi'_{k-1}] \ . \\ \Pi'_k &:= (\forall)_{pol}[\Sigma'_{k-1}] \ . \end{aligned}$$

Der Satz von Celia Wrathall besagt, dass $\Sigma'_k = \Sigma_k$ und $\Pi'_k = \Pi_k$, d.h., die „neue“ Hierachie ist die „alte“ Hierarchie in verkleideter Form. Der Beweis dieser Aussage ist nichttrivial und wird in Abschnitt 8.2 geführt.

Iterierte Anwendung von Definition 8.7 liefert sofort eine Charakterierung der Klassen $\Sigma'_k$ und $\Pi'_k$ vermöge alternierender Quantorenketten:

**Lemma 8.8** *Für alle $k \geq 1$ gilt:*

$$\begin{aligned} L \in \Sigma'_k &\Leftrightarrow \exists L_0 \in P : L = \{x : (\exists y_1)_{pol}(\forall y_2)_{pol}(\exists y_3)_{pol} \cdots (Q_k y_k)_{pol} \langle y_1, \ldots, y_k, x\rangle \in L_0\} \ . \\ L \in \Pi'_k &\Leftrightarrow \exists L_0 \in P : L = \{x : (\forall y_1)_{pol}(\exists y_2)_{pol}(\forall y_3)_{pol} \cdots (\bar{Q} y_k)_{pol} \langle y_1, \ldots, y_k, x\rangle \in L_0\} \ . \end{aligned}$$

Aus der Charakterisierung der Klassen $\Sigma'_k, \Pi'_k$ mit alternierenden Quantorenketten lassen sich unmittelbar eine paar Schlussfolgerungen ziehen.

**Folgerung 8.9**

1. *Für alle $k \geq 0$ gilt $\Pi'_k = co\text{-}\Sigma'_k$.*
2. *Für alle $k \geq 1$ gilt $\Sigma'_k = (\exists)_{pol}[\Sigma'_k]$.*
3. *Für alle $k \geq 1$ gilt $\Pi'_k = (\forall)_{pol}[\Pi'_k]$.*
4. *Für alle $k \geq 0$ gilt $\Sigma'_k \subseteq \Sigma'_{k+1} \cap \Pi'_{k+1}$ und $\Pi'_k \subseteq \Pi'_{k+1} \cap \Sigma'_{k+1}$.*

**Beweis** Wir geben jeweils nur die entscheidende Idee an. (Die technischen Details sind leicht auszufüllen.)

1. Kombiniere Lemma 8.8 mit Lemma 8.2.
2. Die Grundidee ist, dass $(\exists)_{pol}(\exists)_{pol}$ gleichwertig zu $(\exists)_{pol}$ ist, da sich zwei aufeinanderfolgende Quantoren des gleichen Typs miteinander verschmelzen lassen.
3. Das geschilderte Verschmelzungsargument gilt natürlich auch für $(\forall)_{pol}$-Quantoren. Alternativ kann $\Pi'_k = (\forall)_{pol}[\Pi'_k]$ auch leicht durch Dualisierung gezeigt werden, wobei dann die bereits bewiesenen Aussagen $\Pi'_k = \text{co-}\Sigma'_k$ und $\Sigma'_k = (\exists)_{pol}[\Sigma'_k]$ benutzt werden.
4. Die „Beschreibungskraft“ einer Quantorenkette kann nicht abnehmen, wenn wir sie um einen neuen Quantor verlängern.

□

Die Ausführung der technischen Details zu dem „Verschmelzungsargument“ sind Bestandteil der Übungen. Man kann sich dabei an der technischen Beweisführung des folgenden Lemmas orientieren.

**Lemma 8.10** *Für jedes $k \geq 0$ sind die Klassen $\Sigma'_k$ und $\Pi'_k$ abgeschlossen unter Vereinigung und Durchschnitt von Sprachen.*

**Beweis** Für $k = 0$ ist die Aussage offensichtlich. Sei $k \geq 1$. Seien $L', L'' \in \Sigma'_k$. Gemäß Lemma 8.8 existieren dann Sprachen $L'_0, L''_0 \in P$, so dass

$$\begin{aligned} L' &= \{x : (\exists y'_1)_{pol}(\forall y'_2)_{pol}(\exists y'_3)_{pol} \cdots (Q_k y'_k)_{pol} \langle y'_1, \ldots, y'_k, x\rangle \in L'_0\} \ . \\ L'' &= \{x : (\exists y''_1)_{pol}(\forall y''_2)_{pol}(\exists y''_3)_{pol} \cdots (Q_k y''_k)_{pol} \langle y''_1, \ldots, y''_k, x\rangle \in L''_0\} \ . \end{aligned}$$

Hieraus ergibt sich

$$\begin{aligned} L' \bigcup L'' &= \{x : (\exists y_1)_{pol}(\forall y_2)_{pol}(\exists y_3)_{pol} \cdots (Q_k y_k)_{pol} \, (y_1 = \langle y'_1, y''_1\rangle \wedge \cdots \wedge y_k = \langle y'_k, y''_k\rangle) \\ &\quad \wedge \ (\text{Längenbedingung}) \ \wedge (\langle y'_1, \ldots, y'_k, x\rangle \in L'_0 \vee \langle y''_1, \ldots, y''_k, x\rangle \in L''_0)\} \ . \\ L' \cap L'' &= \{x : (\exists y_1)_{pol}(\forall y_2)_{pol}(\exists y_3)_{pol} \cdots (Q_k y_k)_{pol} \, (y_1 = \langle y'_1, y''_1\rangle \wedge \cdots \wedge (y_k = \langle y'_k, y''_k\rangle) \\ &\quad \wedge \ (\text{Längenbedingung}) \ \wedge (\langle y'_1, \ldots, y'_k, x\rangle \in L'_0 \wedge \langle y''_1, \ldots, y''_k, x\rangle \in L''_0)\} \ . \end{aligned}$$

Hierbei steht (Längenbedingung) für eine Bedingung, welche kontrolliert, dass $y'_1, \ldots, y'_k$ und $y''_1, \ldots, y''_k$ die Maximallänge besitzen, die durch die Definition von $L'$ bzw. $L''$ vorgeschrieben ist. Wenn also $p'_i$ bzw. $p''_i$ die konkreten Polynome sind, welche die Maximallängen von $y'_i$ bzw. $y''_i$ festlegen, dann ist (Längenbedingung) die logische Konjunktion der Bedingungen

$$|y'_i| \leq p'_i(|x|) \wedge |y''_i| \leq p''_i(|x|)$$

für $i = 1, \ldots, k$. Man beachte, dass sowohl die Längenbedingung als auch die logische Konjunktion der Bedingungen $y_i = \langle y'_i, y''_i\rangle$ für $i = 1, \ldots, k$ in Polynomialzeit getestet werden kann.[52] Eine Inspektion der obigen Definitionsbedingungen für $L' \cup L''$ bzw. $L' \cap L''$ verbunden mit einer erneuten Anwendung von Lemma 8.8 liefert $L' \cup L'', L' \cap L'' \in \Sigma'_k$.
Die entsprechenden Abschlusseigenschaften von $\Pi'_k$ ergeben sich leicht durch Dualisierung (s. Übungen). □

## 8.2 Der Satz von Celia Wrathall und sein Beweis

Im vorangegangenen Abschnitt haben wir genug Vorarbeit geleistet, um jetzt dem Hauptresultat zu Leibe zu rücken:

**Satz 8.11 (Wrathall, 1977)** *Für alle $k \geq 0$ gilt $\Sigma'_k = \Sigma_k$ und $\Pi'_k = \Pi_k$.*

**Beweis** Für $k = 0$ ist die Behauptung richtig, da auf Stufe 0 alle beteiligten Sprachklassen mit $P$ zusammenfallen. Wir können daher für ein beliebiges aber festes $k \geq 1$ induktiv $\Sigma'_{k-1} = \Sigma_{k-1}$ und $\Pi'_{k-1} = \Pi_{k-1}$ voraussetzen. Aus $\Sigma'_k = \Sigma_k$ ergibt sich $\Pi'_k = \Pi_k$ durch Dualisierung:

$$\Pi'_k = \text{co-co-}\Pi'_k = \text{co-}\Sigma'_k = \text{co-}\Sigma_k = \Pi_k \ .$$

[52] Beim effizienten Testen der Längenbedingung nutzen wir die Platzkonstruierbarkeit der betreffenden Polynome aus.

Es genügt also der induktive Nachweis von $\Sigma'_k = \Sigma_k$.
Die Inklusion $\Sigma'_k \subseteq \Sigma_k$ ergibt sich wie folgt:

$$\Sigma'_k = (\exists)_{pol}[\Pi'_{k-1}] = (\exists)_{pol}[\Pi_{k-1}] \subseteq (\exists)_{pol}[\Sigma_k] = \Sigma_k \quad .$$

Hierbei haben wir nacheinander die Definition von $\Sigma'_k$, die Induktionsvoraussetzung, die Inklusion $\Pi_{k-1} \subseteq \Sigma_k$ (die nach Anwendung des $(\exists)_{pol}$-Operators gültig bleibt) und schließlich Lemma 8.6 angewendet.
Zum Nachweis der Inklusion $\Sigma_k \subseteq \Sigma'_k$ werden wir härter arbeiten müssen. Sei $L \in \Sigma_k = NP[\Sigma_{k-1}]$. Sei weiter $M$ die polynomielle NOTM, die in Kooperation mit einem Orakel für $L' \in \Sigma_{k-1}$ als Akzeptor von $L$ auftritt. $M$ darf als nichtdeterministische Maschine „raten“ und als Orakel-Maschine zusätzlich pol($|x|$)-oft ihr $L'$-Orakel befragen. Zum Nachweis von $L \in \Sigma'_k = (\exists)_{pol}[\Pi'_{k-1}]$ müssen wir die Fähigkeiten von $M$ irgendwie mit Hilfe des $(\exists)_{pol}$-Operators nachbilden. Die Korrespondenz von „Raten“ und $(\exists)_{pol}$-Quantifizierung schreckt uns nicht wirklich, da wir diese vom Rate- und Verifikationsprinzip her gewohnt sind. Bleibt aber das Problem, auch die Kommunikation zwischen $M$ und dem $L'$-Orakel mit Hilfe des $(\exists)_{pol}$-Quantors nachzubilden. Wir verwenden dabei die folgende

**Zentrale Beweisstrategie** Antizipiere die Ratebits von $M$ *und* die gesamte Kommunikation zwischen $M$ und dem $L'$-Orakel durch einen einzigen (raffiniert entworfenen) String polynomiell beschränkter Länge und wende die $(\exists)_{pol}$-Quantifizierung auf diesen String an.

Es folgt die technische Umsetzung. $M$ stoppe nach höchstens $T = \text{pol}(|x|)$ Rechenschritten und habe oBdA pro Schritt nur zwei Handlungsalternativen. Die Rechnung von $M$ hängt von folgenden Faktoren ab:

- den nichtdeterministischen Entscheidungen repräsentiert durch einen Ratestring $y \in \{0,1\}^T$,
- den JA/NEIN-Antworten des $L'$-Orakels.

Für $r, s \leq T$ seien $u, v$ Strings von der Form

$$u = \langle u_1, \ldots, u_r \rangle \text{ und } v = \langle v_1, \ldots, v_s \rangle \quad .$$

Der String $\langle y, u, v, x \rangle$ heiße *konsistentes Rechenprotokoll*, falls folgendes gilt:

- Wenn $M$ auf Eingabe $x$ gemäß Ratestring $y$ rechnet und wenn alle Orakelanfragen nach einem Wort aus $U := \{u_1, \ldots, u_r\}$ mit JA und alle Orakelanfragen nach einem Wort aus $V := \{v_1, \ldots, v_s\}$ mit NEIN beantwortet werden, dann stellt $M$ nur Anfragen nach Wörtern aus $U \cup V$ und akzeptiert nach höchstens $T$ Schritten.

Beachte, dass wir in dieser Definition *nicht* verlangen, dass die Antworten korrekt sind. Stattdessen sind die (evtl. falschen) Antworten durch die Strings $u, v$ vorgegeben.[53] Die Sprache

$$L_0 := \{\langle y, u, v, x \rangle : \langle y, u, v, x \rangle \text{ ist ein konsistentes Rechenprotokoll}\}$$

[53] Dies ist eine subtile Angelegenheit: unsere Analyse der NOTM $M$ beschäftigt sich mit dem Verhalten von $M$, für den Fall, dass wir sie vom Orakel abkoppeln und ihr auf andere Weise Antworten verabreichen. In mathematischen Beweisen ist alles erlaubt! Warten wir mal ab, wozu solche dubiosen Manöver gut sind.

gehört daher zu $P$.[54] Wir können nämlich auf Eingaben der Form $\langle y, u, v, x\rangle$ eine effiziente Simulation von $M$ auf $x$ starten[55], welche die konsistenten Rechenprotokolle (und nur diese) akzeptiert. Wir betrachten zwei Fälle:

**Fall 1** $M$ (mit Ratestring $y$) stelle während der ersten $T$ simulierten Schritte nur Orakelanfragen aus $U \cup V$.
Dann können wir ohne Probleme $T$ Schritte von $M$ simulieren, da die Antwort auf Orakelanfragen effizient aus dem String $u$ bzw. $v$ abgelesen werden kann. Wenn $M$ nach spätestens $T$ Schritten (akzeptierend oder verwerfend) stoppt, dann stoppe die Simulation mit dem gleichen Ergebnis. Wenn $M$ nach $T$ Schritten nicht stoppt[56], dann stoppe die Simulation nach $T$ simulierten Schritten verwerfend.[57]

**Fall 2** $M$ (mit Ratestring $y$) stelle während der ersten $T$ simulierten Schritte eine Orakelanfrage außerhalb $U \cup V$.
Bei der ersten Anfrage dieser Art stoppe die Simulation verwerfend.

Aus der Definition der konsistenten Rechenprotokolle ergibt sich direkt, dass unsere Simulation diese und nur diese akzeptiert. Somit gilt $L_0 \in P$.
Wir haben mit dem Nachweis von $L_0 \in P$ bereits einen Teilsieg errungen. Auf Dauer können wir uns aber nicht davor drücken, die Korrektheit von Antworten auf Orakelanfragen zu kontrollieren. Dieser Kontrolle dienen die folgenden beiden Sprachen:

$$\begin{aligned} L_1 &:= \{\langle y, u, v, x\rangle : u_1, \ldots, u_r \in L'\} \ . \\ L_2 &:= \{\langle y, u, v, x\rangle : v_1, \ldots, v_s \in \bar{L}'\} \ . \end{aligned}$$

Wegen $L' \in \Sigma_{k-1}$ und $\bar{L}' \in \text{co-}\Sigma_{k-1} = \Pi_{k-1}$ ist unter Verwendung der Induktionsvoraussetzung und des Lemmas 8.5 leicht zu sehen (s. Übung), dass folgende Aussagen gelten:

$$\begin{aligned} L_1 \in \Sigma_{k-1} &= \Sigma'_{k-1} \subseteq \Sigma'_k \ . \\ L_2 \in \Pi_{k-1} &= \Pi'_{k-1} \subseteq \Sigma'_k \ . \end{aligned}$$

Da $L_0 \in P \subseteq \Sigma'_k$ und $\Sigma'_k$ abgeschlossen unter Durchschnittsbildung ist, gilt dann auch $L_0 \cap L_1 \cap L_2 \in \Sigma'_k$. Weiterhin sind $L_0, L_1, L_2$ gerade so entworfen, dass

$$x \in L \Leftrightarrow (\exists w)_{pol} w = \langle y, u, v\rangle \wedge \langle y, u, v, x\rangle \in L_0 \cap L_1 \cap L_2 \ .$$

Hieraus ergibt sich leicht $L \in (\exists)_{pol}[\Sigma'_k]$ und wegen $(\exists)_{pol}[\Sigma'_k] = \Sigma'_k$ (gemäß Folgerung 8.9) folgt schließlich $L \in \Sigma'_k$. Da $L$ eine beliebige Sprache aus $\Sigma_k$ ist, ergibt sich die angestrebte Inklusion $\Sigma_k \subseteq \Sigma'_k$. □

In Verbindung mit Lemma 8.8 erhalten wir unmittelbar die

[54] Dazu diente das Manöver! Der Nichtdeterminismus wird mit Hilfe des Strings $y$ eliminiert; das Orakel wird mit Hilfe der Strings $u, v$ überflüssig gemacht; bleibt eine deterministische polynomielle Rechnung.
[55] Eingaben, die schon syntaktisch von der Form $\langle y, u, v, x\rangle$ abweichen, können sofort verworfen werden.
[56] was man wegen der $M$ zugespielten falschen Antworten nicht ausschließen kann
[57] An dieser Stelle verwenden wir die Zeitkonstruierbarkeit von $T = \text{pol}(|x|)$.

**Folgerung 8.12** *Für alle $k \geq 1$ gilt:*

$$L \in \Sigma_k \;\Leftrightarrow\; \exists L_0 \in P : L = \{x : (\exists y_1)_{pol}(\forall y_2)_{pol}(\exists y_3)_{pol} \cdots (Q_k y_k)_{pol} \langle y_1, \ldots, y_k, x\rangle \in L_0\}$$
$$L \in \Pi_k \;\Leftrightarrow\; \exists L_0 \in P : L = \{x : (\forall y_1)_{pol}(\exists y_2)_{pol}(\forall y_3)_{pol} \cdots (\bar{Q} y_k)_{pol} \langle y_1, \ldots, y_k, x\rangle \in L_0\}$$

Die „Rechenkraft“ einer zu $\Sigma_k$ korrespondierenden polynomiellen NOTM (mit einem Orakel aus $\Sigma_{k-1}$) entspricht also exakt der „Beschreibungskraft“ einer alternierenden mit $(\exists)_{pol}$ beginnenden Quantorenkette der Länge $k$, gefolgt von einer in Polynomialzeit überprüfbaren Aussage. Die analoge Entsprechung finden wir für die Klasse $\Pi_k$ und alternierende mit $(\forall)_{pol}$ beginnende Quantorenketten der Länge $k$. Es hat sich somit eine erstaunliche Querbeziehung zwischen Berechnungskomplexität und prädikatenlogischer Beschreibungskomplexität ergeben.

## 8.3 Anwendungen des Satzes von Celia Wrathall

Für die harte Arbeit, die zum Beweis des Satzes von Celia Wrathall nötig war, werden wir in diesem Abschnitt entlohnt. Einige elegante Anwendungen dieses Satzes werden uns nun zufallen wie reife Früchte. Eine erste Anwendung besteht darin, die Einordnung einer Sprache in die polynomielle Hierarchie nicht durch „Programmierung“ einer geeigneten OTM vorzunehmen, sondern durch Angabe einer geeigneten „Definition“ der Sprache. Die Länge der alternierenden Quantorenkette in der Definitionsbedingung wird uns dann verraten, zu welcher Stufe der polynomiellen Hierchie die Sprache gehört. Eine zweite Anwendung ist die Analyse, unter welchen Bedingungen die polynomielle Hierachie auf eine endliche Stufe kollabiert. Eine dritte Anwendung besteht in der Angabe von $\Sigma_k$- oder $\Pi_k$-vollständigen Problemen und im Nachweis der Vollständigkeit.

### 8.3.1 Anwendung 1: Deskriptiver Nachweis der Mitgliedschaft in PH

Ein „algorithmischer“ Nachweis von $L \in PH$ besteht in der Angabe einer passenden OTM. Ein „deskriptiver“ Nachweis von $L \in PH$ macht sich Folgerung 8.12 zunutze und beschreibt $L$ mit einer passenden prädikatenlogischen Formel. Wir illustrieren dies an einem

**Beispiel 8.13** *Wir betrachten erneut die Sprache MEC (Minimum Equivalent Circuit). Wir haben früher bereits einen algorithmischen Nachweis von $MEC \in \Sigma_2$ geführt, und zwar durch Angabe einer polynomiellen NOTM, die mit Hilfe eines $SAT^*$-Orakels als Akzeptor von MEC auftrat. Es folgt nun ein deskriptiver Nachweis für die gleiche Aussage:*

$$MEC = \{\langle S, k\rangle : (\exists S')_{pol}(\forall a)_{pol} \langle S', a, S, k\rangle \in MEC_0\} \;,$$

*wobei $MEC_0$ die Sprache aller Strings der Form $\langle S', a, S, k\rangle$ sei, so dass zusätzlich die folgenden Bedingungen gelten:*

1. *$S$ repräsentiert einen Booleschen Schaltkreis. Die Anzahl der Eingangsknoten in $S$ sei mit $n$ bezeichnet.*

2. *$k$ repräsentiert eine nicht-negative ganze Zahl.*

3. *$S'$ repräsentiert einen Booleschen Schaltkreis mit $n$ Eingangsknoten und höchstens $k$ Bausteinen.*

4. *$a \in \{0,1\}^n$ und die Schaltkreise $S$ und $S'$ berechnen auf Eingabe $a$ das gleiche Ausgabebit.*

*Offensichtlich gilt $MEC_0 \in P$. Aus Folgerung 8.12 ergibt sich unmittelbar $MEC \in \Sigma_2$.*

### 8.3.2 Anwendung 2: Hinreichende Bedingungen für einen Kollaps von PH

Es wird vermutet, dass die polynomielle Hierarchie aus unendlich vielen voneinander verschiedenen Stufen besteht. Der folgende Satz formuliert eine hinreichende Bedingung für die Negation dieser Vermutung.

