Lowness-Eigenschaften und Erlernbarkeit von Booleschen Schaltkreisklassen

Habilitationsschrift

der Fakultät für Informatik
der Universität Ulm

für das Fach
Theoretische Informatik

vorgelegt von
Johannes Köbler

Ulm, 5. Juni 1995

Viele Menschen haben mir bei der Anfertigung dieser Arbeit geholfen. Ganz besonders bin ich Uwe Schöning zu Dank verbunden, der mir jede erdenkliche Unterstützung zukommen ließ. Seine Kompetenz und sein mitreißendes Engagement schaffen eine Atmosphäre, in der eine fruchtbare Zusammenarbeit aufs beste gedeihen kann. Danken möchte ich auch Vikraman Arvind, Martin Mundhenk und Osamu Watanabe für die vielen interessanten Diskussionen, die ihren Niederschlag in dieser Arbeit gefunden haben.

Nicht zuletzt bin ich meiner Familie für die mir entgegengebrachte Geduld und ihren Rückhalt sehr dankbar.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Eine der Hauptaufgaben der Komplexitätstheorie besteht darin, algorithmische Probleme entsprechend dem mit ihrer Lösung unabdingbar verbundenen Aufwand zu klassifizieren. Durch Beschränkung verschiedenartiger Ressourcen (wie zum Beispiel der Rechenzeit oder des Speicherplatzes) in Abhängigkeit von der Größe der Problemeingaben entsteht eine Vielfalt von Komplexitätsklassen, zwischen denen ein dichtes Geflecht von Wechselbeziehungen besteht. Die einem konkreten algorithmischen Problem innewohnende Komplexität kann daran gemessen werden, zu welchen Komplexitätsklassen es gehört (obere Schranken) beziehungsweise nicht gehört (untere Schranken). Während hierbei die Zugehörigkeit eines Problems zu einer bestimmten Klasse typischerweise durch die Angabe eines entsprechenden Algorithmus' (für das zugrundeliegende Berechnungsmodell) nachgewiesen wird, muß im umgekehrten Fall die Existenz eines solchen Algorithmus' ausgeschlossen werden, was eine aus beweistechnischer Sicht vollkommen andere Ausgangsposition darstellt.

Das $\mathcal{P} =? \mathcal{NP}$ Problem

Die beiden wohl bekanntesten Komplexitätsklassen sind die Klassen $\mathcal{P}$ und $\mathcal{NP}$. Während in $\mathcal{P}$ alle durch einen deterministischen Algorithmus in Polynomialzeit entscheidbaren Sprachen zusammengefaßt sind, enthält $\mathcal{NP}$ alle Sprachen, für die dies durch einen nichtdeterministischen Algorithmus bewerkstelligt werden kann. Diese beiden Klassen enthalten eine Vielzahl von praktisch relevanten Entscheidungsproblemen aus den verschiedensten Bereichen, insbesondere aus der Informatik und der Mathematik. Aus einem etwas mehr beweistheoretisch orientierten Blickwinkel betrachtet gehört eine Sprache A zu $\mathcal{P}$, falls zu jeder Eingabe $x \in A$ in Polynomialzeit ein *Beweis* für die Mitgliedschaft von x in A algorithmisch *generiert* werden kann.

Zu $\mathcal{NP}$ dagegen gehören alle Sprachen, für die sichergestellt ist, daß zu jeder positiven Instanz x ein Zugehörigkeits-Beweis existiert, der in Polynomialzeit (in der Länge von x) *verifiziert* werden kann.

Beispielsweise liegt das Problem, die Lösbarkeit eines gegebenen linearen Programms zu entscheiden, in $\mathcal{P}$, da eine Lösung (sofern eine existiert) in Polynomialzeit berechnet werden kann [Kha79, Kar84]. Für das damit verwandte Problem, die Existenz einer *ganzzahligen* Lösung festzustellen, ist dagegen bislang kein $\mathcal{P}$ Algorithmus bekannt. Es gibt jedoch einen $\mathcal{NP}$ Algorithmus, da im positiven Fall die Existenz einer ganzzahligen Lösung sichergestellt ist, die hinreichend klein und daher in Polynomialzeit verifizierbar ist [BT76].

Trotz sehr intensiver Erforschung seit Anfang der 70er Jahre konnte das $\mathcal{P} =? \mathcal{NP}$ Problem bis heute nicht gelöst werden. In seinem Kern beinhaltet dieses nicht nur für die Theoretische Informatik überaus bedeutsame Problem die grundsätzliche Frage, ob das Berechnungsvermögen von nichtdeterministischen Algorithmen tatsächlich stärker ist als das von deterministischen, falls ihnen vergleichbare Zeitschranken auferlegt werden. Anders formuliert: Um wieviel schwieriger ist es, einen Beweis zu finden (bzw. seine Nichtexistenz festzustellen), als einen bereits vorliegenden (potentiellen) Beweis auf seine Richtigkeit zu überprüfen? Es deutet einiges darauf hin, daß die für eine Lösung dieser und ähnlicher Fragen benötigten Beweistechniken größtenteils erst noch entwickelt werden müssen.

Ein naheliegender Ansatz, die (vermutete) Ungleichheit der beiden Klassen $\mathcal{P}$ und $\mathcal{NP}$ zu beweisen, besteht offenbar darin, für eine geeignet gewählte Sprache in $\mathcal{NP}$ nachzuweisen, daß sie von keinem deterministischen Algorithmus in Polynomialzeit entscheidbar ist. Angesichts der Tatsache, daß weit über Tausend verschiedene $\mathcal{NP}$ Probleme bekannt sind, deren Zugehörigkeit zu $\mathcal{P}$ ungeklärt ist, ist es erstaunlich, daß – bis auf vereinzelte Ausnahmen – all diese Probleme im Fall $\mathcal{P} \neq \mathcal{NP}$ gleichermaßen dazu geeignet sind, die Verschiedenheit von $\mathcal{P}$ und $\mathcal{NP}$ zu bezeugen. Der Grund hierfür ist, daß nahezu alle bekannten $\mathcal{NP}$ Probleme, für die in der Vergangenheit kein effizienter (das heißt, Polynomialzeit-) Algorithmus gefunden werden konnte, $\mathcal{NP}$-vollständig sind [Coo71, Kar72, GJ79]. Das bedeutet, daß diese Entscheidungsprobleme *definitiv* denselben Status haben, was ihre (Nicht-) Zugehörigkeit zu $\mathcal{P}$ betrifft, und daß sie darüberhinaus nur dann in $\mathcal{P}$ enthalten sind, wenn $\mathcal{NP} = \mathcal{P}$ ist.

Unter der (nach heutigem Wissen) plausiblen Voraussetzung, daß überhaupt Sprachen in $\mathcal{NP} - \mathcal{P}$ existieren, ermöglicht daher der strukturelle Begriff der $\mathcal{NP}$-Vollständigkeit für eine Vielzahl von $\mathcal{NP}$ Problemen den „Nachweis“, daß sie nicht effizient lösbar sind.

Dies wirft die Frage nach der Lösbarkeit von $\mathcal{NP}$ Problemen mit Hilfe von Näherungsmethoden oder durch *quasi-effiziente* Verfahren auf, wie dies etwa im Falle des (vermutlich nicht $\mathcal{NP}$-vollständigen) Primzahlproblems möglich ist. Für dieses Problem ist zwar kein $\mathcal{P}$ Algorithmus bekannt, es stehen jedoch eine Reihe von probabilistischen Algorithmen [SS77, Rab76, AH87] zur Verfügung, die sehr effizient sind und eine exponentiell kleine Fehlerwahrscheinlichkeit haben. Unter praktischen Gesichtspunkten sind solche Algorithmen durchaus mit einem $\mathcal{P}$ Algorithmus vergleichbar.

Polynomiale Schaltkreise

Es hat sich gezeigt, daß alle bekannten Verallgemeinerungen der Klasse $\mathcal{P}$, die als effizient lösbar in einem weiteren Sinne angesehen werden können, in der Klasse $\mathcal{P}/poly$ enthalten sind. Das heißt, für jedes der darin enthaltenen Entscheidungsprobleme A gibt es eine Folge von Schaltkreisen c_n, aufgebaut aus einer polynomialen Anzahl von Gattern (darunter n Eingabegatter), die bei Eingabe eines beliebigen (Binär-)Wortes der Länge n entscheiden, ob es zu A gehört oder nicht. Die Vermutung, daß eine $\mathcal{NP}$-vollständige Sprache auch nicht durch den Einsatz von quasi-effizienten Verfahren einer algorithmischen Lösung zugeführt werden kann, fand daher eine frühe Bestätigung durch das folgende von Karp und Lipton (zusammen mit Sipser) in ihrer fundamentalen Arbeit [KL80] bewiesene Resultat.

Satz 1.1 [KL80] $\mathcal{NP} \subseteq \mathcal{P}/poly \Rightarrow \mathcal{PH} = \Sigma_2^{\mathrm{P}}$.

Hierbei bezeichnet $\mathcal{PH}$ die Vereinigung über alle Klassen der Polynomialzeit-Hierarchie, die in einem gewissen Sinne den Schritt von $\mathcal{P}$ $(= \Sigma_0^{\mathrm{P}})$ nach $\mathcal{NP}$ $(= \Sigma_1^{\mathrm{P}})$ extrapolieren:

$$\Sigma_0^{\mathrm{P}} \subseteq \Sigma_1^{\mathrm{P}} \subseteq \Theta_2^{\mathrm{P}} \subseteq \Delta_2^{\mathrm{P}} \subseteq \Sigma_2^{\mathrm{P}} \subseteq \Theta_3^{\mathrm{P}} \subseteq \Delta_3^{\mathrm{P}} \subseteq \Sigma_3^{\mathrm{P}} \subseteq \cdots \subseteq \mathcal{PH}.$$

Es wird vermutet, daß die Polynomialzeit-Hierarchie „echt“ ist, das heißt, daß sämtliche Inklusionen in obiger Kette echt sind. Ein „Kollaps“ von $\mathcal{PH}$ auf Σ_0^{P} ist gleichbedeutend mit der (sehr unwahrscheinlichen) Gleichheit von $\mathcal{P}$ und $\mathcal{NP}$. Ein nicht ganz so tiefer Kollaps auf Σ_1^{P} hingegen ist gleichbedeutend mit der (etwas weniger unwahrscheinlichen) Gleichheit von $\mathcal{NP}$ und *co*-$\mathcal{NP}$.[1] Insgesamt stellt die Polynomialzeit-Hierarchie eine (potentiell) unendliche Folge von zunehmend stärker werdenden Hypothesen

$$\mathcal{PH} \neq \Sigma_0^{\mathrm{P}} \Leftarrow \mathcal{PH} \neq \Sigma_1^{\mathrm{P}} \Leftarrow \mathcal{PH} \neq \Theta_2^{\mathrm{P}} \Leftarrow \mathcal{PH} \neq \Delta_2^{\mathrm{P}} \Leftarrow \mathcal{PH} \neq \Sigma_2^{\mathrm{P}} \Leftarrow \cdots$$

[1] Die Klasse *co*-$\mathcal{NP}$ enthält alle Sprachen, die das Komplement einer $\mathcal{NP}$ Sprache bilden.

zur Verfügung. Da die Zugehörigkeit eines beliebigen $\mathcal{NP}$-vollständigen Problems zu $\mathcal{P}/poly$ die Inklusion von $\mathcal{NP}$ in $\mathcal{P}/poly$ impliziert, und da ein Kollaps der Polynomialzeit-Hierarchie auf Σ_2^{P} als nicht wahrscheinlich gilt, kann Satz 1.1 als ein starkes Indiz dafür gewertet werden, daß kein $\mathcal{NP}$-vollständiges Problem polynomiale Schaltkreise hat.

Die Berman-Hartmanis Vermutung und spärliche Mengen

Nur wenige Jahre später konnte Mahaney [Mah82] zeigen, daß keine $\mathcal{NP}$-vollständige Sprache in Polynomialzeit auf eine spärliche Menge[2] *many-one* reduzierbar ist (es sei denn, es wäre $\mathcal{P} = \mathcal{NP}$). Dieses Ergebnis steht in einer interessanten Beziehung zu Satz 1.1, der unter der stärkeren Bedingung, daß die Polynomialzeit-Hierarchie nicht auf die zweite Stufe kollabiert, die Möglichkeit einer *Turing*-Reduktion von einer $\mathcal{NP}$-vollständigen Sprache auf eine spärliche Menge ausschließt.[3]

Die Motivation für den Satz von Mahaney geht auf die berühmte Hypothese von Berman und Hartmanis [BH77] zurück. Diese besagt, daß alle $\mathcal{NP}$-vollständigen Sprachen „polynomial isomorph" sind, das heißt, es gibt eine bijektive Reduktionsfunktion, die in beide Richtungen in Polynomialzeit berechenbar ist. Tatsächlich konnten Berman und Hartmanis die Existenz eines solchen Isomorphismus' zwischen allen (bis dahin) bekannten $\mathcal{NP}$-vollständigen Sprachen nachweisen. Da die Eigenschaft, spärlich zu sein, unter polynomialen Isomorphismen erhalten bleibt, jedoch keine der bekannten $\mathcal{NP}$-vollständigen Sprachen spärlich ist, schließt die Gültigkeit der Berman-Hartmanis-Vermutung die Existenz spärlicher $\mathcal{NP}$-vollständiger Sprachen aus. Insofern bedeutet das Resultat von Mahaney eine Bestätigung für diese Hypothese.

Zwischen der Vermutung von Berman und Hartmanis und dem $\mathcal{P} =? \mathcal{NP}$ Problem besteht eine interessante Querverbindung: Ist nämlich die Vermutung richtig, so folgt $\mathcal{P} \neq \mathcal{NP}$. Dies liegt daran, daß sicherlich nicht alle many-one vollständigen Sprachen für die Klasse $\mathcal{P}$ polynomial isomorph sein können (da es beispielsweise sowohl endliche als auch unendliche many-one vollständige Sprachen in $\mathcal{P}$ gibt). Hinter dieser Implikation verbirgt sich ein interessanter Ansatz, die Verschiedenheit zweier Komplexitätsklassen $\mathcal{C}$ und $\mathcal{D}$ zu beweisen:

> Finde eine (strukturelle) Eigenschaft, die zwar die eine, nicht aber die andere der beiden Klassen aufweist.

[2] Eine Menge ist spärlich, falls sie von den exponentiell vielen möglichen Wörtern bis zur Länge n jeweils nur polynomial viele enthält; für diese und weitere Definitionen siehe Abschnitt 2.3

[3] Diese Version von Satz 1.1 ist eine Konsequenz aus der Tatsache, daß genau diejenigen Sprachen polynomiale Schaltkreise haben, die auf eine spärliche Menge Turing reduzierbar sind.

Da $\mathcal{NP}$ unter many-one Reduktionen abgeschlossen ist (das heißt, jede zu einer $\mathcal{NP}$ Sprache in Polynomialzeit many-one reduzierbare Sprache gehört zu $\mathcal{NP}$), die Klasse $\mathcal{E} = \mathit{DTIME}(2^{O(n)})$ jedoch nicht, läßt sich auf diese Art beispielsweise die Verschiedenheit von $\mathcal{NP}$ und $\mathcal{E}$ nachweisen.

Das Lowness-Konzept

Neben den erwähnten Kollaps-Resultaten in [KL80, Mah82] wurde in der Folgezeit eine ganze Reihe von weiteren Resultaten erzielt, die einen mehr oder weniger tiefen Kollaps der Polynomialzeit-Hierarchie unter der Voraussetzung implizieren, daß $\mathcal{NP}$-vollständige Probleme in bestimmten Teilklassen von $\mathcal{NP}$ enthalten sind. Dies bedeutet umgekehrt, daß Entscheidungsprobleme, von denen wir wissen, daß sie in einer solchen Teilklasse enthalten sind, höchstwahrscheinlich nicht $\mathcal{NP}$-vollständig sein können. Beispielhaft seien an dieser Stelle nur die drei folgenden algorithmischen Probleme genannt:

- Das bereits erwähnte Primzahlproblem (als ein Vertreter der Klasse $\mathcal{ZPP}$ aller durch einen probabilistischen „Las Vegas"-Algorithmus (siehe Abschnitt 2.3) entscheidbarer Sprachen),
- das damit verwandte, aber vermutlich schwierigere Faktorisierungsproblem (dessen Formulierung als Entscheidungsproblem in der Klasse $\mathcal{UP} \cap co\text{-}\mathcal{UP}$ liegt[4] [FK92]), und schließlich
- das Graphisomorphieproblem, das in $\mathcal{NP} \cap co\text{-}\mathcal{AM}$ enthalten ist [GS89].

(Die Klasse $\mathcal{AM}$ [Bab85] ist eine Verallgemeinerung von $\mathcal{NP}$, die aus allen Sprachen gebildet wird, für die ein interaktiver Zugehörigkeitsbeweis unter Verwendung von Zufallsbits geführt werden kann.)

In der Rekursionstheorie konnten Eigenschaften, deren Vorhandensein bei einem Entscheidungsproblem darauf schließen läßt, daß es sich nicht um ein vollständiges Problem für die Klasse aller rekursiv aufzählbaren Sprachen handelt (zu der die Klasse $\mathcal{NP}$ gewisse Analogien aufweist), als *Lowness*-Eigenschaften klassifiziert werden. Das bedeutet, diese Sprachen erweisen sich nicht als nützlich, wenn man sie in der Funktion eines Orakels bei der Lösung von Entscheidungsproblemen innerhalb der arithmetischen Hierarchie verwendet (siehe zum Beispiel [Ler83, Soa87]).

[4] $\mathcal{UP}$ ist eine Teilklasse von $\mathcal{NP}$ und steht für *Unique Polynomial Time*, siehe Abschnitt 2.3.

Motiviert durch diese Analogie führte Schöning [Sch83] die $\mathcal{L}ow$- und $\mathcal{H}igh$-Hierarchien innerhalb von $\mathcal{NP}$ ein. Diese klassifizieren alle Sprachen A in $\mathcal{NP}$ nach ihrer Nützlichkeit als Informationsquelle bei der Lösung von algorithmischen Problemen, die zur Polynomialzeit-Hierarchie gehören. Anschaulich kann man sich die $\mathcal{L}ow$-Hierarchie als eine „Projektion" der Polynomialzeit-Hierarchie in die Klasse $\mathcal{NP}$ vorstellen:

$$\mathcal{L}ow_0^{\Sigma} \subseteq \mathcal{L}ow_1^{\Sigma} \subseteq \mathcal{L}ow_2^{\Theta} \subseteq \mathcal{L}ow_2^{\Delta} \subseteq \mathcal{L}ow_2^{\Sigma} \subseteq \mathcal{L}ow_3^{\Theta} \subseteq \cdots \subseteq \mathcal{NP}.$$

Je tiefer ein NP-Problem innerhalb der Low-Hierarchie angesiedelt ist, desto stärker ist die Evidenz, daß es nicht NP-vollständig ist. Die Annahme, daß ein NP-vollständiges Problem in einer bestimmten Stufe der $\mathcal{L}ow$-Hierarchie enthalten ist, impliziert nämlich einen Kollaps von $\mathcal{PH}$ auf die entsprechende Stufe innerhalb der Polynomialzeit-Hierarchie. Das liegt daran, daß eine Sprache A definitionsgemäß dann zu einer bestimmten Stufe der $\mathcal{L}ow$-Hierarchie gehört, wenn ihre Verwendung als Orakel nicht dazu führt, daß auf der entsprechenden Stufe innerhalb der Polynomialzeit-Hierarchie mehr Sprachen entscheidbar sind (siehe Abschnitt 4.1).

Die Identifizierung von Lowness-Eigenschaften (d.h. von Eigenschaften, die eine Zugehörigkeit zur Low-Hierarchie implizieren) bildet folglich eine sinnvolle Ergänzung zur strukturellen NP-Vollständigkeitstheorie. Tatsächlich konnten seit ihrer Einführung Anfang der 80er Jahre viele Teilklassen von $\mathcal{NP}$ innerhalb der $\mathcal{L}ow$-Hierarchie eingeordnet werden. So konnte die Klasse aller $\mathcal{NP}$ Sprachen mit polynomialer Schaltkreiskomplexität von Ko und Schöning [KS85] innerhalb der dritten Sigma-Stufe der $\mathcal{L}ow$-Hierarchie lokalisiert werden. Verschiedene probabilistische Teilklassen von $\mathcal{NP}$, wie zum Beispiel die bereits erwähnte Klasse $\mathcal{ZPP}$ oder die Klasse $\mathcal{NP} \cap \mathcal{BPP}$ aller $\mathcal{NP}$ Sprachen, die durch einen sog. „Monte Carlo"-Algorithmus entscheidbar sind, sind sogar in kleineren Stufen der $\mathcal{L}ow$-Hierarchie enthalten: $\mathcal{ZPP}$ etwa in $\mathcal{L}ow_1^{\Sigma}$ und $\mathcal{NP} \cap \mathcal{BPP}$ in $\mathcal{L}ow_2^{\Sigma}$ [Sch86]. Weitere Beispiele sind Reduktionsklassen zu spärlichen Mengen, Sprachen mit niedriger Instanzenkomplexität ($\mathcal{NP} \cap \mathcal{IC}[log, poly]$ [OKSW94], enthalten in $\mathcal{L}ow_2^{\Theta}$) und schließlich das Komplement von Sprachen, die in bestimmten Verallgemeinerungen der Klasse $\mathcal{NP}$ enthalten sind. In diese Kategorie fällt beispielsweise die bereits erwähnte Klasse $\mathcal{NP} \cap co\text{-}\mathcal{AM}$, in der das Graphisomorphieproblem enthalten ist, und die ebenfalls unterhalb der $\mathcal{L}ow_2^{\Sigma}$ Stufe angesiedelt ist [Sch86].

Ein interessanter Aspekt ist hierbei, daß Sprachen in der $\mathcal{L}ow$-Hierarchie auch nicht in bezug auf allgemeinere Reduzierbarkeiten $\mathcal{NP}$-vollständig sein können, ohne einen Kollaps der Polynomialzeit-Hierarchie zu verursachen. Beispielsweise ergibt sich aus der Zugehörigkeit des Graphisomorphieproblems zur zweiten Sigma-Stufe der $\mathcal{L}ow$-Hierarchie, daß dieses Problem auch nicht bezüglich der allgemeineren γ-Reduzierbarkeit $\mathcal{NP}$-vollständig ist (eine Frage, die in [AM77] gestellt wurde), außer wenn die Polynomialzeit-Hierarchie auf Σ_2^{P} kollabiert [Sch86].

Die erweiterte $\mathcal{L}ow$-Hierarchie und die Klasse $\mathcal{P}/poly$

Um auch eine Klassifizierung von Sprachen außerhalb von $\mathcal{NP}$ zu ermöglichen, wurden die $\mathcal{L}ow$- und $\mathcal{H}igh$-Hierarchien von Balcázar, Book und Schöning zur erweiterten $\mathcal{L}ow$-Hierarchie ausgedehnt [BBS86]. In dieser Arbeit wurde auch gezeigt, daß die Klasse aller $\mathcal{P}/poly$ Sprachen in der dritten Sigma-Stufe $E\mathcal{L}_3^{\Sigma}$ der erweiterten $\mathcal{L}ow$-Hierarchie enthalten ist. Die Frage, ob es möglich ist, die Klasse $\mathcal{P}/poly$ in einer tieferen Stufe der erweiterten $\mathcal{L}ow$-Hierarchie zu lokalisieren, blieb über mehrere Jahre offen. Das Interesse an dieser Frage verstärkte sich noch, als Anfang der 90er Jahre von Allender und Hemachandra [AH92] bewiesen werden konnte, daß eine Einordnung von $\mathcal{P}/poly$ innerhalb der untersten Stufen der erweiterten $\mathcal{L}ow$-Hierarchie bis einschließlich $E\mathcal{L}_2^{\Sigma}$ ausgeschlossen ist.

Schließlich wurde die Frage der exakten Einordnung von $\mathcal{P}/poly$ in [Köb94] dadurch gelöst, daß $\mathcal{P}/poly$ innerhalb der kleinsten der übrig gebliebenen drei Stufen $E\mathcal{L}_3^{\Theta}$, $E\mathcal{L}_3^{\Delta}$ und $E\mathcal{L}_3^{\Sigma}$ lokalisiert wurde. Den Schlüssel zu diesem Ergebnis lieferte eine Verbesserung der bis dahin bekannten Komplexitätsschranken für das Problem, zu einer Sprache A in $\mathcal{P}/poly$ eine Folge von Schaltkreisen c_n zu generieren, die A entscheiden. Die effizientere Generierung solcher Schaltkreise wurde unter anderem durch Verwendung von Konzepten aus den Bereichen des universalen Hashings[5] [CW79] und des algorithmischen Lernens [Ang88] ermöglicht.

Als eine unmittelbare Konsequenz aus der Zugehörigkeit aller $\mathcal{P}/poly$ Sprachen zu $E\mathcal{L}_3^{\Theta}$ ergibt sich, daß alle $\mathcal{NP}$ Sprachen, die polynomiale Schaltkreise haben, in $\mathcal{L}ow_3^{\Theta}$ enthalten sind. Folglich kann *keine* Sprache in der $\mathcal{H}igh_3^{\Theta}$ Stufe der $\mathcal{H}igh$-Hierarchie (und damit beispielsweise auch keine $\leq_T^{SN}$-vollständige[6] Sprache für $\mathcal{NP}$) polynomiale Schaltkreise haben, außer wenn die Polynomialzeit-Hierarchie auf die Stufe Θ_3^{P} kollabiert.

[5] Die ursprünglich von Carter und Wegman [CW79] stammende Methode des Universalen Hashings wurde erstmals von Sipser [Sip83] im Bereich der strukturellen Komplexitätstheorie als eine Beweistechnik eingesetzt, um die Zugehörigkeit aller $\mathcal{BPP}$ Sprachen zu Σ_2^{P} zu beweisen.

[6] Die Gesamtheit aller $\leq_T^{SN}$-vollständigen Sprachen für $\mathcal{NP}$ bildet die $\mathcal{H}igh_1^{\Sigma}$ Stufe der $\mathcal{H}igh$-Hierarchie. Die Stufen der $\mathcal{H}igh$-Hierarchie insgesamt stellen zunehmende Verallgemeinerungen des Vollständigkeitsbegriffs innerhalb von $\mathcal{NP}$ dar (siehe Abschnitt 4.1).

Auswirkungen von Selbstreduzierbarkeit auf die Lowness Eigenschaften einer Sprache

Vergleichen wir die im letzten Absatz beschriebene Folgerung mit Satz 1.1, so stellen wir fest, daß dieser einen tieferen Kollaps auf Σ_2^{P} unter einer stärkeren Annahme herbeiführt. Während ein Kollaps auf Θ_3^{P} aus der Existenz eines beliebigen $\leq_T^{SN}$-vollständigen Problems mit polynomialen Schaltkreisen folgt, nützt der Beweis von Satz 1.1 insbesondere die Annahme aus, daß ein ganz bestimmtes $\leq_T^{SN}$-vollständiges Problem (nämlich das bekannte $\mathcal{NP}$-vollständige Problem SAT) in $\mathcal{P}/poly$ enthalten ist. Und zwar macht der Beweis von Satz 1.1 sehr stark von der strukturellen Selbstreduzierbarkeits-Eigenschaft von SAT Gebrauch. Wie Ko und Schöning gezeigt haben, kann der tiefere Kollaps in Satz 1.1 in der Tat damit erklärt werden, daß für selbstreduzierbare Sprachen in $\mathcal{P}/poly$ stärkere Lowness-Eigenschaften gelten.

Satz 1.2 [KS85] *Alle selbstreduzierbaren $\mathcal{NP}$ Sprachen in $\mathcal{P}/poly$ sind in der $\mathcal{L}ow_2^{\Sigma}$ Stufe der $\mathcal{L}ow$-Hierarchie enthalten.*

Um einen möglichst tiefen Kollaps der Polynomialzeit-Hierarchie unter der Voraussetzung $\mathcal{NP} \subseteq \mathcal{P}/poly$ herbeizuführen, empfiehlt es sich daher, sich den strukturellen Aufbau von speziellen $\mathcal{NP}$-vollständigen Sprachen zunutze zu machen. Wie wir gesehen haben ist hierbei die Frage von besonderem Interesse, welche Auswirkungen die Selbstreduzierbarkeit einer Sprache auf ihre Lowness-Eigenschaften hat. Die Motivation für die Erforschung der Lowness-Eigenschaften von selbstreduzierbaren Sprachen wird noch dadurch verstärkt, daß außer der Klasse $\mathcal{NP}$ viele andere Komplexitätsklassen ebenfalls vollständige Sprachen mit einer Selbstreduzierbarkeits-Struktur besitzen. Tatsächlich trifft dies auf alle Komplexitätsklassen zwischen $\mathcal{P}$ und $\mathcal{EXP}$ zu,[7] von denen bekannt ist, daß sie vollständige Sprachen besitzen. Folglich „erben“ diese Komplexitätsklassen unter der Annahme, daß sie in $\mathcal{P}/poly$ enthalten sind, sämtliche Lowness-Eigenschaften, die für die Klasse aller selbstreduzierbaren Sprachen in $\mathcal{P}/poly$ gelten.

