Strukturelle Komplexität von Anzahlproblemen

Von der Fakultät Informatik
der Universität Stuttgart
zur Erlangung der Würde eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)
genehmigte Abhandlung

vorgelegt von
Johannes Köbler
aus Ellwangen/Jagst

Hauptberichter: Prof. Dr. Uwe Schöning
Mitberichter: Prof. Dr. Walter Knödel
Tag der mündlichen Prüfung: 12. Mai 1989

Institut für Informatik der Universität Stuttgart
1989

Herrn Prof. Dr. Uwe Schöning möchte ich ganz besonders für seine stete Unterstützung und engagierte Betreuung danken.

Zu Dank verbunden bin ich auch Herrn Prof. Dr. Walter Knödel für die Übernahme des Mitberichts und das Interesse an der vorliegenden Arbeit.

Inhaltsverzeichnis

Einleitung

Die Komplexität von Problemen, die mithilfe eines Computers lösbar sind, kann nach verschiedenen Gesichtspunkten gemessen werden. Die wohl bekanntesten sind

- der Verbrauch an Rechenzeit
- der Bedarf an Speicherplatz

durch deterministische Algorithmen. Dabei wird der Aufwand noch als bewältigbar angesehen, wenn er nicht schneller als ein Polynom mit der Größe der Problemeingabe wächst. Entscheidungs- oder ja-nein-Probleme, die mit polynomiellem Zeitaufwand gelöst werden können, werden daher in der Zeitkomplexitätsklasse P und die mit polynomiellem Platzbedarf lösbaren in der Platzkomplexitätsklasse PSPACE zusammengefaßt. Für viele in der Praxis auftretende Probleme ist die Bestimmung ihrer deterministischen Komplexität eine bis heute ungelöste Aufgabe.

Für einen Teil dieser Probleme scheinen allgemeinere Berechnungsverfahren wie nichtdeterministische oder probabilistische Algorithmen besser zugeschnitten zu sein. So faßt man die in Polynomzeit durch nichtdeterministische bzw. probabilistische Algorithmen lösbaren Probleme in der Klasse NP bzw. PP zusammen.

Die Komplexität vieler praktisch relevanter Probleme läßt sich nun auf dieser erweiterten Grundlage exakt einordnen. Es stellt sich nämlich heraus, daß sie genau soviel an Komplexität beinhalten, wie mit nichtdeterministischen bzw. probabilistischen Algorithmen in vertretbarem Zeitaufwand zu bewältigen ist (sog. vollständige Probleme für NP bzw. PP).

Die Frage nach der Komplexität spezieller Praxisprobleme ist damit gleichbedeutend mit der Frage nach den strukturellen Beziehungen unter den genannten Komplexitätsklassen. Zum Beispiel sind bereits alle NP-Probleme in Polynomzeit lösbar (d.h. P = NP), wenn auch nur ein NP-vollständiges Problem in P liegt. Die Tatsache, daß für keines von über tausend aus den verschiedensten Bereichen der Mathematik stammenden NP-vollständigen Problemen ein effizienter Algorithmus gefunden wurde, wird als starkes Indiz für P $\neq$ NP gewertet.

Bei der Berechnungskomplexität kombinatorischer Anzahl- und Optimierungsprobleme lassen sich ähnliche Querverbindungen beobachten. So erhält man interessante Komplexitätsklassen, wenn man nichtdeterministische oder probabilistische Berechnungen zuläßt. Ein Beispiel ist Valiant's [Val79a] Klasse #P aller Zählfunktionen f, zu denen es einen NP-Algorithmus gibt, der bei Eingabe x genau $f(x)$ akzeptierende Rechnungen besitzt. Bei den meisten NP-Algorithmen korrespondiert eine akzeptierende Rechnung mit einer gefundenen „Lösung“ für das zugrundeliegende Problem. f bestimmt also die Anzahl der Lösungen für eine Problemeingabe. Für viele NP-vollständige Probleme ist das zugehörige Anzahlproblem #P-vollständig. Ein Beispiel ist die Bestimmung der Anzahl von Hamiltonlinien in einem gegebenen Graphen. Es gibt aber auch eine ganze Reihe von #P-vollständigen Problemen, deren ja-nein-Version, d.h. das Problem, ob eine Lösung existiert, in P liegt. Zum Beispiel ist das Problem, die Anzahl vollständiger Paarungen (perfect matchings) in bipartiten Graphen zu bestimmen, #P-vollständig [Val79a], obwohl die Frage nach der Existenz in Polynomzeit entschieden werden kann [HK73]. Allgemein liegt das Problem, die Anzahl aller Lösungen für eine Problemeingabe x zu bestimmen in #P, wenn in Polynomzeit die Richtigkeit einer vorgeschlagenen Lösung überprüft werden kann.

Nach Einführung der benötigten komplexitätstheoretischen Grundbegriffe und Rekapitulierung der für das Folgende wichtigen Resultate untersuchen wir im 1. Kapitel die Komplexität von kombinatorischen

Anzahl- und Optimierungsproblemen. Im Mittelpunkt steht dabei die Klasse span-P, die alle Funktionen enthält, die wie #P-Funktionen für eine Problemeingabe x die Anzahl verschiedener Lösungen berechnen. In Verallgemeinerung von #P muß dabei das Problem, ob eine vorgeschlagene Lösung richtig ist, nicht in P, sondern in NP liegen. Wir untersuchen die von span-P, Valiant's Klassen #P und #NP, sowie Krentel's Klasse opt-P von Optimierungsproblemen dargestellte Inklusionsstruktur. Für die Fälle, in denen die Inklusionsbeziehung nicht geklärt werden kann, geben wir äquivalente offene Probleme der Komplexitätstheorie an. Als Hauptresultat erhalten wir dabei die Äquivalenz von span-P $\subseteq$ #P und NP = UP (zur Definition von UP siehe unten), sowie #NP $\subseteq$ span-P $\Leftrightarrow$ NP = co-NP, d.h. NP ist genau dann unter Komplementbildung abgeschlossen, wenn jedes unter einem NP-Orakel in #P lösbare Problem ohne Orakelbefragungen in span-P berechnet werden kann. Anschließend untersuchen wir die Struktur von span-P unter verschiedenen Reduktionen und geben vollständige Probleme für span-P und für Teilklassen von span-P an.

Im zweiten Kapitel beschäftigen wir uns mit der Frage nach der Komplexität von Orakeln, unter deren Befragung span-P-Funktionen in Polynomzeit durch deterministische und probabilistische Algorithmen approximiert werden können. Stockmeyer [Sto85] und Jerrum, Valiant und Vazirani [JVV86] haben unter Anwendung von Sipser's Hashing-Technik [Sip83] Approximationsalgorithmen für #P-Funktionen entworfen, die deterministisch unter einem Σ_2^p-Orakel und probabilistisch sogar unter einem NP-Orakel laufen. Wir erweitern diese Resultate auf span-P-Funktionen und zeigen, daß zur Berechnung eines Näherungswertes logarithmisch viele Orakelfragen ausreichen. Letzteres wurde für die deterministische Approximation von #P-Funktionen auch in [Pio88] festgestellt. Des weiteren konstruieren wir die Approximationsalgorithmen so, daß sie nur einseitige Fehler machen, d.h. sie geben niemals einen zu kleinen Wert aus. Ein Teil der im ersten und zweiten Kapitel dargestellten Resultate sind in [KST89] erschienen (vgl. auch [KST88]).

Im dritten Kapitel untersuchen wir Probleme, bei denen es für jede Problemeingabe nur wenige Lösungen gibt. Im Extremfall (Valiant's Klasse UP $\subseteq$ NP, [Val76]) verlangt man, daß jede Problemeingabe maximal eine Lösung besitzt. Die Frage, ob alle UP-Probleme bereits in P liegen, hängt eng mit der ungelösten Frage nach der Existenz von Einweg-Funktionen in der Kryptographie zusammen. In Verallgemeinerung dieser Fragestellung führte Allender [All86] die Klasse FewP $\subseteq$ NP aller ja-nein-Probleme ein, deren Problemeingaben maximal polynomiell viele Lösungen besitzen. FewP wurde von Cai und Hemachandra [CH89] weiter auf die Klasse Few aller ja-nein-Probleme verallgemeinert, bei denen wie bei FewP-Problemen die Anzahl $anz(x)$ der Lösungen polynomiell beschränkt ist. Im Gegensatz zu einem FewP-Problem A, wo es nur um die Frage der Existenz von Lösungen geht (d.h. $x \in A \Leftrightarrow anz(x) > 0$), entscheidet bei einem Few-Problem A ein frei wählbares P-Prädikat Q in Abhängigkeit von $anz(x)$ über die Zugehörigkeit von x zu A, d.h. $x \in A \Leftrightarrow Q(x, anz(x))$. Es ist nicht bekannt, ob Few $\subseteq$ NP gilt. Cai und Hemachandra [CH89] haben jedoch gezeigt, daß Few in $\oplus$P enthalten ist. Die Sprachklasse $\oplus$P wurde von Papadimitriou und Zachos [PZ83] als „moderate version of the counting idea“ eingeführt und enthält alle Sprachen A, zu denen ein NP-Algorithmus existiert, der bei Eingaben x in A eine ungerade und sonst eine gerade Anzahl von akzeptierenden Rechnungen durchführt. $\oplus$P ist unter Komplementbildung abgeschlossen [PZ83], weshalb bei der Definition von $\oplus$P „ungerade“ durch „gerade“ ersetzt werden kann. Es ist weder bekannt, ob NP in $\oplus$P, noch ob $\oplus$P in NP enthalten ist. Das Isomorphieproblem von Turniergraphen liegt z.B. in $\oplus$P [GP86], da die Automorphismengruppe eines Turniergraphen immer von ungerader Ordnung ist.

Als Hauptresultat des dritten Kapitels führen wir jedes unter einem Few-Orakel in #P lösbare Anzahlproblem f auf ein in #P (ohne Orakel) lösbares Anzahlproblem zurück, indem wir zu f eine leicht zu berechnende Funktion h und eine #P-Funktion g mit $f = g - h$ konstruieren. Daraus ergeben sich eine ganze Reihe von Folgerungen, so zum Beispiel, daß Few-Orakel bei PP-Rechnungen überflüssig sind ($\mathrm{PP}^{\mathrm{Few}} = \mathrm{PP}$, d.h. Few ist PP-low). Eine andere beantwortet eine von Schöning [Sch88b] gestellte Frage positiv, ob FewP in C=P enthalten ist. C=P wurde von Wagner [Wag86a] als die Klasse aller Sprachen eingeführt, deren Elemente aus den Problemeingaben bestehen, bei denen eine exakt vorgegebene Anzahl von akzeptierenden Rechnungen durchgeführt wird. Einige der Resultate des dritten Kapitels werden in [KSTT89] erscheinen.

Grundlegende Begriffe und Resultate

In diesem Abschnitt stellen wir die benötigten komplexitätstheoretischen Begriffe zusammen. Dies geschieht in erster Linie mit dem Ziel, die verwendete Notation festzulegen. Dabei benützen wir die Gelegenheit, einige im Umfeld der vorliegenden Arbeit liegende Resultate zu erwähnen.

Σ sei ein (endliches) *Alphabet*, bestehend aus mindestens zwei Buchstaben. $\Sigma^* := \bigcup_{i \geq 0} \Sigma^i$ ist die Menge aller *Wörter* über dem Alphabet Σ. Die *Länge* eines Wortes $u \in \Sigma^*$ bezeichnen wir mit $|u|$, d.h. es gilt $u \in \Sigma^{|u|}$. Es gibt genau ein Wort der Länge 0, das *leere Wort* λ. Die Menge $\bigcup_{i=0}^{n} \Sigma^i$ aller Wörter über Σ, deren Länge höchstens n ist, bezeichnen wir mit $\Sigma^{\leq n}$. Eine *Sprache (über Σ)* ist eine Teilmenge von Σ^*. Das *Komplement* einer Sprache L ist die Sprache

$$\overline{L} = \{x \in \Sigma^* \mid x \notin L\}.$$

Wenn nichts anderes gesagt wird, soll Σ immer das *Standardalphabet* $\{0, 1\}$ sein. Für eine Sprachklasse $\mathcal{A} \subseteq \mathcal{P}(\Sigma^*)$ ist co-$\mathcal{A}$ die Klasse aller Sprachen, deren Komplement in $\mathcal{A}$ ist,

$$\text{co-}\mathcal{A} = \{A \subseteq \Sigma^* \mid \overline{A} \in \mathcal{A}\}.$$

$\mathcal{N}$ ist die Menge der natürlichen Zahlen, $\mathcal{N}_0 := \mathcal{N} \cup \{0\}$. Eine natürliche Zahl n stellen wir binär als ein Wort über dem Standardalphabet Σ dar. Die Länge von n in Binärdarstellung ergibt sich dann zu $|n| = \lfloor \log(n+1) \rfloor$. Dabei ist log der Logarithmus zur Basis 2 und $\lfloor r \rfloor$ die größte ganze Zahl $\leq r$. Entsprechend ist $\lceil r \rceil$ die kleinste ganze Zahl $\geq r$ für eine reelle Zahl r. Seien $f, g : \mathcal{N}_0 \to \mathcal{N}_0$. g ist (in) $O(f)$, falls Konstanten k_1, k_2 existieren mit $g(n) \leq k_1 f(n)$ für alle $n \geq k_2$. Die Menge der ganzen Zahlen bezeichnen wir mit $\mathcal{Z}$. Die Mächtigkeit einer endlichen Menge oder Sprache A bezeichnen wir mit $\|A\|$.

Die *Indikator-* oder *charakteristische Funktion* $\chi_A : \Sigma^* \to \{0,1\}$ einer Sprache A ist definiert durch

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

Die auf Σ vorhandene Totalordnung läßt sich kanonisch auf Σ^* erweitern. Seien hierzu $v = v_1 \ldots v_n$ und $w = w_1 \ldots w_m$ aus Σ^*.

$$\begin{aligned} v < w \quad :\Leftrightarrow \quad & |v| < |w|, \text{ d.h. } n < m, \text{ oder} \\ & |v| = |w|, \text{ d.h. } n = m, \text{ und es gibt ein } i, 1 \leq i \leq n, \\ & \text{mit } v_j = w_j \text{ für alle } j < i, \text{ und } v_i < w_i. \end{aligned}$$

Diese Ordnung heißt *lexikographische Ordnung* auf Σ^*. Für eine Sprache A und $x \in \Sigma^*$ sei

$$A^{\leq x} = \{y \in A \mid y \leq x\}$$

die Menge aller Wörter y in A, die x in der lexikographischen Anordnung vorausgehen.

Ein k-Tupel $(x_1, \ldots, x_k)$ von Wörtern $x_1, \ldots, x_k$ aus Σ^* codieren wir durch ein Wort in Σ^*. Sei hierzu $\pi : \Sigma^* \times \Sigma^* \to \Sigma^*$ eine Injektion mit Umkehrabbildung π^{-1}, deren Projektionen wir mit π_1 und π_2 bezeichnen. Eine Möglichkeit wäre z.B. $\pi(x,y) := \widetilde{x}01y$, wobei $\widetilde{x} := x_1x_1 \ldots x_nx_n$ für $x = x_1 \ldots x_n \in \Sigma^n$. π, π_1, π_2 sind dann in Polynomzeit berechenbar und wir werden später davon Gebrauch machen, daß $|\pi(x,y)|$ nur von $|x|$ und $|y|$ abhängt und nicht kleiner als das Maximum von $|x|$ und $|y|$ ist. $(x_1, \ldots, x_k)$ codieren wir durch das Wort $\pi(0^k, \pi(x_1, \pi(\ldots \pi(x_{k-1}, x_k) \ldots))$. Ist demnach x Codierung eines Tupels $(x_1, \ldots, x_k)$, so gilt

$$\begin{aligned} k &= |\pi_1(x)|, \\ x_i &= \pi_1(\underbrace{\pi_2(\ldots \pi_2(}_{i\text{-mal}}x) \ldots)), \quad i = 1, \ldots, k-1, \end{aligned}$$

$$x_k = \pi_2(\underbrace{\pi_2(\ldots\pi_2(}_{(k-1)\text{-mal}}x)\ldots)).$$

Ist f eine Funktion auf Σ^*, so schreiben wir anstatt $f((x_1,\ldots,x_k))$ einfach $f(x_1,\ldots,x_k)$. Wir werden keine Funktionen auf $(\Sigma^*)^n, n > 1$ betrachten, so daß sich hieraus keine Verwechslungsmöglichkeiten ergeben. Da k nicht immer fest sein wird, ist noch von Bedeutung, daß $(x_1,\ldots,x_k)$ in Polynomzeit bzgl. $|x_1|+\cdots+|x_k|$ berechnet werden kann.

Unser Berechnungsmodell ist die Mehrband-Turingmaschine (vgl. Hopcroft und Ullman [HU79]). Eine k-Band-Turingmaschine M besteht aus einer Steuereinheit und k Bändern, auf denen sich je ein mit der Steuereinheit verbundener Lese-Schreib-Kopf befindet. Die Steuereinheit befindet sich zu jedem Zeitpunkt in einem Zustand z aus einer endlichen Zustandsmenge Z. Die Bänder sind in eine nach links und rechts unendlich ausgebreitete Folge von Feldern eingeteilt, die zu jedem Zeitpunkt mit je genau einem Zeichen aus einem festen Alphabet A *(Arbeitsalphabet)* beschriftet sind. Eine in der Steuereinheit verdrahtete Turingtafel kontrolliert die Rechenschritte von M. Ist M *deterministisch*, so besitzt die Turingtafel für jede Kombination $(z, x_1,\ldots,x_k)$, bestehend aus dem momentanen Zustand z und den sich unter den Schreib-Lese-Köpfen befindlichen Zeichen $x_1,\ldots,x_k$, genau einen Eintrag $(z', x'_1,\ldots,x'_k, b_1,\ldots,b_k)$. Dieser bestimmt die durchzuführende Aktion: M wechselt von Zustand z in den Zustand z', ersetzt die sich unter den Köpfen befindlichen Zeichen $x_1,\ldots,x_k$ durch $x'_1,\ldots,x'_k$ und bewegt die Köpfe gemäß $b_1,\ldots,b_k$, d.h. der Kopf auf Band i wird um $b_i \in \{-1,0,1\}$ Felder nach rechts bewegt.

Die Turingtafel einer *nichtdeterministischen* Maschine M kann mehrere Einträge für dieselbe Kombination besitzen. M wählt bei jedem Rechenschritt aus den unter $(z, x_1,\ldots,x_k)$ eingetragenen Aktionen eine aus, wobei z wieder der momentane Zustand und $x_1,\ldots,x_k$ die gerade gelesenen Zeichen sind. Einen solchen Rechenschritt nennen wir nichtdeterministisch, falls M's Turingtafel unter $(z, x_1, \ldots, x_k)$ mehrere Einträge besitzt, sonst deterministisch. Aus technischen Gründen (und ohne Einschränkung der Allgemeinheit) beschränken wir M's Wahlmöglichkeit in einem Rechenschritt auf höchstens zwei Alternativen.

Einen bei der Rechnung von M gemachten „Schnappschuß", bestehend aus dem momentanen Zustand, den Bandinschriften und den Positionen der Bandköpfe nennen wir eine *Konfiguration* von M. Die Zustandsmenge Z enthält immer die drei Zustände z_0 *(Anfangszustand)*, z_a *(akzeptierender Endzustand)* und z_v *(verwerfender Endzustand)*. Das Arbeitsalphabet A enthält außer den Zeichen des *Eingabealphabets* Σ ein Leerzeichen *(Blank)*. Eine Konfiguration heißt *Startkonfiguration* bei Eingabe x, falls sich M im Zustand z_0 befindet, der Kopf von Band 1 *(Eingabeband)* direkt hinter der Eingabe x steht und das restliche Band sowie alle übrigen Bänder leer sind. Jede Konfiguration, bei der ein Endzustand angenommen wird, heißt *Endkonfiguration*. Eine Folge $k_0, \ldots, k_m$ von nacheinander angenommenen Konfigurationen heißt (endliche) *Rechnung* von $M(x)$, falls k_0 Startkonfiguration bei Eingabe x und k_m die erste in der Folge auftretende Endkonfiguration ist. m heißt die Länge der Rechnung. Ist z_a der in k_m angenommene Zustand, so heißt $k_0, \ldots, k_m$ eine *akzeptierende Rechnung*. Tritt während einer Rechnung $k_0, k_1, \ldots$ keine Endkonfiguration auf, so hat diese Rechnung unendliche Länge.