**Satz 8.14** *Falls $\Sigma_k = \Pi_k$ für ein $k \geq 1$, dann kollabiert PH auf den $k$-ten Level, d.h., dann gilt*

$$PH = \bigcup_{i=0}^{\infty} \Sigma_i = \bigcup_{i=0}^{k} \Sigma_i = \Sigma_k \quad .$$

**Beweis** Wir beschränken uns auf die Angabe der zentralen Beweisidee. Die technischen Details sind leicht auszufüllen. Die Klasse $\Sigma_k$ ist im Sinne von Folgerung 8.12 durch eine alternierende Quantorenkette vom Typ $(\exists)_{pol}(\forall)_{pol}(\exists)_{pol}\cdots(Q_k)_{pol}$ gekennzeichnet; die Klasse $\Pi_k$ ist entsprechend durch eine alternierende Quantorenkette vom Typ $(\forall)_{pol}(\exists)_{pol}(\forall)_{pol}\cdots(\bar{Q}_k)_{pol}$ gekennzeichnet. Unter der Voraussetzung $\Sigma_k = \Pi_k$ können wir dann aber auch Sprachen aus $\Pi_k$ mit Hilfe von Quantorenketten vom Typ $(\exists)_{pol}(\forall)_{pol}(\exists)_{pol}\cdots(Q_k)_{pol}$ beschreiben. Demzufolge können wir Sprachen aus $\Sigma_{k+1} = \Sigma'_{k+1} = (\exists)_{pol}[\Pi'_k] = (\exists)_{pol}[\Pi_k]$ auch mit Hilfe von Quantorenketten vom Typ $(\exists)_{pol}(\exists)_{pol}(\forall)_{pol}(\exists)_{pol}\cdots(Q_k)_{pol}$ beschreiben. Durch Verschmelzung der ersten beiden $(\exists)_{pol}$-Quantoren erhalten wir eine Beschreibung vom Typ $(\exists)_{pol}(\forall)_{pol}(\exists)_{pol}\cdots(Q_k)_{pol}$, welche ja (wie bereits erwähnt) gerade Sprachen aus $\Sigma_k$ kennzeichnet. Folglich hat der zusätzliche $(\exists)_{pol}$-Quantor, den Sprachen aus $\Sigma_{k+1}$ zur Verfügung haben, „nichts gebracht“ und es gilt $\Sigma_{k+1} = \Sigma_k$. Aus Dualitätsgründen gilt dann auch $\Pi_{k+1} = \Pi_k$. Die Gleichheit $\Sigma_k = \Pi_k$ auf Stufe $k$ sorgt auf Stufe $k+1$ für $\Sigma_k = \Sigma_{k+1} = \Pi_{k+1}$. Iteration dieses Argumentes (genauer: vollständige Induktion) liefert $\Sigma_k = \Sigma_{k+j} = \Pi_{k+j}$ für alle $j \geq 0$. Es folgt $PH = \cup_{i=0}^{k}\Sigma_k$ und wegen $\Sigma_0 \subseteq \Sigma_1 \subseteq \Sigma_2 \subseteq \cdots$ natürlich auch $PH = \Sigma_k$. □

Aus Satz 8.14 ergibt sich ein kleiner Beitrag zu der $P \neq NP$-Debatte:

**Folgerung 8.15** *$P \neq NP \Leftrightarrow P \neq PH$.*

**Beweis** Wegen $P \subseteq NP \subseteq PH$ folgt aus $P \neq NP$ natürlich $P \subset NP \subseteq PH$ und somit auch $P \neq PH$. (Hierbei ist „$\subset$“ das Zeichen für *echte* Inklusion.) Zum Nachweis von $P \neq PH \implies P \neq NP$ genügt es freilich $P = NP \implies P = PH$ zu zeigen. Wie früher bereits nachgewiesen, impliziert $P = NP$ die Aussage $NP =$ co-$NP$. Wegen $NP = \Sigma_1$

und co-$NP = \Pi_1$ ergibt sich mit Satz 8.14 ein Kollaps von $PH$ auf den ersten Level, d.h., $PH = NP$. Zusammen mit der Voraussetzung $P = NP$ ergibt sich nun auch $PH = P$. □

Folgerung 8.15 bedeutet, dass wir die berühmte $P \neq NP$-Vermutung im Prinzip durch den Nachweis von $P \neq PH$ beweisen könnten. Da $PH$ eine viel mächtigere Komplexitätsklasse ist als $NP$, da sie ja $NP$ bereits auf Stufe 1 zur Teilklasse hat, könnte die Separation von $P$ und $PH$ (durch Nachweis der Existenz einer Sprache in $PH \setminus P$) evtl. leichter zu erreichen sein als die Separation von $NP$ und $P$.

### 8.3.3 Anwendung 3: Vollständige Probleme in PH

Das $NP$-vollständige Problem SAT lässt sich mit einer $(\exists)_{pol}$-quantifizierten Booleschen Formel (in konjunktiver Normalform) beschreiben, da wir nach der Existenz einer erfüllenden Belegung zu einer gegebenen CNF-Formel fragen. Um zu $\Sigma_k$- oder $\Pi_k$-vollständigen Problemen zu gelangen ist es naheliegend mit quantifizierten Booleschen Formeln zu operieren, wobei die Quantifizierung über eine alternierende Quantorenkette der Länge $k$ erfolgt. Um solche Formeln bequem zu notieren, teilen wir die Booleschen Variablen in $k$ Typen ein:

**Definition 8.16** *Sei $F$ eine Boolesche Formel und $m$ die Anzahl der in $F$ verwendeten Booleschen Variablen. Eine doppelt indizierte Boolesche Variable der Form $v_{ij}$ heiße* Variable vom Typ *$i$. Wir sagen $F$ ist eine Boolesche Formel* mit $k$ Variablentypen*, wenn $m_1, \ldots, m_k$ existieren, so dass $m = m_1 + \cdots + m_k$ und $F$ verwendet genau $m_i$ Variablen vom Typ $i$. Dies seien oBdA die Variablen $\bar{v}_i = (v_{i1}, \ldots, v_{im_i})$ für $i = 1, \ldots, k$. Wir schreiben dann auch $F(\bar{v}_1, \ldots, \bar{v}_k)$ statt $F$, um die Abhängigkeit von den Variablen hervorzuheben.*

Die folgenden Sprachen sind eine naheliegende Verallgemeinerung der Sprache SAT, wobei an die Stelle eines einzelnen $(\exists)_{pol}$-Quantors eine alternierende Quantorenkette tritt und wir nicht mehr darauf bestehen, dass die Formel in konjunktiver Normalform (also eine CNF-Formel) ist.

**Definition 8.17** *Die Sprache $\mathcal{B}_k$ enthalte alle (Kodierungen von) Booleschen Formeln mit $k$ Variablentypen, welche die Bedingung*

$$\exists \bar{a}_1 \in \{0,1\}^{m_1} \forall \bar{a}_2 \in \{0,1\}^{m_2} \exists \bar{a}_3 \in \{0,1\}^{m_3} \ldots Q_k \bar{a}_k \in \{0,1\}^{m_k} : F(\bar{a}_1, \ldots, \bar{a}_k) = 1 \quad (40)$$

*erfüllen.*

Eine zu $\mathcal{B}_k$ duale Klasse ergibt sich wie folgt:

**Definition 8.18** *Die Sprache $\tilde{\mathcal{B}}_k$ enthalte alle (Kodierungen von) Booleschen Formeln mit $k$ Variablentypen, welche die Bedingung*

$$\forall \bar{a}_1 \in \{0,1\}^{m_1} \exists \bar{a}_2 \in \{0,1\}^{m_2} \forall \bar{a}_3 \in \{0,1\}^{m_3} \ldots \bar{Q}_k \bar{a}_k \in \{0,1\}^{m_k} : F(\bar{a}_1, \ldots, \bar{a}_k) = 1 \quad (41)$$

*erfüllen.*

Der folgende Satz ist das Pendant zum Cook'schen Theorem:

**Satz 8.19** *Für alle $k \geq 1$ gilt: $\mathcal{B}_k$ ist $\Sigma_k$-vollständig und $\tilde{\mathcal{B}}_k$ ist $\Pi_k$-vollständig.*

**Beweis** Der Beweis ähnelt dem Beweis des Cook'schen Theorems. Wir skizzieren die wesentlichen Ideen und weisen von Zeit zu Zeit auf die Analogie zum Beweis des klassischen Cook'schen Theorems hin. Genauere technische Details lassen sich leicht ausfüllen.
Wir diskutieren zunächst den Fall einer ungeraden Anzahl $k$ von Variablentypen. In diesem Fall gilt $Q_k = \exists$, d.h., die alternierende Quantorenkette in (40) endet mit einem $\exists$-Quantor. Sei $L \in \Sigma_k$. Gemäß Folgerung 8.12 gilt (unter Beachtung von $k$ ungerade):

$$L = \{x : (\exists y_1)_{pol}(\forall y_2)_{pol} \cdots (\exists y_k)_{pol} \langle y_1, \ldots, y_k, x\rangle \in L_0\}$$

für eine geeignete Sprache $L_0 \in P$. Wir können oBdA annehmen, dass $y_1, \ldots, y_k$ Wörter über dem Alphabet $\{0,1\}$ sind (in Analogie zu der Annahme binärer Ratestrings bei NTMs). Sei $M_0$ die polynomielle DTM, welche als Akzeptor von $L_0$ auftritt. Wie im Beweis des Cook'schen Theorems lässt sich die polynomielle deterministische Rechnung von $M_0$ auf $\langle y_1, \ldots, y_k, x\rangle$ mit Hilfe einer Booleschen CNF-Formel $F$ beschreiben. $F$ verwendet Boolesche Variable zur Kodierung der Strings $y_1, \ldots, y_k$ und weitere Hilfvariablen. Für $i = 1, \ldots, k$ und $m_i = |y_i|$ seien $\bar{v}_i = (v_{i1}, \ldots, v_{im_i})$ die Booleschen Variablen vom Typ $i$, deren Belegung zum Binärstring $y_i$ korrespondieren soll (in Analogie zu den Booleschen Variablen für die Ratebits im Beweis des Cook'schen Theorems). Die Hilfsvariablen sind alle vom Typ $k$ und dienen (wie im klassischen Beweis des Cook'schen Theorems) der Beschreibung von Rechnungen von $M_0$ auf Eingaben der Form $\langle y_1, \ldots, y_k, x\rangle$. (Es handelt sich also um Variablen, deren Belegungen den jeweiligen Zustand, die jeweilige Kopfposition und die jeweilige Bandinschrift kodieren.) Sei $m'_k$ die Anzahl dieser Hilfsvariablen, welche wir als $\bar{v}'_k = (v'_{k1}, \ldots, v'_{km'_k})$ notieren. Da $M_0$ eine *polynomielle* DTM ist, ist $m'_k$ polynomiell in $m_1, \ldots, m_k, |x|$ beschränkt. Da $m_i = |y_i|$ polynomiell in $|x|$ beschränkt ist, ist dann schließlich auch $m'_k$ wie auch die Gesamtanzahl $m = m_1 + \cdots + m_k + m'_k$ aller in $F$ vorkommender Boolescher Variablen polynomiell in $|x|$ beschränkt. In Analogie zum Beweis des klassischen Cook'schen Theorems lässt sich nun in pol($|x|$) Schritten eine Formel $F(\bar{v}_1, \ldots, \bar{v}_{k-1}, \bar{v}_k, \bar{v}'_k)$ konstruieren, so dass für alle $x \in L$

$$\begin{aligned}
x \in L &\Leftrightarrow (\exists y_1)_{pol}(\forall y_2)_{pol} \cdots (\exists y_k)_{pol} \langle y_1, \ldots, y_k, x\rangle \in L_0 \\
&\Leftrightarrow (\exists y_1)_{pol}(\forall y_2)_{pol} \cdots (\exists y_k)_{pol} M_0 \text{ akzeptiert } \langle y_1, \ldots, y_k, x\rangle \\
&\Leftrightarrow \exists \bar{a}_1 \in \{0,1\}^{m_1} \forall \bar{a}_2 \in \{0,1\}^{m_2} \ldots \exists \bar{a}_k \in \{0,1\}^{m_k} \exists \bar{a}'_k \in \{0,1\}^{m'_k} : \\
&\quad F(\bar{a}_1, \ldots, \bar{a}_k, \bar{a}'_k) = 1 \ .
\end{aligned}$$

Da wir die letzten beiden $\exists$-Quantoren vor $F(\bar{a}_1, \ldots, \bar{a}_k, \bar{a}'_k)$ verschmelzen können, ist offensichtlich die Abbildung $x \mapsto F$ eine polynomielle Reduktion von $L$ auf $\mathcal{B}_k$. Folglich ist $\mathcal{B}_k$ $\Sigma_k$-vollständig. Die $\Pi_k$-Vollständigkeit von $\tilde{\mathcal{B}}_k$ ergibt sich (für ungerades $k$) durch Dualisierung.
Bleibt der Fall eines geraden $k$ zu diskutieren. Hier ist es technisch leichter zunächst die $\Pi_k$-Vollständigkeit von $\tilde{B}_k$ zu verifizieren und die $\Sigma_k$-Vollständigkeit von $B_k$ durch Dualisierung zu folgern. Es ist nämlich praktischer, wenn die alternierende Quantorenkette erneut mit einem $\exists$-Quantor endet, damit der $\exists$-Quantor für die Hilfsvaribalen der Booleschen Formel $F$ mit dem vorangehenden $\exists$-Quantor verschmolzen werden kann. Der Nachweis der$\Pi_k$-Vollständigkeit von $\tilde{B}_k$ für gerades $k$ geschieht analog zum Nachweis der $\Sigma_k$-Vollständigkeit von $\mathcal{B}_k$ für ungerades $k$. □

Aus dem Beweis von Satz 8.19 ist folgendes ersichtlich (s. Übungen):

**Folgerung 8.20** *Die Einschränkung von $\mathcal{B}_k$ auf CNF-Formeln bei ungeradem $k$ und DNF-Formeln bei geradem $k$ ist $\Sigma_k$-vollständig. Dual dazu ist die Einschränkung von $\tilde{B}_k$ auf DNF-Formeln bei ungeradem $k$ und CNF-Formeln bei geradem $k$ $\Pi_k$-vollständig.*

Folgerung 8.20 liefert im Falle $k = 1$ gerade das Cook'sche Theorem.

# 9 PSpace-vollständige Probleme

## 9.1 Quantifizierte Boolesche Formeln

geplant für 13.07.

**Definition 9.1** *Die Sprache der* Quantifizierten Booleschen Formeln*, notiert als QBF (oder als $\mathcal{B}_*$, um die Verwandtschaft zu den Sprachen $\mathcal{B}_k$ zu betonen), enthalte alle (Kodierungen von) quantifizierten Booleschen Formeln der Form*

$$(\mathcal{Q}_1 v_1) \dots (\mathcal{Q}_m v_m) F(v_1, \dots, v_m) \ , \tag{42}$$

*welche die Korrektheitsbedingung*

$$\mathcal{Q}_1 a_1 \in \{0,1\}, \dots, \mathcal{Q}_m a_m \in \{0,1\} : F(a_1, \dots, a_m) \tag{43}$$

*erfüllen. Hierbei sei $m$ eine beliebige nicht-negative ganze Zahl, $\mathcal{Q}_1, \dots, \mathcal{Q}_m \in \{\exists, \forall\}$, und $F$ sei eine Boolesche Formel in den Booleschen Variablen $v_1, \dots, v_m$.*

**Beispiel 9.2** *Folgende quantifizierten Booleschen Formeln haben die in (42) vorgeschriebene Form:*

$$\begin{array}{lcll} qbf_1 & := & (\forall v_1)(\exists v_2)(\exists v_3) & \bar{v}_3 \wedge ((\bar{v}_1 \wedge v_2) \vee (v_1 \wedge \bar{v}_2)) \\ qbf_2 & := & (\exists v_1)(\forall v_2)(\exists v_3) & \bar{v}_3 \wedge ((\bar{v}_1 \wedge v_2) \vee (v_1 \wedge \bar{v}_2)) \end{array}$$

*Wir werden weiter unten sehen, dass $qbf_1$ die Bedingung (43) erfüllt und somit zu QBF gehört, wohingegen $qbf_2$ diese Bedingung verletzt und somit nicht zu QBF gehört.*

**Lemma 9.3** *Für alle $k \geq 1$ gilt $\mathcal{B}_k \leq_{pol} QBF$.*

**Beweis** Wir verwenden die Reduktionstechnik der Spezialisierung. $\mathcal{B}_k$ ist das Teilproblem von QBF, bei dem die Quantorenkette $\mathcal{Q}_1 \cdots \mathcal{Q}_m$ mit einem $\exists$-Quantor beginnt und nur $(k-1)$-mal zwischen $\exists$- und $\forall$-Quantor hin und her wechselt. □

**Folgerung 9.4** *QBF ist PH-hart.*

**Beweis** Sei $L \in PH$. Dann existiert ein $k \geq 1$ mit $L \in \Sigma_k$. Für diesen Index $k$ gilt dann wegen der $\Sigma_k$-Vollständigkeit von $\mathcal{B}_k$: $L \leq_{pol} \mathcal{B}_k$. Mit Lemma 9.3 und der Transitivität von polynomiellen Reduktionen ergibt sich $L \leq_{pol}$ QBF. □

Vermutlich gilt aber QBF $\notin PH$, da die Anzahl der Quantorenwechsel in $\mathcal{Q}_1, \dots, \mathcal{Q}_m$ durch keine Konstante $k$ beschränkt ist (sondern mit der Eingabelänge wachsen kann).

**Satz 9.5** *QBF $\in$ PSpace.*

**Beweis** Wir verwenden zunächst eine platzineffiziente EXHAUSTIVE SEARCH, um ein kanonisches Auswertungsschema klarzumachen. Anschließend verwenden wir die Technik des *Ariadne Fadens*[58], um die EXHAUSTIVE SEARCH platzeffizient zu gestalten.
Wir setzen oBdA voraus, dass die Eingabe zumindest die in (42) vorgeschriebene Form hat. Mit einem vollständig binären Entscheidungsbaum $T$ der Tiefe $m$ lassen sich alle $2^m$ denkbaren Belegungen von $v_1, \ldots, v_m$ durchprobieren. Wir markieren die inneren Knoten von $T$ der Tiefe $i-1$ mit $\mathcal{Q}_i \in \{\exists, \forall\}$, um deutlich zu machen, wie die Variable $v_i$ quantifiziert ist.

**Einschub** Die Entscheidungsbäume $T_1$ und $T_2$, die zu den Formeln $\mathrm{qbf}_1$ und $\mathrm{qbf}_2$ aus Beispiel 9.2 korrespondieren, sind in den Abbildungen 14 und 15 zu besichtigen. Der Fortgang des Beweises kann an diesen Abbildungen nachvollzogen werden.

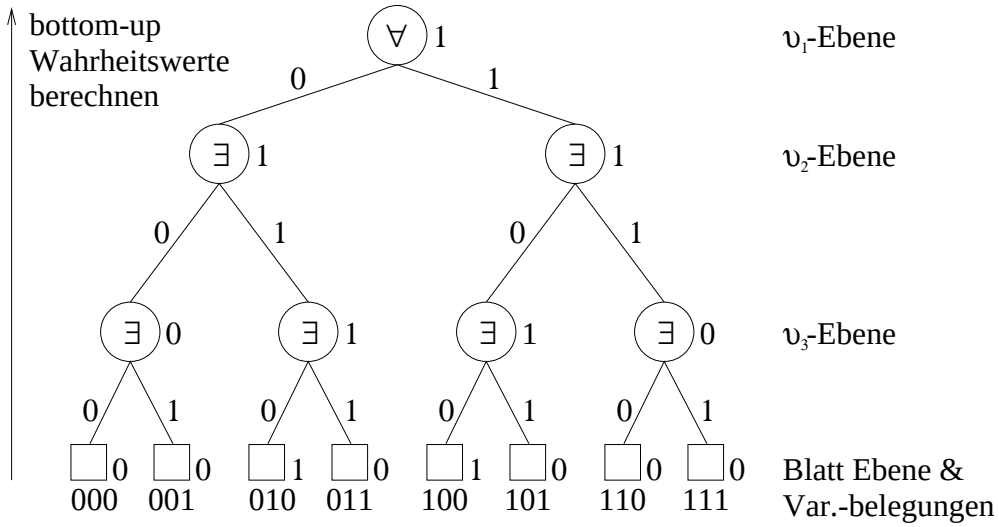



Abbildung 14: Der Entscheidungsbaum $T_1$ für die quantifizierte Boolesche Formeln $\mathrm{qbf}_1$ aus Beispiel 9.2.

Jedes Blatt von $T$ korrespondiert zu einer vollständigen Belegung $(b_1, \ldots, b_m) \in \{0,1\}^m$ der Variablen $(v_1, \ldots, v_m)$. Ein innerer Knoten $u$ der Tiefe $i$ in $T$ korrespondiert zu einer partiellen Belegung von $(v_1, \ldots, v_i)$ mit $(b_1, \ldots, b_i)$. Unser Ziel ist, die Knoten von $T$ „bottom-up“ mit den Booleschen Wahrheitswerten 0 oder 1 zu markieren, wobei ein zur partiellen Belegung $(b_1, \ldots, b_i)$ korrespondierender Knoten $u$ der Tiefe $i$ genau dann mit 1 markiert werden soll, wenn die Bedingung

$$\mathcal{Q}_{i+1} a_{i+1} \in \{0,1\} \cdots \mathcal{Q}_m a_m \in \{0,1\} : F(b_1, \ldots, b_i, a_{i+1}, \ldots, a_m) \tag{44}$$

erfüllt ist. Zur Markierung eines zur vollständigen Belegung $b = (b_1, \ldots, b_m)$ korrespondierenden Blattes, brauchen wir lediglich die Formel $F$ an $b$ auszuwerten. Wenn wir induktiv

[58] der Sage nach erstmals von Ariadne angewendet, um Theseus davor zu schützen, sich im Labyrinth des Minotaurus zu verirren

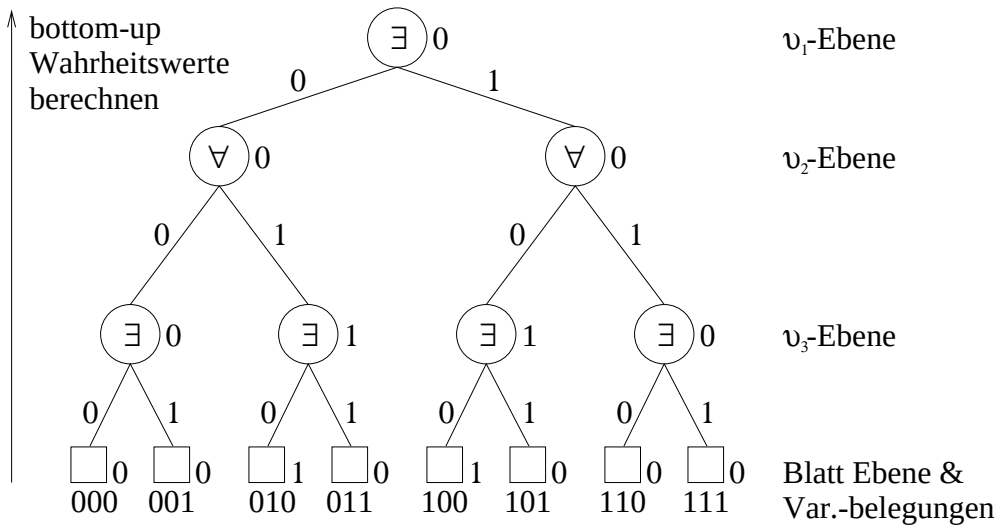



Abbildung 15: Der Entscheidungsbaum $T_2$ für die quantifizierte Boolesche Formel $\text{qbf}_2$ aus Beispiel 9.2.

annehmen, dass die Kinder $u_0, u_1$ eines inneren Knoten $u$ bereits korrekt mit Wahrheitswerten $w_0, w_1$ markiert sind, dann führt folgende Regel zu einer korrekten Markierung von $u$:

- Wenn $u$ mit Quantor $\exists$ markiert ist, dann markiere $u$ mit Wahrheitswert $w_0 \vee w_1$.
- Wenn $u$ mit Quantor $\forall$ markiert ist, dann markiere $u$ mit Wahrheitswert $w_0 \wedge w_1$.