Erst vor kurzem gelang der Nachweis, daß alle selbstreduzierbaren Sprachen A in $\mathcal{P}/poly$ low für die probabilistische Klasse $\mathcal{ZPP}(\mathcal{NP})$ sind (diese Ergebnisse werden in [KW95] erscheinen). Das bedeutet, eine Befragung des Orakels A führt nicht dazu, daß durch $\mathcal{ZPP}(\mathcal{NP})$ Berechnungen mehr Sprachen entscheidbar werden als dies ohne die Hilfe von A möglich ist. Das $\mathcal{ZPP}(\mathcal{NP})$ Lowness-Resultat verbessert Satz 1.2, der besagt, daß A nicht in der Lage ist, das Berechnungsvermögen von Σ_2^{P} $(= \mathcal{NP}(\mathcal{NP}))$ Berechnungen zu erhöhen. Eine weitere Verbesserung zu einem

[7] $\mathcal{EXP}$ ist die Klasse aller in deterministischer Exponentialzeit entscheidbaren Sprachen.

$\mathcal{P}(\mathcal{NP})$ Lowness-Resultat ist zumindest mit relativierbaren Beweistechniken nicht mehr zu erzielen, da Orakel existieren, relativ zu denen $\mathcal{NP}$ zwar in $\mathcal{P}/poly$ enthalten ist (die selbstreduzierbare Sprache SAT also polynomiale Schaltkreise hat), die Polynomialzeit-Hierarchie jedoch nicht auf $\mathcal{P}(\mathcal{NP})$ kollabiert (SAT also nicht low für $\mathcal{P}(\mathcal{NP})$ sein kann) [Hel86, Wil85].

Als Folgerung aus dem $\mathcal{ZPP}(\mathcal{NP})$ Lowness-Resultat ergeben sich unmittelbar eine Reihe von stärkeren Kollaps-Konsequenzen unter der Annahme, daß $\mathcal{NP}$ oder andere Komplexitätsklassen in $\mathcal{P}/poly$ enthalten sind. Beispielsweise erhalten wir dadurch einen tieferen Kollaps in Satz 1.1 zu $\mathcal{PH} = \mathcal{ZPP}(\mathcal{NP})$, was wiederum die Evidenz dafür erhöht, daß $\mathcal{NP}$-vollständige Sprachen nicht mittels polynomialer Schaltkreise entscheidbar sind. Aus dem tieferen Kollaps lassen sich weiterhin neue untere Schranken für die Schaltkreiskomplexität von Sprachen in verschiedenen Komplexitätsklassen wie zum Beispiel $\mathcal{ZPP}(\mathcal{NP})$ herleiten.

Querbezüge zur algorithmischen Erlernbarkeitstheorie

Wie bereits angedeutet, bestehen enge Querbezüge zwischen den Lowness-Eigenschaften einer Sprache A in $\mathcal{P}/poly$ und der Frage, welcher Rechenaufwand für die Generierung einer Schaltkreisfolge c_n für A betrieben werden muß. Diese Frage wiederum stellt sich in ganz ähnlicher Form bei der Erforschung der Erlernbarkeit von Konzepten, falls zu deren Repräsentation Boolesche Schaltkreise gewählt werden. In beiden Fällen geht es nämlich darum, durch eine bestimmte Art von Fragen (im ersten Fall an das Orakel A, im zweiten an einen „Lehrer“) möglichst effizient genügend Informationen zu sammeln, die eine Bestimmung eines geeigneten Schaltkreises c_n erlauben. Tatsächlich kann man an diesem Beispiel sehr schön sehen, wie sich Ideen aus den Bereichen der strukturellen Komplexitätstheorie und der algorithmischen Erlernbarkeitstheorie im Wechselspiel inspiriert haben.

Erst kürzlich konnten Bshouty, Cleve, Kannan und Tamon [BCKT94] unter Verwendung eines Resultats von Jerrum, Valiant und Vazirani [JVV86] (aus der strukturellen Komplexitätstheorie) den Nachweis erbringen, daß Boolesche Schaltkreise durch einen *probabilistischen* Lernalgorithmus mithilfe von Äquivalenzfragen unter zusätzlicher Befragung eines $\mathcal{NP}$ Orakels exakt erlernbar sind. Aus dem Beweis dieses Resultats ist erkennbar, daß für jede Sprache A in $\mathcal{P}/poly$ eine Folge von Schaltkreisen in der probabilistischen Funktionenklasse $\mathcal{FZPP}(\mathcal{NP}(A))$ (siehe Abschnitt 2.3) berechnet werden kann. Durch diese Beobachtung wurde wiederum das $\mathcal{ZPP}(\mathcal{NP})$ Lowness-Resultat [KW95] für alle selbstreduzierbaren Sprachen in $\mathcal{P}/poly$ inspiriert (dessen Beweis in Kapitel 5 dieser Arbeit allerdings auf den in [Sip83, Ang88, Gav92, Köb94] entwickelten Techniken aufbaut).

Ähnliche Wechselbeziehungen können im deterministischen Fall beobachtet werden. Unter Verwendung der Hashing-Methode konnte in [Gav93, BCG+95] ein deterministischer Lernalgorithmus entworfen werden, der Boolesche Schaltkreise in Polynomialzeit mithilfe von Äquivalenzfragen und Fragen an ein Σ_3^{P} Orakel erlernt. Durch eine Weiterentwicklung der Hashing-Technik gelingt es uns, das Σ_3^{P} Orakel durch ein Σ_2^{P} Orakel zu ersetzen. Diesem bislang unveröffentlichten Resultat ist Kapitel 6 gewidmet, das darüberhinaus in die Grundbegriffe des algorithmischen Lernens am Beispiel des exakten Lernens durch Fragen [Ang88] und des PAC-Lernens [Val84] einführt.

Wir geben abschließend einen kurzen Überblick der Kapitel 2 bis 5. Das nächste Kapitel führt in die verwendete Notation ein und beinhaltet eine Zusammenstellung von grundlegenden Definitionen und Resultaten, auf die wir in späteren Abschnitten immer wieder Bezug nehmen. In Kapitel 3 führen wir in die Technik des universalen Hashings ein verfeinern diese weiter. Kapitel 4 enthält eine Einführung in die Lowness-Theorie und ist im übrigen der exakten Einordnung von $\mathcal{P}/poly$ und anderer Advice-Klassen in die erweiterte $\mathcal{L}ow$-Hierarchie gewidmet. Durch die im dritten Kapitel erfolgte Weiterentwicklung der Hashing-Technik sind wir dabei in der Lage, den Beweisgang unter Zurückdrängung von technischen Details auf das Wesentliche zu beschränken. Desweiteren zeigen wir im letzten Abschnitt dieses Kapitels auf, daß die erzielten Lowness-Resultate optimal sind, eine weitergehende Verbesserung also nicht möglich ist. In Kapitel 5 untersuchen wir die Lowness-Eigenschaften von selbstreduzierbaren Sprachen in $\mathcal{P}/poly$ und die sich daraus ergebenden Folgerungen. Wichtige Vereinbarungen, die jedoch nicht den Charakter einer formalen Definition haben, heben wir besonders heraus, um dem Leser das Wiederauffinden zu erleichtern.

Kapitel 2

Grundlegende Definitionen und Resultate

Die in dieser Arbeit dargestellten Resultate bauen auf einer Reihe von Begriffen, Definitionen und Ergebnissen aus verschiedenen Gebieten insbesondere der Theoretischen Informatik auf. Der Zweck dieses (und des nächsten) Kapitels ist es, einen Bausatz von wichtigen Grundbausteinen in übersichtlicher Form zusammenzustellen, so daß wir in den nachfolgenden Kapiteln immer wieder darauf zurückgreifen können. Den an einer ausführlicheren Beschäftigung mit den in diesem Kapitel nur kurz abgehandelten Themengebieten interessierten Leser möchten wir auf die entsprechenden Lehrbücher verweisen, die exzellente Darstellungen dieser Grundkonzepte in leicht verständlicher Form enthalten (siehe zum Beispiel [BC94, BDG95, BDG90, GJ79, GKP89, HU79, Pap94, Sch92, WW86, Weg87]).

2.1 Allgemeines

Für ein (endliches) *Alphabet* Σ ist $\Sigma^* =_{def} \bigcup_{i\geq 0} \Sigma^i$ die Menge aller *Wörter* über dem Alphabet Σ. Wird nichts anderes gesagt, so ist Σ das *Standardalphabet* $\{0,1\}$. Die *Länge* eines Wortes $x \in \Sigma^*$ bezeichnen wir mit $|u|$. Das (einzige) Wort der Länge 0 ist das *leere Wort* und wird mit λ bezeichnet. Die Menge $\bigcup_{i=0}^{n} \Sigma^i$ aller Wörter über Σ, deren Länge höchstens n beträgt, bezeichnen wir mit $\Sigma^{\leq n}$. Eine *Sprache (über Σ)* ist eine beliebige Teilmenge von Σ^*. Das *Komplement* einer Sprache L ist die Sprache

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

Der join-Operator $\oplus$ vereinigt zwei Sprachen A und B, ohne daß dabei die Information verloren geht, aus welcher der beiden Sprachen die einzelnen Wörter in der Vereinigung stammen (die Paarfunktion $\langle\cdot,\cdot\rangle$ ist weiter unten definiert):

$$A \oplus B \quad =_{def} \quad \{\langle x, 1\rangle \mid x \in A\} \cup \{\langle x, 0\rangle \mid x \in B\}.$$

Diese Operatoren lassen sich kanonisch auf Sprachklassen fortsetzen. So bezeichnet für eine Klasse $\mathcal{A}$ von Sprachen *co*-$\mathcal{A}$ die Klasse aller Sprachen, deren Komplement in $\mathcal{A}$ ist,

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

Entsprechend ist

$$\mathcal{A} \oplus \mathcal{B} \quad =_{def} \quad \{A \oplus B \mid A \in \mathcal{A} \text{ und } B \in \mathcal{B}\}.$$

Die Mächtigkeit einer endlichen Menge oder Sprache A bezeichnen wir mit $\|A\|$.

Die Menge der natürlichen Zahlen bezeichnen wir mit $\mathcal{N}$, die der reellen mit $\mathcal{R}$. Eine natürliche Zahl n stellen wir binär als ein Wort über dem Standardalphabet $\Sigma = \{0, 1\}$ dar. Die Länge von n in Binärdarstellung ergibt sich offenbar zu $|n| = \lfloor \log(n + 1)\rfloor$. Dabei ist log der Logarithmus zur Basis 2 und $\lfloor r\rfloor$ die größte ganze Zahl $\leq r$ (für eine reelle Zahl r). Entsprechend bezeichnet $\lceil r\rceil$ die kleinste ganze Zahl $\geq r$. Seien $f, g : \mathcal{N} \to \mathcal{N}$. 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 *Indikator*- oder auch *charakteristische Funktion* einer Sprache A bezeichnen wir ebenfalls mit A. Diese ist definiert durch

$$A(x) \quad =_{def} \quad \begin{cases} 1, & x \in A, \\ 0, & x \notin A. \end{cases}$$

Die auf Σ vorhandene Totalordnung $(0 < 1)$ läßt sich kanonisch auf Σ^* erweitern:

$$\lambda < 0 < 1 < 00 < 01 < 10 < 11 < 000 < \cdots$$

Diese Ordnung heißt *lexikographische Ordnung* auf Σ^*.

Ein k-Tupel $(x_1, \ldots, x_k)$ von Wörtern $x_1, \ldots, x_k$ aus Σ^* kodieren wir durch ein Wort $\langle x_1, \ldots, x_k\rangle$ 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 (für unsere Zwecke genügt es $\pi(x, y) =_{def} \tilde{x}01y$ zu wählen, wobei $\tilde{x} =_{def} x_1x_1 \ldots x_nx_n$ für $x = x_1 \ldots x_n \in \Sigma^n$ ist). Dann können wir $(x_1, \ldots, x_k)$ durch das Wort $\langle x_1, \ldots, x_k\rangle =_{def} \pi(0^k, \pi(x_1, \pi(\ldots \pi(x_{k-1}, x_k) \ldots)))$ kodieren. Ist f eine Funktion auf Σ^*, so schreiben wir anstatt $f(\langle x_1, \ldots, x_k\rangle)$ einfacher $f(x_1, \ldots, x_k)$.

2.2 Wahrscheinlichkeitstheorie

Wir werden es in dieser Arbeit nur mit diskreten (das heißt endlichen oder abzählbar unendlichen) *Wahrscheinlichkeitsräumen* Ω zu tun haben. Die Elemente von Ω werden als *Elementarereignisse* bezeichnet. Jedem Elementarereignis x ist eine Wahrscheinlichkeit zugeordnet, die wir mit $P(x)$ bezeichnen. Die Wahrscheinlichkeit eines *Ereignisses* $E \subseteq \Omega$ ist die Summe der Wahrscheinlichkeiten der in E enthaltenen Elementarereignisse, $P(E) = \sum_{x\in E} P(x)$. In jedem Wahrscheinlichkeitsraum gilt $P(\emptyset) = 0$ und $P(\Omega) = 1$.

Seien $E_1, \ldots, E_m$ Ereignisse. $E_1, \ldots, E_m$ heißen *(stochastisch) unabhängig*, falls für jede nicht-leere Indexmenge $I \subseteq \{1, \ldots, m\}$ die Gleichheit $P(\bigcap_{i\in I} E_i) = \prod_{i\in I} P(E_i)$ gilt. Gilt dies zumindest für alle zweielementigen Indexmengen, so heißen die Ereignisse $E_1, \ldots, E_m$ *paarweise unabhängig*.

Eine *(reelle) Zufallsvariable* Z weist jedem Elementarereignis x eine reelle Zahl $Z(x) \in \mathcal{R}$ zu. Der *Erwartungswert* von Z ist definiert als $E(Z) = \sum_{x\in\Omega} Z(x) \cdot P(x)$. Durch die Zufallsvariable Z wird eine *Verteilung* auf $\mathcal{R}$ definiert, das heißt, jeder Teilmenge A von $\mathcal{R}$ wird eine Wahrscheinlichkeit $P(\{x \in \Omega \mid Z(x) \in A\})$ zugeordnet, die wir mit $\Pr_{x\in_P\Omega}[\, Z(x) \in A\,]$ bezeichnen. Wir werden im folgenden meistens darauf verzichten, den zugrundeliegenden Wahrscheinlichkeitsraum (Ω, P) explizit anzugeben. Er wird jedoch immer eindeutig aus dem Kontext hervorgehen. Sind das Wahrscheinlichkeitsmaß P und/oder die Grundmenge Ω bereits fest vorgegeben, so schreiben wir auch verkürzt $\Pr_{x\in\Omega}[\, Z(x) \in A\,]$ bzw. $\Pr[\, Z \in A\,]$.

Das einfachste nichttriviale Beispiel für eine Zufallsvariable ist die sogenannte *Indikatorvariable* für ein Ereignis E, die anzeigt, ob bei einem Zufallsexperiment das Ereignis E eingetreten ist oder nicht:

$$Z(x) \quad = \quad \begin{cases} 1, & x \in E, \\ 0, & \text{sonst.} \end{cases}$$

Diese Art von Experiment, bei der nur auf das Eintreten („Erfolg“) bzw. Nicht-Eintreten („Mißerfolg“) eines einzelnen Ereignisses E geachtet wird, wird auch als *Bernoulli-Versuch* bezeichnet. Offensichtlich ist $E(Z) = \Pr[\, Z = 1\,] = P(E)$.

Eine sehr wichtige Wahrscheinlichkeitsverteilung ist die *Binomialverteilung*. Diese ergibt sich, wenn man einen Bernoulli-Versuch n-mal unabhängig wiederholt und dabei die Gesamtzahl der Erfolge bestimmt. Für ein festes Ereignis E sei $P(E) = p$, und Z_i sei die Indikatorvariable für „Erfolg“ beim i-ten Versuch, $i = 1, \ldots, n$.

Dann ist $Z = \sum_{i=1}^{n} Z_i$ *binomial verteilt* mit Parametern n und p, und es gilt für $i = 0, 1, \ldots, n$:

$$\Pr[\, Z = i \,] \quad = \quad \binom{n}{i} p^i (1-p)^{n-i}.$$

Der Erwartungswert von Z ist $E(Z) = E(\sum_{i=1}^{n} Z_i) = \sum_{i=1}^{n} E(Z_i) = nP(E) = np$. Der überwiegende Teil der Wahrscheinlichkeitsmasse ist bei einer binomial verteilten Zufallsvariablen Z auf diejenigen Werte i verteilt, die sich in der Nähe des Erwartungswertes $E(Z) = np$ befinden. Für wachsendes n nimmt die Wahrscheinlichkeit sogar exponentiell in n ab, daß Z einen Wert in der Nähe des einen oder anderen Endes des Intervalls $[0, n]$ annimmt. Tatsächlich gibt es eine Vielzahl von Wahrscheinlichkeitsabschätzungen (sogenannte „Chernoff-Schranken“) für diese Randbereiche (siehe zum Beispiel [Che52, Pap94, Sch95]). Für unsere Zwecke genügt jedoch folgender einfach zu beweisende Spezialfall (vgl. auch [Sch86]).

Lemma 2.1 *Sei E ein Ereignis, das mit einer positiven Wahrscheinlichkeit von höchstens $p < 1/2$ eintritt, und sei $q = 1 - p$. Dann ist die Wahrscheinlichkeit, daß E bei n unabhängigen Versuchen ($n \geq 2$) mindestens $n/2$-mal eintritt, kleiner als $(4pq)^{n/2}$.*

Beweis. Sei Z die Zufallsvariable, die die Anzahl der Erfolge bei n unabhängigen Versuchen angibt. Dann ergibt sich wegen $p/q < 1$,

$$\begin{aligned}
\Pr[\, Z \geq n/2 \,] \quad &\leq \quad \sum_{i=\lceil n/2 \rceil}^{n} \binom{n}{i} p^i q^{n-i} \\
&= \quad \sum_{i=\lceil n/2 \rceil}^{n} \binom{n}{i} p^{n/2} q^{n/2} (p/q)^{i-n/2} \\
&< \quad \sum_{i=\lceil n/2 \rceil}^{n} \binom{n}{i} p^{n/2} q^{n/2} \\
&< \quad 2^n p^{n/2} q^{n/2} \\
&= \quad (4pq)^{n/2}.
\end{aligned}$$

■

Bei einer Erfolgswahrscheinlichkeit von beispielsweise $p = 1/4$ strebt demnach die Wahrscheinlichkeit, daß bei n unabhängigen Wiederholungen eines Bernoulli-Versuchs mindestens die Hälfte aller Versuche erfolgreich verläuft, schneller als die Funktion $(\sqrt{3}/2)^n$ gegen 0.

2.3 Komplexitätstheorie

Turingmaschinen-Modell

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 $b_1, \ldots, b_k$, das heißt der Kopf auf Band i wird um $b_i \in \{-1, 0, 1\}$ Felder nach rechts bewegt.

Die Turingtafel einer *nichtdeterministischen* Maschine M kann dagegen 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.

Eine *Konfiguration* von M besteht aus dem momentanen Zustand, den Bandinschriften und den Positionen der Bandköpfe. 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. Wird in k_m der Zustand z_a angenommen, 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.

Die Anzahl aller akzeptierenden Rechnungen von M bei Eingabe x bezeichnen wir mit $acc_M(x)$. Für eine deterministische Turingmaschine gilt notwendigerweise $acc_M(x) \leq 1$.

Zeit- und Platz-Komplexitätsklassen

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

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

Sei $t : \mathcal{N} \to \mathcal{N}$. 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 polynomial zeit- bzw. platzbeschränkt, falls für die Schranke t ein Polynom gewählt werden kann.

Alle Sprachen, die von einer t-zeitbeschränkten (nicht-) deterministischen Turingmaschine erkannt werden, werden in der Komplexitätsklasse *DTIME*$[t(n)]$ (bzw. *NTIME*$[t(n)]$) zusammengefaßt. Entsprechend sind die Klassen *DSPACE*$[t(n)]$ und *NSPACE*$[t(n)]$ definiert.

Die Klassen $\mathcal{P}$, $\mathcal{NP}$, $\mathcal{UP}$ und $\mathcal{F}ew\mathcal{P}$

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

Eine $\mathcal{NP}$-Maschine M heißt *$\mathcal{UP}$-Maschine*, falls sie die Bedingung $acc_M(x) \leq 1$ für alle $x \in \Sigma^*$ erfüllt, das heißt M hat bei Eingaben $x \in L(M)$ genau einen akzeptierenden Pfad. *$\mathcal{UP}$ (Unique Polynomial Time)* [Val76] ist die Klasse aller Sprachen, die von einer $\mathcal{UP}$-Maschine erkannt werden. Eine Sprache L ist in $\mathcal{F}ew\mathcal{P}$

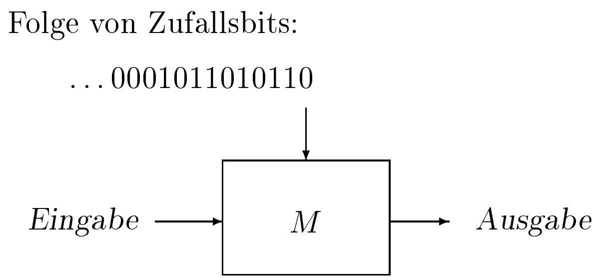
[All85], wenn L von einer $\mathcal{NP}$-Maschine erkannt wird, die für ein Polynom p bei jeder Eingabe x höchstens $p(|x|)$ akzeptierende Rechnungen durchführt, das heißt es gilt

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

für alle $x \in \Sigma^*$.

Probabilistische Berechnungen

Eine *probabilistische* Turingmaschine M ist eine nichtdeterministische Turingmaschine mit einem anderen Akzeptierungsmodus: Wir stellen uns vor, daß M bei jedem nichtdeterministischen Rechenschritt unabhängig mit Wahrscheinlichkeit $1/2$ die eine oder die andere Alternative wählt. Bildlich gesprochen kann M also Zufallsbits von einem idealen Zufallsgenerator abrufen und diese in die Rechnung miteinfließen lassen.



Eine Rechnung von M erhält dadurch die Wahrscheinlichkeit 2^{-t}, wobei t die Anzahl der in der Rechnung ausgeführten nichtdeterministischen Schritte ist. M *akzeptiert* ein Wort $x \in \Sigma^*$, falls M bei Eingabe x mit Wahrscheinlichkeit mindestens $1/2$ eine akzeptierende Rechnung durchführt. Für eine probabilistische Turingmaschine ist $M(x)$ also eine Zufallsvariable und die von M *erkannte Sprache* ist

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

$\mathcal{PP}$ *(Probabilistic Polynomial Time)* [Gil77] ist die Klasse aller von einer probabilistischen polynomial zeitbeschränkten Turingmaschine erkannten Sprachen. Es ist einfach zu sehen, daß $\mathcal{NP}$ in $\mathcal{PP}$ und $\mathcal{PP}$ in $\mathcal{PSPACE}$ enthalten ist [Gil77]. Ein noch junges Resultat von Toda zeigt, daß alle Sprachen der Polynomialzeit-Hierarchie in $\mathcal{P}(\mathcal{PP})$ liegen [Tod91]. Eine interessante Teilklasse von $\mathcal{PP}$ ist die von Wagner [Wag86] eingeführte Klasse $\mathcal{C}_{=}\mathcal{P}$, die für jede $\mathcal{PP}$ Maschine die Sprache $\{x \in \Sigma^* \mid \Pr[M(x) \text{ akzeptiert}] = 1/2\}$ enthält.

Eine weitere wichtige Teilklasse von $\mathcal{PP}$ ist die Klasse $\mathcal{BPP}$ *(Bounded-error Probabilistic Polynomial Time)* [Gil77], die alle Sprachen L enthält, die durch eine probabilistische polynomial zeitbeschränkte Turingmaschine M mit beschränkter Fehlerwahrscheinlichkeit erkannt werden. Das heißt, es gilt für alle x,

$$\Pr[M(x) = L(x)] \quad \geq \quad 3/4.$$

Ein $\mathcal{BPP}$-Algorithmus berechnet also mit großer Wahrscheinlichkeit das richtige Ergebnis. Die Fehlerwahrscheinlichkeit kann durch Ausführung einer polynomialen Anzahl von Rechnungen und anschließendem Mehrheitsentscheid exponentiell verkleinert werden (vgl. Lemma 2.1). Hat M überdies die Eigenschaft, daß sie im Fall $x \notin L$ immer die richtige Antwort ($M(x) = 0$) gibt, so liegt L in der Klasse $\mathcal{RP}$ *(Random Polynomial Time)*. Entsprechend besteht die Klasse *co*-$\mathcal{RP}$ aus allen Sprachen, die von einem $\mathcal{BPP}$-Algorithmus entschieden werden, der im Fall $x \in L$ immer die richtige Antwort ($M(x) = 1$) gibt. $\mathcal{BPP}$-, $\mathcal{RP}$- und auch *co*-$\mathcal{RP}$-Algorithmen geben also mit einer gewissen Wahrscheinlichkeit ein falsches Ergebnis (solche Algorithmen werden auch *Monte-Carlo Algorithmen* genannt). Im Gegensatz hierzu gibt es für alle Sprachen L in $\mathcal{RP} \cap co\text{-}\mathcal{RP}$ einen probabilistischen Algorithmus, der nie ein falsches Ergebnis liefert, sondern lediglich mit einer sehr kleinen Wahrscheinlichkeit „?“ ausgibt (solche Algorithmen werden auch *Las Vegas Algorithmen* genannt). Sind etwa M_1 und M_2 ein $\mathcal{RP}$- bzw. *co*-$\mathcal{RP}$-Algorithmus für L, so gibt folgender Algorithmus M immer das richtige Ergebnis (bzw. ein „?“) aus:

> Bei Eingabe x simuliere die beiden Algorithmen M_1 und M_2 bei Eingabe x. Falls M_1 akzeptiert (das heißt $x \in L$) oder M_2 verwirft (das heißt $x \notin L$), so akzeptiere im ersten Fall und verwerfe im zweiten. Andernfalls gib „?“ aus.

Durch eine polynomiale Anzahl von Wiederholungen der Simulation von M_1 und M_2 kann die Wahrscheinlichkeit für die Ausgabe von „?“ offenbar exponentiell klein gemacht werden. Die Klasse $\mathcal{RP} \cap co\text{-}\mathcal{RP}$, für die ein solcher Algorithmus existiert, wird auch mit $\mathcal{ZPP}$ *(Zero-error Probabilistic Polynomial Time)* [Gil77] bezeichnet. Ein prominentes Mitglied dieser Klasse ist etwa das Primzahlproblem [SS77, Rab76, AH87].

Allgemeiner liegt eine Sprache L in *ZPTIME*$[t(n)]$, falls es eine probabilistische Turingmaschine M gibt, die niemals ein falsches Ergebnis liefert, und die bei allen Eingaben mit einer Wahrscheinlichkeit von höchstens $1/2$ ein „?“ ausgibt.

Berechnung von Funktionen

Ein *Transducer* T ist eine Turingmaschine mit *Ausgabeband*. Dabei ist die von T 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 T deterministisch, so bezeichnen wir die von T bei Eingabe x erzeugte Ausgabe mit $T(x)$, falls T x akzeptiert. Andernfalls ist $T(x)$ undefiniert (hierfür schreiben wir auch $T(x) = \perp$). Ist T dagegen nichtdeterministisch oder probabilistisch, so bezeichnen wir die Menge aller von T bei Eingabe x erzeugten Ausgaben mit *set*-$T(x)$.

Mit $\mathcal{FP}$ bezeichnen wir die Klasse aller totalen Funktionen $f : \Sigma^* \to \Sigma^*$, die von einem deterministischen polynomial zeitbeschränkten Transducer T berechnet werden, das heißt, es gilt $f(x) = T(x)$ für alle $x \in \Sigma^*$. Eine Sprache L heißt $\mathcal{P}$ *-auflistbar*, wenn es eine Funktion $f \in \mathcal{FP}$ gibt mit

$$f(0^n) \quad = \quad \langle x_1, \ldots, x_k \rangle,$$

wobei $x_1, \ldots, x_k$ die Folge aller Wörter in L bis zur Länge n in lexikographischer Reihenfolge darstellt. Es ist klar, daß nur spärliche Sprachen in diesem Sinne auflistbar sein können. Eine Sprache heißt *spärlich*, falls es ein Polynom p gibt, so daß L maximal $p(n)$ viele Wörter bis zur Länge n enthält. Beispielsweise sind alle Sprachen über einem einelementigen Alphabet (sog. *tally* Mengen) spärlich. Wir fassen alle spärlichen (tally) Sprachen in der Klasse SPARSE (bzw. TALLY) zusammen.