M *akzeptiert* ein Wort $x \in \Sigma^*$, falls es eine akzeptierende Rechnung von M bei Eingabe x gibt (kurz $M(x) = 1$). Andernfalls *verwirft* M die Eingabe x ($M(x) = 0$). Die von M *erkannte Sprache* ist

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

Die Anzahl aller akzeptierenden Rechnungen von M bei Eingabe x bezeichnen wir mit $acc_M(x)$. Eine Turingmaschine M erkennt also eine Sprache L genau dann, wenn

$$x \in L \Leftrightarrow acc_M(x) > 0$$

für alle $x \in \Sigma^*$ gilt. Für eine deterministische Turingmaschine gilt notwendigerweise $acc_M(x) \leq 1$, d.h. $acc_M = \chi_{L(M)}$.

Sei $t : \mathcal{N}_0 \rightarrow \mathcal{N}_0$. M heißt *t-zeitbeschränkt*, falls für jede Eingabe x alle Rechnungen von $M(x)$ in der Länge durch $t(|x|)$ beschränkt sind. Besuchen die Lese-Schreib-Köpfe in jeder Rechnung insgesamt höchstens $t(|x|)$ verschiedene Bandfelder, so heißt M *t-platzbeschränkt.* M heißt polynomiell zeit- bzw. platzbeschränkt, falls für die Schranke t ein Polynom gewählt werden kann.

Eine *NP-Maschine* ist eine nichtdeterministische, polynomiell zeitbeschränkte Turingmaschine. *NP (Nondeterministic Polynomial Time)* ist die Klasse aller Sprachen, die von einer NP-Maschine erkannt werden. Entsprechend sind *P-Maschine* und *P (Polynomial Time)* im deterministischen Fall definiert. *PSPACE* ist die Klasse aller von einer polynomiell platzbeschränkten Turingmaschine erkannten Sprachen.

Eine *probabilistische* Turingmaschine M ist eine nichtdeterministische Turingmaschine mit einem anderen Akzeptanzmodus: Wir stellen uns vor, daß M bei jedem nichtdeterministischen Rechenschritt ein Zufallsexperiment ausführt und abhängig vom Ausgang entweder die eine oder die andere Alternative wählt. Dabei sollen beide Alternativen dieselbe Wahrscheinlichkeit 1/2 haben. Eine Rechnung von M erhält dadurch die Wahrscheinlichkeit 2^{-t}, wobei t die Anzahl der in der Rechnung durchgeführten nichtdeterministischen Schritte ist. M *akzeptiert* ein Wort $x \in \Sigma^*$, falls M bei Eingabe x mit Wahrscheinlichkeit $\geq 1/2$ eine akzeptierende Rechnung durchführt. Für eine probabilistische Turingmaschine ist $M(x)$ also eine Zufallsvariable und die von M *erkannte*

Sprache ist

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

PP (Probabilistic Polynomial Time) [Gil77] ist die Klasse aller von einer probabilistischen polynomiell zeitbeschränkten Turingmaschine erkannten Sprachen. Es ist bekannt, daß NP in PP und PP in PSPACE enthalten ist [Gil77]. Mit A, B liegen auch $\overline{A}$ und $A \Delta B := (A \setminus B) \cup (B \setminus A)$ in PP, d.h. PP ist unter Komplementbildung und symmetrischer Differenz abgeschlossen [Rus85]. Ein ungelöstes Problem ist, ob PP bzgl. Durchschnitts- und Vereinigungsbildung abgeschlossen ist, und ob PP in der polynomiellen Hierarchie (siehe unten) liegt. Erst kürzlich hat Toda zeigen können, daß alle Sprachen der polynomiellen Hierarchie in P^{PP} (siehe unten) liegen [Tod].

Eine *Orakelmaschine* M ist eine mit einem speziellen Orakelband ausgestattete Turingmaschine, deren Zustandsmenge Z zusätzlich die drei Zustände $z_?$ *(Fragezustand)*, z_j *(ja-Zustand)* und z_n *(nein-Zustand)* enthält. Eine Orakelmaschine M kann an eine Orakelmenge $A \subseteq \Sigma^*$ Fragen stellen, indem sie ein Wort y auf das Orakelband schreibt und anschließend in den Fragezustand $z_?$ übergeht. Die Frage wird nun dadurch beantwortet, daß das Orakelband gelöscht wird, und M im Fall $y \in A$ in den Zustand z_j und andernfalls in den Zustand z_n übergeht. Alle betrachteten Maschinenmodelle lassen sich mit einem solchen Orakelmechanismus ausrüsten. Wir schreiben dann einfach M^A, acc_M^A oder $L(M, A)$, um die Abhängigkeit vom Orakel auszudrücken. Weiter definieren wir für eine durch eine Menge $\mathcal{C}$ von Turingmaschinen definierte Komplexitätsklasse $\mathcal{K} = \{L(M) \mid M \in \mathcal{C}\}$ die bzgl. eines Orakels A *relativierte Klasse*

$$\mathcal{K}^A = \{L(M, A) \mid M \in \mathcal{C}\}$$

und die bzgl. einer Sprachklasse $\mathcal{A}$ relativierte Klasse

$$\mathcal{K}^{\mathcal{A}} = \bigcup_{A \in \mathcal{A}} \{L(M, A) \mid M \in \mathcal{C}\}.$$

Die *Polynomielle Hierarchie* [MS73, Sto77, Wra75, Wra77] besteht aus den folgenden Sprachklassen.

$$\begin{aligned}
\Sigma_0^p &:= \Pi_0^p := \Delta_0^p := \mathrm{P}; \\
\Delta_{k+1}^p &:= \mathrm{P}^{\Sigma_k^p},\ k \geq 0; \\
\Sigma_{k+1}^p &:= \mathrm{NP}^{\Sigma_k^p},\ k \geq 0; \\
\Pi_{k+1}^p &:= \text{co-}\mathrm{NP}^{\Sigma_k^p},\ k \geq 0; \\
\mathrm{PH} &:= \bigcup_{k \geq 0} \Sigma_k^p.
\end{aligned}$$

Es ist nicht bekannt, ob die Polynomielle Hierarchie kollabiert, d.h. ob es ein k gibt mit $\mathrm{PH} = \Sigma_k^p$. Eine hinreichende (aber vermutlich nicht notwendige) Bedingung hierfür ist $\mathrm{PH} = \mathrm{PSPACE}$ [Wra77]. Aus $\Sigma_k^p = \Pi_k^p$ folgt der Kollaps auf (mindestens) die k-te Stufe, d.h. $\mathrm{PH} = \Sigma_k^p$.

An die Stelle eines Sprachorakels kann auch eine Wortfunktion $f : \Sigma^* \to \Sigma^*$ treten. Eine unter Orakel f arbeitende Orakelmaschine M^f kann im Verlauf einer Rechnung (mehrmals) eine Frage y auf ihr Orakelband schreiben und in den Fragezustand übergehen. Die Beantwortung erfolgt dann dadurch, daß die Orakelbandinschrift y durch $f(y)$ ersetzt wird.

Ein *Transducer* M ist eine Turingmaschine mit *Ausgabeband*. Dabei ist die von M bei einer akzeptierenden Rechnung erzeugte Ausgabe wie folgt definiert. Steht der Ausgabebandkopf nach Beendigung der Rechnung auf einem Leerzeichen, so ist λ die *erzeugte Ausgabe*, ansonsten das Wort aus Σ^*, innerhalb dessen sich der Kopf befindet. Ist M deterministisch, so bezeichnen wir die von M bei Eingabe x erzeugte Ausgabe mit $f_M(x)$, falls M x akzeptiert. Andernfalls ist $f_M(x)$ undefiniert. Ist M nichtdeterministisch, so ist $f_M(x, y)$ die von M bei Eingabe

$x \in \Sigma^*$ auf Pfad $y \in \Sigma^*$ erzeugte Ausgabe, falls M auf Pfad y akzeptiert. Andernfalls ist $f_M(x, y)$ wieder undefiniert. Dabei wird der *Pfad* y als ein Protokoll der von M der Reihe nach in den nichtdeterministischen Schritten getroffenen Entscheidungen interpretiert.

F$\mathcal{S}$ ist die Klasse aller Wortfunktionen, die von einem deterministischen, polynomiell zeitbeschränkten Orakeltransducer mit einem Orakel S aus $\mathcal{S}$ berechnet werden.

Sei $\mathcal{K}^{\mathcal{S}}$ eine bzgl. einer Sprach- bzw. Funktionenklasse $\mathcal{S}$ relativierte Komplexitätsklasse. Weiter sei $\mathcal{G}$ eine Menge von Funktionen $g : \mathcal{N}_0 \to \mathcal{N}_0$. Mit $\mathcal{K}^{\mathcal{S}}[\mathcal{G}]$ bezeichnen wir dann die Menge all jener Sprachen $L(M, S)$ in $\mathcal{K}^{\mathcal{S}}$, bei denen für eine Funktion $g \in \mathcal{G}$ die Orakelmaschine M bei jeder Eingabe x auf jedem Berechnungspfad höchstens $g(|x|)$ Fragen an das Orakel S stellt. Dabei kann $\mathcal{K}$ natürlich auch eine Klasse von Funktionen sein.

Für eine Klasse $\mathcal{F}$ von Wortfunktionen $f : \Sigma^* \to \Sigma^*$ und eine Klasse $\mathcal{G}$ von Funktionen $g : \mathcal{N}_0 \to \mathcal{N}_0$ sei $\mathcal{F}\langle\mathcal{G}\rangle$ die Teilklasse von $\mathcal{F}$, in der nur noch die Funktionen enthalten sind, bei denen die Länge der Funktionswerte über eine Funktion aus $\mathcal{G}$ durch die Argumentlänge beschränkt ist,

$$\mathcal{F}\langle\mathcal{G}\rangle = \{f \in \mathcal{F} \mid \text{ es gibt ein } g \in \mathcal{G} \text{ mit } |f(x)| \leq g(|x|)\}.$$

Eine Funktion $f : \Sigma^* \to \Sigma^*$ heißt *polynomiell beschränkt*, falls es ein Polynom p gibt mit $|f(x)| \leq p(|x|)$ für alle $x \in \Sigma^*$.

Eine Sprache A ist auf eine Sprache B *many-one reduzierbar* ($A \leq_m^p B$), falls eine Funktion $f \in \mathrm{FP}$ existiert mit $f^{-1}(B) = A$, d.h. $x \in A \Leftrightarrow f(x) \in B$ für alle $x \in \Sigma^*$. Ein Problem B heißt bzgl. einer Reduktion $\preceq$ *hart* für eine Komplexitätsklasse $\mathcal{K}$ ($\mathcal{K}$-hart), falls $A \preceq B$ für alle $A \in \mathcal{K}$ gilt. Ein bzgl. $\preceq$ $\mathcal{K}$-hartes Problem, das in $\mathcal{K}$ liegt, heißt *vollständig* für $\mathcal{K}$ ($\mathcal{K}$-vollständig) bzgl. $\preceq$. Wir vereinbaren, daß wir im Fall der many-one Reduktion den Zusatz bzgl. $\leq_m^p$ auch weglassen können. Das erste

Problem, für das der Nachweis der NP-Vollständigkeit gelang [Coo71, Kar72], ist das Erfüllbarkeitsproblem für die aus Aussagenvariablen und den logischen Junktoren $\neg$, $\vee$ und $\wedge$ aufgebauten Booleschen Formeln,

$$\mathrm{SAT} := \{x \in \Sigma^* \mid \text{ es gibt eine erfüllende Belegung für } x\}.$$

Hierbei und in ähnlichen Fällen setzen wir ein festes, aber nicht näher spezifiziertes Codierungsverfahren voraus, das mathematische Objekte wie Formeln oder Graphen durch Wörter aus Σ^* beschreibt. Jedes Wort steht also — abhängig von der Problemstellung — für ein eindeutig (aber nicht unbedingt eineindeutig) bestimmtes mathematisches Objekt. Wir werden dabei nicht besonders zwischen der Codierung und dem codierten Objekt unterscheiden. Ein bekanntes PP-vollständiges Problem ist

$$\mathrm{MAJ} := \{x \in \Sigma^* \mid \text{ mindestens die Hälfte aller Belegungen erfüllt } x\}.$$

Dabei ist x wieder eine Boolesche Formel und es werden nur Belegungen über den in x vorkommenden Aussagenvariablen in Betracht gezogen. Eine schwächere Form von Reduktion ist die *Turing-Reduktion* ($A \leq_T^p B$), die durch $A \in \mathrm{P}^B$ definiert ist. Die many-one Reduktion kann als eine spezielle Turing-Reduktion aufgefaßt werden, bei der nur eine Orakelfrage gestellt werden darf und in direkter Abhängigkeit von der Antwort akzeptiert oder verworfen werden muß. Zwischen der Turing- und der many-one Reduktion existiert eine Vielzahl weiterer Reduktionen wie zum Beispiel die *k-tt Reduktionen*, bei denen höchstens k Fragen unabhängig voneinander gestellt werden dürfen [LLS75] (vgl. auch [KSW87]).

1 Anzahlbestimmende Funktionen

1.1 Einleitung

Wir betrachten Funktionen $f : \Sigma^* \to \mathcal{N}_0$, deren Funktionswerte $f(x)$ sich durch Anwendung eines Operators auf die Menge

$$out_M(x) := \{y \in \Sigma^* \mid \exists z \in \Sigma^* : f_M(x,z) = y\}$$

aller von einem NP-Transducer M bei Eingabe x erzeugten Ausgaben ergeben. Zum Beispiel gilt $acc_M(x) = \|out_M(x)\|$ für einen NP-Transducer M, der bei jeder akzeptierenden Rechnung den zugehörigen Pfad ausgibt. Valiant [Val79a] hat diese Funktionen zur Klasse

$$\#\mathrm{P} = \{acc_M \mid M \text{ ist eine NP-Maschine}\}$$

oder allgemeiner für eine Orakelklasse $\mathcal{A}$ zu

$$\#\mathcal{A} = \{acc_M^A \mid M \text{ ist eine NP-Orakelmaschine und } A \in \mathcal{A}\}$$

zusammengefaßt und gezeigt, daß #P *(Number-P)* natürliche vollständige Probleme besitzt. Viele nichtdeterministische Algorithmen arbeiten nach einer „rate und verifiziere"-Strategie. Das bedeutet, sie versuchen einen von vielen möglichen Wegen und testen, ob der geratene Weg auf eine Lösung des Problems führt. Ist der eingeschlagene Weg erfolgreich, so wird die Eingabe akzeptiert. Der Funktionswert einer #P-Funktion kann somit als die Anzahl verschiedener Lösungswege („witnesses") für eine Problemeingabe interpretiert werden. Ein typisches #P-Problem ist

$$\#\mathrm{SAT}(x) := \|\{h \mid h(x) = 1\}\|,$$

das für eine Boolesche Formel x die Anzahl der erfüllenden Belegungen h über den in x vorkommenden Variablen bestimmt. Jede erfüllende Belegung h für die Boolesche Formel x kann als eine Lösung des

Erfüllbarkeitsproblems für x angesehen werden. Dabei entspricht jede akzeptierende Rechnung genau einer Lösung. Es gibt jedoch Beispiele (siehe unten), bei denen die zu zählenden Lösungen für eine Eingabe x nicht eineindeutig den akzeptierenden Rechnungen von M entsprechen, sondern bei denen ein und dieselbe Lösung auf mehreren Lösungswegen (akzeptierenden Pfaden) gefunden wird. Daher definieren wir für einen nichtdeterministischen Turingtransducer M die folgende Funktion.

Definition 1.1 *Für einen* NP-*Transducer* M *sei*

$$span_M(x) := \|out_M(x)\|,$$

die Funktion, die für jedes $x \in \Sigma^*$ *die Anzahl aller verschiedenen Ausgaben berechnet, die* M *bei Eingabe* x *produzieren kann.*

Krentel [Kre88] hat Funktionen untersucht, die die optimale Lösung einer Problemeingabe für ein NP-Problem bestimmen. Um wieder Funktionen von Σ^* in $\mathcal{N}_0$ zu erhalten, interpretieren wir die von einem Transducer erzeugte Ausgabe $y = y_1 \dots y_n \in \Sigma^*$ als Binärdarstellung des Ausgabewerts $\sum_{i=1}^{n} y_i 2^{n-i}$. O.B.d.A. soll eine als Zahlenwert zu interpretierende Ausgabe immer ohne führende Nullen erfolgen. Durch diese Vereinbarung erreichen wir, daß die Ordnung auf $\mathcal{N}_0$ mit der lexikographischen Ordnung auf den Binärdarstellungen von natürlichen Zahlen übereinstimmt. Für einen nichtdeterministischen Turingtransducer M ist

$$max_M(x) := \max(out_M(x))$$

der maximale von $M(x)$ erzeugte Ausgabewert, falls $out_M(x) \neq \emptyset$ und $max_M(x) := 0$ sonst. Dabei bedeutet es keine Einschränkung, zu verlangen, daß M auf jedem Pfad einen Zahlenwert ausgibt. Anstatt zu verwerfen, kann M nämlich den Wert 0 ausgeben, wodurch sich max_M nicht ändert. Krentel nennt einen solchen Transducer *metrisch*. Analog zu max_M sei $min_M(x)$ der minimale von $M(x)$ erzeugte Ausgabewert, wobei wir wieder voraussetzen, daß M ein metrischer Transducer ist.

1.2 Anzahlprobleme in span-P

In diesem Abschnitt untersuchen wir die Beziehungen von span-P zu anderen Klassen von Berechnungsproblemen, die über einen Operator auf out_M definiert sind. Für eine Klasse $\mathcal{A}$ von Orakeln sei

$$\text{min-}\mathcal{A} = \{min_M^A \mid M \text{ ist ein NP-Orakeltransducer und } A \in \mathcal{A}\},$$

$$\text{max-}\mathcal{A} = \{max_M^A \mid M \text{ ist ein NP-Orakeltransducer und } A \in \mathcal{A}\}$$

und

$$\text{span-}\mathcal{A} = \{span_M^A \mid M \text{ ist ein NP-Orakeltransducer und } A \in \mathcal{A}\}.$$

Man beachte, daß die Vereinigung von max-P und min-P gerade Krentels Klasse opt-P ist (vgl. [Kre87]).

In span-P sind viele interessante Funktionen enthalten, von denen nicht bekannt ist, ob sie auch in #P liegen. Beispiele sind die ranking-Funktionen von NP-Sprachen (siehe unten) und die Funktionen der Klasse max-P, in der viele für die Praxis relevante Berechnungsprobleme liegen [Kre88]. Anlaß zur Einführung von span-P war folgende aus dem Graphisomorphieproblem entstandene Funktion f:

$$f(G_1, G_2) = \begin{cases} n_1!, & \text{falls } G_1 \text{ und } G_2 \text{ isomorph sind,} \\ n_1! + n_2!, & \text{falls } G_1 \text{ und } G_2 \text{ nicht isomorph sind.} \end{cases}$$

Dabei ist n_i die Anzahl der Knoten im Graphen G_i. f liegt in span-P. Wir betrachten hierzu zunächst den folgenden NP-Transducer.

> M rät bei Eingabe (G_1, G_2) eine Zahl $i \in \{1, 2\}$ und eine Permutation π auf der Menge $\{1, \ldots, n_i\}$. Dann berechnet M den Graphen $\pi(G_i)$ und gibt ihn aus.