Auf diese Weise können wir, beginnend bei den Blättern, alle Knoten von $T$ bottom-up mit Wahrheitswerten markieren bis wir schließlich bei der Wurzel anlangen. Aus der Invarianzbedingung (44) geht durch Inspektion des Spezialfalles $i = 0$ hervor, dass die Wurzel von $T$ genau dann mit 1 markiert ist, wenn die gesamte quantifizierte Formel zu QBF gehört. Wir akzeptieren also die Eingabe genau dann, wenn am Ende unserer Markierungsprozedur die Wurzel des Entscheidungsbaumes $T$ mit Wahrheitswert 1 markiert ist.
Entscheidungsbaum $T$ hat $2^m$ Blätter. Seine Größe ist i.A. nicht polynomiell in der Eingabelänge $n$ beschränkt. Eine platzeffiziente EXHAUSTIVE SEARCH darf zu keinem Zeitpunkt den Baum $T$ vollständig abspeichern. Stattdessen verwenden wir beim Durchlaufen von $T$ die Technik des „Ariadne-Fadens“.
Um die eingangs beschriebene Markierungsprozedur platzeffizient durchzuführen, wird $T$ in Präordnung durchlaufen, wobei wir zu jedem Zeitpunkt nur den Pfad von der Wurzel zum aktuellen Knoten (Ariadne-Faden) und die aktuelle partielle Belegung $(b_1, \ldots, b_i)$ abspeichern. Die Abspeicherung des Ariadne-Fadens erfolgt kellerartig. Eine DTM kann zu diesem Zweck eines ihrer Bänder als Keller verwenden.
Es folgen einige Details dieser Organisation. Zunächst erinnern wir an den Begriff der „Präordnung“, „Ariadne-Faden“ und „kellerartige Organisation“. Wenn $T$ ein Baum mit Wurzel $r$, linkem Unterbaum $T_0$ und rechtem Unterbaum $T_1$ ist, dann ist der Präordnungs-Durchlauf rekursiv gegeben durch die folgenden Regeln:

1. Besuche zuerst die Wurzel $r$.

2. Besuche dann (rekursiv) die Knoten von $T_0$ in Präordnung.

3. Besuche schließlich (rekursiv) die Knoten von $T_1$ in Präordnung.

**Einschub** In Abbildung 16 ist der Präordnungs-Durchlauf an einem Beispiel illustriert. Der aktuell besuchte Knoten ist der Knoten $u$ mit Quantormarkierung „$\forall$“. und der Ariadne-Faden von der Wurzel zu diesem Knoten ist graphisch hervorgehoben. Wie in der Abbildung angedeutet läuft man in Präordnung im Prinzip „einmal um den Baum herum“.

Beim Präordnungs-Durchlauf wird jede Kante zweimal durchlaufen: das erste Mal beim *Abstieg* von einem Knoten zu einem seiner Söhne, das zweite Mal bei der *Rückkehr* oder dem *Aufstieg* vom Sohn zum Vaterknoten. *Kellerartige Organisation* des Durchlaufs bedeutet, dass wir beim Abstieg den Sohnknoten (und die mit ihm assoziierte Information) auf den Kellerspeicher „PUSHen“ (Verlängerung des Ariadne-Fadens); beim Aufstieg zum Vaterknoten kann dieses Speichersegment wieder vom Keller „gePOPt“ werden (Verkürzung des Ariadne-Fadens).[59]
Wir klären als nächstes das Speicherformat für die Knoten von $T$. Die zu einem Knoten abgespeicherten Hilfsinformationen sollen uns befähigen, den Präordnungs-Durchlauf in Gang zu halten und nebenbei die eingangs beschriebene Prozedur zur Markierung von Knoten mit Wahrheitswerten durchzuführen. Zu einem inneren Knoten $u$ von $T$ mit linkem Sohn $u_0$ und rechtem Sohn $u_1$ merken wir uns zu diesem Zweck die folgenden Informationen:

1. die Quantormarkierung für $u$ ($\exists$ oder $\forall$).

2. die Wahrheitswertmarkierungen $w_0, w_1, w$ für $u_0, u_1, u$ (mit Eintrag „?“ solange der Wahrheitswert noch nicht bekannt ist).

Aus dem Speicherformat geht hervor, dass pro abgespeichertem Knoten nur konstanter Speicherplatz erforderlich ist.

**Einschub** Die Felder für die Wahrheitswerte eines „Knoten-Records“ enthalten anfangs das Fragezeichen, das bei Bekanntwerden des betreffenden Wahrheitswertes mit diesem überschrieben wird. Im Beispiel von Abbildung 16 hätte das Record des aktuell besuchten Knotens $u$ folgende Einträge:

| $\forall$ | $w_0$ | ? | ? |
|---|---|---|---|
| Quantor | Wahrheitswert linker Sohn | Wahrheitswert rechter Sohn | Wahrheitswert aktueller Knoten |

[59] Wie üblich benutzen wir bei Kellerspeichern die Bezeichnungen PUSH und POP anstelle von INSERT und DELETE.

Es ist eine leichte Übung nachzuweisen, dass man mit diesen Hilfsinformationen (plus die aktuelle partielle Belegung, die ja ebenfalls abgespeichert wird) den Präordnungs-Durchlauf in Gang halten kann, wobei die Hilfsinformationen selbst leicht aktualisierbar sind. Wenn der Durchlauf vom rechten Sohn der Wurzel zur Wurzel zurückkehrt, ist die Wahrheitswertmarkierung der Wurzel bekannt. Die Eingabe wird genau dann akzeptiert, wenn die Wurzel mit 1 markiert ist.
Die Korrektheit dieser Vorgehensweise ergibt sich aus der Korrektheit der eingangs geschilderten Markierungsprozedur, bei der wir ja lediglich die Implementierung platzeffizient gestaltet haben. Die Platzanalyse ist leicht. Zu jedem Zeitpunkt haben wir im Wesentlichen den aktuellen Ariadne-Faden und die aktuelle partielle Belegung abgespeichert. Die partielle Belegung besteht aus maximal $m$ Bits. Der Ariadne-Faden enthält maximal $m+1$ Knoten, wobei wir konstanten Speicherplatz pro Knoten veranschlagen müssen. Die Platzschranke ist somit linear in $m$. Da die Länge $n$ der Eingabe (quantifizierte Boolesche Formel, welche $m$ Boolesche Variable verwendet) nicht kleiner als $m$ sein kann, erhalten wir eine Platzschranke, die auch linear in $n$ ist. Folglich gehört QBF zu PSpace. □

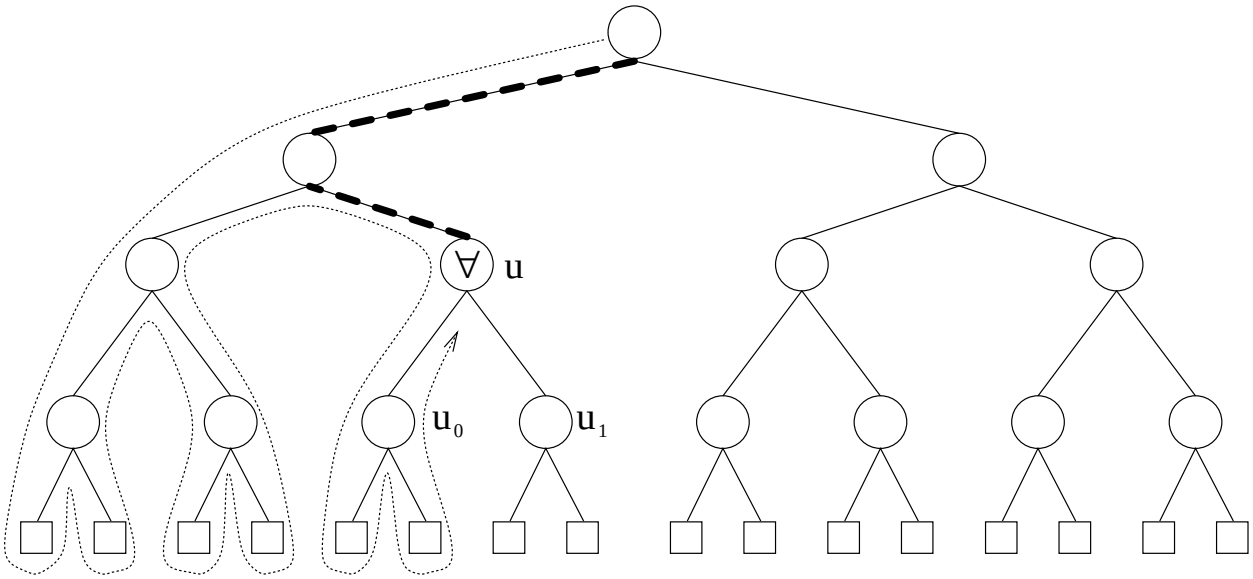



Abbildung 16: Eine Momentaufnahme des Präordnungs-Durchlaufs durch einen Entscheidungsbaum ($u$ mit Söhnen $u_0, u_1$ der aktuell besuchte Knoten, Ariadne-Faden hervorgehoben).

Da QBF *PH*-hart ist, ergibt sich aus Satz 9.5 unmittelbar die

**Folgerung 9.6** *PH* $\subseteq$ *PSpace*.

**Satz 9.7** *QBF ist PSpace-hart*

**Beweis** Sei $L \in$ *PSpace*. Wir haben zu zeigen, dass $L \leq_{pol}$ QBF. Sei $M$ eine DTM mit polynomieller Platzschranke $S(n)$ und $L = L_M$. Wir können oBdA annehmen, dass $M$ nur auf den Zellen mit Nummern $1, \ldots, S(n)$ arbeitet. Sei $\mathcal{K}_n$ die Menge der Konfigurationen von $M$ auf Eingaben der Länge $n$. Eine Konfiguration aus $\mathcal{K}_n$ ist gegeben durch

- den aktuellen Zustand $q \in Q$,
- die aktuelle Kopfposition $h \in \{1, \ldots, S(n)\}$,
- die aktuelle Bandinschrift $I \in \Gamma^{S(n)}$.

Hierbei bezeichnen, wie üblich, $Q$ die Zustandsmenge und $\Gamma$ das Bandalphabet von $M$. Die Anzahl der verschiedenen Konfigurationen ist offensichtlich nach oben beschränkt wie folgt:

$$|\mathcal{K}_n| \leq |Q| \cdot S(n) \cdot |\Gamma|^{S(n)} = 2^{O(S(n))} \quad .$$

Die letzte asymptotische Abschätzung gilt, weil $|Q|$ und $|\Gamma|$ von $n$ unabhängige Konstanten sind. $M$ kann so normalisiert werden, dass keine Konfiguration zweimal angenommen wird. (Ansonsten würde eine Endlosschleife vorliegen.) Folglich rechnet $M$ auf Eingaben der Länge $n$ maximal $|\mathcal{K}_n| - 1$ viele Schritte. Wir können oBdA annehmen, dass $M$ für eine geeignete Konstante $c$ auf Eingaben der Länge $n$ exakt $2^{cS(n)}$ Schritte rechnet und danach akzeptierend oder verwerfend stoppt. Weiter nehmen wir oBdA an, dass für jede Eingabe $x$ der Länge $n$ eine eindeutige Startkonfiguration $K_0(x) \in \mathcal{K}_n$ und eine eindeutige akzeptierende Endkonfiguration $K_+(x) \in \mathcal{K}_n$ existiert. Für $K, K' \in \mathcal{K}_n$ und $s \geq 0$ bezeichne $K \xrightarrow{s} K'$ die folgende Relation:

$$K \xrightarrow{s} K' :\Leftrightarrow \text{Konfiguration } K \text{ wird von } M \text{ in } 2^s \text{ Rechenschritten in } K' \text{ überführt} \quad .$$

Offensichtlich gilt

$$x \in L \Leftrightarrow K_0(x) \xrightarrow{cS(n)} K_+(x) \quad .$$

Aus diesen Vorbemerkungen ergibt sich die folgende

**Beweisstrategie** Um zu einer polynomiellen Reduktion von $L$ auf QBF zu gelangen, konstruieren wir in Polynomialzeit aus $x$ eine quantifizierte Boolesche Formel, welche genau dann die Korrektheitsbedingung (43) erfüllt, wenn $K_0(x) \xrightarrow{cS(n)} K_+(x)$. Salopp formuliert: es geht darum die Bedingung $K_0(x) \xrightarrow{cS(n)} K_+(x)$ in Form einer (effizient konstruierbaren) quantifizierten Booleschen Formel aufzuschreiben.

**Einschub** Wir könnten jetzt die Technik aus dem Beweis des Cook'schen Theorems verwenden und eine CNF-Formel $F_x$ konstruieren, die genau dann erfüllbar ist, wenn $K_0(x) \xrightarrow{cS(n)} K_+(x)$. Das Problem ist aber, dass die Länge der resultierenden Formel $F_x$ kubisch von der Laufzeitschranke abhängt. Bei $2^{cS(n)}$ Rechenschritten hätte $F_x$ die Länge $2^{3cS(n)}$. Diese Formel ist (allein schon wegen ihrer exponentiellen Kodierungslänge) aus $x$ nicht in Polynomialzeit konstruierbar. Unsere Überlegung zeigt, wozu der Einsatz der Quantoren dienen wird: Konstruktion einer extrem komprimierten Formel, die Rechnungen beschreibt, welche eine Exponentialstufe länger sind als die Formel selbst.

Wir werden versuchen, die gewünschte quantifizierte Boolesche Formel rekursiv zu konstruieren. Dazu benutzen wir die folgenden Beobachtungen für beliebige Konfigurationen $K, K' \in \mathcal{K}_n$ und $s \geq 1$:

$$K \xrightarrow{0} K' \quad \Leftrightarrow \quad K' \text{ ist eine unmittelbare Folgekonfiguration von } K \ . \tag{45}$$

$$K \xrightarrow{s} K' \quad \Leftrightarrow \quad \exists K'' \in \mathcal{K}_n : K \xrightarrow{s-1} K'' \wedge K'' \xrightarrow{s-1} K' \ . \tag{46}$$

Wir werden im folgenden ein paar Fertigkeiten reaktivieren, die wir durch den Beweis des Cook'schen Theorems erworben haben:

1. Um eine einzelne Konfiguration in $\mathcal{K}_n$ zu beschreiben, reichen $O(S(n))$ Boolesche Variable aus.[60] Sei $V$ eine Kollektion solcher Variablen. Eine konkrete Konfiguration entspricht eindeutig einer Belegung der Variablen in $V$. Allerdings entspricht umgekehrt nicht jede Belegung der Variablen einer zulässigen Konfiguration. Es gibt aber eine (polynomiell konstruierbare) CNF-Formel $F[V]$ in den Variablen aus $V$, die genau durch die Belegungen erfüllbar ist, welche zulässige Konfigurationen repräsentieren.[61]

2. Weiterhin gibt es für jede konkrete Konfiguration $K \in \mathcal{K}_n$ eine (polynomiell konstruierbare) CNF-Formel $F_K[V]$, so dass $F[V]$ und $F_K[V]$ genau dann gemeinsam erfüllbar sind, wenn die Belegung der Variablen in $V$ die Konfiguration $K$ repräsentiert.[62]

3. Seien schließlich $V$ und $V'$ disjunkte Kollektionen von Booleschen Variablen zur Beschreibung von Konfigurationen. Dann gibt es eine (polynomiell konstruierbare) CNF-Formel $F_0[V, V']$, so dass $F[V]$, $F[V']$ und $F_0[V, V']$ genau dann gemeinsam erfüllbar sind, wenn die Belegung der $V'$-Variablen eine unmittelbare Folgekonfiguration repräsentiert von der durch die $V$-Variablenbelegung repräsentierten Konfiguration.[63]

Zur sprachlichen Vereinfachung identifizieren wir im folgenden Konfigurationen mit den sie beschreibenden Variablenbelegungen. Die Formel $F[V] \wedge F[V'] \wedge F_0[V, V']$ testet in diesem Sinne, ob die $V'$-Variablenbelegung eine unmittelbare Folgekonfiguration der $V$-Variablenbelegung ist. Wir suchen jetzt nach einer (möglichst komprimierten) quantifizierten Formel $F_s[V, V']$ mit freien (sprich unquantifizierten) Variablen aus $V \cup V'$ und weiteren gebundenen (sprich quantifizierten) Hilfsvariablen, so dass $F[V] \wedge F[V'] \wedge F_s[V, V']$ testet, ob die $V'$-Variablenbelegung eine Konfiguration ist, die sich nach $2^s$ Rechenschritten aus der $V$-Variablenbelegung ergibt. Hier ist ein erster (einfacher, aber leider auch ineffizienter) Versuch $F_s$ rekursiv aus $F_{s-1}$ zu gewinnen:

$$F_s[V, V'] := (\exists V'')F[V''] \wedge F_{s-1}[V, V''] \wedge F_{s-1}[V'', V'] \ . \tag{47}$$

Die Expansion von $F_s[V, V']$ in die Formel auf der rechten Seite von (47), erfordert dabei stets eine *neue* Kollektion (quantifizierter) Boolescher Variablen (zur Beschreibung einer Konfiguration). Da die Rekursionsregel (47) eine treue Nachbildung von (46) ist, wird folgende

[60] $|Q|$ „Zustandsvariablen", $S(n)$ „Kopfpositionsvariablen" und $|\Gamma| \cdot S(n)$ „Bandinschriftsvariablen".
[61] Im Beweis zum Cook'schen Theorem enthielt diese CNF-Formel die Klauseln vom Typ I. S. dort.
[62] Im Beweis zum Cook'schen Theorem enthielt diese CNF-Formel die Klauseln vom Typ II. S. dort.
[63] Im Beweis zum Cook'schen Theorem enthielt diese CNF-Formel die Klauseln vom Typ III. S. dort.

Feststellung nicht überraschen (einfacher induktiver Beweis[64]):

$$\begin{aligned} K \xrightarrow{0} K' &\Leftrightarrow (\exists V)(\exists V')F[V] \wedge F_K[V] \wedge F[V'] \wedge F_{K'}[V'] \wedge F_0[V,V'] \in \text{QBF} \ . \\ K \xrightarrow{s} K' &\Leftrightarrow (\exists V)(\exists V')F[V] \wedge F_K[V] \wedge F[V'] \wedge F_{K'}[V'] \wedge F_s[V,V'] \in \text{QBF} \ . \end{aligned}$$

Somit gilt insbesondere

$$K_0(x) \xrightarrow{cS(n)} K_+(x) \Leftrightarrow (\exists V)(\exists V')F[V] \wedge F_{K_0(x)}[V] \wedge F[V'] \wedge F_{K_+(x)}[V'] \wedge F_{cS(n)}[V,V'] \ . \tag{48}$$

Die schlechte Nachricht ist aber, dass die resultierende quantifizierte Boolesche Formel für $K_0(x) \xrightarrow{cS(n)} K_+(x)$ eine in $n$ exponentielle Länge hat. Dies liegt daran, dass wir in (47) *zwei* rekursive Aufrufe vorfinden. Es entsteht somit ein vollständig binärer Rekursionsbaum der Tiefe $cS(n)$ mit $2^{cS(n)}$ Blättern.

**Einschub** Es ist nicht verwunderlich, dass der Plan noch nicht aufgeht. Bisher haben wir ausschließlich $\exists$-Quantoren verwendet. Wenn wir auf diese Weise eine polynomielle Reduktion erhalten hätten, hätten wir $PSpace = NP$ gezeigt (was eine Sensation wäre). Die noch ausstehende Kompression der Formellänge wird wohl Gebrauch von dem $\forall$-Quantor machen müssen.

Wir ändern jetzt die Rekursionsregel (47) unter Einsatz des $\forall$-Quantors so ab, dass nur noch *ein* rekursiver Aufruf erfolgt. Anstelle eine binären Rekursionsbaumes der Tiefe $cS(n)$ erhalten wir dann einen „Rekursionspfad“ der Tiefe $cS(n)$, der folgerichtig zu einer quantifizierten Booleschen Formel der Kodierungslänge $O(S(n))$ führen wird. In einem gewissen Sinn war bisher alles nur Vorgeplänkel und erst jetzt kommt die

**Entscheidende Idee** In die Rekursion (47) fügen wir zwei weitere *neue* Kollektionen von Variablen ein, sagen wir $X$ und $Y$. Wir werden mit *einem* rekursiven Aufruf testen, ob die $Y$-Variablenbelegung eine Konfiguration ist, die durch $2^{s-1}$ Schritte aus der $X$-Variablenbelegung hervorgeht. Mit Hilfe eines $\forall$-Quantors werden wir erreichen, dass $(X,Y)$ einmal in der Rolle von $(V,V'')$ und ein weiteres Mal in der Rolle von $(V'',V')$ auftreten. Dadurch ersetzt der *eine* rekursive Aufruf die *beiden* rekursiven Aufrufe in (47).

Um die neue raffinierte Rekursionsregel einprägsam zu notieren, benutzen wir die Abkürzung $(\forall V)$ in völliger Analogie zu $(\exists V)$. Die Gleichheit $X = V$ zweier Variablenkollektionen ist komponentenweise zu verstehen. Man beachte auch, dass Gleichheit zweier Boolescher Variablen oder Implikation zwischen Booleschen Variablen als Abkürzungen im Sinne von

$$\begin{aligned} a = b &\Leftrightarrow (a \wedge b) \vee (\bar{a} \wedge \bar{b}) \\ a \rightarrow b &\Leftrightarrow \bar{a} \vee b \end{aligned}$$

[64] Die Formeln, die durch Expansion der Rekursion entstehen, sind erst einmal nicht in pränexer Normalform. Streng genommen gilt die Mitgliedschaft zu QBF nicht für die Formel sondern für ihre pränexe Normalform. Der besseren Lesbarkeit zuliebe notieren wir die Transformation in pränexe Normalform nicht explizit.

zu sehen sind. Nach diesen Vorbemerkungen geben wir nun die neue Rekursionsregel an, die (47) ersetzen soll:

$$F_s[V,V'] := \\ (\exists V'')(\forall X)(\forall Y)F[V''] \wedge (((X = V \wedge Y = V'') \vee (X = V'' \wedge Y = V')) \rightarrow F_{s-1}[X,Y]) \quad .$$

Nach etwas kontemplativer Versenkung sollte klar werden, dass die neue Rekursionsregel logisch äquivalent zu (47) ist. Wenn wir jetzt in (48) die rekursive Expansion von $F_{cS(n)}[V,V']$ gemäß der neuen Regel vornehmen, ergibt sich (endlich) die angestrebte polynomielle Reduktion von $L$ nach QBF. Die resultierende quantifizierte Boolesche Formel hat nämlich eine polynomiell beschränkte Länge und lässt sich auch in Polynomialzeit aus $x$ entwickeln. Sie erfüllt genau dann die Korrektheitsbedingung (43) von QBF, wenn $K_0(x) \stackrel{cS(n)}{\longrightarrow} K_+(x)$, was wiederum genau dann der Fall ist, wenn $x \in L$. □

Sätze 9.5 und 9.7 liefern zusammen die

**Folgerung 9.8** *QBF ist PSpace-vollständig.*

## 9.2 Prädikatenlogische Charakterisierung von PSpace

geplant für 15.07.