Wir werden auch mehrwertige Funktionen $f : \Sigma^* \to \Sigma^*$ betrachten. Diese ordnen jedem $x \in \Sigma^*$ eine beliebige endliche Teilmenge von Σ^* zu, die wir mit *set*-$f(x)$ bezeichnen. $\mathcal{NPMV}$ ist die Klasse aller (partiellen) mehrwertigen Funktionen, die von einem nichtdeterministischen polynomial zeitbeschränkten Transducer T berechnet werden, das heißt, es gilt *set*-$f(x) =$ *set*-$T(x)$ für alle $x \in \Sigma^*$. Eine totale mehrwertige Funktion f ist in $\mathcal{FZPP}$, falls es ein Polynom p und einen probabilistischen Polynomialzeit-Transducer T gibt, so daß für alle $x \in \Sigma^*$ gilt:

- T erzeugt bei Eingabe x mit einer Wahrscheinlichkeit von mindestens $1/2$ eine Ausgabe und
- *set*-$f(x) =$ *set*-$T(x)$.

Es ist leicht zu sehen, daß eine Sprache L genau dann in $\mathcal{ZPP}$ enthalten ist, wenn ihre charakteristische Funktion in $\mathcal{FZPP}$ liegt.

Anzahlbestimmende Funktionen

In vielen Anwendungen geht es darum, die Anzahl von Objekten mit einer bestimmten Eigenschaft zu bestimmen. Ein typisches Beispiel hierfür ist die Bestimmung der Anzahl von Lösungen bei $\mathcal{NP}$ Problemen. Valiant [Val79] hat diese Funktionen zur Klasse

$$\#\mathcal{P} \quad =_{\mathit{def}} \quad \{acc_M \mid M \text{ ist eine } \mathcal{NP}\text{-Maschine}\}$$

zusammengefaßt. Eine Verallgemeinerung von $\#\mathcal{P}$ bildet die Klasse $\mathcal{S}pan\mathcal{P}$, die in [KST89] eingeführt und untersucht wurde:

$$\mathcal{S}pan\mathcal{P} \quad =_{\mathit{def}} \quad \{f \mid \text{es gibt einen } \mathcal{NP}\text{-Transducer } T \text{ mit } f(x) = \|set\text{-}T(x)\|\}.$$

Während also eine $\#\mathcal{P}$ Funktion die Anzahl aller Rechnungen einer $\mathcal{NP}$-Maschine bestimmt, gibt eine $\mathcal{S}pan\mathcal{P}$ Funktion die Anzahl aller Ausgabewerte an, die ein $\mathcal{NP}$-Transducer bei einer gegebenen Eingabe produziert.

Ein wichtiges Mittel, um aus bereits bekannten Sprach- oder Funktionenklassen weitere Klassen aufzubauen, bilden sogenannte *Operatoren*, von denen wir weitere im nächsten Unterabschnitt einführen werden. Die folgenden mit min- und max-bezeichneten Operatoren wurden im Kontext von Anzahlfunktionen u.a. in [Vol94] untersucht.

Definition 2.2 *Sei $\mathcal{FC}$ eine Funktionenklasse. Dann bezeichnen wir mit* max-$\mathcal{FC}$ *die Klasse aller Funktionen f, für die es eine Funktion $h \in \mathcal{FC}$ und ein Polynom p gibt, so daß für alle $n \geq 0$ und alle $x \in \{0,1\}^n$,*

$$f(x) \quad = \quad \max_{|y|=p(n)} h(x,y).$$

Die Klasse min-$\mathcal{FC}$ *ist analog definiert.*

Relativierte Berechnungen

Eine *Orakelmaschine M* ist eine mit einem speziellen Orakelband ausgestattete Turingmaschine, deren Zustandsmenge Z zusätzlich die drei Zustände z_{frage} *(Fragezustand)*, z_{ja} *(ja-Zustand)* und z_{nein} *(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_{frage} übergeht. Die Frage wird nun dadurch beantwortet, daß das Orakelband gelöscht wird, und M im Fall $y \in A$ in den Zustand z_{ja} und andernfalls in den Zustand z_{nein} versetzt wird.

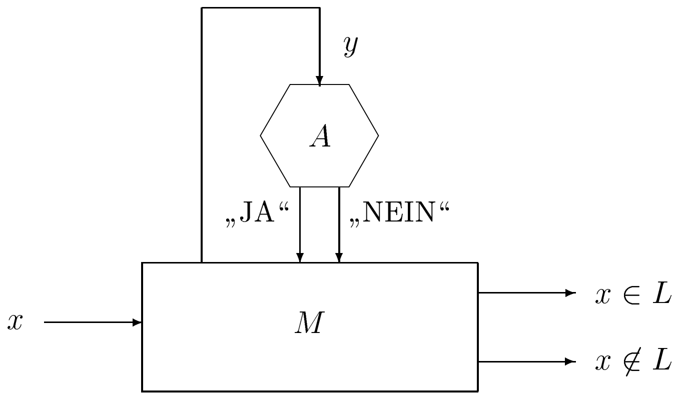
Alle betrachteten Maschinenmodelle lassen sich mit einem solchen Orakelmechanismus ausrüsten. Wir schreiben dann einfach M^A, T^A, acc_M^A oder $L(M, A)$, um die Abhängigkeit vom Orakel auszudrücken. Weiter definieren wir für eine durch eine Menge $\mathcal{C}$ von Turingmaschinen definierte Komplexitätsklasse $\mathcal{K} = \{L(M) \mid M \in \mathcal{C}\}$ (und ganz entsprechend für eine Klasse $\mathcal{F}$ von Funktionen) die bzgl. eines Orakels *A relativierte Klasse*

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

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

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

$\mathcal{P}(A)$ ist also die Klasse aller Sprachen L, die von einer $\mathcal{P}$-Orakelmaschine M relativ zum Orakel A erkannt werden.



Stellt hierbei M bei Eingaben der Länge n nur $O(\log n)$ viele Fragen an A, so nennen wir M eine Θ-Maschine und schreiben $L \in \Theta(A)$ [Wag90, LS91]. $\Theta(A)$ ist in der Klasse $\mathcal{P}_{tt}(A)$ enthalten, die aus allen Sprachen besteht, die relativ zu A von einer *nicht-adaptiven* $\mathcal{P}$-Orakelmaschine M akzeptiert werden. Eine nicht-adaptive Orakelmaschine stellt sämtliche Fragen unabhängig voneinander (das heißt, sie benützt nicht die Antworten auf frühere Fragen, um die nächste Frage zu generieren).

Es ist bekannt, daß auch umgekehrt jede $\mathcal{P}_{tt}(\mathcal{NP}(A))$ Berechnung durch eine Berechnung in $\Theta(\mathcal{NP}(A))$ simuliert werden kann [Hem87, BH91]. Dies impliziert wegen

$$\Theta(\Theta(\mathcal{NP}(A))) \subseteq \mathcal{P}_{tt}(\mathcal{P}_{tt}(\mathcal{NP}(A))) \subseteq \mathcal{P}_{tt}(\mathcal{NP}(A)) \subseteq \Theta(\mathcal{NP}(A))$$

die Abgeschlossenheit aller Klassen der Form $\Theta(\mathcal{NP}(\mathcal{A}))$ unter dem Θ-Operator. Für die analog definierten Funktionenklassen $\mathrm{F}\Theta(\mathcal{NP}(A))$ und $\mathcal{FP}_{tt}(\mathcal{NP}(A))$ hingegen gilt im allgemeinen nur die (triviale) Inklusion von links nach rechts.

Wird eine Sprache L in $\mathcal{NP}(A)$ von einer $\mathcal{NP}$-Orakelmaschine M akzeptiert, die auf jedem Berechnungspfad maximal eine Frage an A stellt, so schreiben wir $L \in \mathcal{NP}_1(A)$. Hat M zusätzlich die Eigenschaft, daß sie genau bei Erhalt einer positiven (negativen) Antwort akzeptiert, so kennzeichnen wir dies durch die Schreibweise $L \in \mathcal{NP}_{\mathrm{m}}(A)$ (bzw. $L \in \mathcal{NP}_{\overline{\mathrm{m}}}(A)$).

Die Polynomialzeit-Hierarchie

Die einzelnen Stufen der *Polynomialzeit-Hierarchie* [MS73, Sto77, Wra75, Wra77, Wag90] sind induktiv wie folgt definiert:

$$\begin{aligned}
\Sigma_0^{\mathrm{P}} &= \Pi_0^{\mathrm{P}} = \Delta_0^{\mathrm{P}} = \Theta_0^{\mathrm{P}} = \mathcal{P};\\
\Theta_{k+1}^{\mathrm{P}} &= \Theta(\Sigma_k^{\mathrm{P}}),\ k \geq 0;\\
\Delta_{k+1}^{\mathrm{P}} &= \mathcal{P}(\Sigma_k^{\mathrm{P}}),\ k \geq 0;\\
\Sigma_{k+1}^{\mathrm{P}} &= \mathcal{NP}(\Sigma_k^{\mathrm{P}}),\ k \geq 0;\\
\Pi_{k+1}^{\mathrm{P}} &= \mathit{co}\text{-}\mathcal{NP}(\Sigma_k^{\mathrm{P}}),\ k \geq 0;\\
\mathcal{PH} &= \bigcup_{k\geq 0} \Sigma_k^{\mathrm{P}}.
\end{aligned}$$

Zu den Delta- und Theta-Stufen lassen sich ganz analog auch funktionale Klassen $\mathcal{F}\Delta_k^{\mathrm{P}}$ und $\mathcal{F}\Theta_k^{\mathrm{P}}$ definieren. Es ist nicht bekannt, ob die Polynomialzeit-Hierarchie kollabiert, das heißt ob es ein k gibt mit $\mathcal{PH} = \Sigma_k^{\mathrm{P}}$. Aus der (unwahrscheinlichen) Annahme $\Sigma_k^{\mathrm{P}} = \Pi_k^{\mathrm{P}}$ folgt jedoch ein Kollaps auf die k-te Stufe ($\mathcal{PH} = \Sigma_k^{\mathrm{P}}$).

Die Sigma- und Pi-Stufen der Polynomialzeit-Hierarchie können neben obiger Orakelmaschinen-Charakterisierung alternativ durch beschränkt alternierende Turingmaschinen [CKS81] oder mittels polynomial längen-beschränkter Quantifizierungen [Sto77, Wra77] charakterisiert werden. Wir werden in dieser Arbeit auch von letzterer Gebrauch machen.

Eine Sprache L ist genau dann in Σ_k^{P}, wenn es eine polynomial zeitbeschränkte Orakelmaschine M und ein Polynom p gibt, so daß für alle $x \in \Sigma^*$ gilt:

$$x \in L \ \Leftrightarrow \exists y_1\, \forall y_2 \ldots Q y_k \,\langle x, y_1, y_2, \ldots, y_k\rangle \in L(M).$$

Hierbei ist $Q = \forall$, falls k gerade ist, und $Q = \exists$ sonst, und die Quantoren erfassen alle Wörter bis zur Länge $p(|x|)$. Eine weitere in der strukturellen Komplexitätstheorie intensiv erforschte Hierarchie ist die von Wagner [Wag86] eingeführte Counting-Hierarchie, die als die Vereinigung (mit $\mathcal{CH}$ bezeichnet) über alle Klassen der Form $\mathcal{PP}$, $\mathcal{PP}(\mathcal{PP})$, $\mathcal{PP}(\mathcal{PP}(\mathcal{PP}))$, usw. gebildet wird.

Reduzierbarkeiten

Eine Sprache A ist auf eine Sprache B *many-one reduzierbar* ($A \leq_m^P B$), falls eine Funktion $f \in \mathcal{FP}$ existiert mit $f^{-1}(B) = A$, das heißt $x \in A \Leftrightarrow f(x) \in B$ 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 $\mathcal{NP}$-Vollständigkeit gelang [Coo71, Kar72], ist das Erfüllbarkeitsproblem für die aus Aussagenvariablen und den logischen Junktoren $\neg, \vee$ und $\wedge$ aufgebauten Booleschen Formeln,

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

Hierbei und in ähnlichen Fällen setzen wir ein festes, aber nicht näher spezifiziertes Kodierungsverfahren 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 Kodierung und dem kodierten Objekt unterscheiden.

Eine schwächere Form von Reduktion ist die *Turing-Reduktion* ($A \leq_T^P B$), die durch $A \in \mathcal{P}(B)$ 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 [LLS75].

Eine weitere wichtige Reduzierbarkeit ist die von Long [Lon82] eingeführte *starke Turing-Reduzierbarkeit.* Wir schreiben $A \leq_T^{SN} B$, falls A in $\mathcal{NP}(B) \cap co\text{-}\mathcal{NP}(B)$ enthalten ist. Die $\leq_T^{SN}$-Reduzierbarkeit ist eine Verallgemeinerung der γ-Reduzierbarkeit (siehe [AM77, GJ79]), die eine many-one Version von $\leq_T^{SN}$ darstellt. Der Beweis des folgenden Satzes (siehe auch [Sch83, Lon85]) beinhaltet eine interessante Technik, mit der ein Orakel A durch ein anderes (möglicherweise „einfacheres“) Orakel B in nichtdeterministischen Berechnungen ersetzt werden kann.

Satz 2.3 [Sel78] *Eine Sprache A ist genau dann $\leq_T^{SN}$-reduzierbar auf eine Sprache B, wenn $\mathcal{NP}(A) \subseteq \mathcal{NP}(B)$ gilt.*

Beweis. Gelte $A \in \mathcal{NP}(B) \cap co\text{-}\mathcal{NP}(B)$ mittels zweier nichtdeterministischer Orakelmaschinen M_1 und M_2, das heißt, $A = L(M_1, B)$ und $\overline{A} = L(M_2, B)$. Sei weiterhin L in $\mathcal{NP}(A)$ mittels einer nichtdeterministischen Orakelmaschine M. Betrachte folgende nichtdeterministische Orakelmaschine M', die L relativ zu B erkennt.

> Bei Eingabe x simuliere $M(x)$ und für jede Orakelfrage y an A rate, ob die Antwort „JA" oder „NEIN" ist. Verifiziere jeweils die geratene Antwort durch Simulation von M_1 (bzw. M_2) bei Eingabe y. Falls M_1 (bzw. M_2) akzeptiert, so fahre in der Simulation von $M(x)$ mit der geratenen Antwort fort. Andernfalls verwerfe die Eingabe x (diese Rechnung kann sich also nicht auf die Zugehörigkeit von x zu $L(M', B)$ auswirken).

Dies beweist die Richtung von links nach rechts. Umgekehrt impliziert die Voraussetzung $\mathcal{NP}(A) \subseteq \mathcal{NP}(B)$ sofort, daß A und $\overline{A}$ in $\mathcal{NP}(B)$ sind. ∎

2.4 Boolesche Schaltkreise und die Komplexitätsklasse $\mathcal{P}/\boldsymbol{poly}$

In diesem Abschnitt betrachten wir Boolesche Schaltkreise, die ein rein kombinatorisches Konzept zur Berechnung von Booleschen Funktionen darstellen. Zwischen der Schaltkreiskomplexität von Booleschen Funktionen (also von endlichen Objekten) und der Turingmaschinen-Komplexität von (unendlichen) Sprachen bestehen interessante Querbezüge, die das Hauptthema dieser Arbeit bilden.

Definition 2.4

a) Ein Boolescher Schaltkreis *mit n Eingängen ist eine Folge $c = (\alpha_1, \ldots, \alpha_m)$, wobei jedes α_i entweder eine Boolesche Konstante ($\alpha_i \in \{0, 1\}$), eine Variable ($\alpha_i \in \{x_1, \ldots, x_n\}$) oder ein Gatter (dargestellt durch ein Tripel (j, k, op) mit $j, k < i$ und $op \in \{\wedge, \vee\}$, bzw. durch ein Paar $(j, \neg)$ mit $j < i$) ist.*

b) Die Größe von c ist m (also gleich der Anzahl aller Gatter in c, sofern wir Eingabegatter und Konstanten mitzählen).

c) *Die von c berechnete n-stellige Boolesche Funktion (diese bezeichnen wir auch mit $c(x)$) ordnet jedem Binärstring $x = x_1 \cdots x_n$ den bei Eingabe x im Gatter α_m (dem sogenannten Ausgabegatter) berechneten Wert zu. Der bei Eingabe $x \in \{0,1\}^n$ in Gatter α_i berechnete Wert $\alpha_i(x)$ ist hierbei induktiv wie folgt definiert:*

$$\alpha_i(x) = \begin{cases} \alpha_i, & \alpha_i \in \{0,1,x_1,\ldots,x_n\}, \\ \alpha_j(x) \wedge \alpha_k(x), & \alpha_i = (j,k,\wedge), \\ \alpha_j(x) \vee \alpha_k(x), & \alpha_i = (j,k,\vee), \\ \neg\alpha_j(x), & \alpha_i = (j,\neg). \end{cases}$$

d) *Sei $f : \{0,1\}^n \to \{0,1\}$ eine Boolesche Funktion. Dann ist die Schaltkreiskomplexität von f (bezeichnet durch $C(f)$) die minimale Größe eines Schaltkreises c mit n Eingängen, der f berechnet.*

Oftmals werden zur Konstruktion auch andere Gatter (außer den hier gewählten $\neg$, $\vee$ und $\wedge$ Gattern) verwendet. Für unsere Betrachtungen ist jedoch nur wichtig, daß die verwendete Basis $\{\neg, \vee, \wedge\}$ vollständig ist, das heißt, daß alle Booleschen Funktionen durch einen Schaltkreis bestehend aus Gattern, die die Basisfunktionen realisieren, berechenbar sind.

Während Turingmaschinen a priori unendliche Objekte berechnen, stellen Schaltkreise ein Berechnungsmodell für endliche Objekte dar. Wollen wir daher Schaltkreise zur Berechnung von (unendlichen) Sprachen heranziehen, so müssen wir diese zuvor in eine Folge von endlichen „Teilstücken“ zerlegen und jedes dieser Teilstücke durch einen anderen Schaltkreis berechnen lassen.

Definition 2.5 *Eine* Schaltkreisfamilie *ist eine Folge von Schaltkreisen c_n, $n \geq 0$, mit der Eigenschaft, daß der n-te Schaltkreis n Eingänge hat. Die von einer solchen Familie berechnete Sprache ist*

$$\{x \in \{0,1\}^* \mid c_{|x|}(x) = 1\}.$$

Sei $s : \mathcal{N} \to \mathcal{N}$ eine Funktion. Wir sagen, eine Sprache A hat Schaltkreise der Größe $s(n)$, *falls es eine Schaltkreisfamilie c_n, $n \geq 0$, gibt, die A berechnet, so daß die Größe von c_n höchstens $s(n)$ ist.*

Mit anderen Worten, interpretieren wir die charakteristische Funktion von $A^{=n}$ als n-stellige Boolesche Funktion, so muß $C(A^{=n}) \leq s(n)$ für alle $n \geq 0$ sein. A hat also Schaltkreise polynomialer Größe, falls $C(A^{=n})$ durch ein Polynom beschränkt werden kann, das heißt, $C(A^{=n}) \in n^{O(1)}$.

Im Gegensatz zu Turingmaschinen stellen somit Boolesche Schaltkreise ein *nicht-uniformes Modell* dar, da für jede Eingabelänge ein anderer Schaltkreis zuständig ist. Der Unterschied zur Turingmaschinen-Komplexität läßt sich sehr schön am Beispiel von tally Sprachen (über einem einelementigen Alphabet) verdeutlichen. Während all diese Sprachen offenbar lineare Schaltkreis-Komplexität haben, läßt sich jedes beliebige Entscheidungsproblem (beispielsweise auch nicht-rekursive wie das Halteproblem) in eine tally Sprache kodieren.

Die in der nächsten Proposition festgehaltene Charakterisierung der Sprachen mit polynomialer Schaltkreis-Komplexität geht auf Pippenger [Pip79] zurück. Diese Äquivalenz inspirierte Karp und Lipton, das fundamentale Konzept der Advice-Klassen einzuführen, mit dem sich nicht-uniforme Komplexitätsklassen in einheitlicher Form darstellen lassen.

Definition 2.6 [KL80] *Sei $\mathcal{C}$ eine Sprachklasse und sei $\mathcal{F}$ eine Klasse von Funktionen von $\mathcal{N}$ nach Σ^*. Dann enthält die Klasse $\mathcal{C}/\mathcal{F}$ alle Mengen A, für die es eine Menge $B \in \mathcal{C}$ und eine Funktion $h \in \mathcal{F}$ gibt mit der Eigenschaft, daß für alle $x \in \Sigma^*$*

$$x \in A \quad \Leftrightarrow \quad \langle x, h(|x|)\rangle \in B$$

gilt. Die Funktion h heißt Advice-Funktion *für A, und B nennen wir die zugehörige* Interpreter-Menge.

Man beachte, daß die Sprache A durch die Interpreter-Menge B und die Advice-Funktion h eindeutig festgelegt ist. Mitunter verwenden wir daher auch die von Karp und Lipton benutzte Schreibweise B/h für A. Aus intuitiver Sicht enthält $\mathcal{C}/\mathcal{F}$ alle Sprachen, die durch Kombination der für die Komplexitätsklasse $\mathcal{C}$ charakteristischen Ressourcen und einem durch eine Funktion in $\mathcal{F}$ zusätzlich zur Verfügung gestellten *Advice* erkannt werden können. Die bei weitem am intensivsten untersuchte Advice-Klasse ist $\mathcal{P}/poly$, die durch Kombination der Interpreter-Klasse $\mathcal{P}$ mit der Klasse *poly* aller Advice-Funktionen h entsteht, deren Länge polynomial beschränkt ist, das heißt,

$$poly \quad =_{def} \quad \{h \mid |h(n)| \in n^{O(1)}\}.$$

Proposition 2.7 [Pip79] *Eine Sprache hat genau dann Schaltkreise polynomialer Größe, wenn sie in $\mathcal{P}/poly$ enthalten ist.*

Beweis. Sei A eine Sprache, berechnet durch eine Familie c_n, $n \geq 0$, von Schaltkreisen polynomialer Größe. Definiere die Advice-Funktion $h(n) = c_n$, die jede Eingabelänge n auf (eine Kodierung des Schaltkreises) c_n abbildet. Da die Interpreter-Sprache

$$I_{univ} \;=_{def}\; \{\langle x, c\rangle \mid c \text{ ist ein Schaltkreis mit } |x| \text{ Eingängen und } c(x) = 1\}$$

offensichtlich in $\mathcal{P}$ entscheidbar ist, folgt $A \in \mathcal{P}/poly$.

Die Rückrichtung ist eine unmittelbare Folgerung aus einem Resultat von Ladner [Lad75a], das zeigt, daß alle Sprachen in $\mathcal{P}$ Schaltkreise polynomialer Größe haben. Der zusätzliche Advice-String, der bei einer Sprache in $\mathcal{P}/poly$ einen Bestandteil der Eingabe ausmacht, kann hierbei fest in den entsprechenden Schaltkreis „verdrahtet“ werden. ■

Aus obigem Beweis ergibt sich die interessante Konsequenz, daß es eine *universelle* Interpreter-Sprache I_{univ} in folgendem Sinne gibt: Für jede Interpreter-Sprache $B \in \mathcal{P}$ gibt es ein Polynom p, so daß für jede Advice-Funktion h eine Advice-Funktion h' existiert mit

- $B/h = I_{univ}/h'$ und
- für alle n ist $|h'(n)| \leq p(n + |h(n)|)$.

Unter Verwendung der Charakterisierung von Karp und Lipton kann relativ einfach gezeigt werden, daß alle Sprachen in $\mathcal{BPP}$ Schaltkreise polynomialer Größe haben.

Proposition 2.8 [BG81] *$\mathcal{BPP} \subseteq \mathcal{P}/poly$.*

Beweis. Sei A eine Sprache in $\mathcal{BPP}$ und sei M eine entsprechende probabilistische Turingmaschine. Wie weiter oben bereits erwähnt, können wir annehmen, daß die Fehlerwahrscheinlichkeit (durch Ausführen einer polynomialen Anzahl von Rechnungen und anschließendem Mehrheitsentscheid) bei Eingaben der Länge n kleiner als 2^{-2n} ist:

$$\Pr[M(x) \neq L(x)] \;<\; 2^{-2|x|}.$$

Sei $p(n)$ eine obere Schranke für die Anzahl von Zufallsbits, die M bei einer Eingabe der Länge n benötigt. Für eine zufällige Folge w von $p(n)$ Zufallsbits beträgt demnach die Wahrscheinlichkeit, daß es eine Eingabe $x \in \{0,1\}^n$ gibt, die M unter Verwendung der Zufallsbitfolge w inkorrekt entscheidet, höchstens $2^n \cdot 2^{-2n} < 1$.

Folglich muß es für jede Eingabelänge n eine geeignete Zufallsbitfolge w_n *geben*,[1] bei deren Verwendung M auf allen Eingaben der Länge n die korrekte Entscheidung trifft. Da die Länge von w_n exakt $p(n)$ ist, und da die von M bei einer Eingabe x bei Zufallsbitfolge w_n ausgeführte Rechnung von einer $\mathcal{P}$-Maschine simuliert werden kann, folgt $A \in \mathcal{P}/poly$. ■

Damit sind natürlich auch die $\mathcal{BPP}$-Teilklassen $\mathcal{RP}$, *co*-$\mathcal{RP}$ und $\mathcal{ZPP}$ in $\mathcal{P}/poly$ enthalten. Wir erwähnen zum Abschluß dieses Kapitels eine weitere Charakterisierung von $\mathcal{P}/poly$ als die Klasse aller Sprachen A, die auf eine spärliche Menge Turing-reduzierbar sind. Diese Beobachtung wird A. Meyer in [BH77] zugesprochen. Äquivalent hierzu ist auch, daß A auf eine tally Menge Turing-reduzierbar ist [Har83].

[1] Die hierbei verwendete Argumentation, durch eine Wahrscheinlichkeitsabschätzung die Existenz eines Objektes mit gewissen Eigenschaften nachzuweisen, ist unter dem Namen „Probabilistische Methode“ bekannt.

Kapitel 3

Universale Hash-Verfahren

Die ursprünglich von Carter und Wegman [CW79] stammende Methode, zufällig gewählte Hashfunktionen für die Generierung von Adressen für Datenschlüssel zu verwenden, wurde erstmals von Sipser [Sip83] im Bereich der strukturellen Komplexitätstheorie eingesetzt. Sipser verwendete sie, um die Anzahl von akzeptierenden Pfaden bei $\mathcal{BPP}$ Berechnungen abzuschätzen, und damit die Zugehörigkeit aller $\mathcal{BPP}$ Sprachen zu Σ_2^{P} zu beweisen (diese wurde unabhängig davon mit einer etwas anderen Technik auch in [Lau83] bewiesen). Die universale Hash-Methode fand daraufhin eine vielfache Anwendung als Beweistechnik und wurde unter anderem in den Arbeiten [Sto85, JVV86, KST89, Pio91, Gav92, Köb94, KW95] weiterentwickelt.

In diesem Kapitel stellen wir eine Reihe von grundlegenden Resultaten zusammen, die auf der Beweismethode des universalen Hashings beruhen. Die in Abschnitt 3.3 dargestellten Ergebnisse sind bislang unveröffentlicht, während die in Abschnitt 3.4 enthaltenen Resultate in [KW95] erscheinen werden.

3.1 Die Methode des Universalen Hashings

Universale Hash-Verfahren wurden von Carter und Wegman [CW79] als eine elegante Methode eingeführt, die Häufigkeit des Auftretens von Kollisionen weitgehend unabhängig davon zu machen, welche Schlüssel in einer bestimmten Anwendung tatsächlich auftreten. Für dieses Vorgehen, das man intuitiv damit umschreiben kann, daß man „die Würfel dem Algorithmus in die Hand gibt“, gibt es viele Beispiele in der Informatik. Eines der bekanntesten ist die probabilistische Variante von Quicksort, bei der durch zufällige Wahl des Pivot-Elementes erreicht wird, daß die zu erwartende Laufzeit vollkommen unabhängig davon ist, in welcher Reihenfolge sich die zu sortierenden Elemente vor Beginn des Sortiervorgangs befinden.

VEREINBARUNG

Eine **Hashfunktion** $h : A \to B$ ist eine Abbildung von einem **Schlüsselraum** A, der alle potentiell vorkommenden **Schlüssel** umfaßt, in einen **Adreßraum** B, der alle zur Verfügung stehenden **Adressen** beinhaltet.
Berechnet eine Hashfunktion h für zwei unterschiedliche Schlüssel x und y dieselbe Adresse $h(x) = h(y)$, so spricht man von einer **Kollision** zwischen x und y unter h.