Ist m_i die Anzahl aller zu G_i isomorphen Graphen ($i \in \{1,2\}$), so ist $span_M(G_1, G_2) = m_1 = m_2$, falls $G_1 \simeq G_2$, und $m_1 + m_2$ sonst. Um auf die Anzahlen $n_1!$ bzw. $n_1! + n_2!$ zu kommen, muß zu $\pi(G_i)$ nur noch ein Automorphismus π' von G_i geraten und das Paar $(\pi(G_i), \pi')$ ausgegeben werden. Folgende Maschine N berechnet $span_N = f$.

> N rät bei Eingabe (G_1, G_2) eine Zahl $i \in \{1,2\}$ und Permutationen π, π' auf der Menge $\{1, \ldots, n_i\}$. Dann berechnet N den Graphen $\pi(G_i)$ und gibt das Paar $(\pi(G_i), \pi')$ aus, falls $\pi'(G_i) = G_i$ ist.

Wie wir in Abschnitt 2 sehen werden, lassen sich alle span-P-Funktionen durch probabilistische Algorithmen mit beliebig kleiner Fehlerwahrscheinlichkeit in Polynomzeit mit einem NP-Orakel approximativ berechnen. Im Falle der Funktion f läßt sich daraus ein Beweis für das kürzlich in [Sch88a] erzielte Resultat, daß GRAPHISOMORPHIE „low“ ist, erhalten.

Abbildung 1 veranschaulicht die in Korollar 1.1 und Satz 1.3 gezeigten Inklusionsbeziehungen.

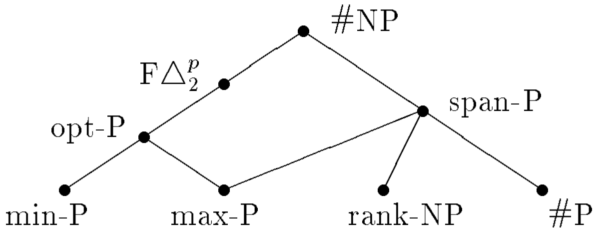

Abbildung 1: span-P und die anderen Funktionenklassen

Ranking-Funktionen wurden von Goldberg und Sipser [GS85] zur Komprimierung von Sprachen eingeführt.

Definition 1.2 *Für eine Sprache A heißt die Funktion*

$$rank_A(x) := \|A^{\leq x}\| = \|\{y \in A \mid y \leq x\}\|$$

ranking-Funktion *von A. Für eine Sprachklasse $\mathcal{A}$ ist*

$$\text{rank-}\mathcal{A} := \{rank_A \mid A \in \mathcal{A}\}.$$

Da rank-NP nur monoton wachsende Funktionen enthält, ist FP nicht in rank-NP enthalten. Es ist jedoch leicht zu sehen, daß FP in den Klassen min-P, max-P und #P enthalten ist. Diese Inklusionsbeziehungen sind relativierbar, zum Beispiel folgt $\mathrm{F}\Delta_2^p \subseteq \#\mathrm{NP}$ durch Relativierung von $\mathrm{FP} \subseteq \#\mathrm{P}$ bzgl. NP. Für die restlichen Inklusionsbeziehungen benötigen wir noch ein wenig Notation. Eine Funktion f ist durch ihren Graphen

$$Graph(f) = \{(x, y) \mid y = f(x)\}$$

eindeutig charakterisiert. Auch läßt sich die Sprache $Graph(f)$ relativ zum Funktionsorakel f mit geringem Aufwand erkennen. Es ist jedoch ein offenes Problem, ob umgekehrt f immer relativ zum Orakel $Graph(f)$ in Polynomzeit berechnet werden kann (vgl. [Val76]). Für eine Funktion $f : \Sigma^* \to \mathcal{N}_0$ definieren wir daher die Sprache

$$Graph_{\leq}(f) := \{(x, y) \mid y \leq f(x)\},$$

mit deren Hilfe $f(x)$ leicht durch binäre Suche bestimmt werden kann (vgl. Abbildung 2). Ist $f(x) < 2^{p(|x|)}$, so reichen hierzu $p(|x|)$ Fragen an $Graph_{\leq}(f)$ aus. Natürlich kann als Orakel genau so gut auch die Sprache

$$Graph_{\geq}(f) := \{(x, y) \mid y \geq f(x)\}$$

verwendet werden. Das folgende Lemma charakterisiert max-P- und min-P-Funktionen f durch die Komplexität von $Graph_{\leq}(f)$.

input: x;
$k := 0$;
for $i := p(|x|) - 1$ **downto** 0 **do**
  if $f(x) \geq k + 2^i$ **then** $k := k + 2^i$;
output k.

Abbildung 2: Bestimmung von $f(x) < 2^{p(|x|)}$ durch binäre Suche

Lemma 1.1 *Für jede polynomiell beschränkte Funktion* $f : \Sigma^* \to \mathcal{N}_0$ *gilt*

i) $\text{Graph}_{\leq}(f) \in \text{NP} \iff \text{Graph}_{\geq}(f) \in \text{co-NP}$,

ii) $f \in \text{max-P} \iff \text{Graph}_{\leq}(f) \in \text{NP}$,

iii) $f \in \text{min-P} \iff \text{Graph}_{\geq}(f) \in \text{NP}$.

Beweis.

i) Klar, da $y \leq f(x) \Leftrightarrow \neg(y - 1 \geq f(x))$.

ii) ($\Rightarrow$) Sei $f = max_M$ für einen NP-Transducer M. Dann ist $Graph_{\leq}(f) = \{(x, y) \mid y \leq f(x)\} = \{(x, y) \mid$ es gibt eine akzeptierende Rechnung von $M(x)$, die eine Ausgabe $z \geq y$ erzeugt$\}$ in NP.
($\Leftarrow$) Sei p ein Polynom mit $|f(x)| \leq p(|x|)$ für alle $x \in \Sigma^*$. Ist $Graph_{\leq}(f)$ in NP, so ist $f = max_M$ für folgenden NP-Transducer M:

$M(x)$ rät eine Zahl y, $|y| \leq p(|x|)$ und gibt y aus, falls die Verifikation von $(x, y) \in Graph_{\leq}(f)$ gelingt.

iii) Unmittelbare Folgerung aus *i*) und *ii*). □

Satz 1.1 [Kre88] max-P *und* min-P *sind in* $\mathrm{F}\triangle_2^p$ *enthalten.*

Beweis. Sei $f = max_M \in$ max-P für einen NP-Transducer M. Nach Lemma 1.1 ist $Graph_{\leq}(f)$ in NP. Sei p ein Polynom mit $f(x) < 2^{p(|x|)}$ für alle $x \in \Sigma^*$. Dann kann $f(x)$ von einem P-Orakeltransducer T durch binäre Suche mit $p(|x|)$ Fragen an das NP-Orakel $Graph_{\leq}(f)$ bestimmt werden (vgl. Abbildung 2). Der Beweis für min-P verläuft analog. □

Der die binäre Suche durchführende Orakeltransducer T im Beweis zu Satz 1.1 besitzt folgende Monotonieeigenschaft.

$$A \subseteq B \quad \Rightarrow \quad \forall x \in \Sigma^* : T^A(x) \leq T^B(x) \tag{1}$$

Wir zeigen im nächsten Satz, daß diese Eigenschaft zur Charakterisierung der max- und min-P-Funktionen verwendet werden kann.

Satz 1.2 *Eine Funktion f liegt genau dann in* max-P *bzw.* min-P, *wenn $f = T^A$ für ein* NP- *bzw.* co-NP-*Orakel A und einen* P-*Transducer T ist, der (1) erfüllt.*

Beweis. Die Richtung von links nach rechts haben wir schon gezeigt. Sei $f = T^A$ für einen P-Orakeltransducer mit Eigenschaft (1) und ein NP-Orakel A. Dann ist $f = max_M$ für folgenden NP-Transducer M.

> $M(x)$ simuliert $T(x)$, wobei M Orakelfragen „$y \in A$?“ selbst zu verifizieren versucht. Gelingt die Verifikation, so führt M die Simulation von T im ja-Zustand, andernfalls im nein-Zustand fort.

Wir können o.B.d.A. annehmen, daß T auf keinem Pfad dieselbe Orakelfrage mehrmals stellt. Daher gibt es zu jeder Ausgabe z von $M(x)$ ein (endliches) Orakel $B \subseteq A$ mit $T^B(x) = z$, d.h. $z \leq T^A(x)$. Andererseits erzeugt ein Pfad, auf dem für jede Orakelfrage y in A die Verifikation gelingt, die Ausgabe $T^A(x)$. Die Charakterisierung von min-P ergibt sich analog. $\square$

Die Klasse UP wurde von Valiant in [Val76] eingeführt, wo die Komplexität zur Berechnung einer Funktion f mit der Komplexität von *Graph*(f) verglichen wird. Es ist nicht bekannt, ob UP vollständige Sprachen besitzt.

Definition 1.3 *Eine* NP-*Maschine* M *heißt* UP-Maschine, *falls sie die Bedingung* $acc_M(x) \leq 1$ *für alle* $x \in \Sigma^*$ *erfüllt, d.h.* M *hat bei Eingaben* $x \in L(M)$ *genau einen akzeptierenden Pfad.* UP (Unique Polynomial Time) *ist die Klasse aller Sprachen, die von einer* UP-*Maschine erkannt werden.*

Als Vorbereitung für Satz 1.3 dient das folgende Lemma 1.2, das eine auch für sich interessante Charakterisierung der #P- und span-P-Funktionen gibt.

Lemma 1.2 *Sei* $\mathcal{A}$ *eine Sprachklasse.*

i) Eine Funktion f *ist genau dann in* $\#\mathrm{P}^{\mathcal{A}}$, *wenn es ein* $\mathrm{P}^{\mathcal{A}}$-*Prädikat (oder gleichwertig:* $\mathrm{UP}^{\mathcal{A}}$-*Prädikat)* R *und ein Polynom* p *gibt, so daß für alle* $x \in \Sigma^*$ *gilt:*

$$f(x) = \|\{y \in \Sigma^{\leq p(|x|)} \mid R(x,y)\}\|.$$

ii) Eine Funktion f *ist genau dann in* span-$\mathrm{P}^{\mathcal{A}}$, *wenn es ein* $\mathrm{NP}^{\mathcal{A}}$-*Prädikat* R *und ein Polynom* p *gibt, so daß für alle* $x \in \Sigma^*$ *gilt:*

$$f(x) = \|\{y \in \Sigma^{\leq p(|x|)} \mid R(x,y)\}\|.$$

Dabei kann R so gewählt werden, daß $|y| = p(|x|)$ für alle x, y mit $R(x,y)$ gilt, d.h. $f(x) = \|\{y \in \Sigma^{p(|x|)} \mid R(x,y)\}\|$.

Beweis.

i) Wir beweisen nur die unrelativierte Fassung. Die Richtung von rechts nach links ist (auch für ein UP-Prädikat R) einfach. Für die umgekehrte Richtung sei $f = acc_M$ eine #P-Funktion f via einer NP-Maschine M. Wir betrachten das P-Prädikat

$$R' := \{(x,y) \mid M(x) \text{ akzeptiert auf Pfad } y\}.$$

Besitzt M die polynomielle Zeitschranke p, so gilt $|y| \leq p(|x|)$ für alle x, y mit $R'(x,y)$. Den Zusatz beweisen wir mit einem *padding*-Argument. Aus R' erhalten wir das gewünschte NP-Prädikat

$$R(x, y10^{p(|x|)-|y|}) :\Longleftrightarrow R'(x,y)$$

mit der Eigenschaft $R(x,y) \Rightarrow |y| = p(|x|)+1$ durch „Ausstopfen“ des zweiten Arguments.

ii) Analog zu *i)*, wobei nun das NP-Prädikat $R'(x,y)$ besagt, daß der Transducer M mit $f = span_M$ bei Eingabe x auf einem akzeptierenden Pfad die Ausgabe y erzeugt. □

In Satz 1.3 beweisen wir die in Abbildung 1 dargestellten Inklusionsbeziehungen zwischen den Klassen span-P, max-P, rank-NP, #P und #NP.

Satz 1.3

i) #P $\subseteq$ span-P,

ii) max-P $\subseteq$ span-P,

iii) rank-NP $\subset$ span-P,

iv) span-P $\subseteq$ #NP.

Beweis.

i) Unmittelbare Folgerung aus Lemma 1.2.

ii) Sei M ein NP-Transducer mit Zeitschranke p. Wir definieren das NP-Prädikat

$$R := \{(x, y) \mid \exists z\,(|z| \leq p(|x|)) : f_M(x, z) > y\}.$$

Dann ist $max_M(x) = \|\{y \in \Sigma^{\leq p(|x|)} \mid R(x, y)\}\|$ nach Lemma 1.2 in span-P.

iii) Sei A eine NP-Sprache, dann ist auch das Prädikat

$$R := \{(x, y) \mid y \leq x \text{ und } y \in A\}$$

in NP, also ist $rank_A(x) = \|\{y \in \Sigma^{\leq p(|x|)} \mid R(x, y)\}\|$ nach Lemma 1.2 in span-P. Da ranking-Funktionen immer monoton sind, ist rank-NP eine echte Teilmenge von span-P.

iv) Nach Lemma 1.2 *ii*) gibt es zu $f \in$ span-P ein NP-Prädikat R und ein Polynom p, so daß $f(x) = \|\{y \in \Sigma^{p(|x|)} \mid R(x, y)\}\|$ ist. Dann gilt $f = acc^R_{M'}$ für eine NP-Maschine M', die bei Eingabe x einen String y der Länge $p(|x|)$ rät, die Frage (x, y) stellt und bei einer positiven Antwort akzeptiert. $\square$

Das folgende Lemma kennzeichnet die Sprachen, deren charakteristische Funktionen in den Funktionenklassen #P, min-P, max-P und span-P liegen. Es wird sich in den folgenden Sätzen als sehr nützlich erweisen. Die Aussagen von Lemma 1.3 gelten natürlich auch relativ zu einem beliebigen Orakel, womit man z.B. für jede Sprache A die Äquivalenz $A \in \mathrm{UP}^{\mathrm{NP}} \Leftrightarrow \chi_A \in \#\mathrm{NP}$ erhält, da $\#\mathrm{P}^{\mathrm{NP}}$ nach Definition gleich #NP ist.

Lemma 1.3 *Für jede Sprache A gilt*

i) $A \in \mathrm{NP} \Longleftrightarrow \chi_A \in$ span-P $\Longleftrightarrow \chi_A \in$ max-P,

ii) $A \in$ co-NP $\Longleftrightarrow \chi_A \in$ min-P,

iii) $A \in \mathrm{UP} \Longleftrightarrow \chi_A \in \#\mathrm{P}$.

Beweis.

i) Es ist klar, daß die charakteristischen Funktionen von NP-Sprachen in span-P und max-P liegen. Umgekehrt nimmt $\chi_A \in$ span-P für ein Argument x genau dann den Wert 1 an, wenn der zugehörige Transducer bei Eingabe x eine Ausgabe erzeugt. Analog gilt $\chi_B(x) = 1$ für eine max-P-Funktion χ_B, falls der zugehörige Transducer bei Eingabe x auf einem Pfad den Ausgabewert 1 erzeugt. A und B sind also in NP.

ii) Ist χ_A eine min-P-Funktion, so liefert sie genau dann den Wert 1, wenn der zugehörige Transducer bei Eingabe x nur Ausgabewerte größergleich 1 erzeugt. Also ist A in co-NP. Die umgekehrte Richtung folgt mit Lemma 1.1 *ii)*, da $Graph_{\leq}(\chi_L)$ auf L many-one reduzierbar ist, und deshalb mit L auch $Graph_{\leq}(\chi_L)$ in co-NP, d.h. χ_L in min-P ist.

iii) Ist $A \in \mathrm{UP}$ via einer UP-Maschine M, so ist $\chi_A = acc_M$ in #P. Ist umgekehrt χ_A in #P, so gibt es eine NP-Maschine M mit $acc_M = \chi_A$. Wegen $acc_M \leq 1$ ist M eine UP-Maschine, d.h. $A = L(M)$ ist in UP. $\square$

Bevor wir das Bestehen weiterer Inklusionsbeziehungen zwischen span-P und den Klassen #P, #NP und max-P untersuchen, bestimmen wir diejenigen Orakelsprachen, auf deren Befragung bei einer #P- bzw. span-P-Berechnung verzichtet werden kann, ohne daß deswegen weniger Funktionen berechnet werden könnten. In diesem Zusammenhang verwenden wir den von Schöning [Sch83] aus der Rekursionstheorie in die Komplexitätstheorie übertragenen Begriff der „lowness".

Definition 1.4 *Eine Sprach- bzw. Funktionenklasse $\mathcal{K}$ heißt* low *für eine Komplexitätsklasse $\mathcal{S}$, kurz $\mathcal{S}$-low, falls $\mathcal{S}^{\mathcal{K}} = \mathcal{S}$ gilt.*

Wir zeigen, daß $\mathrm{UP} \cap \text{co-UP}$-Sprachen #P-low und $\mathrm{NP} \cap \text{co-NP}$-Sprachen span-P-low sind. Dabei greifen wir auf Schönings Resultat [Sch83] der NP-lowness von $\mathrm{NP} \cap \text{co-NP}$ zurück, dessen Beweis zugleich die UP-lowness von $\mathrm{UP} \cap \text{co-UP}$ liefert.

Satz 1.4 *Sei A eine Sprache.*

i) A ist genau dann #P*-low, wenn $A \in \mathrm{UP} \cap \text{co-UP}$ ist.*

ii) A ist genau dann span-P*-low, wenn $A \in \mathrm{NP} \cap \text{co-NP}$ ist.*

Beweis.

i) ($\Rightarrow$) Ist A #P-low, d.h. $\#\mathrm{P}^A = \#\mathrm{P}$, dann sind χ_A und $\chi_{\overline{A}}$ in #P, also ist A in $\mathrm{UP} \cap \text{co-UP}$.
($\Leftarrow$) Ist $A \in \mathrm{UP} \cap \text{co-UP}$ und M eine NP-Orakelmaschine, so existiert nach Lemma 1.2 ein P^A-Prädikat R und ein Polynom p mit $acc_M^A(x) = \|\{y \in \Sigma^{\leq p(|x|)} \mid R(x,y)\}\|$.
Wegen $\mathrm{P}^{\mathrm{UP}\cap\text{co-UP}} \subseteq \mathrm{UP}^{\mathrm{UP}\cap\text{co-UP}} = \mathrm{UP}$ ist R ein UP-Prädikat, also ist $acc_M^A(x)$ nach Lemma 1.2 in #P.

ii) Analog zu *i)* unter Berücksichtigung von $\mathrm{NP}^{\mathrm{NP}\cap\mathrm{co\text{-}NP}} = \mathrm{NP}$. □

In den folgenden Sätzen 1.5 bis 1.7 wird deutlich, daß die Frage nach dem Bestehen weiterer Inklusionsbeziehungen zwischen span-P und den Klassen #P, #NP, max-P eng an offene Probleme in der Komplexitätstheorie geknüpft ist. Zunächst geht es um die Frage, ob es span-P-Funktionen gibt, die nicht in #P liegen.

Satz 1.5 *Folgende Aussagen sind äquivalent*

i) UP = NP

ii) #P = span-P

iii) max-P $\subseteq$ #P

Beweis. *i)* $\Rightarrow$ *ii)* Unmittelbare Folgerung aus Lemma 1.2.
ii) $\Rightarrow$ *iii)* Nach Satz 1.3 ist max-P $\subseteq$ span-P.
iii) $\Rightarrow$ *i)* Es ist $A \in \mathrm{UP} \iff \chi_A \in$ #P und $A \in \mathrm{NP} \iff \chi_A \in$ max-P nach Lemma 1.3. □

Satz 1.5 liefert ein interessantes Korollar, was die Frage betrifft, ob alle ranking-Funktionen für NP-Sprachen schon von #P-Maschinen berechnet werden können. Wegen rank-NP $\subseteq$ span-P würde nämlich eine negative Antwort sofort #P $\neq$ span-P und damit UP $\neq$ NP nach sich ziehen.