Im folgenden Satz wird PSpace mit Hilfe alternierender Quantorenketten charakterisiert. Der wesentliche Unterschied zur Charakterisung von *PH* ist darin zu sehen, dass die Länge der Quantorenkette nicht mehr durch eine feste Konstante beschränkt ist, sondern nur noch durch ein Polynom in der Eingabelänge $|x|$. Wir erhalten die folgende Charakterisierung von PSpace:

**Satz 9.9** *Eine Sprache $L$ gehört genau dann zu PSpace, wenn eine Sprache $L_0 \in P$ und ein Polynom $p$ existieren, so dass*

$$x \in L \Leftrightarrow (\exists y_1)_{pol}(\forall y_2)_{pol}(\exists y_3)_{pol} \cdots (Q_{p(|x|)} y_{p(|x|)})_{pol} \langle y_1, \ldots, y_{p(|x|)}, x\rangle \in L_0 \quad . \qquad (49)$$

**Beweis** Wir setzen zunächst $L \in PSpace$ voraus und weisen Bedingung (49) nach. Wegen $L \leq_{pol}$ QBF gibt es für alle Wörter $x$ eine (aus $x$ polynomiell konstruierbare) Boolesche Formel $F_x$ in $m = \mathrm{pol}(|x|)$ Booleschen Variablen $v_1, \ldots, v_m$ und eine Quantorenkette $\mathcal{Q}_1, \ldots, \mathcal{Q}_m \in \{\exists, \forall\}$ mit der Eigenschaft

$$x \in L \Leftrightarrow (\mathcal{Q}_1 v_1) \ldots (\mathcal{Q}_m v_m) F_x(v_1, \ldots, v_m) \quad . \qquad (50)$$

OBdA gelte $\mathcal{Q}_1 = \exists$. (Ansonsten könnten wir eine redundante Variable mit einem $\exists$-Quantor hinzufügen.) Wir können die Quantorenkette $\mathcal{Q}_1, \ldots, \mathcal{Q}_m$ in maximale Teilketten mit Quantoren nur eines Typs unterteilen und oBdA annehmen (s. Übung), dass die Anzahl und die Länge der Teilketten von $x$ nur über $|x|$ abhängt (und zwar polynomiell). Die Anzahl der Teilketten sei mit $q(|x|)$ bezeichnet. Dann können wir (50) umschreiben wie folgt:

$$x \in L \Leftrightarrow (\exists y_1)_{pol}(\forall y_2)_{pol}(\exists y_3)_{pol} \cdots (Q_{q(|x|)} y_{q(|x|)})_{pol} F_x(y_1, \ldots, y_{q(|x|)}) \quad . \qquad (51)$$

Hierbei bezeichnet $y_i$ die Teilkollektion der Booleschen Variablen aus $v_1, \ldots, v_m$, welche in der $i$-ten Teilkette von $(\mathcal{Q}_1 v_1) \ldots (\mathcal{Q}_m v_m)$ vorkommen. Mit

$$L_0 := \{\langle y_1, \ldots, y_{q(|x|)}, x\rangle : F_x(y_1, \ldots, y_{q(|x|)} = 1\} \in P$$

kann schließlich (51) in die Form (49) gebracht werden.
Nehmen wir umgekehrt an, dass $L$ durch Bedingung (49) gegeben ist. Wir haben $L \in PSpace$ zu zeigen. Dieser Nachweis kann in völliger Analogie zum Nachweis von QBF $\in PSpace$ (Beweis von Satz 9.5) geführt werden. Wir modellieren mit einem Entscheidungsbaum $T$ die möglichen Belegungen der Stringvariablen $y_1, \ldots, y_{q(|x|}$. Wir verwenden wieder eine platzeffiziente EXHAUSTIVE SEARCH durch $T$ unter Einsatz der Ariadne-Faden Technik. Im Laufe der EXHAUSTIVE SEARCH kann eine Markierung der Knoten von $T$ mit Wahrheitswerten vorgenommen werden, so dass $x$ genau dann zu $L$ gehört, wenn die Wurzel dabei mit Wahrheitswert 1 markiert wird. Die Ausfüllung der technischen Details überlassen wir dem Leser und der Leserin. □

## 9.3 Weitere PSpace-vollständige Probleme

In diesem Abschnitt nennen wir ohne Beweis ein paar weitere PSpace-vollständige Probleme, um einen Eindruck zu vermitteln, welcher Problemtypus in diese Kategorie gehört.

### 9.3.1 Probleme mit Automaten und Grammatiken

**Definition 9.10** Reguläre Ausdrücke *über einem Alphabet* $\Sigma$ *und die von ihnen induzierten formalen Sprachen sind induktiv definiert wie folgt:*

1. $\emptyset, \epsilon, \sigma$ *mit* $\sigma \in \Sigma$ *sind reguläre Ausdrücke. Die hiervon induzierten Sprachen sind (in dieser Reihenfolge) die leere Menge, die Menge* $\{\epsilon\}$ *und die Menge* $\{\sigma\}$.

2. *Wenn* $\alpha, \beta$ *reguläre Ausdrücke sind, dann sind auch* $(\alpha + \beta)$ , $(\alpha \cdot \beta)$ *und* $(\alpha^*)$ *reguläre Ausdrücke. Wenn* $L(\alpha)$ *und* $L(\beta)$ *die von* $\alpha$ *und* $\beta$ *induzierten Sprachen bezeichnen, dann sind die von* $(\alpha+\beta)$ , $(\alpha\cdot\beta)$ *und* $(\alpha^*)$ *induzierten Sprachen (in dieser Reihenfolge)* $L(\alpha) \cup L(\beta)$, $L(\alpha) \cdot L(\beta)$ *und* $(L(\alpha))^*$.

Reguläre Ausdrücke werden zum Beispiel bei Editoren oder in der lexikalischen Analyse von Compilern angewendet. Das folgende Problem fragt danach, ob ein gegebener regulärer Ausdruck eine Sprache induziert, die zumindest ein Wort über dem Grundalphabet ausschließt.

**Definition 9.11** *REGULAR EXPRESSION NON-UNIVERSALITY ist das folgende Problem: Zu gegebenem regulären Ausdruck* $\alpha$ *entscheide ob* $L(\alpha) \neq \Sigma^*$.

**Satz 9.12 (Stockmeyer und Meyer, 1973)** *Für jedes mindestens zweielementige Grundalphabet (also zum Beispiel für das binäre Alphabet* $\{0, 1\}$*) gilt: REGULAR EXPRESSION NON-UNIVERSALITY ist PSPACE-vollständig.*

Der Beweis von Stockmeyer und Meyer erfolgt mit einer generischen polynomiellen Reduktion (also ausgehend von einer beliebigen Sprache $L$ aus PSpace).

Ein LBA (ausgeschrieben: Linear Bounded Automaton) ist im wesentlichen eine $O(n)$-platzbeschränkte NTM.[65] Es ist bekannt, dass die Klasse der von LBAs akzeptierbaren Sprachen identisch ist mit der Klasse der sogenannten kontextsensitiven Sprachen (die aus der Vorlesung *Theoretische Informatik* bekannt sein könnten).

**Definition 9.13** *LBA-ACCEPTANCE ist das Problem zu entscheiden, ob ein gegebener LBA $M$ ein gegebenes Eingabewort $x$ akzeptiert. Hierbei sind also sowohl (eine Kodierung von) $M$ als auch $x$ Bestandteil der Eingabe. LINEAR SPACE ACCEPTANCE ist das entsprechende Problem für $O(n)$-platzbeschränkte DTMs.*

**Satz 9.14 (Karp, 1972)** *LBA-ACCEPTANCE und LINEAR SPACE ACCEPTANCE sind beides PSpace-vollständige Probleme.*

Der Beweis von Karp erfolgt über eine (technisch einfache) generische polynomielle Reduktion unter Verwendung eines „padding argument“ (s. Übung). Wegen des engen Zusammenhangs zwischen LBAs und kontextsensitiven Grammatiken (auf deren formale Definition wir hier verzichten) ergibt sich leicht die

**Folgerung 9.15** *Das Problem zu einer gegebenen kontextsensitiven Grammatik $G$ und einem gegebenen Wort $x$ zu entscheiden, ob $x$ zu der von $G$ induzierten kontextsensitiven Sprache gehört, ist PSpace-vollständig.*

### 9.3.2 Spielprobleme

Es gibt einen natürlichen Zusammenhang zwischen Zwei-Personen Spielen und alternierenden Quantorenketten. Nennen wir die Personen WEISS und SCHWARZ. Die Frage, ob WEISS (im Anzug) eine Gewinnkombination über $m$ Züge hat, liest sich in expandierter Form wie folgt:

| | |
|---|---|
| ($\exists$ $\text{Zug}_1$ für WEISS) | ($\forall$ $\text{Zug}_1$ von SCHWARZ) |
| ($\exists$ $\text{Zug}_2$ für WEISS) | ($\forall$ $\text{Zug}_2$ von SCHWARZ) |
| $\ldots$ | $\ldots$ |
| ($\exists$ $\text{Zug}_m$ für WEISS) | Spielstellung ist gewonnen für WEISS |

Die analoge Frage mit WEISS im Nachzug (also SCHWARZ im Anzug) lässt sich analog mit einer alternierenden Quantorenkette formulieren, die mit einem $\forall$-Quantor beginnt. Im Lichte von Satz 9.9 kann es daher nicht überraschen, dass sich über geeignet definierte Spiele PSpace-vollständige Probleme ergeben. Wenn man die Anzahl der Züge zu einer Konstanten gefriert, ergeben sich analog Probleme, die auf dem entsprechenden Level der polynomiellen Hierarchie vollständig sind. Wir konkretisieren dies an folgendem Spiel auf Graphen:

[65] Aus dem Kompressionssatz folgt, dass eine solche NTM oBdA genau $n$ Zellen (also exakt die das Eingabewort enthaltenden Zellen) besucht.

**Definition 9.16** *Sei $G = (V, E)$ ein Graph mit zwei ausgezeichneten Knoten $s, t \in V$. Das Spiel HEX[G] ist ein Zwei-Personen Spiel*[66]*, bei welchem WEISS über weiße und SCHWARZ über schwarze Spielsteine verfügt. Beide Spieler ziehen abwechselnd mit WEISS im Anzug. Ein Zug besteht darin, einen eigenen Stein auf einen noch unbesetzten Knoten aus $V \setminus \{s, t\}$ zu setzen. Das Spiel ist vorüber, wenn alle Knoten aus $V \setminus \{s, t\}$ von (weißen oder schwarzen) Steinen besetzt sind. Die finale Spielstellung ist genau dann eine* Gewinnstellung *für WEISS, wenn die Knoten $s, t$ zusammen mit den von weißen Steinen besetzten Knoten einen Verbindungspfad von $s$ nach $t$ in $G$ enthalten.*

WEISS muss also mit der Strategie spielen, eine Verbindung von $s$ nach $t$ herzustellen; SCHWARZ ist bemüht, solche Verbindungen mit Hilfe von schwarzen Steinen zu vereiteln.

**Definition 9.17** *GENERALIZED HEX ist das Problem zu gegebenem Graphen $G$ mit ausgezeichneten Knoten $s, t \in V$ zu entscheiden, ob WEISS eine Gewinnstrategie besitzt (also das Spiel gewinnen kann egal mit welcher Strategie SCHWARZ spielt).*

**Satz 9.18 (Even und Tarjan, 1976)** *GENERALIZED HEX ist PSpace-vollständig.*

Der Beweis von Even und Tarjan erfolgt durch Angabe einer polynomiellen Reduktion von QBF auf GENERALIZED HEX.

In ähnlicher Weise kann man zeigen, dass (mehr oder weniger) natürliche Verallgemeinerungen[67] populärer Spiele (wie zum Beispiel SCHACH oder GO) ebenfalls PSpace-vollständig sind.

## 9.4 Offene Probleme

Über die (bisher unbewiesene) $P \neq NP$-Vermutung haben wir schon viel berichtet. Äquivalent dazu (wie wir inzwischen wissen) ist die $P \neq PH$-Vermutung. Die Vermutung $P \neq PSpace$ ist vergleichsweise schwächer. Im Lichte der diversen prädikatenlogischen Charakterisierungen kann man diese und verwandte offene Fragen nun auch folgendermaßen ausdrücken:

**P versus NP** Sind prädikatenlogische Aussagen[68] die einen $(\exists)_{pol}$-Quantor verwenden dürfen, mächtiger als rein aussagenlogische (also quantorenfreie) Aussagen?

**P versus PH** Sind prädikatenlogische Aussagen, die eine alternierende Quantorenkette[69] konstanter Länge verwenden dürfen, mächtiger als rein aussagenlogische Aussagen?

**P versus PSpace** Sind prädikatenlogische Aussagen, die eine alternierende Quantorenkette variabler Länge verwenden dürfen, mächtiger als rein aussagenlogische Aussagen?

**Kollaps von PH** Sind alternierende Quantorenketten der Länge $k + 1$ stets mächtiger als alternierende Quantorenketten der Länge $k$?

[66] Für spezielle Graphen in den USA ein handelsübliches Spiel

[67] Um sinnvoll über Komplexität reden zu können, benötigt man Spielbretter einer variablen Größe.

[68] Gemeint ist stets pränexe Normalform mit Quantoren vom Typ $(\exists)_{pol}$ oder $(\forall)_{pol}$ und einer in Polynomialzeit entscheidbaren Aussage.

[69] Gemeint sind stets Ketten, die zwischen dem $(\exists)_{pol}$- und dem $(\forall)_{pol}$-Quantor abwechseln.

**PH versus PSpace** Sind alternierende Quantorenketten variabler Länge mächtiger als alternierende Quantorenketten konstanter Länge?

Wir haben diese Fragen bewusst etwas informell gehalten, damit die Formulierungen griffig sind. Leider haben die prädikatenlogischen Formulierungen der dahinter liegenden berechnungstheoretischen Probleme bisher nicht wirklich zu einer Klärung der offenen Fragen beigetragen.

# 10 Uniforme versus nicht-uniforme Komplexität

Bei einem uniformen Maschinenmodell wenden wir ein einheitliches Programm auf alle möglichen Eingabewörter über einem gegebenen Alphabet an. Wir haben also nicht die Möglicheit, das Programm an die Länge $n$ des Eingabewortes anzupassen. Software-orientierte Maschinenmodelle (wie zum Beispiel die Turing-Maschinen) sind in der Regel von diesem Typ.

Bei einem nicht-uniformen Maschinenmodell darf die „Maschine“ von der Eingabelänge $n$ abhängig sein. Streng genommen hat man dann eine mit $n$ parametrisierte Familie von Maschinen zur Lösung eines vorgegebenen Problems. Hardware-orientierte Maschinenmodelle (wie zum Beispiel Schaltkreisfamilien) sind in der Regel von diesem Typ.

Wir gehen im Folgenden davon aus, dass das Konzept eines Booleschen Schaltkreises (etwa über der Basis $\{\neg, \vee, \wedge\}$) bekannt ist. Zur Konstruktion eines Schaltkreises wird eine endliche Anzahl von Hardwaregattern azyklisch miteinander verdrahtet. Jedes Gatter berechnet dabei eine elementare Boolsche Funktionen (wie etwa die logische Negation eines Bits bzw. die logische Disjunktion oder Konjunktion zweier Bits). Durch die Verdrahtung zu einem Schaltkreis kann dann im Prinzip jede Boolesche Funktion $f : \{0,1\}^n \to \{0,1\}$ berechnet werden. Im Folgenden bezeiche

$$C := (C_n)_{n \geq 0}$$

eine Schaltkreisfamilie (kurz: SKF), wobei $C_n$ eine Boolesche Funktion in $n$ Variablen berechnet. Diese Funktion notieren wir der Einfachheit halber als $C_n(x)$ mit $x \in \{0,1\}^n$. Wir sagen $C$ *realisiert* die Sprache $L \subseteq \{0,1\}^*$, wenn

$$\forall n \geq 0, \forall x \in \{0,1\}^n : C_n(x) = 1 \Leftrightarrow x \in L \ .$$

Die *Größe* eines Schaltkreises $C_n$ ist die Anzahl seiner Gatter und wird als $size(C_n)$ notiert. $C = (C_n)$ heißt *polynomielle SKF*, wenn $size(C_n)$ polynomiell in $n$ beschränkt ist.

Die Nicht-Uniformität eines Maschinenmodells hat weit reichende Folgen. Zum Beispiel gibt es für jede Sprache $L$ (also auch für nicht-entscheidbare oder gar nicht-semi-entscheidbare) eine SKF, welche $L$ realisiert (einfach darum, weil **jede** Boolesche Funktion realisiert werden kann).[70] Beim Vergleich von SKF's und TM's ist es manchmal hilfreich, nur sogenannte „uniforme“ SKF's zuzulassen. Dabei heißt eine SKF $C = (C_n)$ *uniform*, wenn die Abbildung

$$1^n \mapsto desc(C_n)$$

in Polynomialzeit[71] berechenbar ist, wobei $desc(C_n)$ eine „natürliche“ Kodierung von $C_n$ bezeichnet.[72] Eine uniforme SKF kann nicht superpolynomiell viele Gatter haben und ist daher polynomiell.

Dieses Kapitel ist aufgebaut wie folgt:

[70] Dies wird freilich bedeuten, dass die Funktion $n \mapsto C_n$ i.A. nicht berechenbar ist.

[71] Üblicherweise wird sogar logspace-Berechenbarkeit gefordert. Für unsere Zwecke wird aber die Berechnung von $C_n$ in pol($n$) Schritten ausreichend sein.

[72] „desc“ ist Abkürzung von „description“.

- In den Abschnitten 10.1 und 10.2 zeigen wir, dass jede polynomielle DTM in eine äquivalente uniforme SKF transformiert werden kann und umgekehrt.

- In Abschnitt 10.3 diskutieren wir die Familie $P/poly$ der mit polynomiellen SKF's realisierbaren Sprachen.

- In Abschnitt 10.4 gehen wir erneut auf spärliche (= dünne) Sprachen ein. Es wird sich zeigen, dass $P/poly$ übereinstimmt mit der Klasse der auf spärliche Sprachen Turing-reduzierbaren Sprachen. Außerdem lassen sich die (vermutlich falschen) Aussagen $NP \subseteq P/poly$ bzw. $P = NP$ mit Hilfe von spärlichen und sogenannten kospärlichen Sprachen charakterisieren.

- Im abschließenden Abschnitt 10.5 beweisen wir den Satz von Karp und Lipton, welcher besagt, dass $NP \subseteq P/poly$ zur Folge hätte, dass die polynomielle Hierarchie auf ihren zweiten Level kollabiert.

**Hinweis auf laufende Forschung** Da vermutlich $NP \not\subseteq P/poly$, ist es naheliegend, nach Sprachen aus $NP$ zu suchen, die sich beweisbar nicht durch polynomielle SKF's berechnen lassen. Wegen $P \subseteq P/poly$ würde dies $P \neq NP$ implizieren. Leider hat sich das Problem, superpolynomielle untere Schranken für (Sprachen aus $NP$ realisierende) SKF's zu beweisen, als außerordentlich hart erwiesen.[73] Aus „Notwehr“ diskutiert man daher hauptsächlich in ihrer Rechenkraft eingeschränkte SKFs (wie zum Beispiel motonone oder tiefenbeschränkte SKFs). Im Rahmen dieser Vorlesung können wir aber darauf nicht näher eingehen.

## 10.1 Konversion von Software in Hardware

Es sei $M$ eine $S(n)$-platz- und $T(n)$-zeitbeschränkte DTM mit Zustandsmenge $Z$ und Bandalphabet $\Gamma \supset \Sigma$. Zu einer gegebenen Eingabe $x \in \Sigma^n$ stehe $M$ das Bandsegment mit den Zellen $0, 1, \ldots, S(n), S(n)+1$ zur Verfügung, wobei auf den Zellen 0 und $S(n)+1$ Randmarkierungen stehen, die von $M$ nicht überschritten (und auch nicht gelöscht) werden. Der eigentliche Arbeitsbereich besteht aus den Zellen $1, \ldots, S(n)$. Diese Situation ist in Abbildung 17 illustriert. Anfangs befindet sich $M$ im Startzustand $z_0$, ihr Kopf ist auf Zelle 0

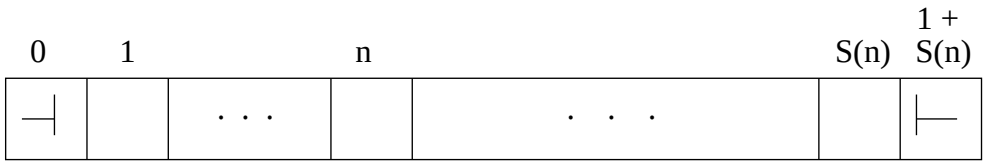



Abbildung 17: Der Arbeitsbereich einer $S(n)$-platzbeschränkten DTM.

positioniert und die Eingabe $x = x_1 \cdots x_n \in \{0,1\}^n$ steht in den Zellen $1, \ldots, n$. Nach exakt $T(n)$ Schritten befindet sich $M$ im Zustand $z_+$, sofern $x \in L_M$, bzw. im Zustand $z_-$, sofern $x \notin L_M$. Zu diesem Zeitpunkt ist das Band (bis auf die Randmarkierungen) gesäubert (also

[73] Bisher sind noch nicht einmal superlineare Schranken bekannt.

leer).[74] Jede Konfiguration von $M$ kann in der offensichtlichen Weise durch einen String aus $K^{2+S(n)}$ über dem Zeichenvorrat

$$K := \Gamma \cup (Z \times \Gamma)$$

beschrieben werden. Das Zeichen aus $Z \times \Gamma$ gibt dabei neben dem aktuellen Zustand auch die Kopfposition an.

**Beispiel 10.1** *Die Anfangskonfiguration (zu Eingabe $x$) ist beschrieben durch*

$$(z_0, \dashv)x_1 \cdots x_n \square \cdots \square \vdash$$

*und die akzeptierende Endkonfiguration durch*

$$(z_+, \dashv)\square \cdots \square \vdash \quad .$$

Die Rechnung von $M$ auf Eingabe $x$ kann durch die *Rechnungstabelle*

$$R := (R_{i,j})_{0 \le i \le T(n), 0 \le j \le S(n)+1}$$

beschrieben werden. Dabei ist $R_{i,j}$ das $j$-te Zeichen der $i$-ten Konfiguration.