Bei einer konkreten Anwendung wird h dazu verwendet, für einen Datensatz mit zugehörigem Schlüssel x eine Adresse $h(x)$ zu berechnen, unter der dieser Datensatz abgespeichert und bei späteren Zugriffen wieder aufgefunden werden kann. Hierbei ist in der Regel der zur Verfügung stehende Adreßraum B deutlich kleiner als der Raum A aller potentiell vorkommenden Schlüssel. Die Anzahl der tatsächlich auftretenden Schlüssel sollte jedoch nur einen Prozentsatz von nicht mehr als 80% der Gesamtheit aller verfügbaren Adressen bilden, da ab dieser Marke die Anzahl der Kollisionen erfahrungsgemäß stark ansteigt.

Die Anzahl der bei einer konkreten Anwendung auftretenden Kollisionen hängt natürlich sehr stark davon ab, welche Schlüssel tatsächlich in Erscheinung treten. Es ist nicht auszuschließen, daß gerade solche Schlüssel durch die Anwendung bevorzugt werden, die unter einer fest gewählten Hashfunktion zu vermehrten Kollisionen führen (vergleichbar etwa dem paradoxerweise ungünstigen Fall einer fast schon sortierten Eingabeliste bei Quicksort). Um der Möglichkeit einer solchen Tendenz entgegenzuwirken, haben Carter und Wegman vorgeschlagen, zur Berechnung der Adressen anstelle einer festen Hashfunktion eine zufällig gewählte Hashfunktion aus einer geeigneten Klasse zu verwenden.

Definition 3.1 *Eine Klasse H von Hashfunktionen von A nach B heißt* universal, *falls für alle Schlüssel x und y in A mit $x \neq y$ gilt,*

$$\mathrm{Pr}_{h\in H}[\,h(x) = h(y)\,] \quad = \quad \frac{1}{\|B\|}.$$

Hierbei bezeichnet $\mathrm{Pr}_{h\in H}[\,h(x) = h(y)\,]$ die Wahrscheinlichkeit, daß eine unter Gleichverteilung gewählte Hashfunktion $h \in H$ zu einer Kollision zwischen den beiden Schlüsseln x und y führt.

H ist also dann universal, wenn eine zufällig aus H gewählte Hashfunktion in bezug auf ein beliebiges Schlüsselpaar mit genau derselben Wahrscheinlichkeit auf eine Kollision führt, wie dies bei einer zufälligen Wahl zweier Adressen aus B (mit Zurücklegen) der Fall wäre.

In ihrer fundamentalen Arbeit [CW79] geben Carter und Wegman bereits mehrere konkrete Beispiele für universale Klassen von Hashfunktionen an. Darunter befindet sich auch die folgende, die eine wichtige Rolle bei unseren Betrachtungen spielen wird.

VEREINBARUNG

Wir bezeichnen die Klasse aller linearen Funktionen von $\{0,1\}^m$ in $\{0,1\}^k$ mit $\mathcal{L}in[m;k]$. Dabei werden die Räume $\{0,1\}^m$ und $\{0,1\}^k$ als Vektorräume über dem zwei-elementigen Körper $(\{0,1\},\oplus,\cdot\,)$ aufgefaßt, den wir wie üblich mit GF[2] bezeichnen.

Jede lineare Hashfunktion h von der Schlüsselmenge $A = \{0,1\}^m$ in den Adreßraum $B = \{0,1\}^k$ entspricht eineindeutig einer Booleschen $(m \times k)$-Matrix

$$C = \begin{pmatrix} c_{11} & \dots & c_{1k} \\ \vdots & \ddots & \vdots \\ c_{m1} & \dots & c_{mk} \end{pmatrix}.$$

Um also eine lineare Hashfunktion aus der Klasse $\mathcal{L}in[m;k]$ unter Gleichverteilung auszuwählen, genügt es, eine Folge von $m \cdot k$ unabhängigen Zufallsbits zu erzeugen und diese zu einer $(m \times k)$-Binärmatrix zusammenzusetzen. Auf diese Weise wird jede Hashfunktion $h \in \mathcal{L}in[m;k]$ mit derselben Wahrscheinlichkeit 2^{-mk} generiert.

Sei h eine lineare Hashfunktion aus $\mathcal{L}in[m;k]$, dargestellt durch eine $(m \times k)$-Binärmatrix C wie oben. Für jeden Schlüssel $x = x_1 \cdots x_m \in \{0,1\}^m$ berechnet sich dann die zugehörige Adresse $h(x)$ zu

$$h(x_1 \cdots x_m) = (x_1 \cdots x_m) \begin{pmatrix} c_{11} & \dots & c_{1k} \\ \vdots & \ddots & \vdots \\ c_{m1} & \dots & c_{mk} \end{pmatrix} = (y_1 \cdots y_k),$$

wobei

$$y_j = (x_1 \cdot c_{1j}) \oplus (x_2 \cdot c_{2j}) \oplus \cdots \oplus (x_m \cdot c_{mj})$$

das Skalarprodukt aus x und der j-ten Spalte der Binärmatrix C ist.

Beispiel 1 *Sei $h \in \mathcal{L}in[4;3]$ diejenige lineare Hashfunktion, die durch die (4×3)-Binärmatrix*

$$C = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$$

dargestellt wird. Dann berechnet h beispielsweise für den Schlüssel $x = 1011$ die Adresse

$$h(1011) = (1\,0\,1\,1)\begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}\begin{matrix} \leftarrow \\ \\ \leftarrow \\ \leftarrow \end{matrix} = (1\,0\,0).$$

Anschaulich ergibt sich $h(x)$ durch bitweises Aufaddieren derjenigen Zeilen von C, deren Positionen jeweils einer „1“ in x entsprechen. Im obigen Beispiel also der ersten, dritten und vierten Zeile: $h(1011) = 101 \oplus 010 \oplus 011 = 100$. Wie man leicht nachrechnet, wird der Schlüssel $y = 0110$ ebenfalls auf die Adresse $h(0110) = 110 \oplus 010 = 100$ abgebildet. Die Hashfunktion h führt also zu einer Kollision zwischen den beiden Schlüsseln 1011 und 0110.

Wir wollen nun beweisen, daß $\mathcal{L}in[m;k]$ tatsächlich eine universale Klasse von Hashfunktionen bildet. Hierbei wird sich das folgende Lemma als sehr nützlich erweisen.

Lemma 3.2 *Sei z ein beliebiger Schlüssel aus $\{0,1\}^m - \{0^m\}$. Dann wird z von einer unter Gleichverteilung aus der Klasse $\mathcal{L}in[m;k]$ gewählten Hashfunktion h mit Wahrscheinlichkeit 2^{-k} auf die Adresse 0^k abgebildet.*

Beweis. Sei $y_1 \cdots y_k$ die (zufällige) Adresse, die eine unter Gleichverteilung gewählte $(m{\times}k)$-Binärmatrix C (bzw. die dadurch dargestellte Hashfunktion) für den (festen) Schlüssel z berechnet. Da y_j das Skalarprodukt aus z und der j-ten Spalte der Binärmatrix C ist, und da die einzelnen Bits von C unabhängig gewählt werden, besteht folglich auch zwischen den einzelnen Bits $y_1 \cdots y_k$ der durch h berechneten Adresse keinerlei stochastische Abhängigkeit. Es genügt also, für jedes einzelne $j \in \{1,\ldots,k\}$ die Wahrscheinlichkeit für das Ereignis $y_j = 0$ zu bestimmen, und diese Wahrscheinlichkeiten aufzumultiplizieren.

Sei S_0 die Menge aller Spaltenvektoren in $\{0,1\}^m$, die auf das Ergebnis $y_j = 0$ führen, und sei S_1 entsprechend definiert. Da $z = z_1 \cdots z_m \neq 0^m$ ist, muß es einen Index

$i \in \{1, \ldots, m\}$ mit $z_i = 1$ geben. Da das Flippen des i-ten Bits eines gegebenen Spaltenvektors offensichtlich eine Bijektion zwischen S_0 und S_1 darstellt, muß jede der beiden Mengen genau die Hälfte aller Vektoren aus $\{0,1\}^m$ enthalten. Da dies $\Pr[\, y_j = 0\,] = \Pr[\, y_j = 1\,] = 1/2$ impliziert, ist hiermit der Beweis erbracht. ∎

Satz 3.3 [CW79] *Für alle $m, k \geq 1$ ist die Klasse $\mathcal{L}in[m;k]$ aller linearen Hashfunktionen von $\{0,1\}^m$ in $\{0,1\}^k$ universal.*

Beweis. Die Universalität von $\mathcal{L}in[m;k]$ ergibt sich nun aufgrund der Linearitätseigenschaft aller in dieser Klasse enthaltenen Funktionen aus Lemma 3.2 ohne Mühe wie folgt. Seien $x, y \in \{0,1\}^m$ zwei verschiedene Schlüssel. Dann ist wegen $x \oplus y \neq 0^m$,

$$\begin{aligned} \Pr[\, h(x) = h(y)\,] &= \Pr[\, h(x) \oplus h(y) = 0^k\,] \\ &= \Pr[\, h(x \oplus y) = 0^k\,] \\ &= 2^{-k}. \end{aligned}$$

∎

Angenommen, wir verwenden in einer konkreten Anwendung eine zufällig aus einer universalen Klasse H gewählte Hashfunktion h. Tritt nun bei dieser Anwendung eine (bei der Festlegung von h noch unbekannte) Menge Y von Schlüsseln auf, so können wir für jeden festen Schlüssel $x \in A$, für den als nächstes eine Adresse zur Verfügung gestellt werden soll, die Wahrscheinlichkeit für das Auftreten einer Kollision abschätzen. Bezeichnen wir das Ereignis, daß h eine Kollision zwischen dem Schlüssel x und einem der in der Menge Y enthaltenen Schlüssel verursacht, mit $\mathit{Kollision}(x, Y, h)$, so erhalten wir

$$\begin{aligned} \Pr[\, \mathit{Kollision}(x,Y,h)\,] &= \Pr[\, \exists y \in Y : x \neq y \wedge h(x) = h(y)\,] \\ &\leq \sum_{y \in Y, x \neq y} \Pr[\, h(x) = h(y)\,] \\ &\leq \frac{\|Y\|}{\|B\|}. \end{aligned}$$

Das heißt also, die Wahrscheinlichkeit für eine Kollision zwischen einem festen Schlüssel $x \in A$ und einer Schlüsselmenge Y ist keinesfalls größer als der *Belegungsfaktor* $\beta = \|Y\|/\|B\|$, ganz gleichgültig, aus welchen Schlüsseln sich die Menge Y tatsächlich zusammensetzt.

In Verallgemeinerung dieser Fragestellung erhalten wir die folgende Abschätzung für die Wahrscheinlichkeit, daß es zwischen mindestens einem Schlüssel x aus einer vorgegebenen Menge $X \subseteq A$ zu einer Kollision mit der Schlüsselmenge Y kommt:

$$\begin{aligned} \Pr[\,\mathit{Kollision}(X,Y,h)\,] &\le \sum_{x \in X} \Pr[\,\mathit{Kollision}(x,Y,h)\,] \\ &\le \|X\| \cdot \|Y\| \,/\, \|B\|. \end{aligned}$$

Insbesondere kann also die Wahrscheinlichkeit für eine Kollision innerhalb einer festen Schlüsselmenge X, die aus s Schlüsseln besteht, folgendermaßen nach oben beschränkt werden:

$$\begin{aligned} \Pr[\,\mathit{Kollision}(X,h)\,] &\le \sum_{x \in X} \Pr[\,\mathit{Kollision}(x, X - \{x\}, h)\,] \\ &\le s(s-1)/\|B\|. \end{aligned}$$

3.2 Abschätzung der Anzahl von Lösungen bei $\mathcal{NP}$-Problemen

Für viele $\mathcal{NP}$ Probleme ist neben dem Entscheidungsproblem (nämlich festzustellen, ob zu einer Eingabe x mindestens eine Lösung existiert) auch das zugehörige Anzahlproblem von großem praktischen Interesse. Dieses besteht darin, für eine Eingabe x festzustellen, wieviele verschiedene Lösungen für diese Eingabe vorhanden sind. Ein Beispiel für diese Art von Problemstellung ist etwa, für einen gegebenen Graphen G herauszufinden, wieviele verschiedene Wege zwischen zwei ausgezeichneten Knoten s und t existieren.

Die Methode des universalen Hashings bildet die Grundlage für eine sehr elegante Technik, mit der sich die Anzahl von Objekten mit gewissen Eigenschaften in einem vorgegebenen Raum *näherungsweise* bestimmen läßt. Sei etwa ein bestimmtes $\mathcal{NP}$ Problem vorgegeben. Dann können wir jede (potentielle) Lösung für eine Eingabe x durch einen Binärstring $y \in \{0,1\}^m$ einer nur von x abhängigen Länge m kodieren. Das heißt, die Gesamtheit aller Lösungen für x bildet eine Teilmenge S_x von $\{0,1\}^m$. Von Sipser stammt nun die Idee, universale Hashfunktionen $h \in \mathcal{H}[m;k]$ von $\{0,1\}^m$ nach $\{0,1\}^k$ dazu zu verwenden, die Größe von S_x approximativ zu bestimmen:

> Versuche für verschiedene Werte von k, die Lösungsmenge S_x durch eine Hashfunktion $h \in \mathcal{H}[m;k]$ kollisionsfrei abzubilden. Ziehe dann aus der Information, für welche Werte von k dies notwendigerweise mit einer

Kollision innerhalb der Lösungsmenge S_x verbunden ist, Rückschlüsse auf die Anzahl der vorhandenen Lösungen.

Sind hierbei $\mathcal{H}[m;k]$, $m,k \geq 1$, universale Klassen von Hashfunktionen, die den Suchraum $\{0,1\}^m$ auf den Raum $\{0,1\}^k$ abbilden, so ergibt sich mit dieser Methode die folgende Abschätzung für die Größe von S_x. Sei k_{min} das kleinste k, für das eine kollisionsfreie Abbildung aller Lösungen in den Adreßraum $\{0,1\}^k$ möglich ist.

- Da es eine Hashfunktion $h \in \mathcal{H}[m;k_{min}]$ gibt, die alle Lösungen $y \in S_x$ kollisionsfrei in den Adreßraum $\{0,1\}^{k_{min}}$ abbildet, kann die Anzahl der Lösungen nach dem „Schubfachprinzip“ keinesfalls größer als $2^{k_{min}}$ sein.

- Andererseits gewährleistet die Universalität von $\mathcal{H}[m;k]$, wie am Ende des letzten Abschnitts gezeigt, daß für jede Menge $S \subseteq \{0,1\}^m$ mit $\|S\| \leq 2^{k/2}$ die Wahrscheinlichkeit für das Auftreten einer Kollision auf S höchstens $2^{k/2} \cdot (2^{k/2} - 1)/2^k < 1$ ist. Folglich erhalten wir aus der Information, daß jede Hashfunktion $h \in \mathcal{H}[m;k_{min} - 1]$ zu einer Kollision innerhalb von S_x führt, die untere Schranke $2^{(k_{min}-1)/2}$ für die Größe von S_x.

Insgesamt ergibt sich also, daß die Anzahl der Lösungen notwendigerweise in dem halboffenen Intervall $(2^{(k_{min}-1)/2}, 2^{k_{min}}]$ liegen muß.

Der mit einer Realisierung dieses Ansatzes verbundene Aufwand hängt entscheidend von der Größe der verwendeten universalen Klasse von Hashfunktionen von $\{0,1\}^m$ nach $\{0,1\}^k$ ab. Würde man etwa die (sicherlich universale) Klasse *aller* Funktionen von $\{0,1\}^m$ nach $\{0,1\}^k$ verwenden, so würde man zur Beschreibung einer einzelnen Hashfunktion eine exponentielle Anzahl von Bits (genauer: $k2^m$ Bits) benötigen. Glücklicherweise kann eine Funktion aus der im Vergleich viel kleineren universalen Klasse $\mathcal{L}in[m;k]$ schon mit mk, also polynomial vielen Bits beschrieben werden. Diese „Kompaktheit“ der Klasse $\mathcal{L}in[m;k]$ hat zur Folge, daß die Frage nach der Existenz einer auf S_x kollisionsfreien Hashfunktion in $\mathcal{L}in[m;k]$ durch ein geeignetes Σ_2^{P}-Orakel beantwortet werden kann:

> Verifiziere, daß es ein $h \in \mathcal{L}in[m;k]$ gibt (dies ist eine *polynomial* beschränkte Existenz-Quantifizierung), so daß für alle $y_1, y_2 \in S_x$ die Werte von $h(y_1)$ und $h(y_2)$ verschieden sind.

Damit ist es also möglich, durch eine $\mathcal{FP}(\Sigma_2^{\mathrm{P}})$-Berechnung ein Intervall der Form $(2^{(k-1)/2}, 2^k]$ zu erhalten, das mit Sicherheit den Wert von $\|S_x\|$ enthält.

Durch zufällige und unabhängige Wahl einer polynomialen Anzahl von Hashfunktionen kann die Kollisionswahrscheinlichkeit exponentiell (in der Anzahl der verwendeten Hashfunktionen) verkleinert werden. Wie wir sehen werden, bewirkt diese Modifikation eine entscheidende Verbesserung der Approximationsgüte. Zuvor müssen wir uns jedoch darüber klar werden, was wir unter einer Kollision verstehen wollen, die von einer Kollektion von Hashfunktionen innerhalb von S_x verursacht wird.

Sei $H = (h_1, \ldots, h_l)$ eine Kollektion von l Hashfunktionen von A nach B. Dann können wir einer Menge von Schlüsseln $x_1, \ldots, x_k$, die in dieser Reihenfolge auftreten, dynamisch eine entsprechende Folge von Adressen in dem (erweiterten) Adreßraum $\{1, \ldots, l\} \times B$ wie folgt zuweisen. Für jedes $i = 1, \ldots, k$ speichere den mit Schlüssel x_i verbundenen Datensatz unter der kleinsten Adresse der Form $(j, h_j(x_i))$ mit $j \in \{1, \ldots, l\}$ ab, die noch frei ist. Sind bereits alle l Adressen von dieser Form durch einen (zuvor abgespeicherten) Datensatz belegt, so führt diese Vorgehensweise offenbar zu einer Kollision zwischen dem Schlüssel x_i und seinen Vorgängern $x_1, \ldots, x_{i-1}$.

Definition 3.4 *Sei $H = (h_1, \ldots, h_l)$ eine Kollektion von Hashfunktionen von A nach B, und seien $x \in A$ und $X, Y \subseteq A$ beliebig.*

a) H verursacht eine Kollision zwischen x und Y (kurz: *Kollision*(x, Y, H)), *falls x unter jeder Hashfunktion h_i aus der Kollektion H mit einem $y \in Y$ kollidiert,*

$$\mathit{Kollision}(x, Y, H) \;\Leftrightarrow\; \exists\, y_1, \ldots, y_l \in Y : x \notin \{y_1, \ldots, y_l\} \text{ und } \text{für alle } j = 1, \ldots, l \text{ ist } h_j(x) = h_j(y_j).$$

b) H verursacht eine Kollision zwischen X und Y (kurz: *Kollision*(X, Y, H)), *falls mindestens ein $x \in X$ mit Y unter H kollidiert,*

$$\mathit{Kollision}(X, Y, H) \;\Leftrightarrow\; \exists\, x \in X : \mathit{Kollision}(x, Y, H).$$

c) H verursacht eine Kollision innerhalb von X (kurz: *Kollision*(X, H)), *falls ein $x \in X$ mit X unter H kollidiert,*

$$\mathit{Kollision}(X, H) \;\Leftrightarrow\; \exists\, x \in X : \mathit{Kollision}(x, X, H).$$

Wir werden nun eine Reihe von sehr nützlichen Wahrscheinlichkeitsabschätzungen für das Auftreten von Kollisionen unter einer zufällig gewählten Kollektion $H = (h_1, \ldots, h_l)$ herleiten. Dabei wird vorausgesetzt, daß die einzelnen Hashfunktionen $h_1, \ldots, h_l$ unabhängig voneinander und unter Gleichverteilung aus einer universalen Klasse $\mathcal{H}$ von Hashfunktionen gewählt werden.

Lemma 3.5 *Sei $\mathcal{H}$ eine universale Klasse von Hashfunktionen von A nach B. Seien weiterhin $x \in A$, $X, Y \subseteq A$, und sei $\beta = \|Y\| \,/\, \|B\|$. Dann gelten für eine unabhängig unter Gleichverteilung aus $\mathcal{H}$ gewählte Kollektion H von l Hashfunktionen die folgenden Wahrscheinlichkeitsabschätzungen.*

(i) $\Pr[\, \mathit{Kollision}(x, Y, H)\,] \;\le\; \beta^l$.

(ii) $\Pr[\, \mathit{Kollision}(X, Y, H)\,] \;\le\; \|X\| \cdot \beta^l$.

(iii) $\Pr[\, \mathit{Kollision}(Y, H)\,] \;<\; \|Y\| \cdot \beta^l$.

Beweis.

(i) Sei $H = (h_1, \ldots, h_l)$. Dann erhalten wir wegen $\Pr[\, \mathit{Kollision}(x, Y, h_i)\,] \le \beta$,

$$\begin{aligned}\Pr[\, \mathit{Kollision}(x, Y, H)\,] &= \Pr[\, \forall i = 1, \ldots, l : \mathit{Kollision}(x, Y, h_i)\,] \\ &= \prod_{i=1}^{l} \Pr[\, \mathit{Kollision}(x, Y, h_i)\,] \\ &\le \beta^l.\end{aligned}$$

(ii) Unter Verwendung von Teil *(i)* ergibt sich sofort

$$\begin{aligned}\Pr[\, \mathit{Kollision}(X, Y, H)\,] &\le \sum_{x \in X} \Pr[\, \mathit{Kollision}(x, Y, H)\,] \\ &\le \|X\| \cdot \beta^l.\end{aligned}$$

(iii) folgt ebenfalls aus Teil *(i)* unter Beachtung der Tatsache, daß die beiden Ereignisse $\mathit{Kollision}(x, Y, H)$ und $\mathit{Kollision}(x, Y - \{x\}, H)$ identisch sind:

$$\begin{aligned}\Pr[\, \mathit{Kollision}(Y, H)\,] &\le \sum_{x \in Y} \Pr[\, \mathit{Kollision}(x, Y - \{x\}, H)\,] \\ &\le \|Y\| \cdot \Big((\|Y\| - 1) \,/\, \|B\|\Big)^l \\ &< \|Y\| \cdot \beta^l.\end{aligned}$$

■

Die in folgendem Korollar festgehaltene Abschätzung der Kollisionswahrscheinlichkeit für eine zufällige Kollektion von linearen Hashfunktionen, der Sipser den Namen „Coding Lemma“ gab, ergibt sich nun unmittelbar aus Teil *(iii)* des vorigen Lemmas.

VEREINBARUNG

$\mathcal{L}in[l;m;k]$ bezeichne die Menge aller Kollektionen $H = (h_1, \ldots, h_l)$ von l (nicht notwendigerweise verschiedenen) linearen Hashfunktionen $h_1, \ldots, h_l \in \mathcal{L}in[m;k]$.

Korollar 3.6 [Sip83, Coding Lemma] *Sei $Y \subseteq \Sigma^m$ eine Menge von höchstens 2^{k-1} Schlüsseln. Dann ist die Wahrscheinlichkeit, daß eine unabhängig unter Gleichverteilung aus $\mathcal{L}in[k;m;k]$ gewählte Kollektion H eine Kollision auf Y verursacht, kleiner als* $1/2$.

Sipsers Coding Lemma bedeutet einen entscheidenden Schritt in Richtung einer verbesserten Abschätzung der Anzahl aller Lösungen für eine $\mathcal{NP}$-Problemeingabe x. Im Unterschied zu der unteren Schranke $\|S_x\| > 2^{(k_{min}-1)/2}$, die wir aus der Nicht-Existenz einer einzelnen auf S_x kollisionsfreien Hashfunktion in $\mathcal{L}in[m;k_{min}-1]$ erhielten, können wir aufgrund von Korollar 3.6 aus der Nicht-Existenz einer auf S_x kollisionsfreien Kollektion von $k_{min}-1$ solcher Hashfunktionen auf die Gültigkeit von $\|S_x\| > 2^{k_{min}-1}$ schließen. Wie das folgende ebenfalls von Sipser stammende Lemma zeigt, müssen wir zwar gleichzeitig eine leichte Verschlechterung der entsprechenden oberen Schranke (von $2^{k_{min}}$ auf $k_{min} \cdot 2^{k_{min}}$) in Kauf nehmen. Insgesamt jedoch erhalten wir ein im Vergleich zu $(2^{(k_{min}-1)/2}, 2^{k_{min}}]$ wesentlich kleineres Intervall $(2^{k_{min}-1}, k_{min} \cdot 2^{k_{min}}]$ von möglichen Werten für $\|S_x\|$. Entscheidend ist, daß sich hierbei der Quotient aus oberer und unterer Schranke größenordnungsmäßig von $\|S_x\|^{1/2}$ auf $\log \|S_x\|$ verbessert hat.

Lemma 3.7 [Sip83] *Ist eine Kollektion $H = (h_1, \ldots, h_l)$ von l Hashfunktionen von A nach B kollisionsfrei auf einer Menge $S \subseteq A$, so muß notwendigerweise $\|S\| \leq l \cdot \|B\|$ sein.*

Beweis. Sei $H = (h_1, \ldots, h_l)$ kollisionsfrei auf S. Dann ist für jedes $x \in S$ die Abbildung

$$\kappa(x) =_{\text{def}} \min\{1 \leq i \leq l \mid \forall y \in S - \{x\} : h_i(x) \neq h_i(y)\}$$

wohldefiniert. Folglich ist die Funktion $f : S \to \{1, \ldots, l\} \times B$ mit

$$f(x) \quad = \quad (\kappa(x), h_{\kappa(x)}(x))$$

eine Injektion. Mit dem Schubfachprinzip folgt daher $\|S\| \leq l \cdot \|B\|$. ∎

Die durch Verwendung von Kollektionen von Hashfunktionen erzielte Verbesserung der Genauigkeit bildet die Grundlage dafür, daß durch Anwendung folgender von Stockmeyer [Sto85] stammenden Idee eine erheblich präzisere Abschätzung erzielt werden kann:

> Um eine im Prinzip beliebig genaue Abschätzung für die Anzahl aller Lösungen in S_x zu erhalten, schätze für einen hinreichend groß gewählten Exponenten r die Anzahl aller in $(S_x)^r$ enthaltenen r-Tupel von Lösungen ab, und berechne hieraus eine Abschätzung für $\|S_x\|$ durch Ziehen der r-ten Wurzel.

Wir präsentieren an dieser Stelle das Resultat von Stockmeyer [Sto85] ohne Beweis, da wir in Abschnitt 3.3 ein allgemeineres Resultat herleiten, das den Satz von Stockmeyer als Spezialfall enthält.

Satz 3.8 [Sto85] *Zu jeder $\#\mathcal{P}$-Funktion f existiert eine Funktion g in $\mathcal{FP}(\Sigma_2^{\mathrm{P}})$, die für alle $x \in \{0,1\}^*$ und $l \geq 1$ den Wert von $f(x)$ wie folgt abschätzt:*

$$f(x) \ \leq \ g(x, 0^l) \ \leq \ \left(1 + \frac{1}{l}\right) f(x)$$

Anders ausgedrückt bedeutet dies, daß $f(x)$ für alle $x \in \{0,1\}^*$ und $l \geq 1$ in dem Intervall $[\frac{l}{l+1} g(x, 0^l), g(x, 0^l)]$ liegen muß. Interessanterweise werden in Satz 3.8 zur Berechnung der Approximationsfunktion $g \in \mathcal{FP}(\Sigma_2^{\mathrm{P}})$ nur $O(\log(n \cdot l))$ Fragen an das verwendete Σ_2^{P}-Orakel benötigt (vgl. [Pio91]). Satz 3.8 wurde überdies von $\#\mathcal{P}$ auf die vermutlich umfassendere Klasse aller $\mathcal{S}pan\mathcal{P}$-Funktionen verallgemeinert (siehe [KST89]).

Wir erwähnen abschließend, daß Gavaldà [Gav92] Sipsers Coding Lemma (Korollar 3.6) auf den Fall einer Kollektion $\mathcal{Y}$ einer exponentiellen Anzahl von Mengen $Y_1, \ldots, Y_{2^n} \subseteq \{0,1\}^m$ erweiterte. Er bewies, daß die Wahrscheinlichkeit für das Auftreten einer Kollision auf mindestens einer der Mengen in $\mathcal{Y}$ kleiner als $1/2$ ist, falls eine Kollektion von $(n+1)k$ Hashfunktionen (anstelle von k Hashfunktionen wie in Korollar 3.6) zufällig gewählt wird. Durch eine etwas effizientere Wahrscheinlichkeitsabschätzung läßt sich allerdings zeigen, daß es hierfür bereits ausreicht, eine Kollektion von $n + k$ Hashfunktionen zu wählen (dies ergibt sich als ein Spezialfall von Lemma 3.9 *(ii)* im nächsten Abschnitt, falls man die Parameterwerte $d = n$ und $r = 1$ einsetzt).