Korollar 1.1 *Falls es eine* NP*-Sprache gibt, deren ranking-Funktion nicht in* #P *liegt, ist* UP $\neq$ NP.

In Satz 1.6 gehen wir der Frage nach, ob span-P eine echte Teilklasse von #NP ist oder mit #NP zusammenfällt. Aufgrund von Satz 1.6

ist es sehr unwahrscheinlich, daß #NP und span-NP exakt dieselben Funktionen enthalten. Wir werden jedoch später sehen (Satz 1.9), daß zur Berechnung einer #NP-Funktion nur unwesentlich mehr Rechenzeit als für eine entsprechende span-P-Funktion aufgebracht werden muß.

Satz 1.6 *Folgende Aussagen sind äquivalent*

i) NP = co-NP

ii) span-P = #NP

iii) min-P $\subseteq$ span-P

iv) min-P = max-P

Beweis. *i)* $\Rightarrow$ *ii)* Ist NP = co-NP, so folgt mit Satz 1.3 (relativiert) und Satz 1.4 $\#\mathrm{NP} \subseteq \#\mathrm{P}^{\mathrm{NP}\cap\text{co-NP}} \subseteq \text{span-P}^{\mathrm{NP}\cap\text{co-NP}} = \text{span-P}$.
ii) $\Rightarrow$ *iii)* Klar, da min-P $\subseteq \mathrm{F}\Delta_2^p$ nach Satz 1.1.
iii) $\Rightarrow$ *i)* Sei $A \in$ co-NP. Dann ist χ_A nach Lemma 1.3 in min-P und damit nach Voraussetzung in span-P, d.h. A ist in NP.
i) $\Leftrightarrow$ *iv)* Direkte Folgerung aus Lemma 1.1 und Lemma 1.3. □

Die Kombination von Satz 1.5 und Satz 1.6 führt auf äquivalente Formulierungen der Frage $\#\mathrm{P} \stackrel{?}{=} \#\mathrm{NP}$.

Korollar 1.2 *Folgende Aussagen sind äquivalent*

i) UP = co-NP

ii) #P = #NP

iii) min-P $\subseteq$ #P

Mit Toda's neuem Resultat ergibt sich, daß #P-Funktionen nur im Falle eines Kollapses der polynomiellen Hierarchie in Polynomzeit unter einem Orakel aus PH berechnet werden können. Eine Inklusion von #P in $\mathrm{F}\Delta_k^p$ für ein $k \in \mathcal{N}$ impliziert nämlich wegen $\mathrm{PH} \subseteq \mathrm{P}^{\mathrm{PP}}$ [Tod] und $\mathrm{P}^{\mathrm{PP}} = \mathrm{P}^{\#\mathrm{P}}$ [BBS86] den Kollaps von PH auf Δ_k^p. Wie der nächste Satz zeigt, ist eine Berechnung der #P-Funktionen durch P-Orakeltransducer mit der Monotonieeigenschaft (1) unter einem NP-Orakel mit dem unwahrscheinlichen Resultat PP = NP äquivalent.

Satz 1.7 $\#\mathrm{P} \subseteq$ opt-P $\Longleftrightarrow$ PP = NP.

Beweis. ($\Rightarrow$) Gelte $\#\mathrm{P} \subseteq$ opt-P. Dann ist #SAT in opt-P enthalten. Mit Lemma 1.1 folgt daraus, daß die Sprache $\mathit{Graph}_{\leq}(\#\mathrm{SAT})$ in $\mathrm{NP} \cup \text{co-NP}$ liegt. Da PP, NP und co-NP unter many-one Reduktionen abgeschlossen sind und offensichtlich $\mathrm{MAJ} \leq_m^p \mathit{Graph}_{\leq}(\#\mathrm{SAT})$ gilt, folgt daraus $\mathrm{PP} \subseteq \mathrm{NP}$ oder $\mathrm{PP} \subseteq$ co-NP, d.h. NP = PP.
($\Leftarrow$) Ist PP = NP, so ist die bzgl. many-one Reduktionen #P-vollständige Funktion #SAT (vgl. Abschnitt 1.3) in max-P. Da max-P unter many-one Reduktionen abgeschlossen ist (siehe Satz 1.10), folgt $\#\mathrm{P} \subseteq$ max-P. □

Wegen $\#\mathrm{P} \subseteq$ span-P und PP = NP $\Rightarrow$ NP = co-NP erhalten wir aus Satz 1.7 das folgende Korollar.

Korollar 1.3 *Falls* span-P $\subseteq$ opt-P*, so kollabiert* PH *auf die erste Stufe.*

Im Gegensatz zur Aussage von Korollar 1.3 besagt der folgende Satz 1.8, daß max-P und span-P dieselben Funktionen enthalten, soweit die Länge ihrer Funktionswerte logarithmisch in der Argumentlänge beschränkt ist. Wie in Lemma 1.4 gezeigt wird, ist letzteres damit gleichbedeutend, daß die Funktionswerte selbst durch ein Polynom in der Argumentlänge beschränkt sind.

Lemma 1.4 *Sei $\mathcal{F}$ eine Klasse von Funktionen $f : \Sigma^* \to \mathcal{N}_0$. Eine Funktion $f \in \mathcal{F}$ ist genau dann in $\mathcal{F}\langle O(\log n)\rangle$, wenn $f(x) < p(|x|)$ für ein Polynom p ist.*

Beweis. ($\Rightarrow$) Sei $f \in \mathcal{F}\langle O(\log n)\rangle$. D.h. es gibt eine Konstante k mit $|f(x)| \leq k \log(|x|)$ und damit $f(x) < 2^{|f(x)|} \leq 2^{k \log(|x|)} = |x|^k$ für fast alle $x \in \Sigma^*$.
($\Leftarrow$) Sei $f \in \mathcal{F}$ und p ein Polynom mit $f(x) < p(|x|)$ für alle $x \in \Sigma^*$. Dann gilt $|f(x)| = \lfloor \log(f(x) + 1) \rfloor \leq \lfloor \log p(|x|) \rfloor \leq \log(|x|^k) = k \log |x|$ für eine Konstante k und fast alle x in Σ^*. $\square$

Jetzt können wir zeigen, daß jede span-P-Funktion f mit $f(x) \leq p(|x|)$ für ein Polynom p bereits in max-P liegt.

Satz 1.8 max-P$\langle O(\log n)\rangle$ = span-P$\langle O(\log n)\rangle$

Beweis. Sei $span_M \in$ span-P$\langle O(\log n)\rangle$ für einen NP-Transducer M, d.h. nach Lemma 1.4 $span_M(x) \leq q(|x|)$ für ein Polynom q. Dann ist jedoch

$$Graph_{\leq}(span_M) = \{(x, y) \mid y \leq q(|x|) \text{ und es gibt (mindestens) } y \text{ akzeptierende Pfade von } M(x) \text{ mit } y \text{ verschiedenen Ausgaben}\}$$

in NP. Mit Lemma 1.1 folgt daher $span_M \in$ max-P. $\square$

Im folgenden Satz 1.9 zeigen wir, daß sich die Komplexitäten einer exakten Berechnung von span-P- und #NP-Funktionen kaum unterscheiden. Zur Vorbereitung dient folgende überraschende Charakterisierung der #NP-Funktionen.

Lemma 1.5 *Eine Funktion f ist genau dann in* #NP*, wenn es ein* NP*-Prädikat R und ein Polynom p gibt, so daß für alle $x \in \Sigma^*$ gilt:*

$$f(x) = \|\{y \in \Sigma^{p(|x|)} \mid \neg R(x, y)\}\|.$$

Beweis. Sei $f = acc_M^A$ für eine NP-Orakelmaschine M und eine Sprache $A \in$ NP. Zu A gibt es ein P-Prädikat Q und ein Polynom q, so daß $x \in A \Leftrightarrow \exists y\,(|y| = q(|x|)) : Q(x, y)$ für alle $x \in \Sigma^*$ gilt. Wir betrachten zunächst folgendes Prädikat R':

$R'(x, w) \quad :\Longleftrightarrow \quad w = (z, (u_1, v_1), \ldots, (u_k, v_k))$ und
z ist eine akzeptierende Rechnung von M bei Eingabe x, in der der Reihe nach genau die Fragen $u_1, \ldots, u_k$ gestellt werden,
jede Frage u_i, $i = 1, \ldots, k$ wird dabei genau dann mit ja beantwortet, wenn $Q(u_i, v_i)$ und $|v_i| = q(|u_i|)$ gilt; wird eine Frage u_i mit nein beantwortet, so ist $v_i = 1^{q(|u_i|)}$,
$v_1, \ldots, v_k$ sind jeweils die lexikographisch kleinsten Wörter mit dieser Eigenschaft, d.h. für alle $v \in \Sigma^{q(|u_i|)}$ mit $v < v_i$ gilt $\neg Q(u_i, v)$.

Aus der Definition von R' folgt, daß jeder akzeptierenden Rechnung von $M^A(x)$ genau ein Wort w mit $R'(x, w)$ entspricht. Deshalb ist $acc_M^A(x) = \|\{w \mid R'(x, w)\}\|$ für alle $x \in \Sigma^*$. Weiter ist R' ein co-NP-Prädikat und es gibt ein Polynom p mit $|w| < p(|x|)$ für alle x und w, die $R'(x, w)$ erfüllen. Durch „padding“ von R' (vgl. Lemma 1.2) erhalten wir somit das Komplement des gewünschten Prädikats. Die Richtung von rechts nach links ist klar. □

Man beachte die Asymmetrie von NP und co-NP in den Charakterisierungen von span-P (Lemma 1.2) und #NP (Lemma 1.5). Zwar läßt sich wegen span-P $\subseteq$ #NP zu jedem NP-Prädikat R und Polynom p ein co-NP-Prädikat R' und ein Polynom p' finden, so daß für alle $x \in \Sigma^*$ $\|\{y \in \Sigma^{p(|x|)} \mid R(x,y)\}\| = \|\{y \in \Sigma^{p'(|x|)} \mid R'(x,y)\}\|$ ist. Aufgrund von Satz 1.6 gilt die umgekehrte Richtung jedoch nicht (falls NP $\neq$ co-NP). Wir beschließen diesen Abschnitt mit der Feststellung, daß sich jede #NP-Funktion als Differenz einer sehr einfach aufgebauten FP-Funktion und einer span-P-Funktion berechnen läßt.

Satz 1.9 *Zu jeder Funktion f in* #NP *gibt es ein Polynom p und eine* span-P-*Funktion g, so daß $f(x) = 2^{p(|x|)} - g(x)$ ist.*

Beweis. Sei M eine NP-Orakelmaschine und A eine Sprache in NP mit $f = acc_M^A$. Nach Lemma 1.5 gibt es zu M und A ein NP-Prädikat R und ein Polynom p, so daß $f(x) = \|\{y \in \Sigma^{p(|x|)} \mid \neg R(x,y)\}\|$ für alle $x \in \Sigma^*$ gilt. Dann ist $g(x) := \|\{y \in \Sigma^{p(|x|)} \mid R(x,y)\}\|$ nach Lemma 1.2 in span-P, woraus die Behauptung folgt. $\square$

1.3 Reduktionen auf Funktionenklassen

Wir betrachten drei Arten von Reduktionen zwischen Wortfunktionen. Die erste entspricht der many-one Reduktion bei Sprachen und wurde auch von Wagner [Wag86b] verwendet. Angewandt auf #P-Probleme ist sie mit Simon's *parsimonious transformation* identisch [Sim75]. Die zweite ist Krentel's metrische Reduktion [Kre88], die der 1-*tt* Reduktion bei Sprachen entspricht (vgl. Lemma 1.6) und ebenfalls in [Wag86b] betrachtet wurde. Die dritte schließlich ist die Turing-Reduktion, die wir bei der Definition einer bzgl. einer Funktionenklasse $\mathcal{F}$ relativierten Komplexitätsklasse $\mathcal{K}^{\mathcal{F}}$ bereits implizit benutzten. Alle drei Reduktionen sind reflexiv und transitiv. Die Definitionen von $\mathcal{K}$-hart und $\mathcal{K}$-vollständig bzgl. einer bestimmten Reduktion übertragen sich unmittelbar auf Funktionenklassen. Coo71's Beweis der NP-Vollständigkeit von SAT [Coo71] führt geradewegs zur #P-Vollständigkeit von #SAT bzgl. $\leq_m^p$ (vgl. [Sim75]). Im Gegensatz zur Übereinkunft bei Sprachreduktionen vereinbaren wir für Funktionenklassen $\mathcal{F}$, daß eine $\mathcal{F}$-vollständige Funktion bzgl. der Turing-Reduktion ($\leq_T^p$) vollständig für $\mathcal{F}$ ist.

Definition 1.5

a) Eine Funktion g ist auf eine Funktion h many-one reduzierbar *($g \leq_m^p h$), falls es eine Funktion $f \in \mathrm{FP}$ gibt mit $g(x) = h(f(x))$ für alle $x \in \Sigma^*$.*

b) g ist auf h metrisch reduzierbar *($g \leq_{\mathrm{metric}} h$), falls es Funktionen $f_1, f_2 \in \mathrm{FP}$ gibt mit $g(x) = f_2(x, h(f_1(x)))$ für alle $x \in \Sigma^*$.*

c) g ist auf h Turing-reduzierbar *($g \leq_T^p h$), falls $g \in \mathrm{FP}^h$ ist.*

Lemma 1.6 gibt eine äquivalente Definition für metrische Reduzierbarkeit.

Lemma 1.6 *Eine Funktion g ist genau dann auf h metrisch reduzierbar, wenn g in* $\mathrm{FP}^h[1]$ *liegt, d.h. wenn $g(x)$ in Polynomzeit mit einer Orakelfrage an die Funktion h berechnet werden kann.*

Beweis. ($\Rightarrow$) Gilt $g \leq_{metric} h$, so gibt es FP-Funktionen f_1, f_2 mit $g(x) = f_2(x, h(f_1(x)))$. Nun ist $g(x) = f_M^h(x)$ für einen Orakeltransducer M, der bei Eingabe x die Orakelfrage $f_1(x)$ stellt und bei Antwort y $f_2(x, y)$ ausgibt.
($\Leftarrow$) Ist $g(x) = f_M^h(x)$ für einen Orakeltransducer M, der bei jeder Eingabe (o.B.d.A. genau) eine Orakelfrage stellt, so ist $g(x) = f_2(x, h(f_1(x)))$, wobei $f_1(x)$ die von M bei Eingabe x gestellte Orakelfrage und $f_2(x, y)$ die von M bei Eingabe x und Orakelantwort y erzeugte Ausgabe ist. □

In Satz 1.10 untersuchen wir die Abgeschlossenheitseigenschaften der hier betrachteten Funktionenklassen unter den genannten Reduktionen. Eine Funktionenklasse $\mathcal{F}$ ist unter einer Reduktion $\preceq$ *abgeschlossen*, falls mit $f \in \mathcal{F}$ jede Funktion $g \preceq f$ wieder in $\mathcal{F}$ ist. Die Klasse aller Funktionen g, zu denen ein $f \in \mathcal{F}$ mit $g \preceq f$ existiert, heißt der *Abschluß* von $\mathcal{F}$ unter $\preceq$.

Satz 1.10

i) #P, max-P, min-P, span-P *und* #NP *sind unter many-one Reduktionen abgeschlossen.*

ii) $\mathrm{F}\triangle_2^p$ *ist unter Turing-Reduktionen abgeschlossen.*

iii) Zu jeder $\mathrm{F}\triangle_2^p$*-Funktion f gibt es eine* max-P*-Funktion g und eine* FP*-Funktion h mit $f = h \circ g$, d.h.* $\mathrm{F}\triangle_2^p$ *ist der Abschluß von* max-P *unter metrischen Reduktionen. Dasselbe gilt für* min-P *anstatt* max-P.

input: x;
Simuliere $M(x)$, wobei Orakelfragen y_i nichtdeterministisch mit 'ja' ($a_i = 1$) oder 'nein' ($a_i = 0$) beantwortet werden. Falls M verwirft, verwerfe ebenfalls. Andernfalls seien $y_1, \ldots, y_n$ die der Reihe nach gestellten Fragen, $a_1, \ldots, a_n$ die geratenen Antworten und $z = z_1 \ldots z_m$ sei die von M erzeugte Ausgabe.
for $i := 1$ **to** n **do**
  if $a_i = 1$ **then**
    begin
      Simuliere $N(y_i)$ nichtdeterministisch;
      if $N(y_i)$ verwirft **then** stoppe und verwerfe
    end;
output $a_1 \ldots a_n 0^{p(|x|)-n} z_1 \ldots z_m 1 0^{p(|x|)-m}$.

Abbildung 3: Ein NP-Transducer M' mit $f_M^{L(N)} \leq_{metric} max_{M'}$

Beweis.

i) Bei allen Maschinenmodellen kann die Eingabe zunächst einer FP-Transformation unterworfen werden.

ii) Klar.

iii) Sei $f \in \mathrm{F}\Delta_2^p$. Wir konstruieren eine max-P-Funktion g, aus deren Funktionswert $g(x)$ sich $f(x)$ ablesen läßt (vgl. Blass und Gurevich [BG84] und Krentel [Kre88]).
Zu f gibt es ein NP-Orakel A, eine NP-Maschine N mit $A = L(N)$ und einen P-Orakeltransducer M, der f unter Orakel A berechnet. p sei eine polynomielle Zeitschranke für M. Wir konstru-

ieren einen NP-Transducer M', aus dessen maximaler Ausgabe $\max_{M'}(x)$ sich $f(x)$ direkt ablesen läßt (vgl. Abbildung 3). M' rät die Antworten auf M's Orakelfragen, verifiziert die positiven Antworten und gibt eine „gepaddete" Version der geratenen Orakelantwortfolge und berechneten Ausgabe aus. Durch Induktion über $i = 1, \ldots, n$ folgt für die maximale Ausgabe $\max_{M'}(x) = a_1 \ldots a_n 0^{p(|x|)-n} z_1 \ldots z_m 10^{p(|x|)-m}$ von M' bei Eingabe x die Äquivalenz

$$a_i = 1 \iff y_i \in A$$

für $i = 1, \ldots, n$. Damit ist $z = z_1 \ldots z_m$ die von $M^A(x)$ erzeugte Ausgabe. Diese kann leicht durch eine FP-Funktion h aus $g(x) = \max_{M'}(x)$ herausgefiltert werden. □

Im nächsten Satz halten wir fest, daß der Abschluß von #NP und span-P unter metrischen Reduktionen beidesmal derselbe ist. Für span-P$\langle O(\log n)\rangle$ erhalten wir die Inklusion im Abschluß von #P unter metrischen Reduktionen.