**Zentrale Beobachtung** Wegen der lokalen Arbeitsweise von DTMs ist $R_{i+1,j}$ nur von $R_{i,j-1}, R_{i,j}, R_{i,j+1}$ abhängig, d.h.,

$$R_{i+1,j} = f_\delta(R_{i,j-1}, R_{i,j}, R_{i,j+1}) \ ,$$

wobei $f_\delta$ eine durch die Überführungsfunktion $\delta$ implizit gegebene Funktion ist.[75]

Jedes Zeichen aus $K$ kann auf einfache Weise mit

$$k := \lceil \log |K| \rceil$$

Bits kodiert werden. Es bezeichne

$$R'_{i,j} \in \{0,1\}^k$$

die Kodierung von $R_{i,j}$. Dann gilt

$$R'_{i+1,j} = f'_\delta(R'_{i,j-1}, R'_{i,j}, R'_{i,j+1})$$

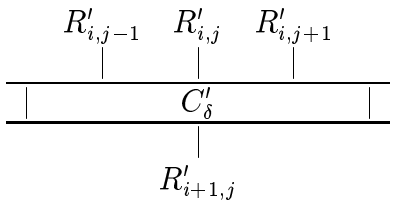
für eine (durch $\delta$ implizit gegebene) Boolesche Funktion

$$f'_\delta : \{0,1\}^{3k} \to \{0,1\}^k \ .$$

Es bezeichne $C'_\delta$ einen Schaltkreis (konstanter Größe), der $f'_\delta$ berechnet:

[74] Durch diese Normierungen haben wir nicht nur eine eindeutige Anfangskonfiguration, sondern auch eine eindeutige (akzeptierende bzw. verwerfende) Endkonfiguration.

[75] Argument $R_{i,j-1}$ entfällt für $j = 0$ (linker Rand); Argument $R_{i,j+1}$ entfällt für $j = S(n) + 1$ (rechter Rand).




Es sollte klar sein, dass durch Parallelschaltung von $2+S(n)$ Duplikaten von $C'_\delta$ (mit Besonderheiten am Rand) ein Schaltkreis $C_\delta$ entsteht, der aus der $i$-ten Zeile der Rechnungstabelle $R'$ die $(i+1)$-te Zeile berechnet. Durch Serienschaltung von $T(n)$ Duplikaten von $C_\delta$ entsteht ein Schaltkreis $C'_M$, der die $T(n)$-te Zeile von $R'$ aus der 0-ten Zeile berechnet. Wenn wir einen Schaltkreis $C_{in}$ voranschalten, der aus $x_1 \cdots x_n$ die $k(2+S(n))$ Bits

$$R'_{0,0} \cdots R'_{0,k(S(n)+2)-1}$$

(also die 0-te Zeile von $R'$) berechnet, und einen Schaltkreis $C_{out}$ hinterherschalten, der die Abbildung

$$R'_{T(n),0} \cdots R'_{T(n),k-1} \mapsto \begin{cases} 1 & \text{falls } R_{T(n),0} = (z_+, \vdash) \\ 0 & \text{falls } R_{T(n),0} = (z_-, \vdash) \end{cases}$$

berechnet, so erhalten wir einen Schaltkreis

$$C_M = C_{out} \circ C'_M \circ C_{in} \quad ,$$

der $L_M$ realisiert, d.h.,

$$C_M(x) = 1 \Leftrightarrow M \text{ akzeptiert } x \Leftrightarrow x \in L_M \quad .$$

$C'_M$ enthält insgesamt

$$(2 + S(n))T(n) = O(S(n)T(n))$$

Duplikate von $C'_\delta$ und hat daher die Größe $O(ST) = O(T^2)$. Es ist nicht schwer zu sehen, dass auch der Gesamtschaltkreis $C_M$ die Größe $O(ST) = O(T^2)$ besitzt und in $O(ST) = O(T^2)$ Schritten konstruiert werden kann.

Wir ziehen aus der gesamten Diskussion das folgende Fazit:

**Satz 10.2** *Falls $L \in P$, dann kann $L$ durch eine uniforme (und somit auch polynomielle) SKF $C = (C_n)$ realisiert werden.*

## 10.2 Konversion von Hardware in Software

Ziel dieses Abschnittes ist die Umkehrung von Satz 10.2:

**Satz 10.3** *Es sei $C = (C_n)$ eine uniforme SKF und*

$$L_C := \{x \in \{0,1\}^* |\ C_{|x|}(x) = 1\}$$

*die von ihr realisierte Sprache. Dann gilt $L_C \in P$.*

**Beweis** Es bezeichne CIRCUIT EVALUATION die Sprache

$$\{\langle desc(C_n), a\rangle |\ n \geq 0, a \in \{0,1\}^n \text{ und } C_n(a) = 1\}\ .$$

Das Mitgliedschaftsproblem zu dieser Sprache besteht im Wesentlichen darin, einen gegebenen Schaltkreis $C_n$ (mit $n$ Boolschen Eingabewerten) an einem gegebenen Booleschen Vektor $a \in \{0,1\}^n$ auszuwerten, was natürlich in Polynomialzeit geschehen kann. Es gilt also

$$\text{CIRCUIT EVALUATION} \in P\ .$$

Der Beweis $L_C \in P$ kann daher geführt werden, indem wir $L_C$ polynomiell auf CIRCUIT EVALUATION reduzieren. Die dazu passende Reduktionsabbildung ist

$$x \mapsto \langle desc(C_{|x|}), x\rangle\ .$$

Sie kann wegen der Uniformität von $C$ in Polynomialzeit berechnet werden und erfüllt (per Definition der beteiligten Sprachen) die Bedingung

$$x \in L_C \Leftrightarrow \langle desc(C_{|x|}), x\rangle \in \text{CIRCUIT EVALUATION}\ .$$

□

Aus den Sätzen 10.2 und 10.3 ziehen wir die

**Folgerung 10.4** *$P$ ist die Klasse aller durch uniforme SKFs realisierbaren Sprachen.*[76]

## 10.3 P/poly: die nicht-uniforme „Schwester“ von P

**Definition 10.5** *$P/poly$ bezeichne die Klasse aller Sprachen, die durch polynomielle (aber nicht notwendig uniforme) SKFs realisiert werden können.*

Aus Folgerung 10.4 ergibt sich unmittelbar die die Inklusion $P \subseteq P/poly$. Am Ende dieses Abschnittes werden wir nachweisen, dass diese Inklusion echt ist. Die eigentliche (und bis heute ungeklärte) Frage lautet: wie mächtig ist die Klasse $P/poly$? Wir werden in diesem und den beiden folgendenen Abschnitten etwas Licht auf diese Frage werfen und beginnen mit dem

**Satz 10.6** *Eine Sprache $L$ gehört zur Klasse $P/poly$ gdw ein Polynom $q$, eine Sprache $L_0 \in P$ und eine Folge $(a_n)_{n\geq 0}$ von binären Strings $a_n \in \{0,1\}^*$ einer durch $q(n)$ beschränkten Länge existieren, so dass*

$$L = \{x |\ \langle a_{|x|}, x\rangle \in L_0\}\ .$$

[76] Streng genommen müssten wir ein binäres Alphabet $\Sigma = \{0,1\}$ zugrunde legen. Für Sprachen über einem erweiterten Alphabet $\Sigma$ gilt die Folgerung aber für die davon induzierte Sprache über $\{0,1\}$ (nach Festlegung eines Binärcodes für die Symbole von $\Sigma$).

**Beweis** Wir beweisen zunächst die Richtung von links nach rechts und setzen daher $L \in P/poly$ voraus. Es sei $C = (C_n)$ eine $L$ realisierende polynomielle SKF. Mit

$$a_n := desc(C_n) \text{ und } L_0 := \{\langle desc(C_{|x|}), x\rangle |\ C_{|x|}(x) = 1\}$$

ergibt sich die gewünschte Charakterisierung von $L$: die Strings $|a_n|$ haben nämlich eine polynomiel beschränkte Länge, da $C$ eine polynomielle SKF ist und $L_0$ gehört (als Teilproblem von CIRCUIT EVALUATION) zur Klasse $P$.
Für die Beweisrichtung von rechts nach links setzen wir jetzt die im Satz beschriebene Charakterisierung von $L$ voraus und weisen $L \in P/poly$ nach. Sei also $q(n) \geq n$ ein Polynom, $a_n$ eine Folge von Strings mit $|a_n| \leq q(n)$, $L_0 \in P$ und $L = \{x|\ \langle a_{|x|}, x\rangle \in L_0\}$. Betrachte eine Boolesche Eingabe $x \in \{0,1\}^n$. Wir können einen Schaltkreis $C'_n$ zur Berechnung von

$$x \mapsto \langle a_n, x\rangle$$

und einen Schaltkreis $C''_n$ zum Entscheiden von

$$\langle a_n, x\rangle \in L_0?$$

in Serie schalten und erhalten so eine SKF $C = (C_n)$ mit $C_n = C''_n \circ C'_n$, welche $L$ realisiert. Wegen $|a_n| \leq q(n)$ gilt sowohl $|\langle a_n, x\rangle| = O(q(n))$ als auch $size(C'_n) = O(q(n))$. $C'$ ist also eine polynomielle SKF. Zusammen mit $L_0 \in P$ und Satz 10.2 ergibt sich nun, dass $C''$ und somit auch $C$ eine polynomielle SKF ist. Insgesamt hat sich also $L \in P/poly$ ergeben. □

Die Bits in $a_n$ heißen auch *Hinweisbits (advice bits)*. Satz 10.6 lässt sich dann auch folgendermaßen lesen: die Sprachen aus $P/poly$ sind dadurch charakterisiert, dass sie von einer DTM in Polynomialzeit akzeptiert werden können, wenn diese neben der eigentlichen Eingabe $x \in \Sigma^n$ polynomiell viele zusätzliche Hinweisbits $a_n$ (die von $x$ nur über $n = |x|$ abhängen dürfen) bekommt. So gesehen beruht der Beweis von Satz 10.6 auf zwei Grundideen:

- Gegeben eine uniforme $L$ realisierende SKF: dann kann $desc(C_n)$ einer ($C_n$ simulierenden) DTM in Form von Hinweisbits zugänglich gemacht werden.
- Gegeben der polynomielle Algorithmus zum Erkennen von Strings aus $L$ mit Hilfe zusätzlicher Hinweisbits $a_n$: dann können die Bits aus $a_n$ in einen (den Erkennungsalgorithmus realisierenden) Schaltkreis $C_n$ „gehardwired“ werden.

Das Verifizieren der Aussage $x \in L$ mit Hinweisbits (wie bei Sprachen aus $P/poly$) und das Verifizieren von $x \in L$ mit Ratebits (wie bei Sprache aus $NP$) scheinen verwandt zu sein. Wir machen aber auf zwei wichtige Unterschiede aufmerksam:

| *NP* | *P/poly* |
|---|---|
| Zertifikat $y$ zum Nachweis von $x \in L$ kann in Abhängigkeit von $x$ gewählt werden. | Hinweisbits $a_n$ zum Nachweis von $x \in L$ hängen nur von $n = \|x\|$ ab (also gleiche Hinweisbits für Eingaben gleicher Länge). |
| Für $x \notin L$ existiert kein falsches Zertifikat (kein Ratestring $y$, der zum Akzeptieren von $x$ führt. | Es könnte falsche Hinweise geben (Strings $a'_n$ mit $\langle a'_n, x\rangle \in L_0$ obwohl $x \notin L$ bzw. Strings $a'_n$ mit $\langle a'_n, x\rangle \notin L_0$ obwohl $x \in L$ |

Wir werden sehen, dass *NP* und *P/poly* sehr unterschiedliche Klassen sind. Zum Beispiel sind wegen $NP \subseteq PSpace$ alle Sprachen in *NP* entscheidbar. *P/poly* hingegen enthält u.a. auch nicht entscheidbare bzw. noch nicht einmal semi-entscheidbare Sprachen, wie aus dem nächsten Resultat hervorgeht:

**Satz 10.7** *Jede unäre Sprache $L \subseteq \{0\}^*$ gehört zu P/poly.*

**Beweis** Das Hinweisbit

$$a_n := \begin{cases} 1 & \text{falls } 0^n \in L \\ 0 & \text{falls } 0^n \notin L \end{cases}$$

reicht aus, um $L$ in Polynomialzeit zu entscheiden: die Dekodierung von $a_n$ aus $\langle a_n, x\rangle$ liefert die Antwort. □

In der Vorlesung *Theoretische Informatik* wurden zahlreiche nicht entscheidbare (bzw. nicht einmal semi-entscheidbare) Sprachen $L \subseteq \{0,1\}^*$ präsentiert. Zu $x \in \{0,1\}^*$ bezeichne $N(x) \in \mathbb{N}$ die durch die Bitfolge $1x$ binär kodierte natürliche Zahl. Da die Abbildung

$$x \mapsto 0^{N(x)} \text{ (unäre Kodierung von } N(x))$$

berechenbar ist, ist mit $L$ auch die unäre Sprache

$$L' := \{0^{N(x)} |\ x \in L\}$$

nicht (semi-)entscheidbar. Somit gibt es zu jeder nicht (semi-)entscheidbaren Sprache ein unäres Pendant.

**Folgerung 10.8**

1. *P/poly enthält insbesondere alle unären nicht (semi-)entscheidbaren Sprachen.*
2. *$P \subset P/poly$.*
3. *P/poly ist keine Teilmenge der semi-entscheidbaren Sprachen und kann daher in keiner uniformen (durch eine Platz- oder Zeitschranke gegebenen) Komplexitätsklasse enthalten sein.*

## 10.4 Spärliche und kospärliche Sprachen

Wir erinnern an Definition 6.93: die *Dichte* einer Sprache $L$ ist gegeben durch

$$dens_L(n) := |\{x \in L :\ |x| \leq n\}|$$

und $L$ heißt *dünn* oder auch *spärlich*, wenn ihre Dichte polynomiell in $n$ beschränkt ist. In diesem Abschnitt benötigen wir weiterhin die

**Definition 10.9** *Eine Sprache $L \subseteq \{0,1\}^*$ heißt* kospärlich*, wenn ihr Komplement $\bar{L} = \{0,1\}^* \setminus L$ spärlich ist.*

Der folgende Satz charakterisiert $P/poly$ mit Hilfe von spärlichen Sprachen:

**Satz 10.10** *$P/poly$ stimmt überein mit der Klasse der auf spärliche Sprachen Turing-reduzierbaren Sprachen.*

**Beweis** Wir setzen zunächst $L \in P/poly$ voraus und wollen eine spärliche Sprache $S$ mit $L \leq_T S$ ausfindig machen. Gemäß Satz 10.6 existiert ein Polynom $q$ (mit Koeffizienten aus $\mathbb{N}$), Hinweise $(a_n)_{n\geq 0}$ mit $|a_n| \leq q(n)$ und eine Sprache $L_0 \in P$, so dass

$$L = \{x |\ \langle a_{|x|}, x\rangle \in L_0\} \ .$$

**Idee** Entwerfe $S$ so, dass das $S$-Orakel nach den Bits von $a_n$ (wobei $n = |x|$) befragt werden kann. Eine DOTM $M[S]$ kann dann $x \in L$ testen wie folgt:

1. Ermittle $a_n$ bitweise durch $q(n)$ entsprechende Anfragen an das $S$-Orakel.
2. Setze die polynomiell zeitbeschränkte DTM $M_0$ für die Sprache $L_0$ auf $\langle a_n, x\rangle$ an und akzeptiere gdw $M_0$ akzeptiert.

Kommen wir zur Umsetzung der geschilderten Idee und definieren für alle $n \geq 0$ und alle $i = 1, \ldots, q(n)$ die Strings

$$s_i^n := 0^{i-1}10^{q(n)-i} \in \{0,1\}^{q(n)} \ .$$

Die Sprache $S$ sei gegeben durch

$$S := \{s_i^n |\ n \geq 0, 1 \leq i \leq q(n) \text{ und das } i\text{-te Bit von } a_n \text{ ist } 1\} \ .$$

Auf Anfrage $s_i^n$ muss das $S$-Orakel mit dem $i$-ten Bit von $a_n$ antworten. $S$ ist spärlich, da offensichtlich[77]

$$|S \cap \{0,1\}^m| \leq m \ .$$

Damit wäre gezeigt, dass jede Sprache aus $P/poly$ auf eine spärliche Sprache Turing-reduzierbar ist.
Für die umgekehrte Beweisrichtung setzen wir $L \leq_T S$ voraus, wobei $S$ eine spärliche Sprache bezeichnet, d.h., es gibt ein Polynom $p(n)$ , so dass $|S \cap \{0,1\}^n| \leq p(n)$. Wir haben zu zeigen, dass sich das Wortproblem für $L$ mit Hilfe geeigneter (polynomiell längenbeschränkter) Hinweise $(a_n)_{n\geq 0}$ in Polynomialzeit lösen lässt.

**Idee** Gib eine Liste $L_S$ aller Elemente von $S$ (bis zu einer geeigneten Maximallänge) als Hinweis. Mit Hilfe von $L_S$ kann das $S$-Orakel simuliert werden.

Kommen wir zur Umsetzung der Idee. Es sei $M'[S]$ die polynomiell zeitbeschränkte DOTM, die für jedes $x \in \Sigma^n$ in $q(n)$ Schritten entscheiden kann, ob $x \in L$. $M'$ kann maximal $q(n)$ viele maximal $q(n)$ lange Anfragen an das $S$-Orakel stellen. Sei $s_1, \ldots, s_{r(n)}$ eine Auflistung

[77] Hierbei nutzen wir aus, dass $q(n)$ (als Polynom mit Koeffizienten aus $\mathbb{N}$) mit $n \in \mathbb{N}_0$ streng monoton wächst. Parameter $m$ hat also maximal ein Urbild $n$ mit $m = q(n)$.

aller Elemente von $S$ der Maximallänge $q(n)$. Wegen der Spärlichkeit von $S$ ist $r(n) \leq p(q(n))$ polynomiell beschränkt. Setze

$$a_n := \langle s_1, \ldots, s_{r(n)} \rangle$$

und beachte, dass

$$|a_n| = O(q(n)p(q(n))) \ .$$

Nun kann (wie geplant) das Wortproblem für $L$ entschieden werden, indem eine DTM $M$ (ohne Orakel), angesetzt auf Eingabe $\langle a_n, x \rangle$ mit $x \in \Sigma^n$, die Rechnung der DOTM $M'$ auf Eingabe $x$ effizient simuliert. Eine etwaige Anfrage an das $S$-Orakel kann durch Inspektion von $a_n$ beantwortet werden. □

Wir wissen bereits, dass es vermutlich keine spärliche Sprache gibt, die *NP*-hart (unter polynomieller Reduktion) ist, denn nach dem Satz von Mahaney (s. Satz 6.95) hätte dies $P = NP$ zur Folge. Dies schließt aber noch nicht die Existenz von spärlichen Sprachen aus, die *NP*-hart unter (den mächtigeren) Turing-Reduktionen sind. Das folgende Resultat legt jedoch die Vermutung nahe, dass dies nicht der Fall ist:

**Folgerung 10.11** *Es gilt $NP \subseteq P/poly$ gdw es eine spärliche Sprache $S$ gibt, die NP-hart unter Turing-Reduktionen ist.*

**Beweis** Die Voraussetzung $NP \subseteq P/poly$ impliziert, dass SAT $\in P/poly$, was gemäß Satz 10.10 die Existenz einer spärlichen Sprache $S$ mit SAT $\leq_T S$ zur Folge hat. Folglich ist $S$ *NP*-hart unter Turing-Reduktionen.
Sei umgekehrt vorausgesetzt es gäbe eine spärliche und zugleich unter Turing-Reduktionen *NP*-harte Sprache $S$. Somit gilt $L \leq_T S$ für jede Sprache $L \in NP$. Gemäß Satz 10.10 folgt hieraus $NP \subseteq P/poly$. □

$NP \subseteq P/poly$ würde bedeuten, dass alle Sprachen aus *NP* (inklusive der *NP*-vollständigen) durch SKFs polynomieller Größe realisiert werden können. Es wird jedoch vermutet, dass dies nicht der Fall ist. Wir fassen die diversen Vermutungen noch einmal zusammen:

**Vermutung 1** Es gibt keine spärliche Sprache, die *NP*-hart (unter polynomiellen Reduktionen) ist. (Ansonsten wäre $P = NP$.)

**Vermutung 2** Es gibt keine spärliche Sprache, die *NP*-hart unter Turing-Reduktionen ist. (Ansonsten wäre $NP \subseteq P/poly$.)

Die gleichen Vermutungen werden auch für kospärliche Sprachen gehegt. Dies wird (zumindest für polynomielle Reduktionen) gestützt durch folgenden

**Satz 10.12** *Es gilt $P = NP$ gdw es eine kospärliche Sprache $S$ gibt, die NP-hart (unter polynomiellen Reduktionen) ist.*

**Beweis** Wir setzen zunächst $P = NP$ voraus. Die Menge $S := \{0,1\}^* \setminus \{0\}$ ist trivialerweise kospärlich. Darüberhinaus ist $S$ aber auch $NP$-hart, da jede Sprache $L \in NP = P$ mit der Reduktionsabbildung

$$x \mapsto \begin{cases} 1 & \text{falls } x \in L \\ 0 & \text{falls } x \notin L \end{cases}$$

polynomiell auf $S$ reduzierbar ist.
Für die umgekehrte Beweisrichtung gehen wir von einer kospärlichen und zugleich $NP$-harten Sprache $S$ aus und weisen $P = NP$ nach, indem wir einen Polynomialzeitalgorithmus $A$ für SAT entwerfen. Wegen der $NP$-Härte von $S$ gilt SAT $\leq_{pol} S$ via eine geeignete Reduktionsabbildung $R : \Sigma^* \to \Sigma^*$. Der Algorithmus $A$ für SAT, den wir im Folgenden entwerfen, ist ähnlich gestrickt wie der Algorithmus für SAT, den wir im Beweis des Satzes von Berman (s. Satz 6.96) verwendet haben. Wir „recyclen“ ein paar wesentliche Bestandteile dieses Beweises:

- Zu einer CNF-Formel $F$ über $n$ Booleschen Variablen $v_1, \ldots, v_n$ und einer partiellen Belegung $a \in \{0,1\}^i$ mit $i \in \{0, \ldots, n\}$ assoziieren wir wieder die vereinfachte CNF-Formel $F[a]$, die sich aus $F$ durch Elimination aller bereits erfüllter Klauseln und aller bereits falsifizierter Literale ergibt. Wir erinnern daran, dass $F[\epsilon] = F$ und $F[a] \in \{0,1\}$ für jedes $a \in \{0,1\}^n$ (die Grenzfälle $i = 0$ bzw. $i = n$). Wir erinnern weiterhin daran, dass $F[a]$ erfüllbar ist gdw $F[a0]$ oder $F[a1]$ erfüllbar ist.