3.3 Verfeinerung der Hashing-Technik

In diesem Abschnitt entwickeln wir verfeinerte Techniken des universalen Hashings, die es uns erlauben, nicht nur $\#\mathcal{P}$- oder $\mathcal{S}pan\mathcal{P}$-Funktionen, sondern beispielsweise auch max-$\mathcal{S}pan\mathcal{P}$- und min-$\mathcal{S}pan\mathcal{P}$-Funktionen mit im Prinzip beliebiger Genauigkeit zu approximieren. Diese Techniken stellen ein wichtiges Hilfsmittel dar, auf die wir in den folgenden Kapiteln mehrfach zurückgreifen werden.

Vereinbarung

Für alle $m, k \geq 1$ bezeichne $\mathcal{H}[m; k]$ eine beliebige universale Klasse von Hashfunktionen h von $\{0,1\}^m$ nach $\{0,1\}^k$. Mit $\mathcal{H}[l; m; k]$ bezeichnen wir die entsprechende Klasse aller Kollektionen H von l Hashfunktionen $h_1, \ldots, h_l$ aus $\mathcal{H}[m; k]$.

Das folgende Lemma ist ein sehr mächtiges Werkzeug. Es bereitet den Weg für die Lösung des folgenden Problems:

> Finde bis auf einen im Prinzip beliebig kleinen relativen Fehler ε heraus, ob die größte einer gegebenen Folge von Teilmengen $X_1, \ldots, X_{2^n} \subseteq \{0,1\}^m$ mehr oder weniger als eine vorgegebene Anzahl a von Elementen enthält?

Die durch Lemma 3.9 ermöglichte Lösung dieses Problems ist die folgende: Bestimme in Abhängigkeit von ε und m geeignete natürliche Zahlen r, k und l, und teste, ob jede Kollektion $H \in \mathcal{H}[l; m \cdot r; k]$ notwendigerweise für mindestens ein $i \in \{1, \ldots, 2^n\}$ zu einer Kollision auf dem r-fachen Produkt

$$(X_i)^r \quad = \quad \{x_1 \cdots x_r \mid \text{für } j = 1, \ldots, r \text{ ist } x_j \in X_i\} \quad \subseteq \quad \{0,1\}^{m \cdot r},$$

der Sprache X_i führt. Ist dies der Fall, so enthält mindestens ein X_i definitiv mehr als a Elemente. Andernfalls können wir zwar nicht darauf schließen, daß alle X_i höchstens a Elemente enthalten. Aber wir wissen dann jedenfalls, daß keine der Mengen $X_1, \ldots, X_{2^n}$ mehr als $(1 + \varepsilon)\, a$ Elemente enthält.

Lemma 3.9 *Seien $a \geq 1$ und $d \geq 0$, sowie eine Fehlerschranke ε, $0 < \varepsilon \leq 1$, vorgegeben. Dann gelten für alle $r \geq 1 + 4d/\varepsilon + 8m/\varepsilon^2$, sowie für $k = 1 + \lceil r \log a \rceil$ die folgenden Implikationen.*

(i) *Jede Teilmenge $X \subseteq \{0,1\}^m$, für die die Produktmenge X^r von einer Kollektion $H \in \mathcal{H}[d+k; m \cdot r; k]$ kollisionsfrei abgebildet werden kann, enthält nicht mehr als $(1+\varepsilon)\, a$ Elemente.*

(ii) *Für eine Folge $X_1, \ldots, X_{2^n}$ von beliebigen Teilmengen von $\{0,1\}^m$, von denen keine mehr als a Elemente enthält, ist die Wahrscheinlichkeit dafür, daß eine zufällig aus $\mathcal{H}[d+k; m \cdot r; k]$ gewählte Kollektion H innerhalb mindestens einer der Produktmengen $(X_1)^r, \ldots, (X_{2^n})^r$ eine Kollision verursacht, kleiner als 2^{n-d-1}:*

$$\Pr[\, \exists i \in \{1, \ldots, 2^n\} : \mathit{Kollision}((X_i)^r, H)\,] \quad < \quad 2^{n-d-1}.$$

Beweis. Zum Nachweis der ersten Implikation sei angenommen, daß eine Kollektion $H \in \mathcal{H}[d+k; m \cdot r; k]$ eine einzelne Teilmenge X kollisionsfrei abbildet. Wir müssen zeigen, daß dann notwendigerweise $\|X\| \leq (1+\varepsilon)\, a$ ist. Wir bemerken zunächst, daß dies im Fall $a \geq 2^m$ trivialerweise gilt, das heißt, wir können $a < 2^m$ voraussetzen. Nach Lemma 3.7 kann es nur dann eine auf einer Menge $Y \subseteq \{0,1\}^{m \cdot r}$ kollisionsfreie Kollektion in $\mathcal{H}[l; m \cdot r; k]$ geben, wenn $\|Y\| \leq l \cdot 2^k$ ist. Dies führt auf die folgende Abschätzung für die Mächtigkeit der Menge X^r:

$$\begin{aligned} \|X^r\| &\leq (d+k) \cdot 2^k \\ &= (1 + d + \lceil r \log a \rceil) \cdot 2^{1+\lceil r \log a \rceil} \\ &< 4(2 + d + r \log a) \cdot a^r \\ &< (1+\varepsilon)^r \cdot a^r. \end{aligned}$$

Die letzte Ungleichung in obiger Kette ergibt sich hierbei unter Verwendung der vorausgesetzten Ungleichungen $r \geq 1 + 4d/\varepsilon + 8m/\varepsilon^2$ und $\varepsilon \leq 1$ wie folgt:

$$\begin{aligned} (1+\varepsilon)^r &\geq 1 + \underbrace{r\varepsilon}_{>8+4d} + r\,\underbrace{(r-1)\varepsilon^2/2}_{>4m} \\ &> 9 + 4d + 4rm \\ &> 4(2 + d + r \log a), \text{ da } 2^m > a. \end{aligned}$$

Damit ist *(i)* gezeigt. Für den Beweis von *(ii)* sei angenommen, daß $\|X_i\| \leq a$ für alle $i \in \{1, \ldots, 2^n\}$ ist. Unter Verwendung von Lemma 3.5 erhalten wir dann für jedes feste i die folgende Abschätzung der Wahrscheinlichkeit, daß eine zufällig aus $\mathcal{H}[d+k; m \cdot r; k]$ gewählte Kollektion H eine Kollision innerhalb der Produktmenge

$(X_i)^r$ verursacht,

$$\begin{aligned}
\Pr[\,\mathit{Kollision}((X_i)^r, H)\,] &< \|(X_i)^r\| \cdot \left(\frac{\|(X_i)^r\|}{2^k}\right)^{d+k} \\
&\leq a^r \cdot \left(\frac{a^r}{2^{1+\lceil r \log a \rceil}}\right)^{1+d+\lceil r \log a \rceil} \\
&< 2^{-d-1}.
\end{aligned}$$

Also ist die Wahrscheinlichkeit, daß eine zufällig aus $\mathcal{H}[d+k; m \cdot r; k]$ gewählte Kollektion H innerhalb mindestens einer der Mengen $(X_1)^r, \ldots, (X_{2^n})^r$ eine Kollision verursacht,

$$\begin{aligned}
\Pr[\,\exists X_i : \mathit{Kollision}((X_i)^r, H)\,] &\leq \sum_{i=1}^{2^n} \Pr[\,\mathit{Kollision}((X_i)^r, H)\,] \\
&< 2^n \cdot 2^{-d-1} = 2^{n-d-1}.
\end{aligned}$$

Diese Abschätzung gilt sogar unabhängig von der Wahl von r (das heißt, für alle $r \geq 1$, vorausgesetzt, daß $k = 1 + \lceil r \log a \rceil$ ist). ■

Unter Verwendung von Lemma 3.9 folgt nun leicht das folgende Resultat, auf das wir im folgenden mehrfach zurückgreifen werden.

Satz 3.10 *Zu jeder max-$\mathcal{S}$pan$\mathcal{P}$-Funktion f und zu jedem Polynom q gibt es eine Sprache $A \in$ co-$\mathcal{NP}$ und ein Polynom $\hat{p}$, so daß für alle $n, a \geq 0$ und $l \geq 1$ folgendes gilt:*

(i) Für alle $x \in \{0,1\}^n$ und alle $y \in \{0,1\}^{\hat{p}(n+l)}$ gilt:

$$\langle x, y, a, 0^l \rangle \in A \Rightarrow f(x) \leq \left(1 + \frac{1}{l}\right) a.$$

(ii) Für ein zufällig aus $\{0,1\}^{\hat{p}(n+l)}$ gewähltes y ist die Wahrscheinlichkeit dafür, daß für alle $x \in \{0,1\}^n$ mit $f(x) \leq a$ das Tupel $\langle x, y, a, 0^l \rangle$ in A enthalten ist, mindestens $1 - 2^{-q(n)}$:

$$\Pr[\,\forall x \in \{0,1\}^n : f(x) \leq a \Rightarrow \langle x, y, a, 0^l \rangle \in A\,] \;\geq\; 1 - 2^{-q(n)}.$$

Beweis. Sei f eine $max\text{-}\mathcal{S}pan\mathcal{P}$-Funktion. Dann gibt es ein Polynom p und eine Sprache B in $\mathcal{NP}$, so daß sich für alle $n \geq 0$ und alle $x \in \{0,1\}^n$ der Wert von $f(x)$ wie folgt darstellen läßt:

$$f(x) \quad = \quad max_{|u|=p(n)} \| \underbrace{\{w \in \{0,1\}^{p(n)} \mid \langle x,u,w \rangle \in B\}}_{=_{def} W_{(x,u)}} \|.$$

Seien d, r und k Funktionen, definiert als

$$\begin{aligned} d(n) &=_{def} p(n) + q(n) + n, \\ r(n,l) &=_{def} 1 + l \cdot d(n) + 8p(n)l^2, \\ k(n,l,a) &=_{def} 1 + \lceil r(n,l) \log a \rceil. \end{aligned}$$

Dann gelten aufgrund von Lemma 3.9 für alle $n \geq 0$ und $l, a \geq 1$ die beiden folgenden Aussagen:

- Falls eine Kollektion $H \in \mathcal{L}in[d(n) + k(n,l,a); p(n)r(n,l); k(n,l,a)]$ für ein $x \in \{0,1\}^n$ und ein $u \in \{0,1\}^{p(n)}$ die Menge $(W_{(x,u)})^{r(n,l)}$ kollisionsfrei abbildet, dann ist $\|W_{(x,u)}\| \leq (1 + \frac{1}{l})\, a$.
- Mit einer Wahrscheinlichkeit von weniger als $2^{-q(n)-1}$ verursacht eine zufällig aus $\mathcal{L}in[d(n) + k(n,l,a); p(n)r(n,l); k(n,l,a)]$ gewählte Kollektion H auf einer der Mengen $(W_{(x,u)})^{r(n,l)}$ mit $x \in \{0,1\}^n$, $u \in \{0,1\}^{p(n)}$ und $\|W_{(x,u)}\| \leq a$ eine Kollision.

Sei $\hat{p}$ ein Polynom, das für alle $n, l \geq 0$ die folgende Ungleichung erfüllt:

$$\hat{p}(n+l) \geq (d(n) + k(n,l,2^{p(n)})) \cdot p(n)r(n,l) \cdot k(n,l,2^{p(n)}).$$

Dann kann jeder String y der Länge $\hat{p}(n+l)$ für ein vorgegebenes a, $1 \leq a < 2^{p(n)}$, als Kodierung einer Kollektion $H \in \mathcal{L}in[d(n) + k(n,l,a); p(n)r(n,l); k(n,l,a)]$ aufgefaßt werden, indem nur die ersten $(d(n) + k(n,l,a)) \cdot p(n)r(n,l) \cdot k(n,l,a)$ Bits für die Spezifikation von H verwendet werden und der Rest ignoriert wird.

Betrachte nun die Sprache A, definiert als

$\langle x, y, a, 0^l \rangle \in A \quad \Leftrightarrow_{def}$

entweder es gilt $a = 0$ und für alle $u \in \{0,1\}^{p(n)}$ ist $W_{(x,u)} = \emptyset$, oder es gilt $a \geq 2^{p(n)}$, oder es gilt $0 < a < 2^{p(n)}$ und y ist ein String der Länge $\hat{p}(n+l)$, der eine Kollektion H aus $\mathcal{L}in[d(n) + k(n,l,a); p(n)r(n,l); k(n,l,a)]$ kodiert, die für jedes $u \in \{0,1\}^{p(n)}$ die Menge $(W_{(x,u)})^{r(n,l)}$ kollisionsfrei abbildet.

Offensichtlich ist A eine *co-*$\mathcal{NP}$-Sprache, die für alle $n, a \geq 0$ und $l \geq 1$ die beiden folgenden Bedingungen erfüllt:

- Für alle $x \in \{0,1\}^n$ und alle $y \in \{0,1\}^{\hat{p}(n+l)}$ gilt:

$$\langle x, y, a, 0^l \rangle \in A \Rightarrow f(x) \leq \left(1 + \frac{1}{l}\right) a,$$

- Für ein zufällig aus $\{0,1\}^{\hat{p}(n+l)}$ gewähltes y ist die Wahrscheinlichkeit dafür, daß es ein $x \in \{0,1\}^n$ mit $f(x) \leq a$ gibt, für das $\langle x, y, a, 0^l \rangle \notin A$ ist, kleiner als $2^{-q(n)}$:

$$\Pr[\, \exists x \in \{0,1\}^n : f(x) \leq a \text{ und } \langle x, y, a, 0^l \rangle \notin A \,] \quad < \quad 2^{-q(n)}.$$

Damit ist der Satz bewiesen. ∎

Der folgende Satz erlaubt für eine große Klasse von Funktionen f einen approximativen Vergleich des Funktionswertes $f(x)$ an einer beliebigen Stelle x mit einem vorgegebenen Wert a durch eine Frage an ein Orakel in Σ_2^{P}.

Satz 3.11 *Zu jeder min-max-*$\mathcal{S}$*pan*$\mathcal{P}$*-Funktion f existiert eine Sprache $A \in \Sigma_2^{\mathrm{P}}$, so daß für alle $l \geq 1$, $a \geq 0$ und alle $x \in \{0,1\}^*$ folgendes gilt:*

$$\langle x, a, 0^l \rangle \in A \quad \Rightarrow \quad f(x) \leq \left(1 + \frac{1}{l}\right) a,$$

$$\langle x, a, 0^l \rangle \notin A \quad \Rightarrow \quad f(x) > a.$$

Beweis. Sei h eine max-$\mathcal{S}$pan$\mathcal{P}$-Funktion und p ein Polynom, so daß für alle $n \geq 0$ und alle $x \in \{0,1\}^n$, $f(x) = min_{|u|=p(n)} h(x,u)$. Nach Satz 3.10 gibt es ein Polynom $\hat{p}$ und eine Sprache U in *co-*$\mathcal{NP}$, die für alle $n, a \geq 0$ und $l \geq 1$ die beiden folgenden Bedingungen erfüllt:

- für alle $x \in \{0,1\}^n$, $u \in \{0,1\}^{p(n)}$ und alle $y \in \{0,1\}^{\hat{p}(n+l)}$ gilt:

$$\langle x, u, y, a, 0^l \rangle \in U \Rightarrow h(x,u) \leq \left(1 + \frac{1}{l}\right) a,$$

- es gibt ein $y \in \{0,1\}^{\hat{p}(n+l)}$, so daß für alle $x \in \{0,1\}^n$ und $u \in \{0,1\}^{p(n)}$ gilt:

$$\langle x, u, y, a, 0^l \rangle \notin U \Rightarrow h(x,u) > a.$$

Betrachte nun folgende Sprache A, definiert als

$$\langle x, a, 0^l \rangle \in A \quad \Leftrightarrow_{def} \quad \text{es gibt ein } y \in \{0,1\}^{\hat{p}(n+l)} \text{ und ein } u \in \{0,1\}^{p(n)}, \text{ so daß } \langle x, u, y, a, 0^l \rangle \in U \text{ ist.}$$

Offensichtlich ist A eine Σ_2^{P}-Sprache, und aufgrund von Satz 3.10 gelten für alle $n, a \geq 0$, $l \geq 1$ und alle $x \in \{0,1\}^n$ die folgenden Implikationen:

$$\begin{aligned}
\langle x, a, 0^l \rangle \in A \quad &\Rightarrow \quad \exists y \in \{0,1\}^{\hat{p}(n+l)}\, \exists u \in \{0,1\}^{p(n)} : \langle x, u, y, a, 0^l \rangle \in U \\
&\Rightarrow \quad \exists u \in \{0,1\}^{p(n)} : h(x,u) \leq \left(1 + \frac{1}{l}\right) a \\
&\Rightarrow \quad f(x) \leq \left(1 + \frac{1}{l}\right) a, \\
\langle x, a, 0^l \rangle \notin A \quad &\Rightarrow \quad \forall y \in \{0,1\}^{\hat{p}(n+l)}\, \forall u \in \{0,1\}^{p(n)} : \langle x, u, y, a, 0^l \rangle \notin U \\
&\Rightarrow \quad \forall u \in \{0,1\}^{p(n)} : h(x,u) > a \\
&\Rightarrow \quad f(x) > a.
\end{aligned}$$

Damit ist der Beweis erbracht. ■

Schließlich läßt sich mit Hilfe von Satz 3.11 noch das folgende Resultat herleiten, das den in Abschnitt 3.2 angegebenen Satz von Stockmeyer [Sto85] verallgemeinert.

Satz 3.12 *Zu jeder min-max-$\mathcal{S}$pan$\mathcal{P}$-Funktion f existiert eine Funktion g in $\mathcal{FP}(\Sigma_2^{\mathrm{P}})$ mit folgender Eigenschaft: Für alle $x \in \{0,1\}^*$ und $l \geq 1$ ist*

$$f(x) \;\leq\; g(x, 0^l) \;\leq\; \left(1 + \frac{1}{l}\right) f(x).$$

Beweis. Nach Satz 3.11 gibt es eine Sprache $U \in \Sigma_2^{\mathrm{P}}$, so daß für alle $l \geq 1$, $a \geq 0$ und alle $x \in \{0,1\}^*$ die folgenden Implikationen gelten:

$$\langle x, a, 0^l \rangle \notin U \quad \Rightarrow \quad f(x) > a, \tag{3.1}$$

$$\langle x, a, 0^l \rangle \in U \quad \Rightarrow \quad f(x) \leq \left(1 + \frac{1}{3l}\right) a. \tag{3.2}$$

Die Approximationsfunktion g wird nun bei Eingabe $\langle x, 0^l \rangle$ durch binäre Suche wie folgt berechnet. Sei

$$a_k \;=\; \begin{cases} 0, & k \leq 0, \\ \left(1 + \dfrac{1}{3l}\right)^{k-1}, & k \geq 1. \end{cases}$$

Wegen $\log(1+\frac{1}{3l}) > \frac{1}{3l}$ hat dann a_k spätestens für $k \geq 1 + 3lp(n)$ einen Wert von

$$a_k \;=\; 2^{(k-1)\log(1+\frac{1}{3l})} \;>\; 2^{\frac{k-1}{3l}} \;\geq\; 2^{p(n)}$$

erreicht. Betrachte nun die folgende Orakelmaschine M.

> Bei Eingabe $\langle x, 0^l\rangle$ bestimme durch binäre Suche $\hat{k}$ als den kleinsten Index k in $\{0, 1, \ldots, 3lp(n)\}$, für den das Tupel $\langle x, a_k, 0^l\rangle$ in U liegt, und gib den Wert $\left(1+\frac{1}{3l}\right) a_{\hat{k}}$ aus.

Da das Tupel $\langle x, a_{\hat{k}}, 0^l\rangle$ in U liegt, folgt mit (3.2), daß $f(x) \leq \left(1+\frac{1}{3l}\right) a_{\hat{k}} = g(x, 0^l)$ ist. Da weiterhin $\langle x, a_{\hat{k}-1}, 0^l\rangle$ nicht in U liegt, folgt mit (3.1), daß $\hat{k} = 0$ oder $f(x) > a_{\hat{k}-1}$ ist. In beiden Fällen gilt $f(x) \geq a_{\hat{k}-1}$. Demnach erhalten wir für den durch M berechneten Wert $g(x, 0^l) = \left(1+\frac{1}{3l}\right) a_{\hat{k}}$ die folgende Abschätzung:

$$\begin{aligned} f(x) &\leq g(x, 0^l) \\ &\leq \left(1+\frac{1}{3l}\right)^2 a_{\hat{k}-1} \;\leq\; \left(1+\frac{1}{l}\right) a_{\hat{k}-1} \;\leq\; \left(1+\frac{1}{l}\right) f(x). \end{aligned}$$

Dies zeigt, daß die Funktion g die gewünschte Approximationsgüte aufweist. ■

Wir bemerken abschließend, daß im Beweis des vorigen Satzes zur Berechnung der Approximationsfunktion $g(x, 0^l)$ in $\mathcal{FP}(\Sigma_2^{\mathrm{P}})$ nur $\log(3lp(n)) = l + O(\log n)$ Fragen an das Σ_2^{P}-Orakel U benötigt werden.

3.4 Partielle Kollisionen

In diesem Abschnitt führen wir den Begriff einer durch eine Kollektion von Hashfunktionen verursachten partiellen Kollision zwischen zwei Schlüsselmengen ein. Dieser wird eine zentrale Rolle im Beweis des Hauptresultats von Kapitel 5 spielen.

Definition 3.13 *Sei $H = (h_1, \ldots, h_l)$ eine Kollektion von Hashfunktionen von A nach B, und sei $0 \leq \alpha \leq 1$. H* verursacht eine α-partielle Kollision *zwischen zwei Teilmengen X und Y von A (kurz: α-Kollision(X, Y, H)), falls es ein $x \in X$ gibt, das mindestens unter einem α-Anteil aller Hashfunktionen in H mit der Schlüsselmenge Y kollidiert,*

$$\alpha\text{-Kollision}(X, Y, H) \;\Leftrightarrow\; \exists\, x \in X\, \exists\, y_1, \ldots, y_l \in Y : x \notin \{y_1, \ldots y_l\} \text{ und}$$
$$\|\{1 \leq i \leq l \mid h_i(x) = h_i(y_i)\}\| \geq \alpha \cdot l.$$

Demnach ist eine Kollision im herkömmlichen Sinne nichts anderes als eine 1-partielle Kollision. Das nächste Lemma zeigt eine interessante Beziehung zwischen Kollisionen und (1/2)-Kollisionen auf.

Lemma 3.14 *Falls eine Kollektion $H = (h_1, \ldots, h_l)$ eine Kollision zwischen einer Menge X und einer Menge $Y = Y_1 \cup Y_2$ verursacht, dann verursacht H eine (1/2)-Kollision zwischen X und Y_1 oder zwischen X und Y_2.*

Beweis. Da H eine Kollision zwischen X und Y verursacht, gibt es ein $x \in X$ und $y_1, \ldots, y_l \in Y$ mit $\|\{1 \leq i \leq l \mid h_i(x) = h_i(y_i)\}\| = l$. Wegen $Y = Y_1 \cup Y_2$ enthält zumindest eine der beiden Mengen $\{y_1, \ldots, y_l\} \cap Y_1$ oder $\{y_1, \ldots, y_l\} \cap Y_2$ mindestens $l/2$ Elemente. Folglich verursacht H eine (1/2)-Kollision zwischen X und Y_1 oder zwischen X und Y_2. ∎

Wir werden Lemma 3.14 in folgendem Kontext anwenden. Angenommen, es liege eine Menge X, sowie eine Folge von 2^n Partitionen von X in je zwei Teilmengen Y_i' und Y_i'', $1 \leq i \leq 2^n$, vor. Wollten wir nun für eine gegebene Kollektion H von Hashfunktionen überprüfen, ob sie für jedes i eine Kollision auf zumindest einer der beiden Teilmengen Y_i' und Y_i'' verursacht, so würde dies zu einem Π_2^{P}-Prädikat führen. Dagegen erfordert die Feststellung, daß H zu einer Kollision auf X führt, nur ein $\mathcal{NP}$-Prädikat. Aufgrund von Lemma 3.14 ist dadurch jedoch sichergestellt, daß H für jedes i zumindest eine (1/2)-Kollision zwischen X und einer der beiden Teilmengen Y_i' und Y_i'' verursacht. Um das soeben geschilderte Verfahren erfolgreich anwenden zu können, muß allerdings gewährleistet sein, daß auf hinreichend kleinen Mengen sogar (1/2)-Kollisionen relativ unwahrscheinlich sind. Wie das folgende Lemma zeigt, nimmt die Wahrscheinlichkeit für das Auftreten einer (1/2)-Kollision mit der Größe der betrachteten Mengen in der Tat beinahe ebenso schnell ab wie die Wahrscheinlichkeit für das Auftreten einer Kollision.

Lemma 3.15 *Sei $\mathcal{H}$ eine universale Klasse von Hashfunktionen von A nach B, $X, Y_1, \ldots, Y_{2^n}$ seien Teilmengen von A, und sei $\beta = \max_{1 \leq i \leq 2^n} \|Y_i\| \,/\, \|B\|$. Dann verursacht eine unabhängig unter Gleichverteilung aus $\mathcal{H}$ gewählte Kollektion $H = (h_1, \ldots, h_l)$ von l Hashfunktionen mit einer Wahrscheinlichkeit von höchstens $2^n \|X\| (4\beta(1-\beta))^{l/2}$ eine (1/2)-Kollision zwischen X und mindestens einer der Mengen Y_i, $1 \leq i \leq 2^n$:*

$$\Pr[\, \exists i \in \{1, \ldots, 2^n\} : (1/2)\text{-}\mathit{Kollision}(X, Y_i, H)\,] \quad \leq \quad 2^n \|X\| (4\beta(1-\beta))^{l/2}.$$

Beweis. Nach Lemma 3.5 *(i)* ist die Wahrscheinlichkeit für das Auftreten einer Kollision zwischen einem festen Schlüssel x und einer festen Menge Y_i für eine zufällig unter Gleichverteilung aus $\mathcal{H}$ gewählte Hashfunktion h (das heißt, wir verwenden Lemma 3.5 mit Parameterwert $l = 1$) nicht größer als $\|Y_i\| \,/\, \|B\| \leq \beta$.

Mit Lemma 2.1 ergibt sich hieraus eine Abschätzung von $(4\beta(1-\beta))^{l/2}$ für die Wahrscheinlichkeit, daß eine unabhängig unter Gleichverteilung aus $\mathcal{H}$ gewählte Kollektion $H = (h_1, \ldots, h_l)$ von l Hashfunktionen eine (1/2)-Kollision zwischen x und Y_i verursacht. Daraus ergibt sich die im Lemma behauptete Abschätzung unmittelbar durch Aufsummieren dieser Wahrscheinlichkeiten über alle $x \in X$ und über alle Teilmengen Y_i, $1 \leq i \leq 2^n$. ∎

Kapitel 4

Lowness-Eigenschaften von Advice-Klassen

Das Hauptresultat dieses Kapitels ist die exakte Lokalisierung der Klasse $\mathcal{P}/poly$ innerhalb der dritten Theta-Stufe $E\mathcal{L}_3^\Theta$ der erweiterten $\mathcal{L}ow$-Hierarchie. Eine unmittelbare Konsequenz dieser Einordnung ist, daß alle $\mathcal{NP}$-Sprachen mit polynomialer Schaltkreiskomplexität in der dritten Theta-Stufe $\mathcal{L}ow_3^\Theta$ der $\mathcal{L}ow$-Hierarchie enthalten sind, und folglich keine $\leq_T^{SN}$-vollständige Sprache für $\mathcal{NP}$ Schaltkreise polynomialer Größe haben kann, außer wenn die Polynomialzeit-Hierarchie auf die Stufe Θ_3^{P} kollabiert. Den Schlüssel zu diesen Resultaten liefert eine verbesserte obere Komplexitätsschranke für das Problem, eine Advice-Funktion für eine Sprache A in einer nicht-uniformen Klasse der Form $\mathcal{C}/poly$ zu berechnen.

Die in diesem Kapitel dargestellten Ergebnisse sind erstmals in der Arbeit [Köb94] erschienen.

4.1 Die Low- und High-Hierarchien

Die $\mathcal{L}ow$- und $\mathcal{H}igh$-Hierarchien wurden von Schöning [Sch83] als Analogon der entsprechenden Hierarchien in der Rekursionstheorie (siehe zum Beispiel [Ler83, Soa87]) eingeführt. Der Leitgedanke ist hierbei, den Informationsgehalt einer Sprache A daran zu messen, wie nützlich sich A als Orakel-Sprache bei der Lösung algorithmischer Probleme innerhalb der Polynomialzeit-Hierarchie erweist.