Satz 1.11

i) *Jede Funktion aus* #NP *ist metrisch auf eine* span-P-*Funktion reduzierbar.*

ii) *Jede Funktion aus* span-P$\langle O(\log n)\rangle$ *ist metrisch auf eine* #P-*Funktion reduzierbar.*

Beweis.

i) Unmittelbare Folgerung aus Satz 1.9.

ii) Nach Satz 1.8 stimmen die Funktionenklassen max-P$\langle O(\log n)\rangle$ und span-P$\langle O(\log n)\rangle$ überein. Aus dem Beweis von Korollar 1.1 folgt max-P$\langle O(\log n)\rangle \subseteq \mathrm{FP}^{\mathrm{NP}}[O(\log n)]$. Papadimitriou und Zachos [PZ83] haben gezeigt, daß $\mathrm{P}^{\mathrm{NP}}[O(\log n)] \subseteq \mathrm{P}^{\#\mathrm{P}}[1]$ ist. Der in [PZ83] gegebene Beweis läßt sich ohne weiteres auf $\mathrm{FP}^{\mathrm{NP}}[O(\log n)] \subseteq \mathrm{FP}^{\#\mathrm{P}}[1]$ verallgemeinern, womit nach Lemma 1.6 die Behauptung bewiesen ist. $\square$

Wir wenden uns nun der Frage nach der Existenz vollständiger Funktionen in den betrachteten Klassen zu. Wir werden sehen, daß die Klassen span-P$\langle k\rangle, k \geq 1$, span-P$\langle O(\log n)\rangle$ und span-P vollständige Probleme besitzen. Für ein Orakel A bestimmt die Funktion

$$\#_A(x_1, \ldots, x_n) := \sum_{i=1}^{n} \chi_A(x_i)$$

die Anzahl der Wörter $x_i, i = 1, \ldots, n$, die im Orakel A enthalten sind. Durch Einschränkung auf (codierte) k-Tupel für ein festes $k \in \mathcal{N}$ erhalten wir folgende Funktion (vgl. auch [Bei87])

$$\#_A^k(x_1, \ldots, x_k) := \sum_{i=1}^{k} \chi_A(x_i).$$

Satz 1.12

i) $\#_{\mathrm{SAT}}$ *ist* span-P$\langle O(\log n)\rangle$*-vollständig bzgl.* $\leq_m^p$.

ii) $\#_{\mathrm{SAT}}^{2^k-1}$ *ist* span-P$\langle k\rangle$*-vollständig bzgl.* $\leq_m^p$.

Beweis.

i) $\#_{\mathrm{SAT}}$ ist in span-P via einem NP-Transducer M, der bei Eingabe $(x_1, \ldots, x_n)$ nichtdeterministisch eine Zahl $i \in \{1, \ldots, n\}$ und eine

Belegung h für x_i wählt und im Fall $h(x_i) = 1$ den Wert i ausgibt. Wegen

$$|\#_{\mathrm{SAT}}(x_1, \ldots, x_n)| \leq |n| \leq \log(|(x_1, \ldots, x_n)|)$$

ist damit $\#_{\mathrm{SAT}}$ in span-P$\langle O(\log n)\rangle$. Ist nun $g \in$ span-P$\langle O(\log n)\rangle$ via M, d.h. $g(x) = span_M(x) \leq c \log(|x|)$ für eine Konstante c und (fast) alle $x \in \Sigma^*$, dann ist

$$g(x) = \#_{\mathrm{SAT}}(h(x, 0^1), \ldots, h(x, 0^{c \log(|x|)})),$$

wobei h die NP-Sprache

$$A := \{(x, 0^i) \mid span_M(x) \geq i\}$$

many-one auf SAT reduziert.

ii) Vollkommen analog zu *i*). □

Da nach Satz 1.8 die beiden Funktionenklassen span-P$\langle O(\log n)\rangle$ und max-P$\langle O(\log n)\rangle$ übereinstimmen, übertragen sich sämtliche Vollständigkeitsresultate für max-P$\langle O(\log n)\rangle$ [Kre88] auf span-P$\langle O(\log n)\rangle$.

Aus [Val79a] stammt das folgende Berechnungsproblem #NSAT: Gegeben ist eine aussagenlogische Formel $\alpha(u_1, \ldots, u_n, v_1, \ldots, v_m)$ in den Variablen $u_1, \ldots, u_n, v_1, \ldots, v_m$. #NSAT($\alpha$) ist die Anzahl von Belegungen h_u für die u-Variablen, so daß $\alpha(h_u(u_1), \ldots, h_u(u_n), v_1, \ldots, v_m)$ eine erfüllbare Formel ist. Valiant führte #NSAT als Beispiel einer Turing-vollständigen Funktion in #NP ein. Wir zeigen in Satz 1.13, daß #NSAT many-one vollständig für span-P ist. Daraus ergibt sich mit Satz 1.11, daß #NSAT auch #NP-vollständig bzgl. metrischen Reduktionen ist (Korollar 1.4).

Satz 1.13 #NSAT *ist* span-P-*vollständig bzgl.* $\leq_m^p$.

Beweis. #NSAT liegt wegen

$$\#\text{NSAT}(\alpha) = \|\{h_u \mid \alpha(h_u(u_1), \ldots, h_u(u_n), v_1, \ldots, v_m) \in \text{SAT}\}\|$$

nach Lemma 1.2 in span-P. Dabei ist $\alpha = \alpha(u_1, \ldots, u_n, v_1, \ldots, v_m)$ und $h_u : \{u_1, \ldots, u_n\} \to \{0, 1\}$. Sei nun $f = span_M$ für einen NP-Transducer M. Zu M und $x \in \Sigma^*$ läßt sich in Polynomzeit eine aussagenlogische Formel $\alpha_x(u_1, \ldots, u_n, v_1, \ldots, v_m)$ mit folgenden Eigenschaften erstellen: Jede Belegung h_u der u-Variablen entspricht eineindeutig einer potentiellen Ausgabe von $M(x)$ und jeder Berechnungspfad von M bei Eingabe x entspricht (mindestens) einer Belegung h_v der v-Variablen. α wird bei einer Belegung (h_u, h_v) genau dann wahr, wenn $M(x)$ auf dem der Belegung h_v entsprechenden Pfad die der Belegung h_u entsprechende Ausgabe erzeugt. Daraus ergibt sich $f(x) = \#\text{NSAT}(\alpha_x)$ für alle $x \in \Sigma^*$, d.h. $f \leq_m^p$ #NSAT. $\square$

Korollar 1.4 #NSAT *ist metrisch vollständig für* #NP.

Obwohl sich nach Satz 1.13 und Korollar 1.4 die Komplexitäten einer exakten Berechnung von span-P- und #NP-Funktionen nicht unterscheiden, scheint dies für eine näherungsweise Berechnung nicht zuzutreffen. Zwar gelingt es uns im nächsten Abschnitt, sowohl deterministische (mit einem Σ_2^p-Orakel) als auch probabilistische (mit einem NP-Orakel) Algorithmen zu konstruieren, die span-P-Funktionen mit beliebig kleinem relativen Fehler approximieren. Doch konnten wir für #NP-Funktionen keine solchen Algorithmen finden, ohne eine höhere Komplexität für das Sprachorakel in Kauf zu nehmen.

2 Approximative Berechnung von span-P-Funktionen

2.1 Einleitung

Wie wir im Anschluß an Korollar 1.2 bereits ausgeführt haben, ist eine exakte Berechnung aller #P-Funktionen in Polynomzeit unter Befragung eines PH-Orakels nur im Falle eines Kollapses der polynomiellen Hierarchie möglich. In [Sto85] wurde gezeigt, daß #P-Funktionen f für jedes Polynom p durch eine unter einem Σ_2^p-Orakel in Polynomzeit berechenbare Funktion g mit einem von der Argumentlänge $|x|$ abhängigen relativen Fehler kleiner $1/p(|x|)$ approximiert werden können, d.h. es gilt für alle $x \in \Sigma^*$ mit $f(x) > 0$

$$\frac{|g(x) - f(x)|}{f(x)} < 1/p(|x|)$$

und $g(x) = f(x)$, falls $f(x) = 0$. Dabei kann angenommen werden, daß f von g nicht unterschätzt wird, d.h. es gilt sogar

$$0 \leq \frac{g(x) - f(x)}{f(x)} < 1/p(|x|)$$

für alle $x \in \Sigma^*$ mit $f(x) > 0$. Wird nämlich f durch eine Funktion g' mit relativem Fehler kleiner $1/(2p+1)$ approximiert, so ist $g = (p/(p-1))g'$ eine Approximation mit relativem Fehler kleiner $1/p$ und es gilt $f \leq g$. Eine Funktion $g \geq f$ mit $g(x) - f(x) \leq e(|x|)f(x)$ nennen wir eine *e-Approximation* an f.

In [JVV86] wurde gezeigt, daß zur Approximation von #P-Funktionen durch probabilistische Algorithmen bereits ein NP-Orakel ausreicht. Dabei kann die gewünschte Approximationsgüte über die Eingabe gesteuert werden: Für jede #P-Funktion f gibt es einen probabilistischen Orakeltransducer M und ein NP-Orakel A, so daß M^A bei

Eingabe von $(x, 0^n, m)$ mit Wahrscheinlichkeit größer $1-1/m$ eine Ausgabe mit relativem Fehler kleiner $1/n$ erzeugt.

Wir erweitern in diesem Abschnitt die genannten Resultate auf span-P-Funktionen. Die Anzahl der Orakelfragen bei der Berechnung eines Näherungswertes von $f(x)$ begrenzen wir dabei durch Anwendung binärer Suche auf $O(\log(|x|))$ (vgl. auch [Pio88]). Des weiteren zeigen wir, daß auch bei der probabilistischen Approximation M so konstruiert werden kann, daß M niemals zu kleine Werte ausgibt.

2.2 Deterministische Approximation unter einem Σ_2^p-Orakel

Die Beweise der Hauptresultate in diesem Kapitel beruhen auf universellem Hashing [CW79]. Beim universellen Hashing wählt der Hashalgorithmus zu einer Schlüsselmenge $S \subseteq A$ zufällig eine Hashfunktion $H : A \to B$ aus einer Klasse von Hashfunktionen $\mathcal{H} \subseteq B^A$ aus. Die Hashfunktion H bestimmt für jeden Schlüssel $s \in S$ eine Adresse $H(s)$, unter der ein mit dem Schlüssel s versehener Datensatz abgespeichert oder aufgefunden werden kann (vgl. z. B. [Meh86]).

Definition 2.1 *Seien A, B endliche Mengen, $S \subseteq A$.*

a) Eine Hashfunktion $H : A \to B$ ist kollisionsfrei *auf S, falls H auf S injektiv ist, d.h. es gilt $H(x) \neq H(y)$ für alle $x, y \in S$ mit $x \neq y$.*

b) Eine Kollektion $H_1, \ldots, H_k$ von Hashfunktionen ist kollisionsfrei *auf S, falls gilt:*

$$\forall x \in S \,\exists i\, (1 \leq i \leq k) \forall y \in S : (x \neq y \Rightarrow H_i(x) \neq H_i(y)).$$

Kollisionsfreie Kollektionen kann es nur für Schlüsselmengen S geben, die nicht zu groß sind.

Lemma 2.1

i) *Ist $H : A \to B$ kollisionsfrei auf $S \subseteq A$, so gilt $\|S\| \leq \|B\|$.*

ii) *Ist eine Kollektion von Hashfunktionen $H_1, \ldots, H_k$ kollisionsfrei auf $S \subseteq A$, so gilt $\|S\| \leq k\,\|B\|$.*

Beweis.

i) Klar.

ii) Nach Voraussetzung existiert für jedes $x \in S$ die Zahl

$$m(x) := \min\{1 \leq i \leq k \mid \forall y \in S : (y \neq x \Rightarrow H_i(x) \neq H_i(y))\}.$$

Dann ist aber folgende Funktion $H : S \to B \times \{1, \ldots, k\}$ kollisionsfrei auf S

$$H(x) := (H_{m(x)}(x), m(x)).$$

Mit *i*) folgt daher $\|S\| \leq k\,\|B\|$. □

Wir beschränken uns im folgenden auf lineare Hashfunktionen von $A = \Sigma^m$ in $B = \Sigma^n$.

Beispiel 2.1 Lineare Hashfunktionen H von Σ^m in Σ^n stellen wir durch Boolesche $m \times n$-Matrizen dar:

$$H(x) = y, \text{ wobei } \begin{pmatrix} h_{11} & \ldots & h_{1m} \\ \vdots & \ddots & \vdots \\ h_{n1} & \ldots & h_{nm} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_m \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix},$$

also $y_i = \sum_{j=1}^{m} h_{ij} x_j$ ist. Summe und Produkt werden dabei in $GF[2]$, d.h. modulo 2 gebildet. Mit der bitweisen Addition (ohne Übertrag) $\oplus$ als Verknüpfung bildet $(\Sigma^k, \oplus)$ für jedes $k \in \mathcal{N}$ eine Gruppe (jedes

Element ist dabei selbstinvers) und es gilt $H(x \oplus y) = H(x) \oplus H(y)$ für alle $x, y \in \Sigma^m$. Fassen wir alle linearen Hashfunktionen von Σ^m in Σ^n in der Menge $\mathcal{L}_{m,n}$ zusammen, dann bildet $(\mathcal{L}_{m,n}, \oplus)$ wieder eine Gruppe, wobei $H_1 \oplus H_2$ durch $(H_1 \oplus H_2)(x) := H_1(x) \oplus H_2(x)$ für alle $x \in \Sigma^m$ definiert ist.

Das nächste Lemma stammt aus [Sip83]. Darin wird die Wahrscheinlichkeit, bei zufälliger Auswahl eine kollisionsfreie Kollektion von linearen Hashfunktionen für S zu erhalten, in Abhängigkeit von der Größe von S nach unten abgeschätzt.

Lemma 2.2 [Sip83] *Sei $S \subseteq \Sigma^m$ gegeben. Wird jede Kollektion $H_1, \ldots, H_k$ linearer Hashfunktionen von Σ^m in Σ^n mit derselben Wahrscheinlichkeit (2^{-mnk}) gewählt, so beträgt die Wahrscheinlichkeit, eine auf S kollisionsfreie Kollektion zu erhalten, mindestens $1 - 2^{-nk}\|S\|^{k+1}$.*

Beweis. Wir zeigen, daß es höchstens $2^{(m-1)nk}\|S\|^{k+1}$ Kollektionen $H_1, \ldots, H_k$ von linearen Hashfunktionen von Σ^m in Σ^n gibt, die auf S nicht kollisionsfrei sind.

Zu $0^m \neq x \in \Sigma^m$ gibt es 2^{m-1} lineare Hashfunktionen h von Σ^m in Σ mit $h(x) = 0$ und ebenso viele mit $h(x) = 1$, da die Abbildung $h \mapsto h(x)$ wegen $x \neq 0^m$ ein nichttrivialer Gruppenhomomorphismus von $(\mathcal{L}_{m,1}, \oplus)$ auf $(\Sigma, \oplus)$ ist.

Daher gibt es $2^{(m-1)n}$ lineare Hashfunktionen $H = (h_1, \ldots, h_n)$ von Σ^m in Σ^n mit $H(x) = (h_1(x), \ldots, h_n(x)) = 0^n$.

Wegen $H(x) = H(y) \Leftrightarrow H(x \oplus y) = 0^n$ folgt damit, daß zu $x \neq y \in \Sigma^m$ genau $2^{(m-1)n}$ lineare Hashfunktionen H von Σ^m in Σ^n existieren mit $H(x) = H(y)$.

D.h. für festes $x \in S$ gibt es weniger als $2^{(m-1)n}\|S\|$ Hashfunktionen H, so daß $H(x) = H(y)$ für ein von x verschiedenes $y \in S$ gilt.

Also existieren für festes $x \in S$ höchstens $2^{(m-1)nk}\|S\|^k$ Kollektionen

$H_1, \ldots, H_k$, so daß es für jedes $i, 1 \leq i \leq k$, ein von x verschiedenes $y \in S$ mit $H_i(x) = H_i(y)$ gibt.
Damit gibt es höchstens $\|S\| \cdot \|S\|^k \cdot 2^{(m-1)nk}$ Kollektionen, die auf S nicht kollisionsfrei sind. $\square$

Eine unmittelbare Folgerung von Lemma 2.2 ist, daß für hinreichend kleines S kollisionsfreie Kollektionen von linearen Hashfunktionen existieren müssen.

Korollar 2.1 *Zu jeder Teilmenge S von Σ^m mit $\|S\| \leq 2^{k-1}, k \geq 1$, gibt es eine auf S kollisionsfreie Kollektion von linearen Hashfunktionen $H_1, \ldots, H_k$ von Σ^m in Σ^k.*

Beweis. Wir zeigen, daß unter den genannten Voraussetzungen die Wahrscheinlichkeit p_{koll} für eine Kollision kleiner als 1 ist. Wir wenden Lemma 2.2 mit $k = n$ an und erhalten

$$\begin{aligned} p_{koll} &\leq 2^{-k^2} \|S\|^{k+1} \\ &\leq 2^{-k^2} 2^{(k-1)(k+1)} \\ &= 2^{-1}. \end{aligned}$$

Damit ist sogar mindestens die Hälfte aller Kollektionen von linearen Hashfunktionen von Σ^m in Σ^k kollisionsfrei auf S. $\square$

Mit Lemma 2.1 *ii)* und Korollar 2.1 gelingt nun eine erste grobe Größenabschätzung einer endlichen Menge $S_x \subseteq \Sigma^m$. Gibt es nämlich eine auf S_x kollisionsfreie Kollektion von linearen Hashfunktionen $H_1, \ldots, H_k$ von Σ^m in Σ^k, so folgt mit Lemma 2.1 *ii)* $\|S_x\| \leq k2^k$. Existiert diese jedoch nicht, so ist $\|S_x\|$ nach Korollar 2.1 größer als 2^{k-1}. Einen groben Schätzwert für $\|S_x\|$ bildet demnach der Wert $k_{\min} 2^{k_{\min}}$, wobei $k_{\min}$ das kleinste k ist, für das eine kollisionsfreie Kollektion der oben beschriebenen Art existiert. Für die hier vorliegende Anwendung liegt $k_{\min}$ im Bereich $0, \ldots, m+1$ und m ist polynomiell in $|x|$ beschränkt. Zur Bestimmung von $k_{\min}$ durch binäre Suche muß daher nur $O(\log |x|)$-mal die Frage nach der Existenz der oben beschriebenen Kollektion gestellt werden (vgl. auch [Pio88]). Diese Überlegungen führen auf den folgenden Satz 2.1. Dabei ist $\mathrm{F}\triangle_3^p[O(\log n)] = \mathrm{FP}^{\Sigma_2^p}[O(\log n)]$ die Klasse aller in Polynomzeit mit $O(\log |x|)$ Fragen an ein Σ_2^p-Orakel berechenbaren Funktionen.

Satz 2.1 *Jede* span-P-*Funktion f läßt sich mit einem relativen Fehler $O(|f|)$ durch eine* $\mathrm{F}\triangle_3^p[O(\log n)]$-*Funktion g approximieren, die f nicht unterschätzt.*

Beweis. Nach Lemma 1.2 gibt es zu f ein NP-Prädikat R und ein Polynom p, so daß für alle $x \in \Sigma^*$ $f(x) = \|\{y \in \Sigma^{p(|x|)} \mid R(x,y)\}\|$ gilt. Es ist also für jedes x die Größe der Menge

$$S_x := \{y \in \Sigma^{p(|x|)} \mid R(x,y)\}$$

abzuschätzen. Wir definieren ein NP-Prädikat KOLLISION, das auf ein Tupel $(x, H_1, \ldots, H_k)$, $k \geq 0$ genau dann zutrifft, wenn $H_1, \ldots, H_k$ eine auf S_x nicht kollisionsfreie Kollektion von linearen Hashfunktionen von $\Sigma^{p(|x|)}$ in Σ^k ist.

$$\begin{aligned} &\mathrm{KOLLISION}(x, H_1, \ldots, H_k) \\ &\quad :\Longleftrightarrow \exists x \in S_x \, \forall i \, (1 \leq i \leq k) \, \exists y \in S_x : (x \neq y \wedge H_i(x) = H_i(y)) \end{aligned}$$

Man beachte, daß im Fall $k = 0$ das Prädikat KOLLISION(x) genau dann erfüllt ist, wenn S_x nicht leer ist. Wegen KOLLISION $\in$ NP liegt das folgende Prädikat HASH $\subseteq \Sigma^* \times \mathcal{N}_0$ in Σ_2^p, das für ein Paar (x,k) die Existenz einer auf S_x kollisionsfreien Kollektion von linearen Hashfunktionen $H_1, \ldots, H_k$ von $\Sigma^{p(|x|)}$ in Σ^k behauptet:

$$\mathrm{HASH}(x,k) \;:\Longleftrightarrow\; \exists H_1, \ldots, H_k : \neg\mathrm{KOLLISION}(x, H_1, \ldots, H_k).$$

Die Strategie der $\mathrm{F}\triangle_3^p$-Maschine M, die die Approximation g berechnet, ist einfach (vgl. Abbildung 4). M bestimmt bei Eingabe x durch

---

input: x;
links $:= 0$;
rechts $:= p(|x|) + 1$;
$k := \lfloor rechts/2 \rfloor$;
repeat
  if HASH(x, k) **then**
    rechts $:= k$
  else
    links $:= k + 1$;
  $k := \lfloor (links + rechts)/2 \rfloor$
until *links* $= k =$ *rechts*;
output $k2^k$.