- Es bezeichne $T$ den vollständigen binären Baum der Tiefe $n$, dessen $2^n$ Blätter in einer 1-zu-1 Beziehung zu den möglichen Belegungen $a \in \{0,1\}^n$ der Booleschen Variablen $v_1, \ldots, v_n$ stehen. Jeder Knoten der Tiefe $i$ entspricht eindeutig einer partiellen Belegung $a \in \{0,1\}^i$. Wir verwenden für diesen Knoten die Notation $u[a]$.

- Ein Exponentialzeitalgorithmus könnte in einem „top-down pass“ den Baum $T$ aufbauen und in einem „bottom-up pass“ die Knoten des Baumes mit Booleschen Wahrheitswerten (0 oder 1) beschriften, so dass der Knoten $u[a]$ mit 1 beschriftet wird gdw die Formel $F[a]$ erfüllbar ist. Damit wäre die ursprünglich gegebene Formel $F$ erfüllbar gdw die Wurzel von $T$ (also der Knoten $u[\epsilon]$) mit 1 beschriftet ist.

- Anstatt nun aber $T$ (in Exponentialzeit) vollständig aufzubauen, versuchen wir die gleiche algorithmische Grundidee auf einem „Anfangsstück“ von $T$ (einer polynomiell in der Kodierungslänge $N$ der Eingabeformel $F$ beschränkten Größe) durchzusetzen. Dabei werden die Konzepte der „Lazy Evaluation“ und des „Hashing“ eingesetzt.

- Zur Implementierung von „lazy evaluation“ erfolgen die Arbeitsschritte während eines Durchlaufes von $T$ in Präordnung. Wenn aber die (zuerst inspizierte) Formel $F[a0]$ sich als erfüllbar erwiesen hat (der Knoten $u[a0]$ ist dann mit 1 markiert), können wir $F[a]$ als erfüllbar deklarieren (also den Knoten $u[a]$ mit 1 markieren), ohne den von $u[a1]$ induzierten Teilbaum aufzubauen (Idee des „Pruning“).

- Als Hashfunktion verwenden wir die Reduktionsabbildung $R$. Wenn sich eine Formel $F[a]$ als unerfüllbar erwiesen hat, dann sind auch alle Formeln $F[a']$ mit $R(F[a']) =$

$R(F[a])$ unerfüllbar. Wenn daher in $T$ ein Knoten $u[a]$ mit 0 markiert wird, dann können alle Knoten $u[a']$ mit $R(F[a']) = R(F[a])$ (sowie deren Nachfolgerknoten, sofern diese schon aufgebaut wurden) ebenfalls mit 0 markiert werden. Überflüssig zu erwähnen, dass wir den Teilbaum eines bereits mit 0 markierten Knoten nicht aufzubauen brauchen und dass auch keine neuen Knoten $u[a']$ aufgebaut werden, die die Bedingung $R(F[a']) = R(F[a])$ für einen bereits mit 0 markierten Knoten $u[a]$ erfüllen.

Wir verzichten auf eine formale Beschreibung des resultierenden Algorithmus $A$. Anhand der geschilderten Vorgehensweise sollte klar sein, dass $A$ erstens $F$ akzeptiert (also die Wurzel von $T$ mit 1 markiert) gdw $F$ erfüllbar ist (Korrektheit). Weiterhin sollte klar sein, dass die Laufzeit polynomiell in $N$ beschränkt ist, sofern nur ein polynomiell kleines Anfangsstück $T'$ von $T$ aufgebaut wird. Zu diesem Zweck betrachten wir die „Hashtabelle“

$$H := \{t \in \bar{S} |\ \exists u[a] \in T' :\ u[a] \text{ ist mit 0 markiert und } R(F[a]) = t\}\ .$$

Beachte, dass diese Tabelle wegen der Kospärlichkeit von $S$ eine in $n \leq N$ polynomiell beschränkte Größe hat, sagen wir $|H| \leq q(n)$ für ein Polynom $q$. Die polynomielle Größenbeschränkung von $T'$ ergibt sich nun leicht aus folgender

**Hilfsbehauptung** Der Durchlauf in Präordnung habe einen Knoten $u[a] \in T'$ mit $a \in \{0,1\}^i$ und $i \in \{0, \ldots, n-1\}$ erreicht und befinde sich in „top-down“ Richtung. Dann wird während des Aufenthaltes im von $u[a]$ induzierten Unterbaum nach Aufnahme von höchstens $n - i$ neuen Knoten in $T'$ einer der folgenden beiden Fälle eingetreten sein:

**Fall 1** $u[a]$ hat eine Boolesche Markierung (0 oder 1) erhalten.

**Fall 2** $u[a]$ hat noch keine Boolesche Markierung erhalten, aber die Hashtabelle $H$ hat sich um mindestens 1 vergrößert.

Wir zeigen zunächst, wie unser Beweis mit der Hilfsbehauptung abgeschlossen werden kann. Zu diesem Zweck zerlegen wir den Präordnungsdurchlauf in Arbeits- und Backtracking-Phasen. Zu Beginn einer Arbeitsphase sei der Durchlauf beim Knoten $u[a]$ angelangt (anfangs $u[\epsilon]$) und befinde sich in „top-down“ Richtung. Die Arbeitsphase endet, mit dem Eintreten von Fall 1 bzw. Fall 2. Wenn Fall 1 eintritt, dann belasten wir (beweistechnisch) den Knoten $u[a]$ mit „Kosten“ $n$. Beachte, dass die in der Arbeitsphase neu in $T'$ aufgenommenen Knoten (auch in Zukunft) nicht mit Kosten belastet werden, da der Durchlauf den von $u[a]$ induzierten Unterbaum nicht mehr betreten wird. Wenn Fall 2 eintritt, nennen wir die während der Arbeitsphase in $T'$ neu aufgenommenen Knoten *$H$-generiert*. Nach der Arbeitsphase beginnt die (evtl. leere) Backtracking-Phase die solange währt, bis der Durchlauf wieder die „top-down“ Richtung anvisiert.[78] Beachte, dass der Knoten, bei welchem dann die nächste Arbeitsphase beginnt, $H$-generiert (oder die Wurzel) sein muss. Wegen $|H| \leq q(n)$ kann es höchstens $nq(n)$ $H$-generierte Knoten in $T'$ geben. Die Anzahl der weiteren Knoten in $T'$ ist nach oben durch die Gesamtkosten der $H$-generierten Knoten (plus der Wurzel) beschränkt. Diese betragen höchstens $(1 + nq(n))n$. Folglich ist die Anzahl der Knoten in $T'$ polynomiell

[78] Bechte, dass während des „backtracking“ keine neuen Knoten in $T'$ aufgenommen werden.

in $n \leq N$ beschränkt.
Der Beweis des Satzes wird nun durch den Beweis der Hilfsbehauptung abgeschlossen. Wir verwenden vollständige Induktion nach $i = n-1, \ldots, 0$. Für $i = n-1$ ist der Durchlauf bei einem Knoten $u[a]$ mit $a \in \{0,1\}^{n-1}$ angelangt und befindet sich in „top-down" Richtung. Die Knoten $u[a0]$ und $u[a1]$ sind dann Blätter in $T$. Falls Knoten $u[a0]$ in $T'$ aufgenommen wird, dann gilt entweder $F[a0] = 1$ oder $F[a0] = 0 \wedge R(F[a0]) \notin H$. Im ersten Fall werden $u[a0]$ und $u[a]$ mit 1 markiert und Fall 1 der Hilfsbehauptung ist eingetreten. Im zweiten Fall hat sich die Hashtabelle um ein neues Element vergrößert und Fall 2 der Hilfsbehauptung ist eingetreten. Nehmen wir nun an, dass $u[a0]$ nicht in $T'$ aufgenommen wurde. Dies kann nur daran liegen, dass ein Abgleich mit $H$ ergeben hatte, dass $F[a0]$ nicht erfüllbar ist. In diesem Fall versucht der Präordungsdurchlauf nun sein Glück beim Knoten $u[a1]$. Es gelten nun die analogen Betrachtungen wie zuvor beim Knoten $u[a0]$. Insgesamt hat sich (wie gewünscht) ergeben, dass nach Aufnahme von maximal einem neuen Knoten in $T'$ der Fall 1 oder der Fall 2 der Hilfsbehauptung eingetreten ist. Wir betrachten nun einen Index $i < n-1$ und setzen induktiv voraus, dass die Hilfsbehauptung sich für $i+1$ bereits als richtig erwiesen hat. Der Durchlauf ist also bei einem Knoten $u[a]$ mit $a \in \{0,1\}^i$ angelangt und befindet sich in „top-down" Richtung. Wenn $u[a0]$ in $T'$ aufgenommen wird, dann gilt per Induktionsvoraussetzung, dass nach Aufnahme von höchstens $n-(i+1)$ weiteren Knoten in $T'$ (für $u[a0]$) Fall 1 oder Fall 2 eingetreten ist. Wenn für $u[a0]$ Fall 2 eintritt (neues Element in $H$), dann ist Fall 2 somit auch für den Knoten $u[a]$ nach Aufnahme von höchstens $n-i$ Knoten eingetreten. Wenn für $u[a0]$ Fall 1 eingetreten ist, dann unterscheiden wir zwei Unterfälle. Hat $u[a0]$ die Boolesche Markierung 1, dann wird auch $u[a]$ mit 1 markiert und für $u[a]$ ist dann ebenfalls Fall 1 eingetreten. Hat aber $u[a]$ die Boolesche Markierung 0, dann wird $R(F[a0])$ als neues Element in $H$ aufgenommen[79] und für $u[a]$ ist Fall 2 der Hilfsbehauptung eingetreten. Dies schließt den induktiven Beweis der Hilfsbehauptung und damit auch den Gesamtbeweis ab. □

## 10.5 Der Satz von Karp und Lipton

Dieser Abschnitt ist dem Beweis des folgenden Resultates gewidmet:

**Satz 10.13 (Karp und Lipton, 1980)** $NP \subseteq P/poly \Rightarrow PH = \Sigma_2 = \Pi_2$.

**Beweis** Wir merken zunächst an, dass es ausreicht, $\Pi_2 \subseteq \Sigma_2$ zu zeigen. Hieraus folgt nämlich durch Dualisierung $\Sigma_2 = \text{co-}\Pi_2 \subseteq \text{co-}\Sigma_2 = \Pi_2$ und somit $\Sigma_2 = \Pi_2$, was wiederum (Anwendung 2 des Satzes von Wrathall) $PH = \Sigma_2$ zur Folge hat.
Sei nun $L$ eine beliebige aber fest ausgewählte Sprache aus $\Pi_2$. Somit existieren ein Polynom $p$ und eine Sprache $L' \in \Sigma_1 = NP$, so dass

$$L = \{x |\ \forall y \in \{0,1\}^{p(|x|)} : \langle y, x \rangle \in L'\} \ . \tag{52}$$

Wir haben $L \in \Sigma_2$ zu zeigen und gemäß dem Satz von Wrathall reicht es dazu aus, eine Beschreibung der Sprache $L$ vom „$\exists\forall$"-Typ anzufertigen.

[79] Wenn dieses Element schon zu $H$ gehören würde, dann wäre entweder $u[a0]$ gar nicht erst aufgebaut worden oder für $u[a0]$ wäre der Fall 2 der Hilfsbehauptung eingetreten.

Gemäß unserer Voraussetzung $NP \subseteq P/poly$ gehört die Sprache 3-SAT zu $P/poly$. Im Folgenden repräsentiere $F_n$ eine CNF-Formel über den Booleschen Variablen $v_1, \ldots, v_n$ mit Klauseln der Länge höchstens 3 (kurz: 3-Klauseln).[80] Insgesamt gibt es nur $O(n^3)$ solcher 3-Klauseln, weswegen die binäre Kodierungslänge von $F_n$ polynomiell in $n$, sagen wir durch $q(n)$, nach oben beschränkt ist. Im folgenden identifizieren wir die Formel $F_n$ mit ihrer binären (auf Länge $q(n)$ ausgepolsterten) Kodierung, d.h., $F_n \in \{0,1\}^{q(n)}$. Wegen 3-SAT $\in P/poly$ gibt es eine 3-SAT realisierende polynomielle SKF ($C_n^{3-SAT}$). Schaltkreis $C_n^{3-SAT}$ erhält also als Eingabe eine CNF-Formel $F_n$ und gibt eine 1 aus gdw $F_n$ erfüllbar ist. Auch $C_n^{3-SAT}$ identifizieren wir mit seiner binären Kodierung, d.h., $C_n^{3-SAT} \in \{0,1\}^{r(n)}$ für ein geeignet gewähltes Polynom $r(n)$. Der Schlüssel zum Beweis liegt in folgender

**Definition** Eine Kollektion $(C_0, \ldots, C_m)$ von Schaltkreisen, wobei $C_i \in \{0,1\}^{r(i)}$ einen Schaltkreis mit $q(i)$ Eingangsknoten repräsentiert, heißt *3-SAT Tester*, falls

$$\forall i = 0, \ldots, m, \forall F_i \in \{0,1\}^{q(i)} : C_i(F_i) = 1 \Leftrightarrow F_i \in \text{3-SAT} \ .$$

Offensichtlich ist zum Beispiel $(C_0^{3-SAT}, \ldots, C_m^{3-SAT})$ ein 3-SAT Tester.
Wir verwenden jetzt das Konzept des 3-SAT Testers, um, ausgehend von der Beschreibung (52), zu einer neuen Beschreibung (vom „$\exists\forall$-Typ“) der Sprache $L$ zu gelangen. Hierzu nutzen wir $L' \leq_{pol}$ 3-SAT aus und bedienen uns einer Reduktionsabbildung

$$\langle y, x \rangle \mapsto F_{s(n)}^{x,y} \ ,$$

die einem String $\langle y, x \rangle$ eine CNF-Formel $F_{s(n)}^{x,y}$ über den Booleschen Variablen $v_1, \ldots, v_{s(n)}$ mit Klauseln der Länge höchstens 3 zuordnet, so dass

$$\langle y, x \rangle \in L' \Leftrightarrow F_{s(n)}^{x,y} \in \text{3-SAT} \ .$$

Die Beschreibung (52) von $L$ lässt sich dann umschreiben wie folgt:

$$\begin{aligned} L \ = \ \Big\{ x \ \Big| \ & \exists (C_0, \ldots, C_{s(|x|)}) \in \times_{i=0}^{s(|x|)} \{0,1\}^{r(i)}, \forall y \in \{0,1\}^{p(|x|)} : \\ & C_{s(|x|)} \left( F_{s(|x|)}^{x,y} \right) = 1 \text{ und } (C_0, \ldots, C_{s(|x|)}) \text{ ist ein 3-SAT Tester} \Big\} \ . \end{aligned}$$

Die Bedingung $C_{s(|x|)} \left( F_{s(|x|)}^{x,y} \right) = 1$ lässt sich in Polynomialzeit testen (CIRCUIT EVALUATION). Falls die Sprache der 3-SAT Tester zu co-$NP = \forall[P]$ gehört, dann können wir aus der neuen Beschreibung von $L$ folgern, dass

$$L \in (\exists)_{\text{pol}} (\forall)_{pol} (\forall)_{pol} [P] = (\exists)_{\text{pol}} (\forall)_{pol} [P] \ ,$$

womit der Beweis für $L \in \Sigma_2$ erbracht wäre. Bleibt also nur noch zu zeigen, dass

$$\text{3-SAT Tester} \in (\forall)_{pol} [P] \ .$$

[80] Es ist hier ausnahmsweise einmal praktischer auch Klauseln mit weniger als 3 Literalen zuzulassen.

Zu diesem Zweck nutzen wir wieder aus, dass

$$F_n \in \text{3-SAT} \Leftrightarrow F_n[0] \in \text{3-SAT} \vee F_n[1] \in \text{3-SAT} \ ,$$

wobei $F_n[a]$ wieder für die CNF-Formel stehe, die aus $F_n$ vermöge einer partiellen Belegung $a \in \{0,1\}^j$ hervorgeht. Für $(C_0, \ldots, C_m) \in \times_{i=0}^{m} \{0,1\}^{r(i)}$ erhalten wir folgende äquivalente Bedingungen:

$(C_0, \ldots, C_m)$ ist ein 3-SAT Tester

$$\begin{aligned}
&\Leftrightarrow \quad \left(\forall F_m \in \{0,1\}^{q(m)} : C_m(F_m) = C_{m-1}(F_m[0]) \vee C_{m-1}(F_m[1])\right) \\
&\qquad \wedge \left((C_0, \ldots, C_{m-1}) \text{ ist ein 3-SAT Tester}\right) \\
&\Leftrightarrow \\
&\ldots \\
&\Leftrightarrow \quad \left( \forall (F_1, \ldots, F_m) \in \times_{i=1}^{m} \{0,1\}^{q(i)}, \forall i \in \{1, \ldots, m\} : \overbrace{C_i(F_i) = C_{i-1}(F_i[0]) \vee C_{i-1}(F_i[1])}^{\text{in Polynomialzeit testbar}} \right) \\
&\qquad \wedge \underbrace{\left(C_0 \text{ ist ein 3-SAT Tester}\right)}_{\text{in konstanter Zeit testbar}}
\end{aligned}$$

Wir haben damit für die Sprache der 3-SAT Tester eine Beschreibung vom „∀-Typ“ gefunden, womit der gesamte Beweis abgeschlossen ist. □

# 11 Probabilistische Komplexitätsklassen

Eine *probabilistische Turing-Maschine (PTM)* arbeitet wie eine NTM mit zwei Entscheidungsalternativen pro Schritt. Der Unterschied zwischen NTMs und PTMs macht sich technisch erst bei der Definition der von $M$ erkannten Sprache $L_M$ bemerkbar. Eine NTM $M$ akzeptiert einen String $x \in \Sigma^*$ (d.h. $x \in L_M$) gdw $M$ mindestens eine akzeptierende Rechnung auf Eingabe $x$ besitzt. Bei einer PTM hingegen werden wir i.A. fordern, dass es

- auf Eingaben $x \in L_M$ „viele“
- auf Eingaben $x \notin L_M$ „wenige“

akzeptierende Rechnungen gibt. Um die Begriffe „viele“ und „wenige“ zu quantifizieren, sehen wir die Rechnung einer PTM $M$ als zufällig an: in jedem Schritt wird eine der beiden Handlungsalternativen mit Wahrscheinlichkeit $1/2$ gewählt. Wie schon bei NTMs gibt es bei PTMs ein „online“- und ein „offline“-Modell:

**online-Nichtdeterminismus** In jedem Schritt wählt $M$ nicht-deterministisch eine von (oBdA) zwei Handlungsalternativen aus, sagen wir „Aktion 0“ oder „Aktion 1“.

**online-Randomisierung** In jedem Schritt bestimmt $M$ ein perfektes Zufallsbit $b \in \{0,1\}$ (etwa durch den Wurf einer perfekten Münze) und wählt dann Aktion $b$.

**offline-Nichtdeterminismus** Für eine $T(n)$-zeitschrbeschränkte NTM $M$ nehmen wir die $T(n)$ nicht-deterministichen Entscheidungen vorweg, indem wir die Eingabe $x$ um einen (nicht-deterministisch gewählten) Ratestring $y \in \{0,1\}^{T(n)}$ ergänzen und $M$ deterministisch auf $\langle y, x\rangle$ rechnen lassen, wobei im $i$-ten Schritt die Aktion $y_i$ gewählt wird (Rate-Verifikationsschema).

**offline-Randomisierung** Wir stellen uns vor, die $T(n)$-zeitbeschränkte PTM $M$ hätte Zugriff auf einen (bezüglich der uniformen Verteilung) zufälligen String $y \in \{0,1\}^{T(n)}$. Auf $\langle y, x\rangle$ rechnet $M$ dann deterministisch, wobei (in Analogie zu NTMs) im $i$-ten Schritt die Aktion $y_i$ gewählt wird.

Wir wählen im Folgenden das Modell der offline-Randomisierung und setzen zudem oBdA voraus, dass eine $T(n)$-zeitbeschränkte PTM $M$ auf jeder Eingabe $\langle y, x\rangle$ mit $x \in \Sigma^n$ und $y \in \{0,1\}^{T(n)}$ exakt $T(n)$ Schritte rechnet. Da $y$ bezüglich der uniformen Verteilung auf $\{0,1\}^{T(n)}$ ausgewählt wurde, lassen sich die Wahrscheinlichkeiten zum Akzeptieren bzw. Verwerfen von $x$ durch „Zählen“ ermitteln (Anzahl der günstigen Fälle dividiert durch die Anzahl aller Fälle):

$$\begin{aligned} \Pr_y[M(y,x)=1] &= \frac{|\{y \in \{0,1\}^{T(n)} | M(y,x)=1\}|}{2^{T(n)}} \\ \Pr_y[M(y,x)=0] &= \frac{|\{y \in \{0,1\}^{T(n)} | M(y,x)=0\}|}{2^{T(n)}} \end{aligned}$$

Wir merken kurz an, dass alle auftretenden Wahrscheinlichkeiten Vielfache von $2^{-T(n)}$ sind.

Beim Erkennen einer Sprache $L$ kann $M$ folgende Fehler begehen:

**Fehler 1. Art** $M$ akzeptiert $x$ obwohl $x \notin L$.

**Fehler 2. Art** $M$ verwirft $x$, obwohl $x \in L$.

Die diversen probabilistischen Komplexitätsklassen unterscheiden sich darin, welche Fehlerwahrscheinlichkeiten toleriert werden. Wir führen nun eine allgemeine generische Notation ein, aus welcher sich die gängigen Klassen leicht ableiten lassen:

**Definition 11.1** *Seien $0 \leq \alpha < \beta \leq 1$, $\alpha, \beta \in \mathbb{Q}$ und $L \subseteq \{0,1\}^*$. Wir sagen $L$ gehört zur Klasse $R_{\leq\alpha,\geq\beta}$ gdw eine PTM $M$ mit polynomieller Zeitschranke $T(n)$ existiert, so dass für alle $n \geq 0$ und alle $x \in \{0,1\}^n$ folgende Bedingungen gelten:*

$$\begin{aligned} x \notin L &\implies \Pr_y[M(y,x) = 1] \leq \alpha \\ x \in L &\implies \Pr_y[M(y,x) = 1] \geq \beta \end{aligned}$$

*Die Klassen $R_{<\alpha,\geq\beta}$, $R_{\leq\alpha,>\beta}$ und $R_{<\alpha,>\beta}$ sind analog definiert. Für diese Klassen ist auch der Fall $\alpha = \beta$ (mehr oder weniger) sinnvoll.*

Im Folgenden nennen wir eine PTM mit einer polynomiellen Zeitschranke kurz eine PPTM.