Ist allgemeiner $\mathcal{C}$ eine beliebige Komplexitätsklasse, die auf der Basis eines relativierbaren Maschinenmodells (etwa durch Beschränkung gewisser Ressourcen) definiert ist, so definiert die Klasse $\mathcal{C}$ (genauer: das zugrundeliegende Maschinenmodell) einen Operator $\mathcal{C}(\cdot)$ auf der Klasse aller Sprachen. Und zwar ordnet dieser Operator jeder Sprache A die Klasse $\mathcal{C}(A)$ aller Sprachen zu, die durch Berechnungen des durch die Klasse $\mathcal{C}$ festgelegten Typs unter Befragung des Orakels A entscheidbar sind. Es ist naheliegend, den Informationsgehalt der Sprache A (aus Sicht der Berechnungen vom Typ $\mathcal{C}$) um so höher einzuschätzen, je mehr Sprachen in der Klasse $\mathcal{C}(A)$ im Vergleich zur unrelativierten Klasse $\mathcal{C}$ erkannt werden können. Besonders aussagekräftig ist hierbei der Extremfall, daß jede $\mathcal{C}(A)$ Berechnung durch eine entsprechende Berechnung in $\mathcal{C}$ ohne Zugriff auf das Orakel A simuliert werden kann.

Definition 4.1 *Sei $\mathcal{C}$ eine beliebige relativierbare Komplexitätsklasse. Dann heißt eine Sprache A* **low** *für $\mathcal{C}$ (kurz: $A \in \mathcal{C}$-low), falls $\mathcal{C}(A) = \mathcal{C}$ ist.*

Zur Bestimmung des Informationsgehaltes einer $\mathcal{NP}$ Sprache A eignen sich insbesondere Operatoren, die den einzelnen Stufen der Polynomialzeit-Hierarchie entsprechen. Dabei spielt es keine Rolle, welche Charakterisierung der Polynomialzeit-Hierarchie wir wählen (etwa mittels beschränkt alternierender Turingmaschinen, mittels polynomial längen-beschränkter Quantifizierungen oder mittels Orakelmaschinen). Alle bekannten Charakterisierungen sind nämlich relativ zu einem beliebigen Orakel äquivalent und führen daher auf dieselben Operatoren $\Sigma_k^{\mathrm{P}}(\cdot)$, $\Delta_k^{\mathrm{P}}(\cdot)$ und $\Theta_k^{\mathrm{P}}(\cdot)$.

Betrachten wir etwa den $\Sigma_0^{\mathrm{P}}(\cdot) = \mathcal{P}(\cdot)$ Operator, so reicht die Bandbreite des Informationsgehalts, der von einer $\mathcal{NP}$-Sprache A für relativierte Polynomialzeit-Berechnungen zur Verfügung gestellt werden kann, von $A = \emptyset$ als Minimum, das genau von den Sprachen in $\mathcal{P}$ angenommen wird, bis zu einem Maximum, das genau von den Turing-vollständigen Sprachen für $\mathcal{NP}$ erreicht wird. Unter der Annahme, daß $\mathcal{P} \neq \mathcal{NP}$ ist, existieren auch Sprachen in $\mathcal{NP}$, die „zwischen" diesen beiden Extremen liegen, das heißt diese Sprachen sind weder Turing-vollständig für $\mathcal{NP}$ noch liegen sie in $\mathcal{P}$ [Lad75b]. Ziehen wir zur Bestimmung des Informationsgehalts von A die mächtigeren Operatoren $\Sigma_k^{\mathrm{P}}(\cdot)$, $k > 0$, heran, so ergeben sich offenbar mit wachsendem k zunehmende Verallgemeinerungen sowohl der Klasse $\mathcal{P}$ als auch der Klasse der Turing-vollständigen Sprachen für $\mathcal{NP}$, während der „Zwischenbereich" immer mehr zusammenschrumpft. Konkret spiegelt sich der Informationsgehalt von A für Σ_k^{P}-Berechnungen in der Größe der Klasse $\Sigma_k^{\mathrm{P}}(A)$ wider:

$$\Sigma_k^{\mathrm{P}} = \Sigma_k^{\mathrm{P}}(\emptyset) \;\subseteq\; \Sigma_k^{\mathrm{P}}(A) \;\subseteq\; \Sigma_k^{\mathrm{P}}(\mathrm{SAT}) = \Sigma_{k+1}^{\mathrm{P}}.$$

Für jedes k ergibt sich hieraus eine Dreiteilung der Klasse $\mathcal{NP}$ (entsprechendes gilt natürlich für die Delta- und Theta-Operatoren).

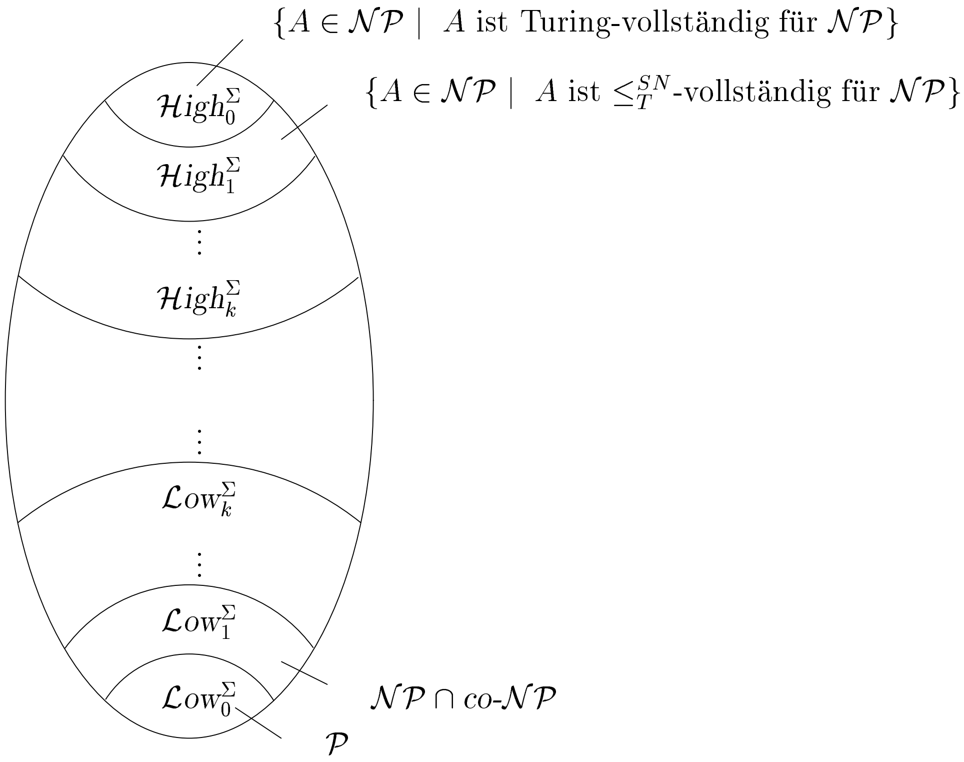


Abbildung 4.1: Die $\mathcal{L}ow$- und $\mathcal{H}igh$-Hierarchien in $\mathcal{NP}$.

- Im ersten Fall, daß $\Sigma_k^{\mathrm{P}}(A) = \Sigma_k^{\mathrm{P}}(\emptyset) = \Sigma_k^{\mathrm{P}}$ ist, die Nützlichkeit von A für Σ_k^{P}-Berechnungen also der der leeren Menge entspricht, gehört A der k-ten Sigma-Stufe der $\mathcal{L}ow$-Hierarchie an.

- Im entgegengesetzten Fall, daß $\Sigma_k^{\mathrm{P}}(A)$ alle Sprachen der nächsthöheren Stufe $\Sigma_k^{\mathrm{P}}(\mathrm{SAT}) = \Sigma_{k+1}^{\mathrm{P}}$ enthält, sich A als Orakel also ebenso nützlich erweist wie ein optimales Orakel aus $\mathcal{NP}$, wird A in die k-te Sigma-Stufe der $\mathcal{H}igh$-Hierarchie eingeordnet.

- Im verbleibenden Fall, daß eine Befragung von A das Erkennen eines gewissen Teils der Sprachen in $\Sigma_{k+1}^{\mathrm{P}} - \Sigma_k^{\mathrm{P}}$ gestattet, liegt A sozusagen „zwischen“ der k-ten Sigma-Stufe der $\mathcal{L}ow$-Hierarchie und der der $\mathcal{H}igh$-Hierarchie (siehe Abb. 4.1).

Die Sigma-Stufen der $\mathcal{L}ow$- und der $\mathcal{H}igh$-Hierarchie wurden in [Sch83] definiert. Um eine feinere Unterteilung innerhalb der beiden Hierarchien zu erhalten, wurden im nachhinein [Sch86, LS91] Zwischenstufen auf der Basis der Delta- und Theta-Stufen der Polynomialzeit-Hierarchie eingeführt.

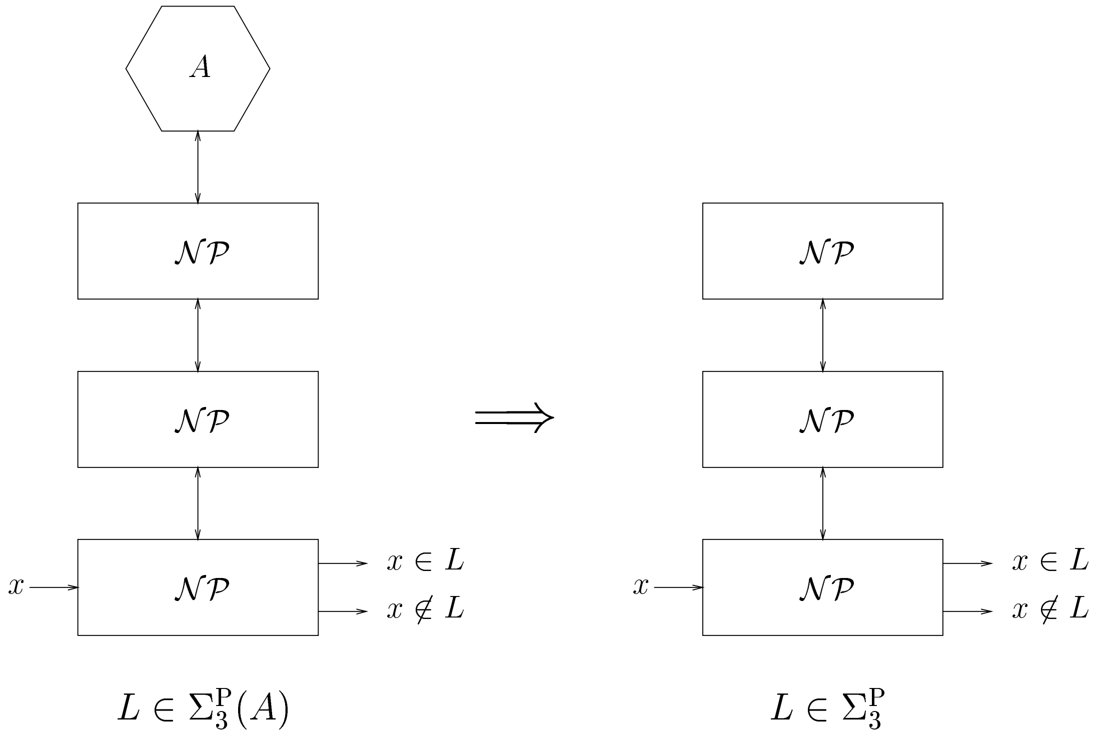


Abbildung 4.2: Eine Menge A in $\mathcal{L}ow_3^{\Sigma}$.

Definition 4.2 [Sch83, Sch86, LS91]
Die Σ, Δ, und Θ-Stufen der $\mathcal{L}ow$- und $\mathcal{H}igh$-Hierarchie sind folgendermaßen definiert:

a) $\mathcal{L}ow_k^{\Sigma} = \{A \in \mathcal{NP} \mid \Sigma_k^{\mathrm{P}}(A) \subseteq \Sigma_k^{\mathrm{P}}\}$, $\mathcal{H}igh_k^{\Sigma} = \{A \in \mathcal{NP} \mid \Sigma_{k+1}^{\mathrm{P}} \subseteq \Sigma_k^{\mathrm{P}}(A)\}$,

b) $\mathcal{L}ow_k^{\Delta} = \{A \in \mathcal{NP} \mid \Delta_k^{\mathrm{P}}(A) \subseteq \Delta_k^{\mathrm{P}}\}$, $\mathcal{H}igh_k^{\Delta} = \{A \in \mathcal{NP} \mid \Delta_{k+1}^{\mathrm{P}} \subseteq \Delta_k^{\mathrm{P}}(A)\}$,

c) $\mathcal{L}ow_k^{\Theta} = \{A \in \mathcal{NP} \mid \Theta_k^{\mathrm{P}}(A) \subseteq \Theta_k^{\mathrm{P}}\}$, $\mathcal{H}igh_k^{\Theta} = \{A \in \mathcal{NP} \mid \Theta_{k+1}^{\mathrm{P}} \subseteq \Theta_k^{\mathrm{P}}(A)\}$.

Beispielsweise ist also eine Menge A dann in $\mathcal{L}ow_3^{\Sigma}$, wenn es möglich ist, jede $\Sigma_3^{\mathrm{P}}(A)$-Berechnung für eine Sprache L durch eine Σ_3^{P}-Berechnung ohne Zugriff auf das Orakel A zu simulieren (siehe Abb. 4.2).

Es ist leicht zu sehen, daß genau die Turing-vollständigen $\mathcal{NP}$-Sprachen in $\mathcal{H}igh_0^{\Sigma}$ liegen, während $\mathcal{L}ow_0^{\Sigma}$ mit $\mathcal{P}$ zusammenfällt. Desweiteren besteht die $\mathcal{H}igh_1^{\Sigma}$ Stufe der $\mathcal{H}igh$-Hierarchie aus allen $\leq_T^{SN}$-vollständigen $\mathcal{NP}$-Sprachen, was sich unmittelbar aus der in Satz 2.3 gezeigten Charakterisierung der $\leq_T^{SN}$-Reduzierbarkeit wie folgt ergibt:

$$\begin{aligned} \mathcal{NP}(\mathrm{SAT}) \subseteq \mathcal{NP}(A) &\Leftrightarrow \mathrm{SAT} \in \mathcal{NP}(A) \cap co\text{-}\mathcal{NP}(A), \text{ nach Satz 2.3} \\ &\Leftrightarrow \mathrm{SAT} \leq_T^{SN} A \\ &\Leftrightarrow A \text{ ist } \leq_T^{SN}\text{-hart für } \mathcal{NP}. \end{aligned}$$

Entsprechend besteht $\mathcal{L}ow_1^\Sigma$ aus genau den Mengen, die im sogenannten Null-Grad $\{A \mid \mathcal{NP}(A) \subseteq \mathcal{NP}(\emptyset)\} = \mathcal{NP} \cap co\text{-}\mathcal{NP}$ der $\leq_T^{SN}$-Reduzierbarkeit liegen (siehe auch Proposition 4.3).

Anschaulich bilden die Stufen der $\mathcal{H}igh$-Hierarchie also zunehmende Erweiterungen der Klasse aller Turing-vollständigen $\mathcal{NP}$-Probleme, während die Stufen der $\mathcal{L}ow$-Hierarchie den Schritt von $\mathcal{P}$ nach $\mathcal{NP} \cap co\text{-}\mathcal{NP}$ fortsetzen. Diese Vorstellung läßt sich folgendermaßen präzisieren. Verallgemeinern wir die Turing-Reduzierbarkeit $\leq_T^{\mathcal{P}}$ und Longs „starke nichtdeterministische Turing-Reduzierbarkeit" $\leq_T^{SN}$ weiter zu den starken Turing-Reduzierbarkeiten

$$A \leq_T^{strong\text{-}\Sigma_k^{\mathrm{P}}} B \quad \Leftrightarrow_{def} \quad \Sigma_k^{\mathrm{P}}(A) \subseteq \Sigma_k^{\mathrm{P}}(B),$$

so folgt unmittelbar, daß die $\mathcal{H}igh_k^\Sigma$-Stufen jeweils genau die $\mathcal{NP}$-vollständigen Probleme bzgl. der $\leq_T^{strong\text{-}\Sigma_k^{\mathrm{P}}}$-Reduzierbarkeit enthalten. Entsprechend korrespondieren die verschiedenen Stufen der $\mathcal{L}ow$-Hierarchie offensichtlich zu den Null-Graden dieser Reduzierbarkeiten, das heißt, sie enthalten genau diejenigen $\mathcal{NP}$ Mengen, die auf die leere Menge reduzierbar sind. Schöning [Sch83] hat das oben erwähnte Resultat von Ladner dahingehend erweitert, daß er (unter der stärkeren Annahme, daß die Polynomialzeit-Hierarchie echt ist) die Existenz von Sprachen $A \in \mathcal{NP}$ nachgewiesen hat, die weder low noch high sind. Das heißt, für alle $k \geq 0$ gilt $\Sigma_k^{\mathrm{P}} \neq \Sigma_k^{\mathrm{P}}(A) \neq \Sigma_{k+1}^{\mathrm{P}}$.

Die wichtigste Konsequenz die sich aus der Einordnung einer Menge A in die $\mathcal{L}ow$-Hierarchie ergibt, ist die Zusicherung, daß A nicht gleichzeitig in der $\mathcal{H}igh$-Hierarchie (und folglich auch nicht $\mathcal{NP}$-vollständig) sein kann, ohne daß die Polynomialzeit-Hierarchie kollabiert. Der Kollaps der Polynomialzeit-Hierarchie fällt dabei um so tiefer aus, je tiefer A in der $\mathcal{L}ow$-Hierarchie angesiedelt ist. Nehmen wir etwa an, eine Sprache A wäre sowohl in $\mathcal{H}igh_k^\Sigma$ als auch in $\mathcal{L}ow_l^\Sigma$ enthalten, so würde dies für $k = max\{l, m\}$ sofort

$$\begin{aligned} \Sigma_k^{\mathrm{P}}(\mathrm{SAT}) &\subseteq \Sigma_k^{\mathrm{P}}(A), \text{ da } A \in \mathcal{H}igh_m^\Sigma \subseteq \mathcal{H}igh_k^\Sigma, \\ &\subseteq \Sigma_k^{\mathrm{P}}, \text{ da } A \in \mathcal{L}ow_m^\Sigma \subseteq \mathcal{L}ow_k^\Sigma \end{aligned}$$

zur Folge haben. Das heißt, die Polynomialzeit-Hierarchie würde zumindest auf die k-te Stufe kollabieren.

In der nächsten Proposition fassen wir nochmals die bisher erwähnten grundlegenden Eigenschaften der $\mathcal{L}ow$- und $\mathcal{H}igh$-Hierarchien zusammen.

Proposition 4.3 [Sch83, Sch86, LS91]

(i) $\mathcal{L}ow_0^\Sigma = \mathcal{L}ow_1^\Theta = \mathcal{L}ow_1^\Delta = \mathcal{P}$.

(ii) $\mathcal{L}ow_1^\Sigma = \mathcal{NP} \cap co\text{-}\mathcal{NP}$.

(iii) $\mathcal{L}ow_{k-1}^\Sigma \subseteq \mathcal{L}ow_k^\Theta \subseteq \mathcal{L}ow_k^\Delta \subseteq \mathcal{L}ow_k^\Sigma$, $k \geq 1$.

(iv) $\mathcal{H}igh_0^\Sigma = \mathcal{H}igh_1^\Theta = \mathcal{H}igh_1^\Delta = \{A \mid A \text{ is } \leq_T^P\text{-vollständig for } \mathcal{NP}\}$.

(v) $\mathcal{H}igh_1^\Sigma = \{A \mid A \text{ ist } \leq_T^{SN}\text{-vollständig für } \mathcal{NP}\}$.

(vi) $\mathcal{H}igh_{k-1}^\Sigma \subseteq \mathcal{H}igh_k^\Theta \subseteq \mathcal{H}igh_k^\Delta \subseteq \mathcal{H}igh_k^\Sigma$, $k \geq 1$.

(vii) *Falls es eine Menge A in $\mathcal{H}igh_k^\Sigma \cap \mathcal{L}ow_k^\Sigma$ gibt, dann kollabiert $\mathcal{PH}$ auf Σ_k^{P}. Analoge Konsequenzen gelten für die Δ- und Θ-Stufen.*

Beispiel 2 *Die Klasse aller spärlichen $\mathcal{NP}$ Sprachen ist in $\mathcal{L}ow_2^\Theta$ enthalten* [Kad89, LS91]. *Der Beweis beruht auf einem interessanten „Zensus-Argument". Ist nämlich S eine spärliche $\mathcal{NP}$ Sprache, so ist die Zensus-Funktion $h : 0^n \mapsto 0^{\|S^{\leq n}\|}$, die für jedes n die Anzahl der Wörter in S bis zur Länge n angibt, durch binäre Suche unter Befragung des $\mathcal{NP}$ Orakels O, definiert als*

$$\{\langle 0^n, 0^k\rangle \mid \textit{es gibt mindestens } k \textit{ Wörter in } S, \textit{ deren Länge} \leq n \textit{ ist }\},$$

in $\mathcal{F}\Theta_2^{\mathrm{P}}$ berechenbar. Ist nun L eine Sprache in $\mathcal{NP}(S)$, so gibt es eine $\mathcal{NP}$ Orakelmaschine M mit $L(M,S) = L$, die für ein Polynom p bei Eingaben der Länge n nur Fragen bis zu einer Länge von $p(n)$ an S stellt. Betrachten wir nun die $\mathcal{NP}$ Sprache L', definiert als

$$\langle x, 0^k\rangle \in L' \quad \Leftrightarrow_{\mathrm{def}} \quad \begin{array}{l} \textit{es gibt } k \textit{ Wörter } y_1,\ldots,y_k \textit{ der Länge} \leq p(|x|) \\ \textit{in } S, \textit{ so daß } x \textit{ zu } L(M,\{y_1,\ldots,y_k\}) \textit{ gehört,} \end{array}$$

so ist leicht zu sehen, daß ein Wort x genau dann in L enthalten ist, wenn das Paar $\langle x, h(0^{p(|x|)})\rangle$ zu L' gehört. Da somit L durch eine logarithmische Anzahl von Fragen an das $\mathcal{NP}$ Orakel O und eine weitere Frage an L' entscheidbar ist, erhalten wir die Inklusion von $\mathcal{NP}(S)$ in Θ_2^{P}. Da Θ_2^{P} jedoch unter dem $\Theta(\cdot)$-Operator abgeschlossen ist, folgt hieraus die Zugehörigkeit von S zu $\mathcal{L}ow_2^\Theta$:

$$\Theta_2^{\mathrm{P}}(S) \;=\; \Theta(\mathcal{NP}(S)) \;\subseteq\; \Theta(\Theta_2^{\mathrm{P}}) \;\subseteq\; \Theta_2^{\mathrm{P}}.$$

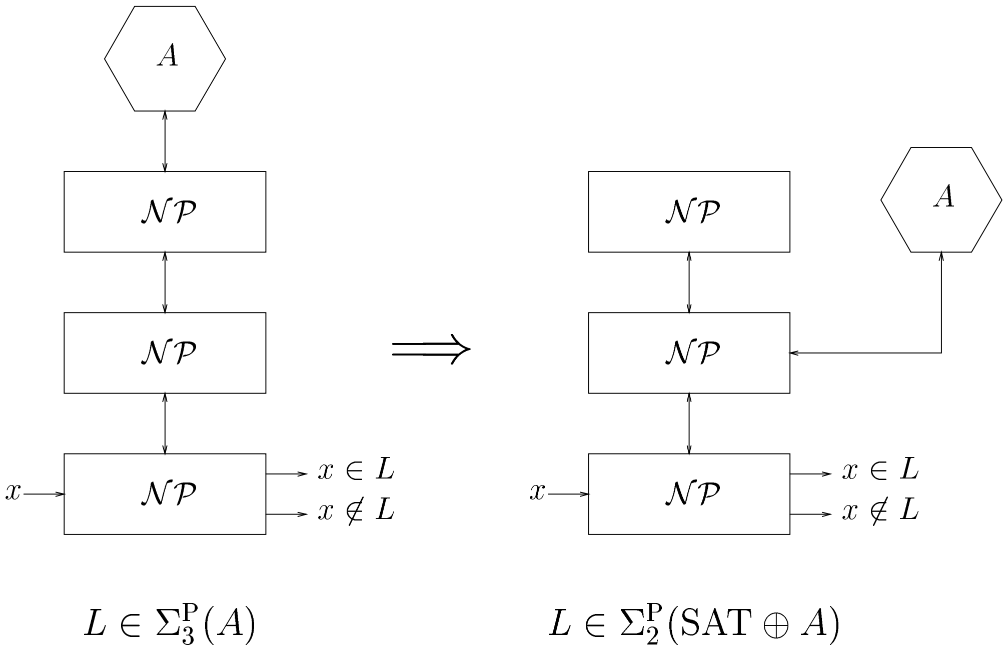


Abbildung 4.3: Eine Menge A in $E\mathcal{L}_3^{\Sigma}$.

Um auch Sprachen außerhalb von $\mathcal{NP}$ nach ihrem Informationsgehalt klassifizieren zu können, haben Balcázar, Book und Schöning [BBS86] die erweiterten $\mathcal{L}ow$- und $\mathcal{H}igh$-Hierarchien eingeführt. Die entsprechenden Delta- und Theta-Stufen wurden einige Jahre später in [AH92, LS91] definiert. Da wir hier die erweiterte $\mathcal{H}igh$-Hierarchie nicht betrachten werden, begnügen wir uns damit, eine Definition für die erweiterte $\mathcal{L}ow$-Hierarchie anzugeben.

Definition 4.4 [BBS86, AH92, LS91] *Die Σ, Δ, und Θ-Stufen der erweiterten $\mathcal{L}ow$-Hierarchie sind folgendermaßen definiert:*

a) $E\mathcal{L}_k^{\Sigma} = \{A \mid \Sigma_k^{\mathrm{P}}(A) \subseteq \Sigma_{k-1}^{\mathrm{P}}(\mathrm{SAT} \oplus A)\}$, $k \geq 1$,

b) $E\mathcal{L}_k^{\Delta} = \{A \mid \Delta_k^{\mathrm{P}}(A) \subseteq \Delta_{k-1}^{\mathrm{P}}(\mathrm{SAT} \oplus A)\}$, $k \geq 2$,

c) $E\mathcal{L}_k^{\Theta} = \{A \mid \Theta_k^{\mathrm{P}}(A) \subseteq \Theta_{k-1}^{\mathrm{P}}(\mathrm{SAT} \oplus A)\}$, $k \geq 2$.

Gemäß dieser Definition ist eine Menge A dann in einer bestimmten Stufe der erweiterten $\mathcal{L}ow$-Hierarchie (beispielsweise in $E\mathcal{L}_3^{\Sigma}$) enthalten, wenn die gesamte Information, die über den zugehörigen Operator (im Beispiel wäre dies der $\Sigma_3^{\mathrm{P}}(\cdot)$-Operator) zugänglich ist, auch schon mit dem Operator der nächstkleineren Stufe (hier also $\Sigma_2^{\mathrm{P}}(\cdot)$) aus A gewonnen werden kann. Da dem schwächeren Operator zusätzlich zu A

der Zugriff auf das $\mathcal{NP}$-vollständige Orakel SAT gewährt wird, stehen abgesehen von dem durch das Orakel A bewirkten Informationsgewinn auf beiden Seiten dieselben Ressourcen zur Verfügung. Der einzige Unterschied zwischen den beiden Berechnungsmodi besteht offensichtlich darin, daß der erste Operator einen mächtigeren Zugriff auf das Orakel A hat als der zweite (siehe Abb. 4.3).

Intuitiv steht hinter der Definition der erweiterten $\mathcal{L}ow$-Hierarchie die Idee, daß für die Ausschöpfung des Informationspotentials einer Menge sicherlich dann nur ein begrenzter Aufwand betrieben werden muß, wenn diese Menge nur einen geringen Informationsgehalt hat. In diesem Fall ist nämlich nicht zu erwarten, daß die Bereitstellung von zusätzlichen Ressourcen für die Extraktion der in dieser Menge enthaltenen Information auch zu einem erhöhten Informationsgewinn führt, der das Entscheiden von zusätzlichen Sprachen erlauben würde.

Es ist jedoch zu beachten, daß dasselbe Phänomen auch bei Orakel beobachtet werden kann, die einen hohen Gehalt an gut strukturierter Information haben, da sie in dieser Form ohne großen Aufwand abrufbar ist. Die $\mathcal{PSPACE}$-vollständige Menge QBF, die aus allen wahren quantifizierten Booleschen Aussagen besteht, hat für Σ_k^{P}-Berechnungen sicherlich einen sehr großen Informationsgehalt. Nichtsdestotrotz ist QBF bereits in der ersten Stufe $E\mathcal{L}_2^{\Delta}$ der erweiterten $\mathcal{L}ow$-Hierarchie enthalten, da die gesamte in QBF enthaltene Information bereits durch den $\mathcal{P}(\cdot)$-Operator extrahiert werden kann:

$$\mathcal{P}(\mathcal{NP}(\mathrm{QBF})) \;\subseteq\; \mathcal{P}(\mathrm{SAT} \oplus \mathrm{QBF}) \;\subseteq\; \mathcal{P}(\mathrm{QBF}) \;=\; \mathcal{PSPACE}.$$

Im Unterschied zur $\mathcal{L}ow$-Hierarchie innerhalb von $\mathcal{NP}$ klassifiziert die erweiterte $\mathcal{L}ow$-Hierarchie also eine Sprache A nicht in erster Linie gemäß ihrer Nützlichkeit als Orakel, sondern vielmehr nach dem Aufwand, der nötig ist, um die in dieser Sprache enthaltene Information mit Hilfe einer parallel dazu verlaufenden Befragung von SAT vollständig auszuschöpfen.