Abbildung 4: Eine $\mathrm{F}\triangle_3^p[O(\log n)]$-Approximationsmaschine M für eine span-P-Funktion

---

binäre Suche das kleinste k, für das HASH(x, k) erfüllt ist, und gibt die Approximation $g(x) = k2^k$ aus.

Korrektheit von M: Wegen $\|S_x\| \leq 2^{p(|x|)}$ ist nach Korollar 2.1 HASH$(x, p(|x|) + 1)$ erfüllt. Für die repeat-Schleife gilt somit die Invariante

$$\mathrm{HASH}(x, \mathit{rechts}) \ \textbf{and not} \ \mathrm{HASH}(x, \mathit{links} - 1).$$

Nach Beendigung der repeat-Schleife ist also HASH(x, k), aber nicht HASH$(x, k - 1)$ erfüllt, d.h. $k = k_{\min}$. Im Fall $f(x) = 0$, d.h. $S_x = \emptyset$ ist $k_{\min} = 0$ und es wird die korrekte Ausgabe $g(x) = 0 \cdot 2^0 = 0$ erzeugt. Für den Fall $f(x) > 0$, d.h. $k_{\min} > 0$ erhalten wir unter Verwendung von Lemma 2.1 *ii*) und Korollar 2.1 die Abschätzung

$$\frac{g(x)}{4k_{\min}} = 2^{k_{\min}-2} < f(x) \le k_{\min}2^{k_{\min}} = g(x),$$

was einen relativen Fehler kleiner als $4k_{\min} - 1 = O(|f(x)|)$ bedeutet.

Laufzeitbetrachtung:

Die repeat-Schleife wird höchstens $\lceil \log(p(|x|) + 2) \rceil$-mal durchlaufen, da die Differenz *rechts* $-$ *links* bei jedem Schleifendurchlauf mindestens halbiert wird. Damit werden auch nur $O(\log |x|)$ Fragen an das Orakel HASH gestellt. □

Satz 2.1 erlaubt es, eine span-P-Funktion f mit $|f(x)| \le q(|x|)$ durch eine Funktion $g \in \mathrm{F}\triangle_3^p[O(\log n)]$ mit relativem Fehler $O(q)$ zu approximieren. Die gewünschte Approximationsgüte von $1/p$ für ein beliebiges Polynom p erreichen wir dadurch, daß wir den Approximationsalgorithmus nicht direkt auf f, sondern auf eine Funktion der Bauart $f(x)^{r(|x|)}$ ansetzen (vgl. Satz 2.2). Die Funktion r muß dabei so groß gewählt werden, daß aus einer $O(r \cdot q)$-Approximation $g(x)$ an $f(x)^{r(|x|)}$ eine $1/p$-Approximation $g'(x) = g(x)^{1/r(|x|)}$ an $f(x)$ wird. Im folgenden Lemma 2.3 bestimmen wir die hierfür ausreichende Größenordnung von r. Wir nehmen dabei der Einfachheit halber an, daß $p = q$ ist, was durch Vergrößern der beiden Polynome immer erreicht werden kann.

Lemma 2.3 *Sei q ein Polynom. Dann existiert ein Polynom r, so daß für alle Funktionen $f, g : \Sigma^* \to \mathcal{N}_0$ gilt: Ist $g(x)$ eine $q\cdot r$-Approximation an $f(x)$, so ist $g(x)^{1/r(|x|)}$ eine $1/q$-Approximation an $f(x)^{1/r(|x|)}$.*

Beweis. Wir müssen r so wählen, daß

$$\frac{g^{1/r(|x|)}(x) - f^{1/r(|x|)}(x)}{f^{1/r(|x|)}(x)} \le \frac{1}{q(|x|)}$$

für alle $x \in \Sigma^*$ mit $f(x) > 0$ gilt. Durch Äquivalenzumformungen erhalten wir die Ungleichung

$$(1 - \frac{1}{q(|x|)+1})^{r(|x|)} g(x) \leq f(x).$$

Da nach Voraussetzung g eine $q \cdot r$-Approximation an f ist, genügt es, die Ungleichung

$$(1 - \frac{1}{q+1})^{r} \leq \frac{1}{qr}$$

zu erfüllen. Nach dem Binomischen Lehrsatz ist

$$\begin{aligned}(1 - \frac{1}{q+1})^{-r} &= \sum_{i=0}^{\infty} \binom{-r}{i} \left(\frac{-1}{q+1}\right)^{i} \\ &= 1 + \frac{r}{q+1} + \frac{r(r+1)}{2(q+1)^2} + \ldots,\end{aligned}$$

d.h. es genügt, für r ein Polynom zu wählen, das einen um eins höheren Grad als q hat. □

Da nach Lemma 2.3 für r ein geeignetes Polynom gefunden werden kann, ist $f(x)^{r(|x|)}$ in span-P und kann daher in $\mathrm{F}\triangle_3^p[O(\log n)]$ approximiert werden. Satz 2.2 faßt die Arbeitsweise der deterministischen Approximationsmaschine zusammen.

Satz 2.2 *Zu jedem Polynom p und zu jeder* span-P*-Funktion f existiert eine $1/p$-Approximation g in $\mathrm{F}\triangle_3^p[O(\log n)]$, d.h. g erfüllt für alle $x \in \Sigma^*$ die Ungleichung*

$$0 \leq g(x) - f(x) \leq \frac{f(x)}{p(|x|)}.$$

Beweis. Sei $q \geq p$ ein Polynom mit $|f(x)| \leq q(|x|)$ für alle $x \in \Sigma^*$. Wir konstruieren eine $1/q$-Approximation an f. Mit f liegt für jedes Polynom r auch die Funktion

$$f'(x) := f^{r(|x|)}(x)$$

in span-P. Unter Beachtung von $|f'(x)| \leq r(|x|)q(|x|)$ kann nach Satz 2.1 zu f' eine $O(r \cdot q)$-Approximation $g' \in \mathrm{F}\triangle_3^p[O(\log n)]$ angegeben werden. Nach Lemma 2.3 kann das Polynom r so gewählt werden, daß $g^*(x) := g'(x)^{1/r(|x|)}$ eine $1/q$-Approximation an $f'(x)^{1/r(|x|)} = f(x)$ ist. Mit g' ist auch

$$g := \lfloor g^*(x) \rfloor = \lfloor g'(x)^{1/r(|x|)} \rfloor$$

in $\mathrm{F}\triangle_3^p[O(\log n)]$ und da $f(x)$ ganzzahlig ist, ist mit g^* auch g eine $1/q$-Approximation an f. □

Manchmal erscheint es sinnvoller, die gewünschte Approximationsgüte nicht schon fest in die Approximationsmaschine einzubauen, sondern über die Eingabe zu steuern.

Korollar 2.2 *Zu jeder* span-P-*Funktion f existiert eine Funktion g in* $\mathrm{F}\triangle_3^p[O(\log n)]$ *mit*

$$0 \leq g(x, 0^n) - f(x) \leq \frac{f(x)}{n}$$

für alle $x \in \Sigma^*$.

Beweis. Sei $f'(x,y) := f(x)$. Für die zu $f' \in$ span-P nach Satz 2.2 existierende $1/n$-Approximation g in $\mathrm{F}\triangle_3^p[O(\log n)]$ gilt dann

$$0 \leq \frac{g(x,0^n) - f'(x,0^n)}{f'(x,0^n)} = \frac{g(x,0^n) - f(x)}{f(x)} \leq \frac{1}{|(x,0^n)|} \leq \frac{1}{n}.$$

□

Man beachte, daß der Wert n, der die gewünschte Approximationsgüte $1/n$ steuert, unär eingegeben werden muß, um eine polynomiell beschränkte Laufzeit zu erhalten.

Stockmeyer [Sto85] hat gezeigt, daß ein rekursives Orakel A und eine Funktion $f \in \#\mathrm{P}^A$ existieren, so daß f von keiner $\mathrm{F}\Delta_2^{p,A}$-Funktion mit relativem Fehler $O(|f|)$ approximiert werden kann. Satz 2.2 und Korollar 2.2 können daher mit relativierbaren Techniken nicht in der Weise verbessert werden, daß für jede span-P-Funktion eine Approximationsfunktion in $\mathrm{F}\Delta_2^p$ gefunden werden kann.

2.3 Probabilistische Approximation unter einem NP-Orakel

Wir wenden uns nun der probabilistischen Approximation einer span-P-Funktion f zu. Dabei ist die vom Approximationsalgorithmus bei Eingabe x erzeugte Ausgabe $g(x)$ eine Zufallsvariable, die mit hoher Wahrscheinlichkeit in der Nähe von $f(x)$ liegen soll. Betrachten wir noch einmal die Approximationsmaschine M in Abbildung 4. M verwendet das Prädikat HASH, um die binäre Suche nach $k_{\min}$, dem kleinsten k mit $\mathrm{HASH}(x,k)$, zu steuern. Dabei steht $\mathrm{HASH}(x,k)$ für die Aussage, daß eine auf $S_x \subseteq \Sigma^{p(|x|)}$ kollisionsfreie Kollektion $H_1,\ldots,H_k$ von linearen Hashfunktionen nach Σ^k existiert.

Ein probabilistischer Algorithmus ist in der Lage, bei der binären Suche nach $k_{\min}$ eine zufällige Kollektion linearer Hashfunktionen zu generieren. M's Orakelfragen an HASH können daher durch eine probabilistische Erzeugung einer Kollektion $H_1,\ldots,H_k$ linearer Hashfunktionen und eine anschließende Orakelfrage an das NP-Orakel KOLLISION ersetzt werden (vgl. Abbildung 5). Durch die Orakelantwort erfährt M', ob die zufällig gewählte Kollektion $H_1,\ldots,H_k$ zu einer Kollision auf S_x führt. Bei einer negativen Antwort ist das Prädikat $\mathrm{HASH}(x,k)$ offensichtlich erfüllt und M' verhält sich wie M bei einer Bejahung der Orakelfrage $\mathrm{HASH}(x,k)$. Aus einer positiven Antwort kann zwar nicht die Gültigkeit von $\neg\mathrm{HASH}(x,k)$ gefolgert werden, da ja die Existenz anderer, kollisionsfreier Kollektionen nicht ausgeschlossen werden kann. Da jedoch die Wahrscheinlichkeit, eine kollisionsfreie Kollektion $H_1, \ldots, H_k$ auf S_x zu erhalten, für k größer als $\log(\|S_x\|)$ sehr schnell gegen 1 strebt, kann M' in diesem Fall von der Gültigkeit von $\neg\mathrm{HASH}(x,k)$ ausgehen, ohne daß dies zu einem allzu großen Fehler bei der erzeugten Ausgabe führt. Wir werden sehen, daß sich das von M' bestimmte k mit hoher Wahrscheinlichkeit in der Nähe von k_{min} befindet, d.h. M' gibt mit hoher Wahrscheinlichkeit eine gute Näherung aus.

Satz 2.3 *Für jede* span-P-*Funktion f gibt es einen polynomiell zeitbeschränkten probabilistischen Orakeltransducer M und ein* NP-*Orakel A, so daß für die Ausgabe $g(x, 0^n)$ von M^A bei Eingabe $(x, 0^n)$ mit Wahrscheinlichkeit größer $1 - 1/f(x)$ im Fall $f(x) > 0$ und mit Wahrscheinlichkeit 1 im Fall $f(x) = 0$*

$$0 \leq g(x, 0^n) - f(x) \leq f(x)/n$$

gilt. Dabei stellt M nur logarithmisch viele Fragen an das Orakel A und gibt auf keinem Pfad einen zu kleinen Wert aus, d.h. die Wahrscheinlichkeit für $g(x, 0^n) \geq f(x)$ ist 1.

Beweis. Wir übernehmen die Bezeichnungsweise vom Beweis zu Satz 2.1. Insbesondere sei

$$f(x) = \|\underbrace{\{y \in \Sigma^{p(|x|)} \mid R(x,y)\}}_{S_x}\|$$

für ein Polynom p und ein NP-Prädikat R. Wir betrachten die in Abbildung 5 dargestellte probabilistische Maschine M'. Auf jedem Pfad gilt für die repeat-Schleife die Invariante HASH(*rechts*). Nach Beendigung der repeat-Schleife ist also HASH(k) erfüllt und damit gilt wegen

input: x;
links $:= 0$;
rechts $:= p(|x|) + 1$;
$k := \lfloor \mathit{rechts}/2 \rfloor$;
repeat
  choose randomly
    lineare Hashfunktionen $H_1, \ldots, H_k : \Sigma^{p(|x|)} \to \Sigma^k$;
  if KOLLISION$(x, H_1 \ldots, H_k)$ **then**
    links $:= k + 1$
  else
    rechts $:= k$;
  $k := \lfloor (\mathit{links} + \mathit{rechts})/2 \rfloor$
until *links* $= k =$ *rechts*;
output $k2^k$.

Abbildung 5: Ein probabilistischer Transducer M', der eine span-P-Funktion unter einem NP-Orakel approximiert

Lemma 2.1 *ii)* die Ungleichung

$$f(x) \leq k2^k = g(x).$$

Ist $f(x) = 0$, so kommt es auf jedem Pfad nur im letzten Schleifendurchlauf ($k = 0$) zu einer Kollision, d.h. es wird mit Wahrscheinlichkeit 1 der richtige Wert 0 ausgegeben. Wir betrachten jetzt den Fall $f(x) > 0$. Als erstes schätzen wir die Wahrscheinlichkeit ab, daß die zu Beginn der repeat-Schleife geratenen Hashfunktionen zu einer Kollision führen. Sei hierzu k_S das kleinste k mit $\|S_x\| \leq 2^k$. Für $k > 0$ ist nach Lemma 2.2 die Wahrscheinlichkeit dafür, eine auf S_x nicht kollisionsfreie Kollektion von linearen Hashfunktionen $H_1, \ldots, H_k : \Sigma^{p(|x|)} \to \Sigma^k$ zu erhalten, höchstens

$$\begin{aligned} 2^{-k^2}\|S_x\|^{k+1} &\leq 2^{-k^2+k_S(k+1)} \\ &= 2^{(k+1)(k_S-k)+k}. \end{aligned}$$

Das in der repeat-Schleife ermittelte k kann nur dann größer als k_S+2 sein, wenn für ein $k \geq k_S+2$ zu Beginn der repeat-Schleife eine auf S_x nicht kollisionsfreie Kollektion $H_1, \ldots, H_k$ geraten wird. Die Wahrscheinlichkeit hierfür ist höchstens

$$\sum_{k=k_S+2}^{p(|x|)} 2^{(k+1)(k_S-k)+k} < 2^{-k_S-3} < \frac{1}{f(x)}.$$

Die relative Fehlerabweichung der Ausgabe von $M'(x)$ mit Orakel KOLLISION ist also mit einer Wahrscheinlichkeit größer $1-1/f(x)$ kleiner als $8(k_S+2)-1 = O(|f(x)|)$. Aus M' läßt sich nun genau wie im deterministischen Fall unter Verwendung von Lemma 2.3 eine probabilistische Maschine M konstruieren, deren Fehlerwahrscheinlichkeit nicht größer ist, und die bei Eingabe $(x, 0^n)$ einen relativen Fehler kleiner $1/n$ macht, d.h. für die Ausgabe $g(x, 0^n)$ von M gilt $g(x, 0^n) \geq f(x)$ und

$$P[g(x,0^n) - f(x) \leq f(x)/n] > 1 - 1/f(x)$$

im Fall $f(x) > 0$ und $g(x, 0^n) = f(x)$ im Fall $f(x) = 0$. $\square$

Es stellt sich nun die Frage, ob sich Satz 2.3 auch für #NP-Funktionen beweisen läßt. Die Tatsache, daß jede #NP-Funktion h als Differenz $h(x) = 2^{p(|x|)} - f(x)$ für ein Polynom p und eine span-P-Funktion f dargestellt werden kann (vgl. Satz 1.9), hilft hier nicht weiter, da ein kleiner relativer Fehler für $f(x)$ einen großen relativen Fehler für $h(x)$ bedeuten kann. Es ist vielmehr unwahrscheinlich, daß Satz 2.3 auch für #NP-Funktionen gilt: Die probabilistische Orakelmaschine aus

Satz 2.3 gibt im Fall $f(x) = 0$ auf jedem Pfad den korrekten Wert 0 aus. Zudem unterschätzt sie $f(x)$ nie, d.h. sie gibt im Fall $f(x) > 0$ auf jedem Pfad einen positiven Wert aus. Daher würde aus der Gültigkeit von Satz 2.3 für #NP-Funktionen sofort der Kollaps von PH auf $\triangle_2^p$ folgen.

Zum Abschluß dieses Kapitels geben wir eine Fassung von Satz 2.3, bei der die Fehlerwahrscheinlichkeit nicht an den Funktionswert $f(x)$ gekoppelt ist, sondern wie die gewünschte maximale relative Fehlerabweichung über die Eingabe gesteuert werden kann. Im Vergleich zur relativen Fehlerabweichung, die nur linear mit der Länge der Eingabe abnimmt, nimmt die maximale Fehlerwahrscheinlichkeit sogar exponentiell mit der Eingabelänge ab.

Satz 2.4 *Jede* span-P-*Funktion f läßt sich durch einen in Polynomzeit arbeitenden probabilistische Orakeltransducer M unter einem* NP-*Orakel A in der Art approximieren, daß für die Ausgabe $g(x, 0^n, m)$ von M^A bei Eingabe $(x, 0^n, m)$*

$$P[g(x, 0^n, m) - f(x) \leq f(x)/n] > 1/m$$

gilt. Dabei stellt M nur logarithmisch viele Fragen an das Orakel A und gibt auf keinem Pfad einen zu kleinen Wert aus.