Dass sich die uns vertrauten Klassen $P$ und $NP$ als Grenzfälle von probabilistischen Komplexitätsklassen darstellen lassen, lehrt folgendes

**Beispiel 11.2** *$P = R_{\leq 0,\geq 1}$ und $NP = R_{\leq 0,>0}$.*

Weiterhin merken wir kurz an:

**Lemma 11.3** *Alle von Definition 11.1 induzierten probabilistischen Komplexitätsklassen sind Teilklassen von PSpace.*

**Beweis** Eine Sprache $L \in R_{\leq\alpha,\geq\beta}$ mit einer PPTM $M$ (polynomielle Zeitschranke $T$) als Akzeptor kann deterministisch mit polynomiell beschränktem Platzverbrauch erkannt werden wie folgt:

1. Gegeben Eingabe $x \in \{0,1\}^n$, starte $M$ auf $\langle y, x\rangle$ für jedes $y \in \{0,1\}^{T(n)}$, benutze aber jeweils das gleiche Bandsegment.
2. Zähle nebenbei die anzahl $N_+$ der akzeptierenden Rechnungen.
3. Akzeptiere schließlich $x$ gdw $N_+ \geq \beta 2^{T(n)}$.

Für die Klassen $R_{<\alpha,\geq\beta}$, $R_{\leq\alpha,>\beta}$ und $R_{<\alpha,>\beta}$ ist die Argumentation völlig analog. □

In den folgenden Abschnitten diskutieren wir einige Standardbeispiele für probabilistische Komplexitätsklassen:

$$\begin{aligned} PP &= R_{<1/2,>1/2} \\ BPP &= R_{\leq 1/3,\geq 2/3} \\ RP &= R_{\leq 0,\geq 1/2} \\ ZPP &= RP \cap \text{co-}RP \end{aligned}$$

Heutzutage gilt $BPP$ (mehr noch als die Klasse $P$) als die Klasse der „praktisch lösbaren“ Probleme (unter Einsatz von Randomisierung). Algorithmen, die die Mitgliedschaft eines Problems in der Klasse $RP$ (bzw. $ZPP$) bezeugen, sind auch unter dem Namen „Monte-Carlo Algorithmen“ (bzw. „Las-Vegas Algorithmen“) bekannt.

## 11.1 Die Klasse PP

Es ist nicht sinnvoll, für den Fehler 1. und 2. Art jeweils eine Fehlerwahrscheinlichkeit von $1/2$ zuzulassen: dann könnten wir nämlich die Frage

$$x \in L \ ?$$

mit dem Wurf einer perfekten Münze entscheiden.[81] Es ist daher eine Art Minimalforderung, dass die Fehlerwahrscheinlichkeit für den Fehler 1. und 2. Art zumindest kleiner als $1/2$ sein sollte. Dies führt zur Definition der Klasse $PP$ (Probabilistic Polynomial Time):

$$PP = R_{<1/2,>1/2}$$

Die Minimalforderungen, die wir an die Klasse $PP$ richten, sind allerdings zu schwach. Der Hauptgrund hierfür ist, dass eine Maschine, die einen lediglich „exponentiell kleinen Vorsprung vor zufälligem Raten“ besitzt, in ihrem statistischen Ein/Ausgabeverhalten nicht effizient von zufälligem Raten unterschieden werden kann.[82] Ein weiteres Indiz dafür, dass $PP$ keine Klasse mit praktikablen Erkennungsalgorithmen ist, werden wir im Satz 11.5 liefern, welcher besagt, dass $NP$ eine Teilklasse von $PP$ ist.

Wir beweisen als kleine Aufwärmübung zunächst das folgende

**Lemma 11.4** $PP := R_{<1/2,>1/2} = R_{\leq 1/2,>1/2}$.

**Beweis** Die Inklusion $R_{<1/2,>1/2} =\subseteq R_{\leq 1/2,>1/2}$ ist trivial. Wir zeigen, dass auch die Inklusion

$$R_{\leq 1/2,>1/2} \subseteq R_{<1/2,>1/2}$$

gültig ist. Es sei $L \in R_{\leq 1/2,>1/2}$ und $M$ eine entsprechende PPTM mit Zeitschranke $T(n)$, wobei $T$ ein Polynom mit Koeffizienten aus $\mathbb{N}$ bezeichnet. Da die Wahrscheinlichkeit für das Ereignis „$M$ akzeptiert Eingabe $x \in \{0,1\}^n$“ ein Vielfaches von $2^{-T(n)}$ ist, folgern wir

$$\Pr_y[M(y,x)=1] > \frac{1}{2} \Leftrightarrow \Pr_y[M(y,x)=1] \geq \frac{1}{2} + 2^{-T(n)} \ . \tag{53}$$

Wir erweitern $M$ zu einer PPTM $M'$, die zusätzliche Zufallsbits $b_0, b_1, \ldots, b_{T(n)}$ verwendet und arbeitet wie folgt:

[81] Deshalb hatten wir die Klasse $R_{\leq\alpha,\geq\beta}$ nur im Falle $\alpha < \beta$ zugelassen. $R_{\leq 1/2,\geq 1/2}$ wäre die Klasse aller Sprachen!

[82] Wir verzichten an dieser Stelle auf eine Konkretisierung und einen mathematischen Beweis dieser Aussage. Im Prinzip läuft die Argmentation darauf hinaus, dass man exponentiell viele Experimente machen müsste, um eine infinitesimal unfaire Münze zuverlässig von einer fairen Münze zu unterscheiden.

1. Falls $b_0 = b_1 = \cdots = b_{T(n)} = 0$ (ein Ereignis der Wahrscheinlichkeit $2^{-(1+T(n))}$)) dann verwerfe $x$ und stoppe. Andernfalls mache weiter.

2. Starte $M$ auf $\langle y, x\rangle$ und entscheide wie $M$.

Wenn $M$ die Eingabe $x$ verwirft, so auch $M'$. Es gibt aber auch eine positive Wahrscheinlichkeit dafür, dass $M$ Eingabe $x$ akzeptiert, aber $M'$ sie verwirft, weil alle $b$-Bits Nullen sind. Folglich gilt für alle $x \notin L$:

$$\Pr_{yb}[M'(yb, x) = 1] < \Pr_{y}[M(y, x) = 1] \leq \frac{1}{2} \ .$$

Weiterhin gilt (wegen der Subadditivität von Wahrscheinlichkeitsmaßen) für alle $x \in L$:

$$\begin{aligned} \Pr_{yb}[M(yb, x) = 0] &\leq \Pr_{y}[M(y, x) = 0] + 2^{-(1+T(n))} \\ &\overset{(53)}{\leq} \frac{1}{2} - 2^{-T(n)} + 2^{-(1+T(n))} \\ &= \frac{1}{2} - 2^{-(1+T(n))} < \frac{1}{2} \end{aligned}$$

und somit

$$\Pr_{yb}[M(yb, x) = 1] > \frac{1}{2} \ .$$

PPTM $M'$ bezeugt, dass $L \in R_{<1/2,>1/2}$. □

Wir beschließen diesen Abschnitt mit dem folgenden

**Satz 11.5** *$NP \subseteq PP$.*

**Beweis** Wegen Lemma 11.4 genügt es

$$NP \subseteq R_{\leq 1/2,>1/2}$$

nachzuweisen. Es sei $L$ eine beliebige aber fest ausgewählte Sprache aus $NP = R_{\leq 0,>0}$ und $M$ eine entsprechende PPTM. Wir erweitern $M$ zu einer PPTM $M'$, die ein zusätzliches Zufallsbit $b$ verwendet und arbeitet wie folgt:

1. Falls $b = 1$ (ein Ereignis der Wahrscheinlichkeit $1/2$), dann akzeptiere und stoppe. Andernfalls mache weiter.

2. Starte $M$ auf $\langle y, x\rangle$ und entscheide wie $M$

Offensichtlich gilt für alle $x \notin L$ (welche von $M$ niemals akzeptiert werden):

$$\Pr_{yb}[M(yb, x) = 1] = \frac{1}{2}$$

Unter Verwendung des „Satzes der bedingten Wahrscheinlichleiten“ erhalten wir für alle $x \in L$ (die von $M$ mit einer positiven, obschon evtl. sehr kleinen, Wahrscheinlichkeit akzeptiert werden):

$$\begin{aligned} \Pr_{yb}[M(yb,x)=1] &= \frac{1}{2}\Pr_{yb}[M(yb,x)=1|b=1] + \frac{1}{2}\Pr_{yb}[M(yb,x)=1|b=0] \\ &= \frac{1}{2}\cdot 1 + \frac{1}{2}\underbrace{\Pr_{y}[M(y,x)=1]}_{>0} > \frac{1}{2} \end{aligned}$$

PPTM $M'$ bezeugt, dass $L \in R_{\leq 1/2,>1/2}$. □

## 11.2 Die Klasse BPP

Die Lehre aus dem Abschnitt 11.1 ist, dass die Wahrscheinlichkeiten für den Fehler 1. und 2. Art „deutlich“ kleiner als 1/2 sein sollten, damit die randomisierten Entscheidungen (Akzeptieren versus Verwerfen) sich signifikant von zufälligem Raten unterscheiden. Dies ist bei der Definition der Klasse BPP (Bounded-away from 1/2 Probabilistic Polynomial Time) berücksichtigt:

$$BPP = R_{\leq 1/3,\geq 2/3}$$

Um mit dieser technischen Definition vertraut zu werden, zeigen wir zunächst, dass die Auswahl der Konstanten $1/3, 2/3$ willkürlich ist und es nur darauf ankommt die Fehlerwahrscheinlichkeit „deutlich“ von 1/2 abzugrenzen. Anschließend zeigen wir, dass *BPP* auf dem 2. Level der polynomiellen Hierarchie (oder noch tiefer)[83] angesiedelt ist.

Um die Grenzen der Fehlerwahrscheinlichkeiten auszuloten, welche wir an die Stelle der Konstanten $1/3, 2/3$ setzen können, benötigen wir folgende natürliche Verallgemeinerung von Definition 11.1:

**Definition 11.6** *Es seien $\alpha = (\alpha_n)_{n\geq 0}$ und $\beta = (\beta_n)_{n\geq 0}$ zwei Folgen mit $0 \leq \alpha_n < \beta_n \leq 1$ und $\alpha_n, \beta_n \in \mathbb{Q}$ für alle $n \geq 0$. Wir sagen $L$ gehört zur Klasse $R_{\leq\alpha,\geq\beta}$ gdw eine PTM $M$ mit polynomieller Zeitschranke $T(n)$ (also eine PPTM) existiert, so dass für alle $n \geq 0$ und alle $x \in \{0,1\}^n$ folgende Bedingungen gelten:*

$$\begin{aligned} x \notin L &\implies \Pr_{y}[M(y,x)=1] \leq \alpha_n \\ x \in L &\implies \Pr_{y}[M(y,x)=1] \geq \beta_n \end{aligned}$$

*Die Klassen $R_{<\alpha,\geq\beta}$, $R_{\leq\alpha,>\beta}$ und $R_{<\alpha,>\beta}$ sind analog definiert.*

Wir zeichnen zwei Nullfolgen $\epsilon = (\epsilon_n)$ und $\delta = (\delta_n)$ aus:

$$\epsilon_n = 2^{-n^l} \text{ und } \delta_n = n^{-k} \tag{54}$$

[83] Sogar $BPP = P$ lässt sich beim derzeitigen Stand des Wissens nicht ausschließen.

Hierbei seien $k, l \in \mathbb{N}$ natürlich-zahlige Konstanten. Offensichtlich gilt

$$R_{\leq \epsilon, \geq 1-\epsilon} \subseteq \overbrace{R_{\leq 1/3, \geq 2/3}}^{=BPP} \subseteq R_{\leq 1/2-\delta, \geq 1/2+\delta} \ . \tag{55}$$

$\epsilon_n$ ist eine phantastisch kleine Fehlerrate, die mit exponentieller Geschwindigkeit gegen Null konvergiert. $1/2 - \delta_n$ hingegen kommt verdächtig nahe an die Fehlerrate $1/2$ von zufälligem Raten heran.[84] Obwohl bei oberflchlicher Betrachtung eine „Galaxie“ zwischen den Fehlerraten $\epsilon_n$ und $1/2 - \delta_n$ zu liegen scheint, werden wir jetzt nachweisen, dass alle in (55) aufgeführten Klassen zu *BPP* identisch sind. Hierzu genügt es freilich, die Inklusion

$$R_{\leq 1/2-\delta, \geq 1/2+\delta} \subseteq R_{\leq \epsilon, \geq 1-\epsilon}$$

nachzuweisen. In Worten: Eine PPTM $M$, die lediglich einen polynomiellen Vorteil über zufälliges Raten erzielt, lässt sich in eine PPTM $M'$ mit einer exponentiell kleinen Fehlerrate transformieren. Wir beweisen zu diesem Zweck die folgende (etwas allgemeinere) Aussage:

**Satz 11.7** *Es sei $\tau = (\tau_n)_{n \geq 0}$ eine in poly$(n)$ Schritten berechenbare Folge. Weiter sei $M$ eine PPTM, die für alle $n \geq 0$ und alle $x \in \{0,1\}^n$ die folgenden Bedingungen erfüllt:*

$$\begin{aligned} x \notin L \quad &\Longrightarrow \quad \Pr_y[M(y,x) = 1] \leq \tau_n - \delta_n \\ x \in L \quad &\Longrightarrow \quad \Pr_y[M(y,x) = 1] \geq \tau_n + \delta_n \end{aligned}$$

*Dann existiert eine PPTM $M'$ für $L$ mit Fehlerrate $\epsilon$ (was $L \in R_{\leq \epsilon, \geq 1-\epsilon}$ bezeugt).*

**Beweis** PPTM $M'$ gehe auf Eingabe $x \in \{0,1\}^n$ vor wie folgt:

1. Berechne $\tau_n$ aus $x$. (Hierfür spielt nur die Länge $n$ von $x$ eine Rolle.)
2. Für eine hinreichend große (Präzisierung erfolgt aus didaktischen Gründen später) natürliche Zahl $R$ lasse $M$ $R$-mal auf Eingabe $x$ laufen (mit jeweils neuen, unabhängigen Zufallsbits) und bestimme dabei die absolute Häufigkeit $R_+$, mit welcher $x$ von $M$ akzeptiert wird.
3. Akzeptiere $x$ gdw $R_+ \geq \tau_n R$.

Wir machen folgende Beobachtungen:

- $R_+$ ist eine binomial verteilte Zufallsvariable (Anzahl der Erfolge bei $R$ unabhängig durchgeführten Bernoulli-Experimenten).
- Falls $x \in L$ (und $M$ daher mit einer Wahrscheinlichkeit von mindestens $\tau_n + \delta_n$ akzeptiert), dann gilt

$$E[R_+] \geq (\tau_n + \delta_n) R \ .$$

[84] Da dieser Term jedoch noch um den Kehrwert eines Polynoms von $1/2$ weg-separiert ist, spricht man von „polynomiellen Vorteil über zufälliges Raten“).

- Falls $x \notin L$ (und $M$ daher mit einer Wahrscheinlichkeit von höchstens $\tau_n - \delta_n$ akzeptiert), dann gilt

$$E[R_+] \leq (\tau_n - \delta_n)R \ .$$

- $M'$ begeht einen Fehler höchstens dann, wenn die Zufallsvariable $R_+$ von ihrem Erwartungswert um mindestens $\delta_n R$ abweicht.

Mit Hilfe der (aus der Statistik bekannten)[85] Chernov-Schranken folgt, dass die Abweichung einer binomial-verteilten Zufallsvariable von ihrem Erwartungswert (bei hinreichend großer Anzahl von Versuchen) ein eher unwahrscheinliches Ereignis ist. Genauer:

$$\Pr[|R_+ - E[R_+]| \geq \delta_n R] \leq 2e^{-2\delta_n^2 R}$$

Eine kleine Rechnung ergibt

$$2e^{-2\delta_n^2 R} \Leftrightarrow R \geq \frac{\ln(2/\epsilon_n)}{2\delta_n^2} \ .$$

Wegen $\delta_n = n^{-k}$ und $\epsilon_n = 2^{-n^l}$ ergibt sich nun, dass

$$R := n^{2k+l+1}$$

hinreichend groß ist, um für $M'$ die Fehlerschranke $\epsilon_n$ zu garantieren. □

Der Spezialfall $\tau_n = 1/2$ liefert nun die

**Folgerung 11.8** *Die Klassen $R_{\leq\epsilon,\geq 1-\epsilon}$ und $R_{\leq 1/2-\delta,\geq 1/2+\delta}$ stimmen beide mit der Klasse $BPP = R_{\leq 1/3,\geq 2/3}$ überein.*

Die Technik, eine hohe Fehlerrate durch wiederholte Anwendung eines Zufallsexperimentes signifikant zu verkleinern, wird „Boosting“ genannt. Bei PPTMs mit beidseitigem Fehler haben wir dabei eine Art „Majoritätsvotum“ in Verbindung mit den „Chernov-Schranken“. Bei PPTMs mit einseitigem Fehler wird sich später ein einfacheres Boosting-Argument ergeben.

[85] s. auch Begleitmaterial zur Vorlesung

Wir versuchen im Folgenden, *BPP* in die polynomielle Hierarchie einzuordnen. Zu diesem Zweck definieren wir eine neue Komplexitätsklasse:

**Definition 11.9** *Wir sagen eine Sprache $L \subseteq \Sigma^*$ gehört zur Komplexitätsklasse $\Phi_2$ gdw eine Sprache $L_0 \in P$ existiert, so dass für alle $n \geq 0$ die folgenden Bedingungen gelten:*

$$x \in L \quad \Leftrightarrow \quad (\exists y)_{pol}(\forall z)_{pol} : \langle x, y, z\rangle \in L_0 \tag{56}$$

$$x \notin L \quad \Leftrightarrow \quad (\exists z)_{pol}(\forall y)_{pol} : \langle x, y, z\rangle \notin L_0 \tag{57}$$

Die Klasse $\Phi_2$ wird von $\Delta_2$ und $\Sigma_2 \cap \Pi_2$ „gesandwiched“:

**Lemma 11.10** $\Delta_2 \subseteq \Phi_2 \subseteq \Sigma_2 \cap \Pi_2$.

**Beweis** Wir beweisen zunächst $\Phi_2 \subseteq \Sigma_2 \cap \Pi_2$. Wähle eine beliebige Sprache $L$ aus $\Phi_2$ fest aus. Aus Bedingung (56) lässt sich unmittelbar $L \in \Sigma_2$ ablesen. Aus Bedingung (57) ergibt sich nun $L \in \Pi_2$ wie folgt:

$$L = \{x|\ \neg(x \notin L)\} = \{x|\ \neg((\exists z)_{pol}(\forall y)_{pol} : \langle x, y, z\rangle \notin L_0)\} = \{x|\ (\forall z)_{pol}(\exists y)_{pol} : \langle x, y, z\rangle \in L_0\}$$

Somit hat sich insgesamt $L \in \Sigma_2 \cap \Pi_2$ und daher $\Phi_2 \subseteq \Sigma_2 \cap \Pi_2$ ergeben.
Der Rest des Beweises ist der Inklusion $\Delta_2 \subseteq \Phi_2$ gewidmet. Wir wählen eine beliebige Sprache $L \in \Delta_2 = P[NP]$ fest aus. Dann gibt es eine Sprache $L' \in NP$ und eine polynomiell zeitbeschränkte DOTM $M[L']$, welche Eingaben aus $L$ erkennen kann. Wir wählen eine beliebige Eingabe $x \in \Sigma^n$ fest aus. Es seien

$$(g_1, b_1), \ldots, (g_s, b_s)$$

mit

$$b_i = \begin{cases} 0 & \text{falls } g_i \notin L' \\ 1 & \text{falls } g_i \in L' \end{cases}$$

die Fragen und Antworten in der Kommunikation zwischen $M[L']$ und ihrem $L'$-Orakel. Für alle $i$ mit $g_i \in L'$ bezeichne $h_i$ ein (in Polynomialzeit prüfbares) Zertifikat, das die Mitgliedschaft von $g_i$ in $L'$ bezeugt. Um $L \in \Phi_2$ nachzuweisen, suchen wir nach einer Darstellung von $L$, die den Bedingungen (56) und (57) genügt. Zu diesem Zweck definieren wir ein *Kommunikationscodewort* $W$ gemäß

$$W := \langle w_1, \ldots, w_s\rangle \ ,$$

wobei

$$w_i := \begin{cases} \langle g_i, 0\rangle & \text{falls } g_i \notin L' \\ \langle g_i, 1, h_i\rangle & \text{falls } g_i \in L' \end{cases} \ .$$

$W$ kodiert die (ggf. um Zertifikate erweiterte) Kommunikation zwischen $M[L']$ und ihrem Orakel. Wir nutzen im Folgenden aus, dass $W$ eine effiziente Simulation von $M[L']$ auf Eingabe $x$ erlaubt. Genauer: wir definieren eine polynomiell zeitbeschränkte DTM $M_0$ (die im Wesentlichen $M[L']$ simuliert) und zeigen, dass die von ihr induzierte Sprache $L_0$ den an eine Sprache $L \in \Phi_2$ gerichteten Bedingungen (56) und (57) genügt:

- Eine Eingabe von $M_0$ heiße *zulässig*, falls sie die Form $\langle x, y, z\rangle$ hat, wobei $y = W$ oder $z = W$ gelten soll.
- $M_0$ soll eine zulässige Eingabe $\langle x, y, z\rangle$ mit $y = W$ oder $z = W$ akzeptieren gdw $x \in L$.[86]

Man überlegt sich leicht, dass „syntaktisch inkorrekte" Eingaben, die nicht von der Form $\langle x, y, z\rangle$ sind (wobei mindestens einer der Strings $y, z$ von der Syntax her ein „potenzielles" Kommunikationscodewort sein muss), in Polynomialzeit entlarvt werden. Wir können daher annehmen, dass eine syntaktisch korrekte Eingabe der Form $\langle x, y, z\rangle$ (mit zumindest einem potenziellen Kommunikationscodewort) vorliegt. Ein offensichtliches Dilemma für $M_0$ besteht darin, dass sie nicht weiß, ob $y$ oder $z$ (oder evtl. keiner von beiden) das korrekte Kommunikationscodewort ist. Vergleichsweise harmlos ist dabei der Fall, dass einer der Strings $y, z$ (oder gar beide) eines der folgenden (in Polynomialzeit erkennbaren) „Fouls" begeht:

**Foul 1** Er weicht bereits syntaktisch von einem Kommunikationscodewort ab.

**Foul 2** Er hat die syntaktische Form eines Kommunikationscodewortes, enthält aber nicht exakt die Fragen, die $M[L']$ an ihr Orakel richtet.