Handelt es sich bei A allerdings um eine $\mathcal{NP}$-Sprache, so wird der Zugriff auf A in der $\Sigma_k^{\mathrm{P}}(\mathrm{SAT} \oplus A)$ Berechnung von $L \in \Sigma_{k+1}^{\mathrm{P}}(A)$ überflüssig (siehe auch Abb. 4.3). Formal liegt dies daran, daß für eine $\mathcal{NP}$-Menge A die beiden Sprachen SAT und $\mathrm{SAT} \oplus A$ (many-one) äquivalent sind, das heißt, $\Sigma_k^{\mathrm{P}}(\mathrm{SAT})$ und $\Sigma_k^{\mathrm{P}}(\mathrm{SAT} \oplus A)$ sind ein und dieselbe Klasse, weshalb für eine $\mathcal{NP}$-Menge A Definition 4.2 und Definition 4.4 dieselbe Eigenschaft beschreiben.

Folglich fallen – wie der Name „erweiterte“ $\mathcal{L}ow$-Hierarchie bereits suggeriert – die einzelnen Stufen der erweiterten $\mathcal{L}ow$-Hierarchie mit den entsprechenden Stufen der $\mathcal{L}ow$-Hierarchie zusammen, wenn man nur Mengen innerhalb von $\mathcal{NP}$ in Betracht zieht (siehe Teil *(iv)* in Proposition 4.5). Man beachte jedoch, daß diese Korrespondenz zwischen den einzelnen Stufen der erweiterten $\mathcal{L}ow$-Hierarchie und der $\mathcal{L}ow$-Hierarchie erst ab der Stufe $E\mathcal{L}_2^{\Delta}$ (als der kleinsten Stufe der erweiterten $\mathcal{L}ow$-Hierarchie) beginnt.

Die nächste Proposition gibt einen kompletten Überblick über die Inklusionsstruktur der einzelnen Stufen der erweiterten $\mathcal{L}ow$-Hierarchie. Demnach ist $E\mathcal{L}_2^{\Delta}$ die kleinste Stufe der erweiterten $\mathcal{L}ow$-Hierarchie, deren weitere Stufen $E\mathcal{L}_2^{\Sigma}$, $E\mathcal{L}_3^{\Theta}$ usw. sich – wie in [SL94] gezeigt – zu einer echten Hierarchie aufschichten. Ein weiterer wichtiger Unterschied zur $\mathcal{L}ow$-Hierarchie besteht darin, daß die Stufen der erweiterten $\mathcal{L}ow$-Hierarchie offenbar nur noch unter Turing-Äquivalenz und nicht unter Turing-Reduzierbarkeit abgeschlossen sind.

Proposition 4.5 [BBS86, AH92, LS91, SL94]

(i) $E\mathcal{L}_1^{\Sigma} = E\mathcal{L}_2^{\Theta} = E\mathcal{L}_2^{\Delta}$.

(ii) $E\mathcal{L}_k^{\Delta} \subset E\mathcal{L}_k^{\Sigma} \subset E\mathcal{L}_{k+1}^{\Theta} \subset E\mathcal{L}_{k+1}^{\Delta}$, $k \geq 2$.

(iii) *Alle Stufen der erweiterten $\mathcal{L}ow$-Hierarchie sind unter Turing-Äquivalenz abgeschlossen.*

(iv) $E\mathcal{L}_k^{\Delta} \cap \mathcal{NP} = \mathcal{L}ow_k^{\Delta}$, $E\mathcal{L}_k^{\Sigma} \cap \mathcal{NP} = \mathcal{L}ow_k^{\Sigma}$, $k \geq 2$, *und*
$E\mathcal{L}_k^{\Theta} \cap \mathcal{NP} = \mathcal{L}ow_k^{\Theta}$, $k \geq 3$.

Wir beenden diesen Abschnitt mit einem kurzen Überblick, welchen Stufen der erweiterten $\mathcal{L}ow$-Hierarchie die verschiedenen zum Teil in Abschnitt 2.4 diskutierten Teilklassen von $\mathcal{P}/poly$ zuzuordnen sind.

$E\mathcal{L}_2^{\Delta}$: TALLY [BB86], $\mathcal{APT}$ [LS91], $\mathcal{IC}[log, poly]$ [AKM92],

$E\mathcal{L}_2^{\Sigma}$: $\mathcal{BPP}$ [Sch86], $\mathcal{P}$-*selective* und $\mathcal{P}$-*cheatable* [ABG90],

$E\mathcal{L}_3^{\Theta}$: SPARSE, $\mathcal{P}$-*close*, $\mathcal{P}_{1\text{-}tt}$(SPARSE) [LS91],
$\mathcal{P}_{btt}$(SPARSE), $\mathcal{P}_{ctt}$(SPARSE), $\mathcal{P}_{dtt}$(SPARSE) [AKM92].

4.2 Advice und Lowness

Die Klasse $\mathcal{P}/poly$ enthält alle Sprachen A, die eine niedrige *nicht-uniforme* Komplexität haben. Das heißt, für jede Eingabegröße n gibt es ein „Programm“ polynomialer Größe (etwa eine Turingmaschine oder einen Schaltkreis), das alle Eingaben der Größe n in Polynomialzeit entscheidet. Im Gegensatz hierzu muß es für eine Sprache in $\mathcal{P}$ ein (uniformes) Programm geben, das für jede Eingabegröße n die Sprache entscheidet. Der Unterschied zwischen Uniformität und Nicht-Uniformität manifestiert sich also in einer Vertauschung der beiden Quantifizierungen „es gibt ein Programm“ und „für jede Eingabegröße n“.

Tatsächlich kann die *uniforme* Komplexität einer Sprache A im Vergleich zu ihrer nicht-uniformen Komplexität beliebig groß sein, da für die Mitgliedschaft in einer nicht-uniformen Komplexitätsklasse keinerlei Bedingungen an die Konstruierbarkeit der Programme in Abhängigkeit von der Problemeingabe geknüpft sind. Für viele Sprachen in $\mathcal{P}/poly$ existieren sogar überhaupt keine berechenbaren Advice-Funktionen, das heißt, es gibt keine Turingmaschine, die zu gegebener Eingabegröße n in endlicher Zeit ein geeignetes Programm konstruiert. Solche Sprachen können offenbar nicht rekursiv entscheidbar sein.

Dagegen bestehen zwischen der uniformen Komplexität einer Sprache A in $\mathcal{P}/poly$ und der Komplexität, die mit der Berechnung einer Advice-Funktion für A verbunden ist, bestimmte Querbezüge, deren Erforschung – wie wir in diesem Abschnitt sehen werden – für die Einordnung von $\mathcal{P}/poly$ in die erweiterte $\mathcal{L}ow$-Hierarchie von großer Bedeutung ist.

Offensichtlich läßt sich das Entscheidungsproblem für $A \in \mathcal{P}/poly$ ohne Schwierigkeiten auf das Problem zurückführen, eine Advice-Funktion für A zu berechnen: Bei Eingabe x bestimme man den zugehörigen *Advice* $h(|x|)$, und verwende diesen, um die Zugehörigkeit von x zu A in Polynomialzeit zu entscheiden. Für unsere Betrachtungen spielt hingegen die folgende, in die entgegengesetzte Richtung zielende Fragestellung eine wichtige Rolle:

> Welcher Aufwand ist damit verbunden, die Berechnung einer Advice-Funktion für eine beliebige Sprache A in $\mathcal{P}/poly$ auf das Entscheidungsproblem für A zurückzuführen?

Das heißt, wir sind daran interessiert, möglichst gute obere (und zu deren Bestätigung gegebenenfalls auch untere) Komplexitätsschranken für das Problem zu erzielen, *relativ* zu A eine Advice-Funktion für A zu berechnen. Diese Fragestellung ist auch deshalb von Interesse, da es für ein (Praxis-) Problem A in $\mathcal{P}/poly$

durchaus vorteilhaft sein kann, folgendermaßen zu verfahren: Konstruiere für jede Eingabe x einer Länge n, sofern nicht bereits geschehen, einen Advice w_n und verwende diesen, um x und alle zukünftigen Eingaben der Länge n in Polynomialzeit zu entscheiden. Diese Verfahrensweise erfordert zwar für jede vorkommende Eingabelänge n ein möglicherweise aufwendiges „Pre-Processing"; alle weiteren Eingaben der Länge n können danach jedoch unter Verwendung von w_n effizient bearbeitet werden.

Zwischen der Problemstellung, eine möglichst effiziente Advice-Funktion für Sprachen in nicht-uniformen Komplexitätsklassen zu finden, und den Lowness-Eigenschaften dieser Sprachen besteht nun die folgende Querverbindung. Wie im vorigen Abschnitt beschrieben, ergibt sich die Zugehörigkeit einer Sprache A zur erweiterten $\mathcal{L}ow$-Hierarchie aus der Tatsache, daß es mit einem begrenzten Aufwand möglich ist, die gesamte in A enthaltene Information zu extrahieren. Im Falle eines Orakels A in $\mathcal{P}/poly$ kann die durch A zur Verfügung gestellte Information offenbar durch eine geeignete Folge von Advice-Strings beschrieben werden. Folglich kann der Aufwand, der für die Extraktion des in A enthaltenen Informationsgehalts benötigt wird, durch die Berechnungskomplexität einer beliebigen Advice-Funktion für A nach oben abgeschätzt werden. Das heißt, je effizienter die Berechnung einer Advice-Funktion relativ zu A realisiert werden kann, desto stärker sind die sich hieraus ergebenden Lowness-Konsequenzen für die Sprache A.

Die in vielen Lowness-Beweisen angewandte Methode, die in einem Orakel $A \in \mathcal{P}/poly$ enthaltene Information durch eine Folge von korrekten Advice-Strings zu ersetzen und somit eine Befragung von A überflüssig zu machen, wird in folgendem Lemma präzisiert.

Lemma 4.6 *Sei $A \in \mathcal{P}/poly$ und sei h eine zugehörige Advice-Funktion. Dann gibt es zu jeder Sprache $L \in \Sigma_k^{\mathrm{P}}(A)$, $k \geq 0$, ein Polynom q und eine Sprache $L' \in \Sigma_k^{\mathrm{P}}$, die für alle Tupel der Form $\langle x, w_0, \ldots, w_{q(n)}\rangle$, $|x| = n$, die folgende Bedingung erfüllt:*

- *Falls $w_i = h(i)$ für $i = 0, \ldots, q(n)$ ist, dann ist x genau dann in L, wenn $\langle x, w_0, \ldots, w_{q(n)}\rangle$ in L' ist.*

Beweis. Sei $B \in \mathcal{P}$ die zugehörige Interpreter-Menge. Ist nun L eine beliebige Sprache in $\Sigma_k^{\mathrm{P}}(A)$, dann gibt es eine polynomial zeitbeschränkte Orakelmaschine M und ein Polynom p, so daß für alle $x \in \{0,1\}^*$ gilt:

$$x \in L \quad \Leftrightarrow \exists y_1 \, \forall y_2 \, \ldots Q y_k \, \langle x, y_1, y_2, \ldots, y_k\rangle \in L(M, A).$$

Hierbei ist $Q = \forall$, falls k gerade ist, und $Q = \exists$ sonst, und die Quantoren erfassen alle Wörter bis zur Länge $p(|x|)$. Sei $q(n)$ ein Polynom, das die Länge der Orakelfragen von M an A bei Eingaben der Form $\langle x, y_1, y_2, \ldots, y_k \rangle$ mit $|x| \le n$ beschränkt. Betrachte nun die folgende Turingmaschine M'.

> Bei Eingabe $\langle x, y_1, y_2, \ldots, y_k, w_0, \ldots, w_{q(n)} \rangle$, $|x| = n$, simuliere die Maschine M bei Eingabe $\langle x, y_1, y_2, \ldots, y_k \rangle$, und benutze die Advice-Strings $w_0, \ldots, w_{q(n)}$, um die Antworten auf alle Orakelfragen an A mit Hilfe der Interpreter-Menge B zu erzeugen. Das heißt, falls M die Orakelfrage z stellt, so fahre in der Simulation mit der Antwort $B(z, w_{|z|})$ fort.

Da B in $\mathcal{P}$ ist, ist M' polynomial zeitbeschränkt. Außerdem ist die von M' durchgeführte Simulation mit der von M unter dem Orakel A ausgeführten Rechnung identisch, falls die mit der Eingabe zusätzlich mitgelieferten Advice-Strings $w_0, \ldots, w_{q(n)}$ korrekt sind. Das heißt, unter der Voraussetzung, daß $w_i = h(i)$ für $i = 0, \ldots, q(n)$ ist, erfüllt die Sprache L', definiert durch

$$\begin{aligned} &\langle x, w_0, \ldots, w_{q(n)} \rangle \in L' \quad \Leftrightarrow_{def} \\ &\qquad \exists y_1 \, \forall y_2 \, \ldots Q y_k \, \langle x, y_1, \ldots, y_k, w_0, \ldots, w_{q(n)} \rangle \in L(M') \end{aligned}$$

die gewünschte Äquivalenz und liegt überdies in Σ_k^{P}. ■

Das folgende Lemma formalisiert die Querverbindung zwischen oberen Komplexitätsschranken für die Berechnung von Advice-Funktionen und den daraus resultierenden Lowness-Eigenschaften.

Lemma 4.7 *Falls eine Sprache $A \in \mathcal{P}/poly$ eine Advice-Funktion in der Klasse $\mathcal{FP}(\Sigma_k^{\mathrm{P}}(A \oplus \mathrm{SAT}))$ hat, so folgt $A \in E\mathcal{L}_{k+2}^{\Delta}$.*

Beweis. Sei $B \in \mathcal{P}$ eine Interpreter-Menge und sei $h \in \mathcal{FP}(C)$ eine Advice-Funktion für A, wobei C ein geeignetes Orakel in $\Sigma_k^{\mathrm{P}}(A \oplus \mathrm{SAT})$ ist. Da die Klasse $\Delta_{k+1}^{\mathrm{P}}(A \oplus \mathrm{SAT})$ unter dem $\mathcal{P}(\cdot)$-Operator abgeschlossen ist, genügt es, die Inklusion von $\Sigma_{k+1}^{\mathrm{P}}(A)$ in $\Delta_{k+1}^{\mathrm{P}}(A \oplus \mathrm{SAT})$ zu zeigen. Sei also L eine beliebige Sprache in $\Sigma_{k+1}^{\mathrm{P}}(A)$. Nach Lemma 4.6 gibt es ein Polynom $q(n)$ und eine Sprache L' in $\Sigma_{k+1}^{\mathrm{P}}$, die für alle Tupel der Form $\langle x, w_0, \ldots, w_{q(n)} \rangle$, $|x| = n$, folgende Bedingung erfüllt:

$$w_i = h(i) \text{ für } i = 0, \ldots, q(n) \quad \Rightarrow \quad (x \in L \Leftrightarrow \langle x, w_0, \ldots, w_{q(n)} \rangle \in L').$$

L ist nun in $\mathcal{P}(C \oplus L') \subseteq \Delta_{k+1}^{\mathrm{P}}(A \oplus \mathrm{SAT})$ wie folgt entscheidbar:

Bei Eingabe x berechne zuerst unter Befragung von Orakel C für $i = 0, 1, \ldots, q(|x|)$ geeignete Advice-Strings $w_i = h(i)$, und stelle dann an das Orakel L' die Frage $\langle x, w_0, \ldots, w_{q(n)} \rangle$. Akzeptiere genau dann, wenn diese Frage mit „JA“ beantwortet wird.

Dies zeigt, daß $A \in E\mathcal{L}^{\Delta}_{k+2}$ ist. ■

Beispiel 3 *Es ist unmittelbar klar, daß eine tally Menge A eine Advice-Funktion h in $\mathcal{FP}(A) \subseteq \mathcal{FP}(\Sigma^{\mathrm{P}}_0(A \oplus \mathrm{SAT}))$ besitzt (etwa $h(n) =_{\mathrm{def}} A(0^n)$). Folglich sind alle tally Sprachen in $E\mathcal{L}^{\Delta}_2$ enthalten* [BB86]. *Dies impliziert sofort, daß alle tally $\mathcal{NP}$ Sprachen in $\mathcal{L}ow^{\Delta}_2$ enthalten sind. Wie wir in Beispiel 2 gesehen haben, ist diese Klasse (und allgemeiner die Klasse aller spärlichen $\mathcal{NP}$ Sprachen) jedoch bereits in $\mathcal{L}ow^{\Theta}_2$ enthalten* [Kad89, LS91].

Um die Zugehörigkeit einer $\mathcal{P}/poly$ Sprache A zu einer der Sigma- oder Theta-Stufen der erweiterten $\mathcal{L}ow$-Hierarchie nachzuweisen, verlagert sich das Problem, eine möglichst effiziente Advice-Funktion für A zu berechnen, auf die Erzielung von möglichst guten oberen Komplexitätsschranken für das Problem, den Graph einer solchen Funktion zu entscheiden. Dabei stellt sich heraus, daß unter Miteinbeziehung von *mehrwertigen* Advice-Funktionen oftmals bessere Schranken erzielbar sind.

VEREINBARUNG

Eine **mehrwertige** Funktion f von A nach B ordnet jedem Element $x \in A$ eine Teilmenge von B zu, die wir mit *set-*$f(x)$ bezeichnen. Wir nennen f **total**, falls *set-*$f(x)$ für kein $x \in A$ die leere Menge ist. Der **Graph** von f ist die Menge *graph-*$f(x) = \{\langle x, y \rangle \mid y \in$ *set-*$f(x)\}$.

Eine **mehrwertige Advice-Funktion** für eine Sprache A in bezug auf eine Interpreter-Menge B ist eine totale mehrwertige Funktion f von $\mathcal{N}$ nach $\{0,1\}^*$, so daß für ein Polynom p und alle $n \geq 0$ gilt: *set-*$f(n)$ enthält nur Strings w der Länge $p(n)$ und jedes solche $w \in$ *set-*$f(n)$ hat die Eigenschaft, daß es alle Eingaben x der Länge n korrekt entscheidet, das heißt, es gilt $x \in A \Leftrightarrow \langle x, w \rangle \in B$ für alle $x \in \{0,1\}^n$.

Der Zusammenhang zwischen den Lowness-Eigenschaften einer Sprache A in $\mathcal{P}/poly$ und der Komplexität des Problems, den Graph einer Advice-Funktion für A zu entscheiden, wird im folgenden Lemma aufgezeigt.

Lemma 4.8 *Sei h eine (evtl. mehrwertige) Advice-Funktion für eine Sprache $A \in \mathcal{P}/poly$. Dann gilt:*

(i) $\textit{graph-}h \in \Sigma_k^{\mathrm{P}}(A \oplus \mathrm{SAT}) \Rightarrow A \in E\mathcal{L}_{k+1}^{\Sigma}$,

(ii) $\textit{graph-}h \in \Sigma_k^{\mathrm{P}}(A \oplus \mathrm{SAT})/\mathcal{FP}_{tt}(\Sigma_k^{\mathrm{P}}(A \oplus \mathrm{SAT})) \Rightarrow A \in E\mathcal{L}_{k+2}^{\Theta}$.

Beweis. Sei h eine Advice-Funktion für A, und sei p ein Polynom, das die Länge aller Strings in $\textit{set-}h(n)$ angibt. Für eine Sprache L in $\Sigma_{k+1}^{\mathrm{P}}(A)$ sei im folgenden $q(n)$ ein Polynom, das die Länge der Orakelfragen an A in der entsprechenden $\Sigma_{k+1}^{\mathrm{P}}(A)$-Berechnung bei Eingaben der Länge n begrenzt. Genauso wie im Beweis von Lemma 4.6 läßt sich zu L eine Sprache L' in $\Sigma_{k+1}^{\mathrm{P}}$ definieren, die für alle Tupel der Form $\langle x, w_0, \ldots, w_{q(n)} \rangle$, $|x| = n$, die folgende Bedingung erfüllt:

$$\{\langle 0, w_0\rangle, \ldots, \langle q(n), w_{q(n)}\rangle\} \subseteq \textit{graph-}h \quad \Rightarrow \quad (x \in L \Leftrightarrow \langle x, w_0, \ldots, w_{q(n)}\rangle \in L').$$

Für den Beweis von Teil *(i)* können wir weiterhin voraussetzen, daß $\textit{graph-}h$ in $\Sigma_k^{\mathrm{P}}(A \oplus \mathrm{SAT})$ liegt. Dann ist aber L in $\Sigma_k^{\mathrm{P}}(A \oplus \mathrm{SAT})$ wie folgt entscheidbar:

input x, $|x| = n$
for $i := 0$ **to** $q(n)$ **do guess** w_i, $|w_i| = p(i)$ **end**
if $\{\langle 0, w_0\rangle, \ldots, \langle q(n), w_{q(n)}\rangle\} \subseteq \textit{graph-}h$ **and** $\langle x, w_0, \ldots, w_{q(n)}\rangle \in L'$ **then**
  accept
else reject end

Für den Beweis von Teil *(ii)* bemerken wir zunächst, daß es ausreicht, die Inklusion $\Sigma_{k+1}^{\mathrm{P}}(A) \subseteq \Theta_{k+1}^{\mathrm{P}}(A \oplus \mathrm{SAT})$ zu zeigen. Aufgrund der Abgeschlossenheit der Θ_k^{P}-Klassen unter dem $\Theta(\cdot)$-Operator impliziert dies nämlich sofort $\Theta_{k+2}^{\mathrm{P}}(A) = \Theta(\Sigma_{k+1}^{\mathrm{P}}(A)) \subseteq \Theta(\Theta_{k+1}^{\mathrm{P}}(A \oplus \mathrm{SAT})) \subseteq \Theta_{k+1}^{\mathrm{P}}(A \oplus \mathrm{SAT})$. Sei also h eine Advice-Funktion für A mit $\textit{graph-}h \in \Sigma_k^{\mathrm{P}}(A \oplus \mathrm{SAT})/\mathcal{FP}_{tt}(\Sigma_k^{\mathrm{P}}(A \oplus \mathrm{SAT}))$ und sei $L \in \Sigma_{k+1}^{\mathrm{P}}(A)$. Zu L gibt es wie oben eine geeignete Sprache $L' \in \Sigma_{k+1}^{\mathrm{P}}$, die für alle n und alle $x \in \{0,1\}^n$ die folgende Bedingung erfüllt:

$$x \in L \quad \Leftrightarrow \quad \begin{array}{l} \text{es gibt Advice-Strings } w_0, \ldots, w_{q(n)}, \text{ so daß} \\ \{\langle 0, w_0\rangle, \ldots, \langle q(n), w_{q(n)}\rangle\} \subseteq \textit{graph-}h \text{ und} \\ \langle x, w_0, \ldots, w_{q(n)}\rangle \in L' \text{ ist.} \end{array}$$

Wegen $L' \in \Sigma^{\mathrm{P}}_{k+1}$ und *graph-h* $\in \Sigma^{\mathrm{P}}_k(A \oplus \mathrm{SAT})/\mathcal{FP}_{tt}(\Sigma^{\mathrm{P}}_k(A \oplus \mathrm{SAT}))$ zeigt dies, daß L in $\Sigma^{\mathrm{P}}_k(A \oplus \mathrm{SAT})/\mathcal{FP}_{tt}(\Sigma^{\mathrm{P}}_k(A \oplus \mathrm{SAT})) \subseteq \mathcal{P}_{tt}(\Sigma^{\mathrm{P}}_k(A \oplus \mathrm{SAT})) = \Theta^{\mathrm{P}}_{k+1}(A \oplus \mathrm{SAT})$ enthalten ist. Folglich ist A in $E\mathcal{L}^{\Theta}_{k+2}$. ■

Die Zugehörigkeit aller Sprachen in $\mathcal{P}/poly$ zur erweiterten $\mathcal{L}ow$-Hierarchie konnte erstmals von Balcázar, Book und Schöning [BBS86] gezeigt werden. Den Schlüssel zu diesem Resultat liefern die folgenden oberen Schranken für die Komplexität von Advice-Funktionen.

Satz 4.9 [KS85, Sch86] *Jede Sprache A in $\mathcal{P}/poly$ hat Advice-Funktionen g und h mit folgenden Eigenschaften:*

(i) *graph-g* $\in$ *co-*$\mathcal{NP}(A)$ *und*

(ii) $h \in \mathcal{FP}(\Sigma^{\mathrm{P}}_2(A))$.

Beweis. Sei $A \in \mathcal{P}/poly$. Dann gibt es eine Interpreter-Menge $B \in \mathcal{P}$ und eine Advice-Funktion $\hat{h} \in poly$ für A. Sei p ein Polynom, das die Länge von $\hat{h}(n)$ angibt. Betrachte die (mehrwertige) Funktion g, definiert durch

$$w \in \textit{set-g}(n) \quad \Leftrightarrow_{\mathit{def}} \quad |w| = p(n) \text{ und für alle } x \in \{0,1\}^n \text{ gilt}$$
$$x \in A \Leftrightarrow \langle x, w\rangle \in B.$$

Das heißt, in *set-g*(n) sind alle Strings w enthalten, die als Advice für die Eingabelänge n verwendet werden können, da sie für alle Eingaben dieser Länge die richtige Entscheidung treffen. Da *graph-g* offensichtlich in *co-*$\mathcal{NP}(A)$ liegt, und es nach Voraussetzung für jedes n mindestens einen Advice-String der Länge $p(n)$ in *set-g*(n) gibt, ist g weiterhin total und somit *(i)* gezeigt. Für den Beweis von Teil *(ii)* können wir eine Advice-Funktion h wie folgt definieren:

$$h(n) \quad =_{\mathit{def}} \quad \min\{w \mid w \in \textit{set-g}(n)\}.$$

Das heißt, $h(n)$ ist der lexikographisch kleinste String w in der Menge *set-g*(n). Dieser kann leicht durch Präfix-Suche unter Befragung des Orakels

$$D \quad =_{\mathit{def}} \quad \{\langle 0^n, u\rangle \mid \text{es gibt ein } v \text{ mit } |uv| = p(n) \text{ und } uv \in \textit{set-g}(n)\}$$

bestimmt werden. Da D offensichtlich in $\Sigma^{\mathrm{P}}_2(A)$ entscheidbar ist, folgt die Zugehörigkeit von h zu $\mathcal{FP}(\Sigma^{\mathrm{P}}_2(A))$. ■

Teil *(ii)* von Satz 4.9 besagt, daß der gesamte Informationsgehalt einer $\mathcal{P}/poly$ Sprache A aus dieser über einen $\mathcal{P}(\Sigma_2^{\mathrm{P}}(\cdot))$-Zugriff extrahiert werden kann. Dies impliziert unter Verwendung von Lemma 4.7, daß $\mathcal{P}/poly$ in $E\mathcal{L}_4^{\Delta}$ enthalten ist. Teil *(i)* liefert allerdings ein stärkeres Lowness-Resultat.

Korollar 4.10

(i) [BBS86] $\mathcal{P}/poly \subseteq E\mathcal{L}_3^{\Sigma}$,

(ii) [KS85] $\mathcal{P}/poly \cap \mathcal{NP} \subseteq \mathcal{L}ow_3^{\Sigma}$.

Beweis. *(i)* ist eine unmittelbare Folgerung aus Satz 4.9 *(i)* und Lemma 4.8 *(i)*. *(ii)* folgt aus *(i)* unter Verwendung von Proposition 4.5 *(iv)*. ∎

Wir schließen diesen Abschnitt mit einer Verallgemeinerung von Lemma 4.6 zu Sprachen A in der Klasse $\mathcal{NP}/poly \cap co\text{-}\mathcal{NP}/poly$, die sich leicht durch Kombination mit der im Beweis von Satz 2.3 verwendeten Technik erhalten läßt. Um jedoch nicht zwei verschiedene Advice-Funktionen (eine für A und eine für $\overline{A}$) betrachten zu müssen, verwenden wir stattdessen die hierzu äquivalente Charakterisierung, daß $A \oplus \overline{A}$ in $\mathcal{NP}/poly$ enthalten ist.