Beweis. Sei $f'(x, m) := mf(x)$. Nach Satz 2.3 existiert für die span-P-Funktion f' eine probabilistisch in Polynomzeit unter einem NP-Orakel berechenbare Approximationsfunktion g' mit

$$P\left[\frac{g'(x, 0^n, m) - mf(x)}{mf(x)} < \frac{1}{n}\right] > 1 - \frac{1}{mf(x)}$$

im Fall $f(x) > 0$ und $g'(x, 0^n, m) = 0$ im Fall $f(x) = 0$. Es ist leicht zu sehen, daß dann für $g(x, 0^n, m) := \lfloor g'(x, 0^n, m)/m \rfloor$

$$P\left[\frac{g(x, 0^n, m) - f(x)}{f(x)} < \frac{1}{n}\right] > 1 - \frac{1}{m}$$

gilt. Da $f(x)$ ganzzahlig ist, folgt aus $g'(x, 0^n, m) \geq mf(x)$ die Ungleichung $g(x, 0^n, m) \geq f(x)$, d.h. g unterschätzt f nicht. $\square$

3 Turingmaschinen mit wenigen akzeptierenden Berechnungspfaden

3.1 Einleitung

Wir untersuchen in diesem Kapitel die Komplexität von mit NP verwandten Klassen, die durch bestimmte strukturelle Eigenschaften charakterisiert sind. Viele Resultate der Komplexitätstheorie wurden unter Ausnutzung struktureller Zusatzinformationen erzielt (vgl. z.B. [Sch86, BDG88]). Stellvertretend sei an dieser Stelle die Erforschung selbstreduzierbarer Probleme und dünner Sprachen angeführt (siehe z.B. [Bal87, Mah82]). Ein anderes Beispiel für diesen Ansatz ist Valiant's Klasse UP, der bereits im vorigen Kapitel eine besondere Bedeutung zukam. Besonders interessant ist in diesem Zusammenhang die Frage, ob die Schwierigkeit, NP-Probleme zu lösen, von der Tatsache herrührt, daß die Größe des Lösungsraums innerhalb eines exponentiell großen Bereiches variieren kann (vgl. auch [VV86]). Grollmann und Selman [GS84] haben gezeigt, daß Einweg-Funktionen genau dann existieren, wenn $P \neq UP$ ist. Allender [All85] hat bei der Untersuchung invertierbarer Funktionen die NP-Teilklasse FewP eingeführt (dort noch mit FNP bezeichnet). Durch Ausstattung von Few-Maschinen mit einem flexibleren Akzeptiermechanismus haben Cai und Hemachandra [CH89] FewP zur Klasse Few verallgemeinert. In [CH89] wurde gezeigt, daß Few in der Klasse $\oplus$P [PZ83] enthalten ist. Wir verfeinern die Beweistechnik aus [CH89] und zeigen, daß zu jeder Few-Maschine M und FP-Funktion $g : \Sigma^* \times \mathcal{N}_0 \rightarrow \mathcal{N}_0$ eine NP-Maschine M' konstruiert werden kann, die bei Eingabe x genau $acc_{M'}(x) = g(x, acc_M(x)) + 2^{p(|x|)}$ akzeptierende Pfade hat, wobei p ein von M und g abhängiges Polynom ist. Daraus ergibt sich zusätzlich die Inklusion von Few in C=P, was eine Frage von Schöning [Sch88b] beantwortet. Als eine weitere Folgerung können wir zeigen, daß Few für die Sprachen PP, $\mathrm{C_{=}P}$ und $\oplus$P low ist. In diesen Klassen können also relativ zu einem Few-Orakel nicht mehr Sprachen als ohne Orakel erkannt werden.

3.2 Die Sprachklasse Few

Folgende Definition stammt von Allender [All85].

Definition 3.1 *Eine Sprache L ist in* FewP, *wenn L von einer* NP-*Maschine erkannt wird, die für ein Polynom p bei jeder Eingabe x höchstens $p(|x|)$ akzeptierende Rechnungen durchführt, d.h. es gilt*

$$acc_M(x) \leq p(|x|)$$

für alle $x \in \Sigma^*$.

Wir sagen zu einer NP-Maschine M Few-Maschine, falls M höchstens polynomiell viele akzeptierende Pfade hat. Eine unmittelbare Folgerung aus Lemma 1.4 ist, daß eine NP-Maschine M genau dann eine Few-Maschine ist, wenn acc_M in $\#\mathrm{P}\langle O(\log n)\rangle$ liegt. FewP wurde von Cai und Hemachandra [CH89] zur Klasse Few verallgemeinert.

Definition 3.2 *Eine Sprache L ist in* Few, *falls es eine* NP-*Maschine M, ein Polynom p und ein* P-*Prädikat Q gibt, so daß*

i) $acc_M(x) \leq p(|x|)$,

ii) $x \in L \Leftrightarrow Q(x, acc_M(x))$

für alle $x \in \Sigma^*$ gilt.

$\oplus$P *(Parity Polynomial Time)* wurde von Papadimitriou und Zachos [PZ83] eingeführt.

Definition 3.3 $\oplus$P *ist die Klasse aller Sprachen L, zu denen eine* NP-*Maschine M existiert mit*

$$x \in L \iff acc_M(x) \textit{ ist ungerade}$$

für alle $x \in \Sigma^*$.

$\oplus$P ist unter Komplementbildung abgeschlossen [PZ83]. Durch Voranstellen einer nichtdeterministischen Entscheidung, die Eingabe x entweder sofort zu akzeptieren, oder M bei Eingabe x zu simulieren, erhalten wir nämlich eine NP-Maschine M' mit $acc_{M'}(x) = acc_M(x) + 1$. Der $\mathsf{C}_=$-Operator wurde zusammen mit dem C-Operator von Wagner [Wag86a] bei der Definition der „Counting Hierarchy" eingeführt.

Definition 3.4 *Für eine Sprachklasse $\mathcal{A}$ ist $\mathsf{C}\mathcal{A}$ die Klasse aller Sprachen L, zu denen es eine Sprache $A \in \mathcal{A}$, eine Funktion $f \in$* FP *und ein Polynom p gibt, so daß*

$$x \in L \iff \|\{y \in \Sigma^{\leq p(|x|)} \mid (x,y) \in A\}\| \geq f(x) \tag{2}$$

für alle $x \in \Sigma^*$ gilt.

Der $\mathsf{C}_=$-Operator ist analog definiert, es muß in (2) nur das Zeichen $\geq$ durch $=$ ersetzt werden. Nach Lemma 1.2 ist $\mathsf{C}P^A$ für ein Orakel A die Klasse aller Sprachen, zu denen es eine NP-Orakelmaschine M und eine Funktion $f \in$ FP gibt, so daß

$$x \in L \iff acc_M^A(x) \geq f(x)$$

für alle $x \in \Sigma^*$ gilt. Dabei bedeutet es keinen Unterschied, ob man $f \in \mathrm{FP}$, $f \in \mathrm{FP}^A$ oder gar $f \in \#\mathrm{P}^A$ verlangt, da es nach Lemma 1.2 zu $f \in \#\mathrm{P}^A$ ein Polynom p gibt, so daß $g(x) := 2^{p(|x|)} - f(x)$ in $\#\mathrm{P}^A$ liegt. D.h. es gilt $acc_M^A(x) \geq f(x) \Leftrightarrow acc_M^A(x) + g(x) \geq 2^{p(|x|)}$ für alle $x \in \Sigma^*$, wobei $acc_M^A + g \in \#\mathrm{P}^A$ ist. Es kann sogar $f(x) = 2^{p(|x|)-1}$ verlangt werden, wobei p die Zeitschranke von M ist [Sim75]. Die Klassen PP^A und CP^A stimmen somit überein. Analog ist $\mathrm{C_{=}P}^A$ für ein Orakel A die Klasse aller Sprachen, zu denen es eine NP-Orakelmaschine M und eine Funktion $f \in \#\mathrm{P}^A$ gibt, so daß

$$x \in L \iff acc_M^A(x) = f(x)$$

für alle $x \in \Sigma^*$ gilt. Auch hier läßt sich zeigen, daß o.B.d.A. $f(x) = 2^{p(|x|)-1}$ vorausgesetzt werden kann, wobei p wieder die Zeitschranke von M ist. Dies hat u.a. den Vorteil, daß jede Sprache aus CP^A bzw. $\mathrm{C_{=}P}^A$ allein durch die Angabe einer NP-Maschine spezifiziert ist. Torán [Tor88b] hat bemerkt, daß $\mathrm{C_{=}P}^A \subseteq \mathrm{CP}^A$ gilt, da CP^A unter symmetrischer Differenz abgeschlossen ist [Rus85]. Es ist nicht bekannt, ob hierin P^A durch eine beliebige Sprachklasse $\mathcal{A}$ ersetzt werden kann.

Zunächst geben wir eine Orakelcharakterisierung von Few. Wir zeigen, daß Few aus genau den Sprachen besteht, die in Polynomzeit mit einer Frage an eine $\#\mathrm{P}\langle O(\log n)\rangle$-Orakelfunktion erkannt werden können. Diese Charakterisierungsmöglichkeit der Few-Sprachen wurde auch von Wagner festgestellt (vgl. [CH89]).

Satz 3.1 $\mathrm{Few} = \mathrm{P}^{\#\mathrm{P}\langle O(\log n)\rangle}[1]$.

Beweis. ($\subseteq$) Klar, da acc_M für eine Few-Maschine M in $\#\mathrm{P}\langle O(\log n)\rangle$ liegt. ($\supseteq$) Ist $A = L(M, f)$ für eine Funktion $f \in \#\mathrm{P}\langle O(\log n)\rangle$ und einen P-Orakeltransducer M, der bei jeder Eingabe x höchstens eine Frage stellt, so ist $f = acc_{M'}$ für eine Few-Maschine M'. Wir betrachten folgende Few-Maschine M'':

> $M''(x)$ simuliert M bei Eingabe x. Stellt M eine Orakelfrage y, so simuliert M'' die Maschine M' bei Eingabe y. Ansonsten verwirft M''.

Nun gilt $A = \{x \mid Q(x, acc_{M''}(x))\}$, wobei Q folgendes P-Prädikat ist

$$Q(x,z) \quad \Leftrightarrow \quad \begin{array}{l} M \text{ akzeptiert } x \text{ ohne eine Orakelfrage zu} \\ \text{stellen oder } M(x) \text{ stellt eine Orakelfrage} \\ y \text{ und akzeptiert bei Antwort } z. \end{array}$$

Also ist A eine Few-Sprache. $\square$

Wegen $\#\mathrm{P}\langle O(\log n)\rangle \subseteq \text{span-P}\langle O(\log n)\rangle = \text{max-P}\langle O(\log n)\rangle$ (Satz 1.3 und Satz 1.8) und $\text{max-P}\langle O(\log n)\rangle \subseteq \mathrm{FP}^{\mathrm{NP}}[O(\log n)]$ [Kre88] folgt mit Satz 3.1 das folgende Korollar.

Korollar 3.1 $\mathrm{Few} \subseteq \mathrm{P}^{\mathrm{NP}}[O(\log n)]$.

Auf der anderen Seite erhalten wir durch Anwendung von *Präfixsuche* (siehe Abbildung 6, vgl. auch [Sch86], S. 77) auf die Menge aller akzeptierenden Pfade einer Few-Maschine bei Eingabe x das folgende Resultat.

Satz 3.2 $\mathrm{Few} \subseteq \mathrm{P}^{\mathrm{FewP}}$.

Beweis. Sei $L \in \mathrm{Few}$ via einer Few-Maschine M und einem P-Prädikat Q, d.h. es gilt $x \in L \iff Q(x, acc_M(x))$ für alle $x \in \Sigma^*$. Sei p die Zeitschranke von M und q sei ein Polynom mit $acc_M(x) \leq q(|x|)$ für alle $x \in \Sigma^*$. Für das P-Prädikat

$$R(x,y) \quad \Leftrightarrow \quad \begin{array}{l} y \text{ ist akzeptierender Pfad von } M \text{ bei Ein-} \\ \text{gabe } x. \end{array}$$

ist dann $acc_M(x) = \|\{y \in \Sigma^{\leq p(|x|)} \mid R(x,y)\}\|$ für alle $x \in \Sigma^*$. Wir definieren ein FewP-Orakel

$$\text{PRÄFIX}(x,y) :\Longleftrightarrow \exists z : R(x, yz),$$

```
input: x;
S := {λ};
k := 0;
repeat
  T := ∅;
  for all y ∈ S do
    begin
      if R(x, y) then k := k + 1;
      if PRÄFIX(x, y0) then T := T ∪ {y0};
      if PRÄFIX(x, y1) then T := T ∪ {y1}
    end;
  S := T
until S = ∅;
if Q(x, k) then
  accept
else
  reject.
```

Abbildung 6: Bestimmung von $\|\{y \in \Sigma^{\leq p(|x|)} \mid R(x,y)\}\|$ durch Präfixsuche

unter dessen Befragung die in Abbildung 6 dargestellte Orakelmaschine M' $acc_M(x)$ berechnet. M' durchsucht den Präfixbaum mit der Knotenmenge $\{y \mid \text{PRÄFIX}(x,y)\}$ bei der Wurzel beginnend Stufe für Stufe und zählt dabei alle y mit $R(x,y)$. Da mit $acc_M(x)$ auch die Mächtigkeit der Knotenmenge $\{y \mid \text{PRÄFIX}(x,y)\}$ polynomiell in $|x|$ beschränkt ist, kann die Menge T in jedem Schleifendurchlauf nur polynomielle Größe erreichen. Da die repeat-Schleife höchstens $(p(|x|)+1)$-mal durchlaufen wird, arbeitet der Algorithmus in polynomieller Zeit. $\square$

Während zum Erkennen einer Few-Sprache nach Korollar 3.1 logarithmisch viele Fragen an ein NP-Orakel ausreichen, scheinen bei Verwendung eines FewP-Orakels polynomiell viele Fragen nötig zu sein.

3.3 Wenige akzeptierende Berechnungspfade implizieren eine genau bestimmbare Anzahl

In diesem Abschnitt zeigen wir das Hauptresultat dieses Kapitels. Es besagt, daß bei gegebener FP-Funktion $g : \Sigma^* \times \mathcal{N}_0 \to \mathcal{N}_0$ und Few-Maschine M eine NP-Maschine M' konstruiert werden kann, die bei jeder Eingabe x genau $acc_{M'}(x) = g(x, acc_M(x)) + 2^{p(|x|)}$ akzeptierende Pfade für ein Polynom p hat. Lemma 3.1 zeigt, wie die Maschine M' vom Prinzip her aufgebaut ist. Als unmittelbare Folgerung erhalten wir die Inklusion von Few in C=P und $\oplus$P.

Lemma 3.1 *Sei $b : \Sigma^* \times \Sigma^* \to \mathcal{Z}$ eine Funktion in* FP, *q ein Polynom und M eine* NP*-Maschine. Dann gibt es eine* NP*-Maschine M' und ein Polynom r mit*

$$acc_{M'}(x) = \sum_{k=0}^{q(|x|)} b(x,k)\binom{acc_M(x)}{k} + 2^{r(|x|)}$$

für alle $x \in \Sigma^*$.

Beweis. Zu M gibt es ein Polynom p und ein P-Prädikat Q mit $acc_M(x) = \|\{y \in \Sigma^{p(|x|)} \mid Q(x,y)\}\|$ für alle $x \in \Sigma^*$. Betrachten wir zunächst die in Abbildung 7 beschriebene Maschine M''. Für jedes k hat $M''(x)$ $b(x,k)\binom{acc_M(x)}{k}$ akzeptierende Pfade bei positivem $b(x,k)$ und $|b(x,k)|\left[\binom{2^{p(|x|)}}{k} - \binom{acc_M(x)}{k}\right]$ bei negativem $b(x,k)$. Insgesamt hat also M''

$$b(x,0)\binom{acc_M(x)}{0} + b(x,1)\binom{acc_M(x)}{1} + \ldots + b(x,q(|x|))\binom{acc_M(x)}{q(|x|)}$$

```
input: x;
guess k, 0 ≤ k ≤ q(|x|);
if b(x,k) = 0 then
  reject
else
  guess y ∈ {1, ..., |b(x,k)|};
  guess y_1 < ... < y_k ∈ Σ^{p(|x|)};
  if Q(x, y_i) für alle i, 1 ≤ i ≤ k then
    test := true
  else
    test := false;
  if (test and b(x,k) > 0) or (¬test and b(x,k) < 0) then
    accept
  else
    reject.
```

Abbildung 7: Eine NP-Maschine M'' mit $\sum_{k=0}^{q(|x|)} b(x,k)\binom{acc_M(x)}{k} + h(x)$ akzeptierenden Pfaden

$$+\underbrace{\sum_{k,b(x,k)<0} |b(x,k)|\binom{2^{p(|x|)}}{k}}_{=:\,h(x)}$$

akzeptierende Pfade, wobei $h(x)$ in FP ist. M'' ist eine NP-Maschine und es gibt ein Polynom r mit $h(x) \leq 2^{r(|x|)}$ für alle $x \in \Sigma^*$. Durch Erhöhen der Anzahl akzeptierender Pfade von M'' um $2^{r(|x|)} - h(x)$ erhalten wir die gewünschte Maschine M'. M' macht bei Eingabe x zuerst einen nichtdeterministischen Schritt. Die eine Alternative besteht in der Simulation von M'' bei Eingabe x. Die andere besteht im Ausführen von genau $2^{r(|x|)} - h(x)$ akzeptierenden Rechnungen. Der Berechnungsbaum von M' bei Eingabe x besteht demnach aus zwei Unterbäumen: Dem Berechnungsbaum von M'' bei Eingabe x und einem Berechnungsbaum mit genau $2^{r(|x|)} - h(x)$ akzeptierenden Pfaden. M' hat somit bei Eingabe x insgesamt genau

$$\begin{aligned} acc_{M'}(x) &= 2^{r(|x|)} - h(x) + acc_{M''}(x) \\ &= 2^{r(|x|)} + \sum_{k=0}^{q(|x|)} b(x,k) \binom{acc_M(x)}{k} \end{aligned}$$

akzeptierende Pfade. □

Für Few-Maschinen M ist $acc_M(x)$ durch $q(|x|)$ für ein Polynom q beschränkt, d.h. $acc_M(x)$ kann nur einen der Werte $0, \ldots, q(|x|)$ annehmen. Dies ermöglicht für jede Funktion $g : \Sigma^* \times \mathcal{N}_0 \to \mathcal{N}_0$ die Bestimmung von Zahlen $b(x,0), \ldots, b(x, q(|x|))$ mit der Eigenschaft

$$\sum_{k=0}^{q(|x|)} b(x,k) \binom{acc_M(x)}{k} = g(x, acc_M(x)). \tag{3}$$

Dabei hängen $b(x,0), \ldots, b(x, q(|x|))$ nur von g und nicht von acc_M ab, d.h. (3) gilt für alle in Frage kommenden Werte von $acc_M(x)$. Für die Berechnung von b sind zwei Punkte von besonderer Bedeutung: Der eine ist, daß der Wert von $\sum_{k=0}^{q(n)} b(x,k)\binom{m}{k}$ nur von $b(x,0), \ldots, b(x,m)$ abhängt. Wurden also bereits Werte für $b(x,0), \ldots, b(x,m)$ gefunden, die die Gleichung (3) im Fall $acc_M(x) \leq m$ erfüllen, so gilt dies auch nach einer später erfolgenden Festlegung der Werte von $b(x, m+1), \ldots, b(x, q(|x|))$. Der zweite Punkt ist, daß für $b(x, m+1)$ nach erfolgter Festlegung von $b(x,0), \ldots, b(x,m)$ ein Wert gefunden werden kann, so daß (3) auch im Fall $acc_M(x) = m+1$ erfüllt ist.

Der Grund hierfür ist folgende Gleichheit

$$\underbrace{\sum_{k=0}^{q(n)} b(x,k)\binom{m+1}{k}}_{\stackrel{!}{=}\, g(x,m+1)} = \sum_{k=0}^{m} b(x,k)\binom{m+1}{k} + b(x,m+1),$$

aus der sich $b(x,m+1)$ in Abhängigkeit von $b(x,0),\ldots,b(x,m)$ und $g(x,m+1)$ eindeutig ergibt. Es bleibt nur noch zu zeigen, daß mit g auch b eine FP-Funktion ist, und wir erhalten den folgenden Satz.