Falls weder $y = W$ noch $z = W$ (insbesondere also, falls beide Strings ein Foul begehen), dann darf $M_0$ sowieso machen, was sie will. Nehmen wir also an, dass eine zulässige Eingabe mit $y = W$ oder $z = W$ vorliegt. Falls einer der Strings ein Foul begeht, dann ist (bei einer zulässigen Eingabe) der andere String das Kommunikationscodewort und $M$ kann die Simulation von $M[L']$ problemlos durchführen.

**Böse, böse** Was aber, wenn weder $y$ noch $z$ ein Foul begeht?

Man überlegt sich leicht, dass $M_0$ auch unter diesen (maximal widrigen) Umständen korrekt arbeiten kann, indem sie ein paar Regeln beherzigt:

- Falls $y, z$ zur Anfrage $g_i$ das gleiche Antwortbit enthalten, dann setze die Simulation mit diesem Antwortbit fort.
- Im Konfliktfall benutze das Zertifikat $h_i$, um

  $$g_i \in L' \ ?$$

  zu testen. Falls die Verfikation gelingt, dann setze die Simulation mit Antwortbit 1 fort. Falls nicht, dann mit Antwortbit 0.

Es hat sich also gezeigt, dass eine DTM $M_0$ mit den gewünschten Eigenschaften existiert. Es bezeichne $L_0$ die zugehörige Sprache. Wir wollen nun argumentieren, dass $L, L_0$ in der Beziehung (56) zueinander stehen:

- Die Richtung „$\Rightarrow$" ergibt sich mit $y = W$.

[86] Bei unzulässigen Eingaben machen wir $M_0$ keine Vorschriften.

- Die umgekehrte Richtung kann indirekt bewiesen werden. Für $x \notin L$ kann kein $y$ existieren, so dass für alle $z$ die Bedingung $\langle x, y, z\rangle \in L_0$ erfüllt ist. Zumindest für $z = W$ würde nämlich $M_0$ die Eingabe $\langle x, y, z\rangle$ verwerfen.

Schließlich argumentieren wir, dass $L, L_0$ auch in der Beziehung (57) zueinander stehen:

- Die Richtung „$\Rightarrow$“ ergibt sich mit $z = W$.
- Die umgekehrte Richtung kann indirekt bewiesen werden. Für $x \in L$ kann kein $z$ existieren, so dass für alle $y$ die Bedingung $\langle x, y, z\rangle \notin L_0$ erfüllt ist. Zumindest für $y = W$ würde nämlich $M_0$ die Eingabe $\langle x, y, z\rangle$ akzeptieren.

Somit hat sich $L \in \Phi_2$ und daher auch $\Delta_2 \subseteq \Phi_2$ ergeben. □

Zum Beweis des Hauptresultates benötigen wir noch das folgende

**Lemma 11.11** *Es sei $A \subseteq \{0,1\}^N$ mit $|A| \geq \frac{2}{3}2^N$ und $k = 18N$. Dann existieren $y_1, \ldots, y_k \in \{0,1\}^N$, so dass für alle $z \in \{0,1\}^N$ die Bedingung*

$$|\{i \in \{1, \ldots, k\} : \ y_i \oplus z \in A\}| > \frac{k}{2} \tag{58}$$

*erfüllt ist (wobei „$\oplus$“ die komponentenweise Addition modulo 2 bezeichnet).*

**Beweis** Wir verwenden die probabilistische Methode, d.h., wir zeigen, dass es eine echt positive Wahrscheinlichkeit gibt, eine „passende“ Sequenz $\bar{y} := (y_1, \ldots, y_k) \in \{0,1\}^{kN}$ zufällig zu generieren. Wir nennen die Sequenz $\bar{y}$ „gut“ für $z \in \{0,1\}^N$, falls $\bar{y}$ und $z$ die Bedingung (58) erfüllen. Ansonsten heiße $\bar{y}$ „schlecht“ für $z \in \{0,1\}^N$. In dieser Sprechweise ist zu zeigen, dass eine Sequenz $\bar{y} \in \{0,1\}^{kN}$ existiert, die simultan für alle $z \in \{0,1\}^N$ gut ist. Betrachte eine Sequenz $\bar{y}$ die (bezüglich der uniformen Verteilung) zufällig aus $\{0,1\}^{kN}$ gewählt ist. Für ein beliebig aber fest ausgewähltes $x \in \{0,1\}^N$ sei $Z_i$ die Bernoulli-Variable

$$Z_i := \begin{cases} 1 & \text{falls } y_i \oplus z \in A \\ 0 & \text{falls } y_i \oplus z \notin A \end{cases} .$$

Dann sind $Z_1, \ldots, Z_k$ identisch verteilte unabhängige Bernoulli-Variable mit Erfolgswahrscheinlichkeit $p \geq 2/3$. Damit $\bar{y}$ schlecht für $z$ ist, müsste $Z_1 + \cdots + Z_k$ um mindestens $k/6$ vom Erwartungswert $pk \geq 2k/3$ abweichen. Mit den (uns inzwischen bekannten) Chernov-Schranken (und mit $k = 18N$) ergibt sich, dass die Wahrscheinlichkeit hierfür durch $e^{-2(1/6)^2 k} = e^{-N}$ nach oben beschränkt ist. Mit der Subadditivität von Wahrscheinlichkeitsmaßen folgern wir weiter: die Wahrscheinlichkeit, eine zufällige Sequenz $\bar{y}$ zu generieren, so dass ein $z \in \{0,1\}^N$ existiert, für welche $\bar{y}$ schlecht ist, ist nach oben durch $2^N e^{-N} < 1$ beschränkt. Somit gibt es eine positive Wahrscheinlichkeit, eine Sequenz $\bar{y}$ zu realisieren, die für alle $z \in \{0,1\}^N$ gut ist. □

Nun zum Finale des laufenden Kapitels:

**Satz 11.12** *BPP $\subseteq \Phi_2$.*

**Beweis** Wähle eine beliebige Sprache $L \in BPP$ fest aus. Es sei $M$ eine PPTM für $L$ mit einer polynomiellen Zeitschranke $T(n)$ und einer durch $1/3$ beschränkten Fehlerrate (für die Fehler jeweils beider Arten).

**Ziel** Darstellung von $L$ gemäß der Definition von $\Phi_2$.

Wähle ein $n \geq 0$ und eine Eingabe $x \in \Sigma^n$ beliebig aber fest aus. Setze $N := T(n)$ und $k := 18N$. Es bezechne $\chi_L$ die charakteristische Funktion für die Sprache $L$. Die Menge

$$A := \{y \in \{0,1\}^N |\ M(y,x) = \chi_L(x)\}$$

hat eine Mächtigkeit $|A| \geq \frac{2}{3}2^N$, da $|A|$ die Anzahl der korrekten Rechnungen der PPTM $M$ (mit einer durch $1/3$ beschränkten Fehlerwahrscheinlichkeit) auf Eingabe $x$ ist. Es bezeichne $L_0$ die Sprache bestehend aus allen $\langle x, y, z\rangle$ mit folgenden Eigenschaften:

1. $y$ hat die Form $\langle y_1, \ldots, y_k\rangle$ mit $y_i \in \{0,1\}^N$ für $i = 1, \ldots, k$.
2. $z$ hat die Form $\langle z_1, \ldots, z_k\rangle$ mit $z_i \in \{0,1\}^N$ für $i = 1, \ldots, k$.
3. Eine Mehrheit der $k^2$ Rechnungen $M(y_i \oplus z_j, x)$ ist akzeptierend.

Da $A$ die Bedingungen von Lemma 11.11 erfüllt, folgt leicht, dass $L$ und $L_0$ in den Beziehungen (56) und (57) zueinander stehen. Somit gilt $L \in \Phi_2$ und daher auch $BPP \subseteq \Phi_2$.

□

## 11.3 Die Klasse RP

Die Komplexitätsklassen $PP$ und $BPP$ lassen prinzipiell einen beidseitigen Fehler zu. In diesem Abschnitt lernen wir eine Klasse kennen, bei welcher lediglich ein einseitiger Fehler zugelassen ist. Die Klasse $RP$ (Random Polynomial Time) ist gegeben durch

$$RP = R_{\leq 0, \geq 1/2}\ .$$

Eingaben $x \notin L$ werden also stets verworfen und Eingaben $x \in L$ werden mit einer Wahrscheinlichkeit von mindestens $1/2$ akzeptiert. Eine PPTM, die diesem Kriterium genügt, ist also auf Eingaben $x \notin L$ fehlerfrei. Wir merken kurz an, dass $RP$ wegen

$$RP = R_{\leq 0, \geq 1/2} \subseteq R_{\leq 0, > 0} = NP$$

in $NP$ enthalten ist.

Ähnlich wie im Falle $BPP$ werden wir zeigen, dass die Wahl der Konstante $1/2$ willkürlich ist:

**Lemma 11.13** *Für die in (54) definierten Nullfolgen $\epsilon = (\epsilon_n)_{n\geq 0}$ und $\delta = (\delta_n)_{n\geq 0}$ gilt:*

$$R_{\leq 0, \geq 1-\epsilon} = \overbrace{R_{\leq 0, \geq 1/2}}^{=:RP} = R_{\leq 0, \geq \delta}\ .$$

**Beweis** Wegen der offensichtlichen Inklusion

$$R_{\leq 0,\geq 1-\epsilon} \subseteq R_{\leq 0,\geq 1/2} \subseteq R_{\leq 0,\geq \delta}$$

genügt es,

$$R_{\leq 0,\geq \delta} \subseteq R_{\leq 0,\geq 1-\epsilon}$$

zu beweisen. In Worten: jede PPTM mit einseitigem Fehler, aber einer Fehlerrate von $1-\delta$ auf Eingaben aus $L$, kann in eine PPTM mit einseitigem Fehler transformiert werden, die auch auf Eingaben in $L$ eine exponentiell kleine Fehlerrate aufweist. Sei nun eine Sprache $L \in R_{\leq 0,\geq \delta}$ mit einer hierzu passenden PPTM $M$ vorgegeben, so dass für alle $n \geq 0$ und alle $x \in \Sigma^n$ gilt:

$$\begin{aligned} x \notin L &\Rightarrow \Pr_y[M(y,x) = 1] = 0 \\ x \in L &\Rightarrow \Pr_y[M(y,x)] = 1] \geq \delta_n \end{aligned}$$

Dann sei $M'$ die PPTM, die auf Eingabe $x \in \Sigma^n$ arbeitet wie folgt:

1. Wende $M$ $R$-mal (jeweils mit neuen unabhängigen Zufallsbits) auf Eingabe $x$ an (wobei wir $R$ aus didaktischen Gründen erst später festlegen).

2. Falls $x$ in mindestens einer der $R$ Rechnungen akzeptiert wurde, so stoppe akzeptierend. Andernfalls stoppe verwerfend.

Falls $x \notin L$, dann wird $x$ von $M$ $R$-mal verworfen. Somit stoppt auch $M'$ schließlich verwerfend. Falls $x \in L$, dann macht $M'$ einen Fehler (verwerfendes Stoppen) gdw alle $R$ Rechnungen von $M$ auf Eingabe $x$ verwerfend sind. Da eine einzelne Rechnung von $M$ auf $x$ mit einer Wahrscheinlichkeit von höchstens $1-\delta_n$ verwerfend ist, ergibt sich für die Wahrscheinlichkeit, dass alle $R$ (unabhängig voneinander ausgeführten) Rechnungen von $M$ auf Eingabe $x$ verwerfend sind (gemäß der Produktformel für die Konjunktion unabhängiger Ereignisse) die obere Schranke

$$(1-\delta_n)^R < e^{-\delta_n R} \ .$$

(Hierbei wurde die Formel $1+a \leq e^a$ mit Gleichheit nur für $a=0$ benutzt, die für alle $a \in \mathbb{R}$ gültig ist.) Eine elementare Rechnung ergibt

$$e^{-\delta_n R} \leq \epsilon_n \Leftrightarrow R \geq \frac{\ln(1/\epsilon_n)}{\delta_n} \ .$$

Wegen $\epsilon_n \geq 2^{-n^l}$ und $\delta_n \geq n^{-k}$ ist $R := n^{k+l}$ eine hinreichend große Wahl von Parameter $R$. Unsere Diskussion hat ergeben, dass $M'$ eine PPTM ist, welche $L \in R_{\leq 0,\geq 1-\epsilon}$ bezeugt (was zu beweisen war). □

**Folgerung 11.14** *Für jede Konstante $0 < c < 1$ gilt: $RP = R_{\leq 0,\geq c}$.*

## 11.4 Die Klasse ZPP

Die bisher betrachteten PPTMs lassen für den menschlichen Benutzer einen Rest Unsicherheit übrig. Selbst bei PPTMs mit einseitigem Fehler können wir nicht sicher sein (obschon die Fehlerwahrscheinlichkeit via „Boosting“ vernachlässigbar klein gemacht werden kann), dass eine verworfene Eingabe nicht vielleicht doch zur Sprache gehört und hätte akzeptiert werden sollen. Eine Fehlerwahrscheinlichkeit von, sagen wir, $2^{-1000}$ ist sicher nicht größer als die Wahrscheinlichekeit, dass alle Kernkraftwerke dieser Erde gleichzeitig ihren Supergau erleben. Allein: eine auch noch so kleine Fehlerwahrscheinlichkeit bleibt ein „Stachel im Fleisch“ des wahren Perfektionisten! Voilá, hier ist nun die Definition der PPTM, die Balsam auf die Wunden aller Puristen und Perfektionisten träufelt:

**Definition 11.15** *Eine* fehlerfreie *PPTM $M$, ist eine PPTM, deren Ausgaben* 0 *(für verwerfende Rechnungen) und* 1 *(für akzeptierende Rechnungen) stets absolut verlässlich sind. Sie verfügt über eine dritte Ausgabe „1/2“ (für „unentschieden“), die aber auf jeder Eingabe mit einer Wahrscheinlichkeit vonb höchstens* $1/2$ *produziert wird.*

**Definition 11.16** *Die Klasse ZPP (Zerro Error Probabilistic Polynomial Time) besteht aus allen Sprachen, die eine fehlerfreie PPTM zum Akzeptor haben.*

Wie im Falle von *RP* ist die Wahl der Konstanten $1/2$ in der Definition von fehlerfreien PPTMs willkürlich. Sie kann durch jede Konstante $0 < c < 1$ (oder auch durch $1-\epsilon$ bzw. $\delta$) ersetzt werden, ohne dass die davon induzierte Klasse *ZPP* sich ändert.

Wenn wir bei einer fehlerfreien PPTM für die Sprache $L$ Ausgabe 0 und Ausgabe 1 vertauschen, erhalten wir eine fehlerfreie PPTM für die Komplementärsprache $\bar{L}$. Somit gilt das

**Lemma 11.17** *$ZPP = co\text{-}ZPP$.*

Folgendes Resultat klärt das Verhältnis von *RP* und *ZPP*:

**Satz 11.18** *$ZPP = RP \cap co\text{-}RP$.*

**Beweis** Wir weisen zunächst

$$ZPP \subseteq RP$$

nach. Sei $L \in ZPP$ und $M$ eine dazu passende fehlerfreie PPTM. Die PPTM $M'$ arbeite wie $M$, außer dass statt $1/2$ stets 0 ausgegeben werde. Es folgt:

$$\begin{aligned} x \notin L &\Rightarrow \Pr_y[M'(y,x)=1] = \Pr_y[M(y,x)=1] = 0 \\ x \in L &\Rightarrow \Pr_y[M'(y,x)=1] = \Pr[M(y,x)=1] \geq \frac{1}{2} \end{aligned}$$

$M'$ bezeugt, dass $L \in RP$. Somit gilt $ZPP \subseteq RP$.
Durch Dualisierung erhalten wir

$$ZPP = \text{co-}ZPP \subseteq \text{co-}RP,$$

womit dann auch

$$ZPP \subseteq RP \cap \text{co-}RP$$

nachgewiesen wäre.
Bleibt also zu zeigen, dass

$$RP \cap \text{co-}RP \subseteq ZPP \ .$$

Sei $L \in RP \cap \text{co-}RP$. Es sei weiter $M$ die PPTM, welche $L \in RP$ bezeugt und entsprechend $\bar{M}$ die PPTM, welche $L \in \text{co-}RP$ und somit $\bar{L} \in RP$ bezeugt. Wir betrachten die PPTM $M'$, die arbeitet wie folgt:

1. Wende $M$ und $\bar{M}$ auf Eingabe $x$ an.
2. Falls $M$ akzeptiert, gib 1 aus, falls $\bar{M}$ akzeptiert gib 0 aus, und falls weder $M$ noch $\bar{M}$ akzeptiert, gib 1/2 aus.

Wegen

$$\begin{aligned} x \notin L \quad &\Longrightarrow \quad \left( \Pr_y[M(y,x)=1] = 0 \text{ und } \Pr_y[\bar{M}(y,x)=1] \geq \frac{1}{2} \right) \\ x \in L \quad &\Longrightarrow \quad \left( \Pr_y[\bar{M}(y,x)=1] = 0 \text{ und } \Pr_y[M(y,x)=1] \geq \frac{1}{2} \right) \end{aligned}$$

entscheidet sich $M'$ immer korrekt und ist mit einer Wahrscheinlichkeit von maximal $1/2$ unentschieden. $M'$ bezeugt, dass $L \in ZPP$, womit auch $RP \cap \text{co-}RP \subseteq ZPP$ bewiesen wäre. □

## 11.5 Abschluss unter Komplement

Wir hatten bereits angemerkt, dass $ZPP = \text{co-}ZPP$. In diesem Abschnitt gehen wir der Frage des Abschlusses unter Komplement etwas systematischer nach.

Es sei $M$ eine PPTM, die $L \in R_{\leq\alpha,\geq\beta}$ bezeugt und $\bar{M}$ die PPTM, die aus $M$ durch Vertauschen der Ausgaben 0 und 1 hervorgeht. Hieruas ergibt sich

$$x \in \bar{L} \Rightarrow \Pr_y[M(y,x)=1] \leq \alpha \Rightarrow \Pr_y[M(y,x)=0] \geq 1-\alpha \Rightarrow \Pr_y[\bar{M}(y,x)=1] \geq 1-\alpha$$

und

$$x \notin \bar{L} \Rightarrow x \in L \Rightarrow \Pr_y[M(y,x)=1] \geq \beta \Rightarrow \Pr_y[M(y,x)=0] \leq 1-\beta \Rightarrow \Pr_y[\bar{M}(y,x)=1] \leq 1-\beta \ .$$

Offensichtlich bezeugt $\bar{M}$, dass $L \in R_{\leq 1-\beta,\geq 1-\alpha}$. Aus diesen Überlegungen lassen sich folgende Schlüsse ziehen:

**Folgerung 11.19** *Es gilt:*

$$\begin{aligned} L \in R_{\leq\alpha,\geq\beta} &\Rightarrow \bar{L} \in R_{\leq 1-\beta,\geq 1-\alpha} \\ L \in R_{\leq\alpha,\geq 1-\alpha} &\Rightarrow \bar{L} \in R_{\leq\alpha,\geq 1-\alpha} \end{aligned}$$

*Analoge Aussagen gelten für die Klassen $R_{<\alpha,\geq\beta}$, $R_{\leq\alpha,>\beta}$ und $R_{<\alpha,>\beta}$. Insbesondere gilt*

$$PP = \textit{co-PP} \text{ und } BPP = \textit{co-BPP} \ .$$

Unter den von uns näher diskutierten Klassen ist $RP$ die einzige „asymmetrisch definierte" Klasse, die vermutlich nicht unter Komplement abgeschlossen ist.

## 11.6 Die Landschaft der Komplexitätsklassen

Wir wollen ein Bild entwerfen, das die probabilistischen und deterministischen Komplexitätsklassen (sagen wir zwischen $\mathcal{L}$ und *PSpace*) und ihre Querbeziehungen darstellt. Aus der trivialen (und auch schon mehrfach ausgenutzten) Beziehung

$$0 \leq \alpha \leq \alpha' < \beta' \leq \beta \leq 1 \Rightarrow R_{\leq\alpha,\geq\beta} \subseteq R_{\leq\alpha',\geq\beta'}$$

(und den analogen Beziehungen für die Klassen $R_{<\alpha,\geq\beta}$, $R_{\leq\alpha,>\beta}$, $R_{<\alpha,>\beta}$) und aus

$$P = R_{\leq 0,\geq 1},\ RP = R_{\leq 0,\geq 1/2},\ BPP = R_{\leq 1/3,\geq 2/3},\ PP = R_{<1/2,>1/2},\ NP = R_{\leq 0,>0}$$

lässt sich sofort

$$P \subseteq RP \subseteq BPP \subseteq PP \text{ und } RP \subseteq NP$$

ableiten. Wegen

$$ZPP = RP \cap \text{co-}RP,\ P = \text{co-}P,\ BPP = \text{co-}BPP$$

folgt weiterhin

$$P \subseteq ZPP \subseteq RP \subseteq RP \cup \text{co-}RP \subseteq BPP \text{ und } ZPP \subseteq NP \cap \text{co-}NP \ .$$

Da alle probabilistischen Klassen, die unserer generischen Definition genügen, in *PSpace* liegen, gilt insbesondere

$$PP \subseteq PSpace \ .$$

Schließlich sei an die Inklusionen

$$\Delta_2 \subseteq \Phi_2 \subseteq \Sigma_2 \cap \Pi_2 \text{ und } BPP \subseteq \Phi_2$$

erinnert. Wenn wir diese Puzzlesteine zusammensetzen, erhalten wir die als Begleitmaterial ausgeteilte „Landschaft der Komplexitätsklasen".

Abschließend sei ein Resultat von Seinosuke Toda erwähnt, das ihm nachträglich im Jahre 1998 den renommierten Gödelpreis einbrachte:

**Satz 11.20 (Toda, 1991)** *Jede Sprache aus PH ist Turing-reduzierbar auf eine Sprache aus PP:*

$$PH \subseteq P[PP] \ .$$

Den komlexen Beweis (der u.a. die Technik der „Arithmetisierung von Booleschen Formeln“ verwendet) lassen wir aus, erwähnen aber die

**Folgerung 11.21** *Falls PH nicht auf einen endlichen Level kollabiert, dann gilt* $PP \not\subseteq PH$.

**Beweis** Wir führen den Beweis indirekt. Es ist bekannt, dass $PP$ ein unter polynomiellen Reduktionen vollständiges Problem $L_*$ enthält. Somit erhielten wir unter der Voraussetzung die $PP \subseteq PH$ die Implikationskette

$$PP \subseteq PH \Rightarrow L_* \in PH \Rightarrow \exists k : L_* \in \Sigma_k \Rightarrow \exists k : PP \subseteq \Sigma_k \Rightarrow \exists k : PH \subseteq P[PP] \subseteq P[\Sigma_k] = \Delta_{k+1} \ .$$

Die Inklusion $PP \subseteq PH$ hätte also den Kollaps von $PH$ auf einen endlichen Level zur Folge. □