Satz 3.3 *Zu jeder* FP-*Funktion* $g : \Sigma^* \times \mathcal{N}_0 \to \mathcal{N}_0$ *und* Few-*Maschine* M *gibt es ein Polynom* r*, so daß die Funktion* $x \mapsto g(x, acc_M(x)) + 2^{r(|x|)}$ *in* #P *ist.*

Beweis. Sei q ein Polynom mit $acc_M(x) \leq q(|x|)$ für alle $x \in \Sigma^*$. $b : \Sigma^* \times \mathcal{N}_0 \to \mathcal{Z}$ sei eine Funktion in FP mit der Eigenschaft

$$\sum_{k=0}^{q(|x|)} b(x,k)\binom{m}{k} = g(x,m)$$

für alle $m, 0 \leq m \leq q(|x|)$. Nach Lemma 3.1 gibt es zu M, q und b eine NP-Maschine M' und ein Polynom r mit

$$\begin{aligned} acc_{M'}(x) &= \sum_{k=0}^{q(|x|)} b(x,k)\binom{acc_M(x)}{k} + 2^{r(|x|)} \\ &= g(x, acc_M(x)) + 2^{r(|x|)} \end{aligned}$$

akzeptierenden Pfaden. Wie bereits oben ausgeführt, läßt sich $b(x,k)$ induktiv berechnen:

$$\begin{aligned} b(x,0) &:= g(x,0), \\ b(x,k+1) &:= g(x,k+1) - \sum_{i=0}^{k} b(x,i)\binom{k+1}{i} \end{aligned}$$

für $k = 0, \ldots, q(|x|) - 1$ und $b(x,k) := 0$ für $k > q(|x|)$. Es ist klar, daß b in Polynomzeit berechenbar ist, wenn nur die Funktionswerte von b nicht zu groß werden. Sei $x \in \Sigma^*$ beliebig aber fest, und $g_{\max}$ sei das Maximum der Werte $g(x,k)$, $k = 0, \ldots, q(|x|)$. Wir zeigen durch Induktion über k

$$|b(x,k)| \leq c_k := g_{\max} 2^{(\sum_{i=0}^{k} i)} = g_{\max} 2^{k(k+1)/2}.$$

Es gilt:

$$|b(x,0)| \;\leq\; g_{\max} \;=\; c_0$$

und

$$\begin{aligned}
|b(x,k+1)| &\leq g_{\max} + \sum_{i=0}^{k} |b(x,i)| \binom{k+1}{i} \\
&\leq g_{\max} + \sum_{i=0}^{k} c_i \binom{k+1}{i} \\
&\leq g_{\max} + c_k \sum_{i=0}^{k} \binom{k+1}{i} \\
&= g_{\max} + c_k (2^{k+1} - 1) \\
&\leq c_k 2^{k+1} \;=\; c_{k+1}.
\end{aligned}$$

Also ist b in FP. □

Durch geeignete Wahl der Funktion g in Satz 3.3 erhalten wir neben Cai und Hemachandra's [CH89] Resultat, daß Few in $\oplus$P enthalten ist, die Inklusion von Few in $\mathrm{C_{=}P}$ und damit auch in CP.

Satz 3.4 *Zu jeder Sprache L aus* Few *gibt es eine* NP-*Maschine M' und ein Polynom r mit*

$$acc_{M'}(x) = \begin{cases} 2^{r(n)}, & x \in L, \\ 2^{r(n)} + 1, & x \notin L. \end{cases}$$

Beweis. Sei L aus Few. M sei eine Few-Maschine und Q sei ein P-Prädikat mit $x \in L \Leftrightarrow Q(x, acc_M(x))$ für alle $x \in \Sigma^*$. Dann ist folgende Funktion g in FP

$$g(x,m) := \begin{cases} 0, & Q(x,m), \\ 1, & \neg Q(x,m). \end{cases}$$

Nach Satz 3.3 gibt es zu M und g ein Polynom r, so daß die Funktion $x \mapsto g(x, acc_M(x)) + 2^{r(n)}$ in #P ist. D.h. es gibt eine NP-Maschine M' mit

$$acc_{M'}(x) = \begin{cases} 2^{r(n)}, & x \in L, \\ 2^{r(n)} + 1, & x \notin L. \end{cases}$$

□

Korollar 3.2

i) Few $\subseteq \mathrm{C}_=\mathrm{P}$,

ii) Few $\subseteq \oplus\mathrm{P}$. [CH89]

3.4 Few ist low für PP, $\mathrm{C}_=\mathrm{P}$ und $\oplus\mathrm{P}$

Wir ziehen in diesem Abschnitt weitere Schlußfolgerungen aus Satz 3.3. Als Hilfsmittel benötigen wir dabei die auch für sich interessante Charakterisierung der #Few-Funktionen durch Lemma 3.2. Eine unmittelbare Folgerung aus Lemma 3.2 ist Satz 3.5, der die Gleichheit der beiden Funktionenklassen #Few und #FewP besagt. Schließlich führt Satz 3.4 zusammen mit Lemma 3.2 zur „lowness“ von Few für die Klassen PP, $\mathrm{C}_=\mathrm{P}$ und $\oplus\mathrm{P}$.

Lemma 3.2 ist eine verfeinerte Version von Lemma 1.5. Es ermöglicht die Simulation einer #Few-Berechnung durch eine standardisierte

#FewP-Berechnung: Die simulierende Maschine stellt auf jedem Pfad genau eine Frage an das FewP-Orakel und akzeptiert die Eingabe x in einheitlicher Weise bei einer negativen Antwort. Für die Anwendung von Lemma 3.2 in Satz 3.6 ist dabei noch wichtig, daß alle Orakelfragen eine feste, nur von $|x|$ abhängige Länge haben (vgl. Abbildung 8).

Lemma 3.2 *Zu jeder* NP-*Orakelmaschine M und Sprache $A \in$* Few *gibt es ein* FewP-*Prädikat R und ein Polynom p, so daß für alle $x \in \Sigma^*$ gilt:*

$$acc_M^A(x) = \|\{y \in \Sigma^{p(|x|)} \mid \neg R(x,y)\}\|.$$

Beweis. Zu $A \in$ Few gibt es ein P-Prädikat Q und eine Few-Maschine M' mit $x \in A \Leftrightarrow Q(x, acc_{M'}(x))$ für alle $x \in \Sigma^*$. Wir betrachten zunächst folgendes Prädikat R':

$R'(x,w) \quad :\Longleftrightarrow \quad w = (z, (q_1, y_1^1, \ldots, y_{i_1}^1), \ldots, (q_k, y_1^k, \ldots, y_{i_k}^k))$
und
z ist eine akzeptierende Rechnung von M bei Eingabe x, in der der Reihe nach genau die Fragen $q_1, \ldots, q_k$ gestellt werden;
jede Frage q_j, $j = 1, \ldots, k$ wird genau dann mit ja beantwortet, wenn $Q(q_j, i_j)$ gilt;
für jedes j ist $y_1^j < \ldots < y_{i_j}^j$ und jedes y_i^j, $(1 \leq i \leq i_j)$ ist eine akzeptierende Rechnung von M' bei Eingabe q_j;
$y_1^j, \ldots, y_{i_j}^j$ sind alle akzeptierenden Rechnungen von $M'(q_j)$.

Aufgrund der Definition von R' ist sichergestellt, daß zu jeder akzeptierenden Rechnung von $M^A(x)$ genau ein $w \in \Sigma^*$ mit $R'(x,w)$ existiert. Da M' eine Few-Maschine ist, ist das Komplement von R' ein FewP-Prädikat. Weiter gibt es ein Polynom p mit der Eigenschaft, daß

$|w| < p(|x|)$ für alle x und w mit $R'(x, w)$ ist. Durch „padding" von R' erhalten wir daher das Komplement des gewünschten Prädikats R (vgl. Lemma 1.2). □

Satz 3.5 *#Few = #FewP.*

Beweis. Sei $f = acc_M^A \in$ #Few für eine NP-Orakelmaschine M und ein Few-Orakel A. Nach Lemma 3.2 existiert ein FewP-Prädikat R und ein Polynom p, so daß $acc_M^A(x) = \|\{y \in \Sigma^{p(|x|)} \mid \neg R(x, y)\}\|$ für alle $x \in \Sigma^*$ gilt. Es ist klar, daß die Orakelmaschine M' in Abbildung 8 mit

---

input: x;
guess $y \in \Sigma^{p(|x|)}$;
if $(x, y) \in$ Orakelmenge **then**
  reject
else
  accept.

Abbildung 8: Eine „standardisierte" NP-Orakelmaschine M'

Orakelmenge R die Funktion f in dem Sinne berechnet, daß $f(x) = acc_{M'}^R(x)$ für alle $x \in \Sigma^*$ gilt. □

Man beachte, daß Satz 3.5 auch aus Satz 3.2 folgt, da jede Few-Orakelfrage durch eine Folge von polynomiell vielen Fragen an ein FewP-Orakel ersetzt werden kann. Wie oben bereits ausgeführt, dient Lemma 3.2 dazu, eine #Few-Berechnung von $acc_M^A(x)$ auf eine standardisierte Berechnung (unter einem FewP-Orakel R) zurückzuführen (vgl. Abbildung 8). Wir verwenden nun das in Satz 3.4 hergeleitete Ergebnis, daß zu $R \in$ FewP eine NP-Maschine existiert, die bei Eingaben

$y \in R$ genau $2^{r(|y|)}$ und bei Eingaben $y \notin R$ genau $2^{r(|y|)} + 1$ akzeptierende Rechnungen für ein Polynom r hat. Diese Maschine wird nun auf die FewP-Orakelfragen der standardisierten Berechnung angesetzt und wir erhalten eine NP-Maschine, die genau $acc_M^A(x) + g(x)$ akzeptierende Pfade für eine FP-Funktion g hat.

Satz 3.6 *Zu jeder Funktion* $f \in \#\mathrm{Few}$ *gibt es ein Polynom* q*, so daß die Funktion* $x \mapsto f(x) + 2^{q(|x|)}$ *in* $\#\mathrm{P}$ *ist.*

Beweis. Sei M eine NP-Orakelmaschine und A eine Sprache in Few mit $f = acc_M^A$. Zu M und A seien ein Prädikat $R \in \mathrm{FewP}$ und ein Polynom p gemäß Lemma 3.2 gewählt, d.h. es gilt für alle $x \in \Sigma^*$ $acc_M^A(x) = \|\{y \in \Sigma^{p(|x|)} \mid \neg R(x, y)\}\|$.
Sei M' eine NP-Maschine und r ein Polynom zu R gemäß Satz 3.4, d.h. für alle x, y mit $R(x, y)$ hat $M'(x, y)$ genau $2^{r(|(x,y)|)}$ und sonst genau $2^{r(|(x,y)|)} + 1$ akzeptierende Pfade. Wir definieren $h(x) := 2^{r(|(x,0^{p(|x|)})|)}$ und betrachten folgende Maschine M'':

> $M''(x)$ rät ein Wort y der Länge $p(|x|)$. Dann simuliert M'' die Maschine M' bei Eingabe (x, y).

$M''(x)$ hat insgesamt

$$2^{p(|x|)}h(x) + \|\{y \in \Sigma^{p(|x|)} \mid \neg R(x, y)\}\| = 2^{p(|x|)}h(x) + acc_M^A(x)$$

akzeptierende Pfade. Durch Erhöhen der Anzahl der akzeptierenden Rechnungen von $M''(x)$ kann wie im Beweis von Lemma 3.1 eine NP-Maschine M''' konstruiert werden, die für ein Polynom q die Gleichung $acc_{M'''}(x) = 2^{q(|x|)} + acc_M^A(x)$ erfüllt. $\square$

Aus Satz 3.6 folgt nun unmittelbar die „lowness“ von Few für PP, $\mathrm{C_=P}$ und $\oplus\mathrm{P}$.

Korollar 3.3

i) Few *ist* PP-*low,*

ii) Few *ist* $\mathrm{C}_=\mathrm{P}$-*low,*

iii) Few *ist* $\oplus$P-*low.*

Eine weitere Möglichkeit, die $\oplus$P-lowness von Few zu zeigen, bietet Papadimitriou und Zachos' Resultat [PZ83], daß $\oplus$P $\oplus$P-low ist, d.h. $\oplus\mathrm{P}^{\oplus\mathrm{P}} = \oplus\mathrm{P}$. Wegen $\mathrm{Few} \subseteq \oplus\mathrm{P}$ muß daher auch Few $\oplus$P-low sein. Eine abschließende Folgerung aus Satz 3.6 ist, daß jede #Few-Funktion metrisch auf eine #P-Funktion reduziert werden kann.

Korollar 3.4 #Few *ist im metrischen Abschluß von* #P *enthalten.*

Literatur

[All85] E.W. ALLENDER. *Invertible Functions.* Dissertation, Georgia Institute of Technology, 1985.

[All86] E. ALLENDER. The complexity of sparse sets in P. In *Proceedings of the 1st Structure in Complexity Theory Conference*, Lecture Notes in Computer Science #223, 1–11. Springer-Verlag, 1986.

[Bal87] J. BALCÁZAR. Self-reducibility. In *Proceedings of the 4th Symposium on Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science #247, 136–147. Springer-Verlag, 1987.

[BBS86] J. BALCÁZAR, R. BOOK UND U. SCHÖNING. The polynomial-time hierarchy and sparse oracles. *Journal of the ACM*, **33**(3):603–617, 1986.

[BDG88] J. BALCÁZAR, J. DÍAZ UND J. GABARRÓ. *Structural Complexity I.* Springer-Verlag, 1988.

[Bei87] R. BEIGEL. Bounded queries to SAT and the Boolean hierarchy. Technical Report TR-7, John Hopkins University, Department of Computer Science, Baltimore, MD, 1987.

[BG84] A. BLASS UND Y. GUREVICH. Equivalence relations, invariants, and normal forms. *SIAM Journal on Computing*, **13**(4):682–689, 1984.

[CH89] J. CAI UND L. A. HEMASPAANDRA. The power of parity. In *Proceedings of the 6th Symposium on Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science #349, 229–239. Springer-Verlag, 1989.

[Coo71] S.A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd ACM Symposium on Theory of Computing*, 151–158. ACM Press, 1971.

[CW79] J. L. Carter und M. N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, **18**:143–154, 1979.

[Gil77] J. Gill. Computational complexity of probabilistic complexity classes. *SIAM Journal on Computing*, **6**:675–695, 1977.

[GJ79] M. Garey und D. Johnson. *Computers and Intractability – A Guide to the Theory of NP-Completeness.* Freeman and Company, 1979.

[GP86] L. M. Goldschlager und I. Parberry. On the construction of parallel computers from various bases of boolean functions. *Theoretical Computer Science*, **43**:43–58, 1986.

[GS84] S. Grollmann und A.L. Selman. Complexity measures for public-key cryptosystems. In *Proceedings of the 25th IEEE Symposium on the Foundations of Computer Science*, 495–503. IEEE Computer Society Press, 1984.

[GS85] A. V. Goldberg und M. Sipser. Compression and ranking. In *Proceedings of the 17th ACM Symposium on Theory of Computing*, 440–448. IEEE Computer Society Press, 1985.

[Hem87] L. A. Hemachandra. The strong exponential hierarchy collapses. In *Proceedings of the 19th ACM Symposium on Theory of Computing*, 110–122. ACM Press, 1987.

[HK73] J. E. Hopcroft und R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, **2**:225–231, 1973.

[HU79] J.E. Hopcroft und J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[JVV86] M. Jerrum, L. Valiant und V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science*, **43**:169–188, 1986.

[Kar72] R.M. Karp. Reducibility among combinatorial problems. In R.E. Miller und J.W. Thatcher, editors, *Complexity of Computer Computations*, 85–103. Plenum Press, 1972.

[Kre87] M. Krentel. *The Complexity of Optimization Problems.* Dissertation, Cornell University, New York, 1987.

[Kre88] M. Krentel. The complexity of optimization problems. *Journal of Computer and System Sciences*, **36**:490–509, 1988.

[KST88] J. Köbler, U. Schöning und J. Torán. On counting and approximation. In *Proceedings of the 13th Colloquium on Trees in Algebra and Programming*, Lecture Notes in Computer Science #299, 40–51. Springer-Verlag, 1988.

[KST89] J. Köbler, U. Schöning und J. Torán. On counting and approximation. *Acta Informatica*, **26**(4):363–379, 1989.

[KSTT89] J. Köbler, U. Schöning, S. Toda und J. Torán. Turing machines with few accepting computations and low sets for PP. In *Proceedings of the 4th Structure in Complexity Theory Conference*, 208–215. IEEE Computer Society Press, 1989.

[KSW87] J. Köbler, U. Schöning und K.W. Wagner. The difference and truth-table hierarchies of NP. *R.A.I.R.O. Informatique Théorique et Applications*, **21**(4):419–435, 1987.

[LLS75] R. Ladner, N. Lynch und A. Selman. A comparison of polynomial time reducibilities. *Theoretical Computer Science*, **1**:103–123, 1975.

[Mah82] S. Mahaney. Sparse complete sets for NP: Solution of a conjecture of Berman and Hartmanis. *Journal of Computer and System Sciences*, **25**(2):130–143, 1982.

[Meh86] K. Mehlhorn. *Datenstrukturen und effiziente Algorithmen. Band 1: Sortieren und Suchen.* B. G. Teubner, Stuttgart, 1986.

[MS73] A. Meyer und L. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential time. In *Proceedings of the 13th IEEE Symp. on Switching and Automata Theory*, 125–129, 1973.

[Pau78] W. J. Paul. *Komplexitätstheorie.* B. G. Teubner, Stuttgart, 1978.

[Pio88] M. Piotrów. On complexity of counting. In *Proceedings of the 13th Symposium on Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science #324, 472–482. Springer-Verlag, 1988.

[PZ83] C. Papadimitriou und S. Zachos. Two remarks on the power of counting. In *Proceedings of the 6th Conference on Theoretical Computer Science*, Lecture Notes in Computer Science #145, 269–276. Springer-Verlag, 1983.

[Rus85] D.A. RUSSO. *Structural Properties of Complexity Classes.* Dissertation, University of California, Santa Barbara, 1985.

[Sch83] U. SCHÖNING. A low and a high hierarchy within NP. *Journal of Computer and System Sciences*, **27**:14–28, 1983.

[Sch86] U. SCHÖNING. *Complexity and Structure*, Lecture Notes in Computer Science #211. Springer-Verlag, 1986.

[Sch88a] U. SCHÖNING. Graph isomorphism is in the low hierarchy. *Journal of Computer and System Sciences*, **37**:312–323, 1988.

[Sch88b] U. SCHÖNING. The power of counting. In *Proceedings of the 3rd Structure in Complexity Theory Conference*, 2–9. IEEE Computer Society Press, 1988.

[Sim75] J. SIMON. *On Some Central Problems in Computational Complexity.* Dissertation, Cornell University, New York, 1975.

[Sip83] M. SIPSER. A complexity theoretic approach to randomness. In *Proceedings of the 15th ACM Symposium on Theory of Computing*, 330–335. ACM Press, 1983.

[Sto77] L. STOCKMEYER. The polynomial-time hierarchy. *Theoretical Computer Science*, **3**:1–22, 1977.

[Sto85] L. STOCKMEYER. On approximation algorithms for #P. *SIAM Journal on Computing*, **14**(4):849–861, 1985.

[Tod] S. TODA. PP is $\leq_T^p$-hard for the polynomial-time hierarchy. Manuskript, 1989.

[Tor88a] J. TORÁN. An oracle characterization of the counting hierarchy. In *Proceedings of the 3rd Structure in Complexity Theory Conference*, 213–223. IEEE Computer Society Press, 1988.

[Tor88b] J. Torán. *Structural Properties of the Counting Hierarchies.* Dissertation, Universitat Politècnica de Catalunya, Barcelona, 1988.

[Val76] L. Valiant. Relative complexity of checking and evaluation. *Information Processing Letters*, **5**(1):20–23, 1976.

[Val79a] L. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, **8**:189–201, 1979.

[Val79b] L. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, **8**(3):410–421, 1979.

[VV86] L. Valiant und V. Vazirani. NP is as easy as detecting unique solutions. *Theoretical Computer Science*, **47**:85–93, 1986.

[Wag86a] K.W. Wagner. The complexity of combinatorial problems with succinct input representation. *Acta Informatica*, **23**:325–356, 1986.

[Wag86b] K.W. Wagner. Some observations on the connection between counting and recursion. *Theoretical Computer Science*, **47**:131–147, 1986.

[Wra75] C. Wrathall. *Subrecursive Predicates and Automata.* Dissertation, Harvard University, 1975.

[Wra77] C. Wrathall. Complete sets and the polynomial-time hierarchy. *Theoretical Computer Science*, **3**:23–33, 1977.

[WW86] K.W. Wagner und G. Wechsung. *Computational Complexity.* VEB Deutscher Verlag der Wissenschaften, Berlin und Reidel-Verlag, Dordrecht, 1986.

Stichwortverzeichnis

Symbolverzeichnis