Lemma 4.11 *Sei h eine (evtl. mehrwertige) Advice-Funktion für eine Sprache $A \oplus \overline{A}$ in $\mathcal{NP}/poly$. Dann gibt es zu jeder Sprache $L \in \Sigma_k^{\mathrm{P}}(A)$, $k \geq 1$, ein Polynom q und eine Sprache $L' \in \Sigma_k^{\mathrm{P}}$, die für alle Tupel der Form $\langle x, w_0, \ldots, w_{q(n)} \rangle$, $|x| = n$, die folgende Bedingung erfüllt:*

$$\{\langle 0, w_0 \rangle, \ldots, \langle q(n), w_{q(n)} \rangle\} \subseteq graph\text{-}h \quad \Rightarrow \quad (x \in L \Leftrightarrow \langle x, w_0, \ldots, w_{q(n)} \rangle \in L').$$

Beweis. Nach Voraussetzung gibt es eine Sprache $B \in \mathcal{NP}$, so daß für alle n, alle $z \in \{0,1\}^n$ und alle $w_n \in set\text{-}h(n)$ gilt:

$$\begin{aligned} z \in A \quad &\Leftrightarrow \quad \langle z, 1, w_n \rangle \in B, \\ z \notin A \quad &\Leftrightarrow \quad \langle z, 0, w_n \rangle \in B. \end{aligned}$$

Sei weiterhin B' eine Sprache in $\mathcal{P}$ und p' ein Polynom mit $B = \{\langle z, b, w \rangle \mid \exists u, |u| = p'(|z|) : \langle z, b, w, u \rangle \in B\}$. Ist nun L eine beliebige Sprache in $\Sigma_k^{\mathrm{P}}(A)$, $k \geq 1$, dann gibt es wie im Beweis von Lemma 4.6 eine polynomial zeitbeschränkte Orakelmaschine M und ein Polynom p, so daß für alle $x \in \{0,1\}^*$ gilt:

$$x \in L \quad \Leftrightarrow \exists y_1 \, \forall y_2 \, \ldots Q y_k \, \langle x, y_1, y_2, \ldots, y_k \rangle \in L(M, A).$$

Sei $q(n)$ ein Polynom, das die Länge der Orakelfragen von M an A bei Eingaben der Form $\langle x, y_1, y_2, \ldots, y_k \rangle$ mit $|x| \leq n$ beschränkt, und sei $q'(n)$ ein Polynom, das die Anzahl der Orakelfragen von M an A bei allen solchen Eingaben beschränkt. Betrachte nun die folgende Turingmaschine M'.

> Bei Eingabe $\langle x, y_1, y_2, \ldots, y_k, u_1, \ldots, u_{q'(n)}, w_0, \ldots, w_{q(n)} \rangle$, $|x| = n$, simuliere die Maschine M bei Eingabe $\langle x, y_1, y_2, \ldots, y_k \rangle$, und benutze die Strings u_i und w_j, um die Antworten auf alle Orakelfragen an A wie folgt zu berechnen: Ist z_i die i-te Orakelfrage, so teste ob eines der beiden Tupel $\langle z_i, b, w_{|z_i|}, u_i \rangle$, $b \in \{0, 1\}$, in B' enthalten ist. Ist dies der Fall, so fahre in der Simulation mit der Antwort b fort. Andernfalls verwerfe im Fall, daß k ungerade ist, und akzeptiere im Fall k gerade (dies bewirkt, daß sich diese Rechnung nicht auf die Zugehörigkeit von $\langle x, w_0, \ldots, w_{q(n)} \rangle$ zu der weiter unten definierten Sprache L' auswirken kann).

Da B' eine Sprache in $\mathcal{P}$ ist, ist M' polynomial zeitbeschränkt. Außerdem ist die von M' durchgeführte Simulation mit der von M unter dem Orakel A ausgeführten Rechnung identisch, falls die mit der Eingabe zusätzlich mitgelieferten Advice-Strings $w_0, \ldots, w_{q(n)}$ korrekt sind, und für jede Orakelfrage z_i (genau) eines der beiden Tupel $\langle z_i, 0, w_{|z_i|}, u_i \rangle$ und $\langle z_i, 0, w_{|z_i|}, u_i \rangle$ in B' enthalten ist. Das heißt, unter der Voraussetzung, daß $\{\langle 0, w_0 \rangle, \ldots, \langle q(n), w_{q(n)} \rangle\} \subseteq$ *graph-h* ist, erfüllt die Sprache $L' \in \Sigma_k^{\mathrm{P}}$, definiert durch

$$\begin{aligned} \langle x, w_0, \ldots, w_{q(n)} \rangle \in L' \quad \Leftrightarrow_{def} \; & \exists y_1 \forall y_2 \ldots Q y_k Q u_1 \ldots Q u_{q'(n)} : \\ & \langle x, u_1, \ldots, u_{q'(n)}, y_1, \ldots, y_k, w_0, \ldots, w_{q(n)} \rangle \in L(M') \end{aligned}$$

die gewünschte Äquivalenz. ∎

Unter Verwendung von Lemma 4.11 können Lemma 4.7 und Lemma 4.8 leicht wie folgt verallgemeinert werden.

Lemma 4.12 *Sei h eine (evtl. mehrwertige) Advice-Funktion für eine Sprache $A \oplus \overline{A}$ in $\mathcal{NP}/poly$. Dann gilt:*

(i) *graph-h* $\in \Sigma_k^{\mathrm{P}}(A \oplus \mathrm{SAT}) \Rightarrow A \in E\mathcal{L}_{k+1}^{\Sigma}$,

(ii) *graph-h* $\in \Sigma_k^{\mathrm{P}}(A \oplus \mathrm{SAT}) / \mathcal{FP}_{tt}(\Sigma_k^{\mathrm{P}}(A \oplus \mathrm{SAT})) \Rightarrow A \in E\mathcal{L}_{k+2}^{\Theta}$,

(iii) $h \in \mathcal{FP}(\Sigma_k^{\mathrm{P}}(A \oplus \mathrm{SAT})) \Rightarrow A \in E\mathcal{L}_{k+2}^{\Delta}$.

Auch Satz 4.9 läßt sich leicht zu folgendem Resultat verallgemeinern.

Satz 4.13 *Jede Sprache $A \oplus \overline{A}$ in $\mathcal{NP}/poly$ hat Advice-Funktionen g und h mit folgenden Eigenschaften:*

(i) graph-$g \in$ co-$\mathcal{NP}(A)$ *und*

(ii) $h \in \mathcal{FP}(\Sigma_2^{\mathrm{P}}(A))$.

Dies führt unmittelbar auf die beiden folgenden Lowness-Resultate.

Korollar 4.14 [HHO+93]

(i) $\mathcal{NP}/poly \cap$ co-$\mathcal{NP}/poly \subseteq E\mathcal{L}_3^{\Sigma}$,

(ii) co-$\mathcal{NP}/poly \cap \mathcal{NP} \subseteq \mathcal{L}ow_3^{\Sigma}$.

4.3 Komplexität von Advice-Funktionen

Durch eine neuartige Anwendung der Hashing-Technik in Verbindung mit der sogenannten „Strategie des Mehrheitsvotums" (siehe auch Abschnitt 6.5) gelang es Gavaldà [Gav92], den Zugriff auf A bei der Berechnung einer Advice-Funktion für eine Sprache A wesentlich effizienter zu gestalten. Genauer gesagt fand Gavaldà einen Weg, die relativ „schwierigen" $\Sigma_2^{\mathrm{P}}(A)$-Fragen (vgl. Teil *(ii)* von Satz 4.9) durch vergleichsweise „einfache" $\mathcal{NP}(A)$-Fragen zu ersetzen. Für die Generierung dieser einfacher zu beantwortenden Fragen benötigte Gavaldà allerdings zusätzlich den Zugriff auf ein Σ_3^{P}-Orakel.

Satz 4.15 [Gav92] *Jede Menge A in $\mathcal{P}/poly$ hat Advice-Funktionen sowohl in $\mathcal{FP}(\mathcal{NP}_1(A) \oplus \Sigma_3^{\mathrm{P}})$ als auch in $\mathcal{FP}_{tt}(\mathcal{NP}(A \oplus \Sigma_2^{\mathrm{P}}))$.*

Aufgrund des zusätzlichen Σ_3^{P}-Orakels führen die von Gavaldà erzielten oberen Schranken, die im übrigen mit den in Satz 4.9 bewiesenen unvergleichbar sind, nicht zu stärkeren Lowness-Resultaten für die Klasse $\mathcal{P}/poly$.

Satz 4.15 hat allerdings die interessante Konsequenz, daß ein Beweis für die Existenz von Sprachen A in $\mathcal{P}/poly$, die keine Advice-Funktion in $\mathcal{FP}(\mathcal{NP}_1(A))$ haben, mindestens so schwer zu erbringen ist wie ein Beweis für $\mathcal{NP} \neq \mathcal{P}$ (tatsächlich würde die Existenz einer solchen Sprache A aufgrund von Satz 4.15 sogar $\Sigma_2^{\mathrm{P}} \neq \Delta_2^{\mathrm{P}}$ implizieren). Von daher erscheint es sehr unwahrscheinlich, daß die folgenden unteren Schranken, die fast an $\mathcal{FP}(\mathcal{NP}_1(A))$ herankommen, in absehbarer Zeit verbessert werden.

Satz 4.16 [GW93] *Es gibt Sprachen A in $\mathcal{P}/poly$, die weder eine Advice-Funktion in $\mathcal{FP}(\mathcal{NP}(A) \cap co\text{-}\mathcal{NP}(A))$ noch in $\mathcal{FP}(\mathcal{NP}_m(A))$ (und folglich auch nicht in $\mathcal{FP}(\mathcal{NP}_{\overline{m}}(A))$) haben.*

Schon eher vorstellbar wäre eine Verbesserung der oberen Komplexitätsschranke auf $\mathcal{FP}(\mathcal{NP}_1(A))$ (wie in der Einleitung bereits erwähnt wurde, läßt sich aus dem in [BCKT94] enthaltenen Erlernbarkeitsbeweis eine obere Schranke von $\mathcal{FZPP}(\mathcal{NP}_1(A))$ ableiten). Einen weiteren (mit der $\mathcal{FZPP}(\mathcal{NP}_1(A))$-Schranke unvergleichbaren) Schritt in diese Richtung bedeutet der folgende Satz 4.17, der sowohl die $\mathcal{FP}(\Sigma_2^{\mathrm{P}}(A))$ Schranke aus Satz 4.9 als auch die $\mathcal{FP}(\mathcal{NP}_1(A) \oplus \Sigma_3^{\mathrm{P}})$ aus Satz 4.15 verbessert. Der Beweis verwendet die in Abschnitt 3.3 verfeinerte Hashing-Technik. Interessanterweise würde eine weitere Verbesserung von Satz 4.17 zu der oberen Schranke $\mathcal{FP}(\mathcal{NP}_m(A) \oplus \Sigma_2^{\mathrm{P}})$ oder zu $\mathcal{FP}(\mathcal{NP}_{\overline{m}}(A) \oplus \Sigma_2^{\mathrm{P}})$ aufgrund von Satz 4.16 ebenfalls die Verschiedenheit von Σ_2^{P} und Δ_2^{P} implizieren.

Für den Beweis von Satz 4.17 vereinbaren wir die folgende Notation (siehe auch Abschnitt 6.4).

Vereinbarung

Sei h eine Advice-Funktion für eine Menge A und sei B die zugehörige Interpreter-Menge. Eine **Beispielfolge** s ist eine Folge $\langle x_1, b_1\rangle\#\cdots\#\langle x_k, b_k\rangle$ von Paaren mit $x_i \in \{0,1\}^*$ und $b_i \in \{0,1\}$ für $i = 1, \ldots, k$. Die (unter der Interpreter-Menge B) mit s **konsistenten** Advice-Strings der Länge m fassen wir in der Menge

$$ADV(m,s) \quad = \quad \{w \in \{0,1\}^m \mid \text{ für alle } i = 1, \ldots, k \text{ gilt } B(x_i, w) = b_i\}$$

zusammen.

Satz 4.17 *Für jede Menge A in $\mathcal{P}/poly$ kann eine Advice-Funktion in der Klasse $\mathcal{FP}(\mathcal{NP}_1(A) \oplus \Sigma_2^{\mathrm{P}})$ berechnet werden.*

Beweis. Sei $A \in \mathcal{P}/poly$. Dann gibt es eine Interpreter-Menge $B \in \mathcal{P}$ und eine Advice-Funktion $h \in poly$ für A. Sei p ein Polynom, das die Länge von $h(n)$

input n
$s := \lambda$
loop
$\hat{a} := (2/3)\, g(0^n, s)$
1: **if** $\exists x \in \Sigma^n : \langle s, x, \hat{a}\rangle \in U$ **then**
berechne durch Präfix-Suche das lexikographisch kleinste solche x
$s := s\#\langle x, A(x)\rangle$
2: **else**
$a^* := \hat{a}/2$
berechne durch Präfix-Suche das lexikographisch kleinste y der Länge $q(n)$ mit der folgenden Eigenschaft:

($\star\star\star$) für jedes $x \in \{0,1\}^n$ ist eines der beiden Tupel $\langle s\#\langle x, 0\rangle, y, a^*\rangle$ und $\langle s\#\langle x, 1\rangle, y, a^*\rangle$ in W

3: **if** $\exists x \in \Sigma^n : \langle s\#\langle x, \overline{A}(x)\rangle, y, a^*\rangle \notin W$ **then**
berechne durch Präfix-Suche das lexikographisch kleinste solche x
$s := s\#\langle x, A(x)\rangle$
else exit(**loop**) **end**
end
end loop
berechne den kleinsten korrekten Advice-String w_{min} der Länge $p(n)$ durch eine Präfixsuche unter Befragung des Orakels V
output w_{min}

Abbildung 4.4: Berechnung einer Advice-Funktion in $\mathcal{FP}(\mathcal{NP}_1(A) \oplus \Sigma_2^{\mathrm{P}})$.

angibt. Unser $\mathcal{FP}(\mathcal{NP}_1(A) \oplus \Sigma_2^{\mathrm{P}})$ Algorithmus M (siehe Abb. 4.4) konstruiert sich bei Eingabe n schrittweise eine Beispielfolge $s = \langle x_1, b_1\rangle\# \cdots \#\langle x_k, b_k\rangle$ von Paaren mit $x_i \in \{0,1\}^n$ und $b_i = A(x_i)$ für $i = 1, \ldots, k$. Das heißt, zumindest der Advice-String $h(n)$ ist zu jedem Zeitpunkt der Rechnung mit s konsistent, und folglich gilt $\|ADV(p(n), s)\| \geq 1$ für alle im Verlauf der Rechnung generierten Beispielfolgen s.

Die Grundidee, auf der die von M durchgeführte Konstruktion von s beruht, ist die folgende: Solange sich noch eine Eingabe $x \in \{0,1\}^n$ finden läßt, die von einem gewissen Mindestanteil aller in der Menge $ADV(p(n), s)$ enthaltenen Strings falsch entschieden wird, erweitert M in jedem Durchlauf der loop-Schleife die Beispielfolge s um ein entsprechendes Paar $\langle x, A(x)\rangle$. Erst wenn jedes einzelne $x \in \{0,1\}^n$ von einer großen Mehrheit aller mit s konsistenten Advice-Strings korrekt entschieden wird, bricht M die Konstruktion von s ab und verläßt die loop-Schleife.

Wie wir zeigen werden, werden nur solche Eingaben $x \in \{0,1\}^n$ in die Beispielfolge s aufgenommen, die zum Zeitpunkt ihrer Aufnahme von mindestens einem Achtel aller mit s konsistenten Strings inkorrekt entschieden werden. Mit anderen Worten, die Anzahl der Strings in der Menge $ADV(p(n), s)$ wird durch jede Erweiterung von s in der loop-Schleife um mindestens den Faktor 7/8 verkleinert. Da $ADV(p(n), s)$ zu Beginn, das heißt, wenn s mit der leeren Folge initialisiert wird, genau $2^{p(n)}$ Strings enthält, kann s höchstens auf eine Länge von $\lceil p(n)/\log(8/7)\rceil \leq 7p(n)$ Paaren anwachsen, da spätestens dann $ADV(p(n), s)$ leer werden müßte, was aber aufgrund der Konsistenz von $h(n)$ mit s ausgeschlossen ist.

Grundsätzlich gibt es zwei Möglichkeiten für einen String $x \in \{0,1\}^n$, in die Beispielfolge s aufgenommen zu werden. Das Vorliegen der hierfür nötigen Voraussetzungen wird in den beiden if-Bedingungen (siehe Abb. 4.4) überprüft. Bei der ersten ist die Frage, ob x zu A gehört oder nicht, ohne Bedeutung; wohl aber bei der zweiten. Die erste if-Bedingung (mit 1: markiert) ist nur dann erfüllt, wenn es ein $x \in \{0,1\}^n$ gibt mit $\langle s, x, \hat{a}\rangle \in U$. Wie wir zeigen werden, impliziert dies, daß in $ADV(p(n), s)$ keine 7/8 Mehrheit für oder gegen x vorhanden ist. In diesem Fall fallen also durch die Aufnahme des Paares $\langle x, A(x)\rangle$ in die Beispielfolge s mindestens 1/8 aller bis zu diesem Zeitpunkt in $ADV(p(n), s)$ verbliebenen Strings aus dieser Menge heraus, da sie durch das Paar $\langle x, A(x)\rangle$ widerlegt werden; und zwar gilt dies unabhängig davon, ob x zu A gehört oder nicht.

Das weiter unten definierte Orakel U hat weiterhin die Eigenschaft, daß der in der ersten if-Bedingung ausgeführte Test nur dann negativ ausfallen kann, wenn es für jedes $x \in \{0,1\}^n$ mindestens eine 2/3 Mehrheit gibt, die sich entweder für oder gegen x ausspricht. Das heißt, im else-Teil der ersten if-Anweisung (mit 2: markiert) enthält für jedes $x \in \{0,1\}^n$ eine der beiden Mengen $ADV(p(n), s\#\langle x,0\rangle)$ oder $ADV(p(n), s\#\langle x,1\rangle)$ mindestens 2/3 aller Strings in $ADV(p(n), s)$. Man beachte, daß $ADV(p(n), s\#\langle x,0\rangle)$ und $ADV(p(n), s\#\langle x,1\rangle)$ für jedes x eine Partition von $ADV(p(n), s)$ bilden, so daß dies gleichbedeutend damit ist, daß für jedes x eine der beiden Mengen nicht mehr als 1/3 aller Strings in $ADV(p(n), s)$ enthält.

Wie wir zeigen werden, garantiert diese Bedingung die Existenz eines Wortes $y \in \{0,1\}^{q(n)}$, das die in Abb. 4.4 angegebene Bedingung ($\star\star\star$) erfüllt. Wie sich weiterhin herausstellen wird, kann jedes Wort y mit dieser Eigenschaft dazu verwendet werden, für jedes $x \in \{0,1\}^n$ durch eine Frage an das $co\text{-}\mathcal{NP}$-Orakel W festzustellen, ob sich die (überwiegende) Mehrheit aller Strings in $ADV(p(n), s)$ für oder gegen x ausspricht. Die Eigenschaft ($\star\star\star$) impliziert nämlich, daß für alle $x \in \{0,1\}^n$ die beiden folgenden Bedingungen gelten:

- Das Tupel $\langle s\#\langle x,0\rangle, y, a^*\rangle$ gehört genau dann zu W, wenn sich die Mehrheit aller Strings in $ADV(p(n), s)$ für x entscheidet, das heißt also, wenn

$ADV(p(n), s\#\langle x, 1\rangle) \geq (2/3)\, ADV(p(n), s)$ ist,

- Das Tupel $\langle s\#\langle x, 1\rangle, y, a^*\rangle$ ist genau dann in W, wenn sich die Mehrheit aller Strings in $ADV(p(n), s)$ gegen x entscheidet, das heißt, wenn $ADV(p(n), s\#\langle x, 0\rangle) \geq (2/3)\, ADV(p(n), s)$ ist.

Nachdem nun M ein Wort $y \in \{0,1\}^{q(n)}$ mit diesen Eigenschaften bestimmt hat, testet M in der zweiten if-Bedingung (mit 3: markiert) unter Befragung des Orakels W, ob es ein $x \in \{0,1\}^n$ gibt, das von der Mehrheit aller in $ADV(p(n), s)$ enthaltenen Strings falsch entschieden wird. Falls ja, wird das lexikographisch kleinste solche x in s aufgenommen. Andernfalls kann M die Konstruktion von s erfolgreich abbrechen, da dann mindestens $2/3$ aller Advice-Strings in $ADV(p(n), s)$ für jedes $x \in \{0,1\}^n$ die richtige Entscheidung treffen, welche wiederum durch eine (wahlweise positive oder negative) Frage an das Orakel W erfragt werden kann. In diesem Fall verläßt M daher die loop-Schleife und bestimmt den kleinsten korrekten Advice-String w_{min} der Länge $p(n)$ durch eine Präfixsuche.

Nachdem wir nun grob die Vorgehensweise von M beschrieben haben, geben wir eine detailierte Analyse der Korrektheit und der Komplexität von M.

Da die Funktion $f(0^n, s) =_{def} \|ADV(p(n), s)\|$ in $\#\mathcal{P}$ liegt, existiert nach Satz 3.8 eine Approximationsfunktion $g(0^n, s) \in \mathcal{FP}(\Sigma_2^{\mathrm{P}})$, die für alle $n \geq 0$ und jede Beispielfolge s die folgende Approximationsgüte aufweist:

$$\|ADV(p(n), s)\| \;\leq\; g(0^n, s) \;\leq\; (9/8)\, \|ADV(p(n), s)\|. \tag{4.1}$$

Zur Bestimmung einer Eingabe $x \in \{0,1\}^n$, die mindestens von einer $1/8$-Minderheit der in $ADV(p(n), s)$ verbliebenen Strings inkorrekt entschieden wird, betrachten wir desweiteren die Funktion

$$m(s, x) \;=_{def}\; \max_{b \in \{0,1\}} \|ADV(p(n), s\#\langle x, b\rangle)\|,$$

die angibt, wie groß die Mehrheit der Advice-Strings in $ADV(p(n), s)$ für oder gegen x ist. Offensichtlich ist genau dann keine $7/8$-Mehrheit für oder gegen x vorhanden, wenn $m(s, x) < (7/8)\, \|ADV(p(n), s)\|$ ist.

Da m eine max-$\#\mathcal{P}$-Funktion ist, gibt es nach Satz 3.11 eine Sprache $U \in \Sigma_2^{\mathrm{P}}$, so daß für jede Beispielfolge s, jedes $x \in \{0,1\}^n$ und alle $a \geq 0$ die folgenden Implikationen gelten:

$$\langle s, x, a\rangle \in U \;\Rightarrow\; m(s, x) \leq (9/8)\, a,$$

$$\langle s, x, a\rangle \notin U \;\Rightarrow\; m(s, x) \geq a.$$

Setzen wir also für eine gegebene Beispielfolge s den Wert von a auf $\hat{a} = (2/3)g(0^n, s)$, so erhalten wir hieraus wegen $(9/8)\,\hat{a} = (9/8) \cdot (2/3)\,g(0^n, s) < (7/8)\,\|ADV(p(n), s)\|$ und $\hat{a} \geq (2/3)\,\|ADV(p(n), s)\|$ die folgenden Implikationen:

$$\langle s, x, \hat{a}\rangle \in U \quad\Rightarrow\quad m(s, x) < (7/8)\,\|ADV(p(n), s)\|, \tag{4.2}$$

$$\langle s, x, \hat{a}\rangle \notin U \quad\Rightarrow\quad m(s, x) \geq (2/3)\,\|ADV(p(n), s)\|. \tag{4.3}$$

Aufgrund von (4.2) erfüllt also jedes von M innerhalb der ersten if-Anweisung in die Beispielfolge s aufgenommene x die Forderung, daß die Größe von $ADV(p(n), s)$ mindestens um den Faktor 7/8 abnimmt.

Gibt es dagegen kein $x \in \{0,1\}^n$, für das das Tupel $\langle s, x, \hat{a}\rangle$ in U liegt, dann hat s wegen (4.3) die Eigenschaft, daß für jedes $x \in \{0,1\}^n$ die kleinere der beiden Mengen $ADV(p(n), s\#\langle x, 0\rangle)$ und $ADV(p(n), s\#\langle x, 1\rangle)$ nicht mehr als 1/3 aller Strings in $ADV(p(n), s)$ enthält, das heißt, s erfüllt die Bedingung

$(\star)$ für alle $x \in \{0,1\}^n$ ist $min_{b\in\{0,1\}}\|ADV(p(n), s\#\langle x, b\rangle)\| \leq \dfrac{\|ADV(p(n), s)\|}{3}$.

Unser Ziel ist in diesem Fall, für ein gegebenes x die kleinere der beiden Teilmengen $ADV(p(n), s\#\langle x, 0\rangle)$ und $ADV(p(n), s\#\langle x, 1\rangle)$ wahlweise durch eine $\mathcal{NP}$-Frage oder durch eine $co\text{-}\mathcal{NP}$-Frage zu identifizieren (wir werden auf die Bedeutung dieses Punktes am Ende des Beweises zurückkommen, wenn wir die Komplexität von M analysieren). Zunächst einmal stellt uns Satz 3.10 hierzu eine Sprache $W \in co\text{-}\mathcal{NP}$ und ein Polynom q zur Verfügung, so daß für jede Beispielfolge s und alle $n, a \geq 0$ folgendes gilt (den in Satz 3.10 vorkommenden Eingabeparameter l haben wir hierbei auf den Wert $l = 3$ fixiert):

- für alle $y \in \{0,1\}^{q(n)}$, für alle $x \in \{0,1\}^n$ und für alle $b \in \{0,1\}$ gilt:

$$\langle s\#\langle x, b\rangle, y, a\rangle \in W \quad\Rightarrow\quad \|ADV(p(n), s\#\langle x, b\rangle)\| \leq (4/3)\,a, \tag{4.4}$$

- es gibt ein $y \in \{0,1\}^{q(n)}$, so daß für alle $x \in \{0,1\}^n$ und $b \in \{0,1\}$ gilt:

$$\langle s\#\langle x, b\rangle, y, a\rangle \notin W \quad\Rightarrow\quad \|ADV(p(n), s\#\langle x, b\rangle)\| > a. \tag{4.5}$$

Setzen wir für a den Wert $a^* = (1/3)\,g(0^n, s)$ ein, so erhalten wir aus (4.1) und (4.4) die Gültigkeit folgender Aussage:

($\star\star$) für alle $x \in \{0,1\}^n$, $y \in \{0,1\}^{q(n)}$ und $b \in \{0,1\}$ folgt aus der Zugehörigkeit von $\langle s\#\langle x,b\rangle, y, a^*\rangle$ zu W, daß

$$\|ADV(p(n), s\#\langle x,b\rangle)\| \leq (4/3)\, a^* \leq \|ADV(p(n),s)\|/2$$

sein muß,

Da wir im else-Teil der ersten if-Anweisung die Gültigkeit der unter ($\star$) angegebenen Bedingung voraussetzen können, kann für jedes $x \in \{0,1\}^n$ und jedes $y \in \{0,1\}^{q(n)}$ höchstens eines der beiden Tupel $\langle s\#\langle x,0\rangle, y, a^*\rangle$ und $\langle s\#\langle x,1\rangle, y, a^*\rangle$ zu W gehören. Weiterhin folgt mit (4.1) und (4.5), daß es ein $y \in \{0,1\}^{q(n)}$ gibt, das die in Abb. 4.4 mit ($\star\star\star$) bezeichnete Bedingung erfüllt:

($\star\star\star$) für jedes $x \in \{0,1\}^n$ ist eines der beiden Tupel $\langle s\#\langle x,0\rangle, y, a^*\rangle$ und $\langle s\#\langle x,1\rangle, y, a^*\rangle$ in W enthalten.

Es ist also sichergestellt, daß die von M im else-Teil durchgeführte Präfix-Suche nach einem y mit der in ($\star\star\star$) beschriebenen Eigenschaft immer erfolgreich ist. Außerdem erfüllt jedes solche y aufgrund von ($\star\star$) die Bedingung, daß es für jedes $x \in \{0,1\}^n$ *genau* ein $b \in \{0,1\}$ gibt, für das das Tupel $\langle s\#\langle x,b\rangle, y, a^*\rangle$ in W enthalten ist. Präzise formuliert: falls y die in ($\star\star\star$) beschriebene Eigenschaft hat, dann gelten für alle $x \in \{0,1\}^n$ und $b \in \{0,1\}$ die folgenden Äquivalenzen:

$$\begin{aligned}
\langle s\#\langle x,b\rangle, y, a^*\rangle \in W \quad &\Leftrightarrow \quad \|ADV(p(n), s\#\langle x,b\rangle)\| \leq (1/3)\, \|ADV(p(n),s)\| \\
&\Leftrightarrow \quad \|ADV(p(n), s\#\langle x,\overline{b}\rangle)\| \geq (2/3)\, \|ADV(p(n),s)\| \\
&\Leftrightarrow \quad \langle s\#\langle x,\overline{b}\rangle, y, a^*\rangle \notin W.
\end{aligned}$$

Falls demnach die Bedingung der zweiten if-Anweisung erfüllt ist, also ein $x \in \{0,1\}^n$ existiert, so daß das Tupel $\langle s\#\langle x,\overline{A}(x)\rangle, y, a^*\rangle$ nicht in W enthalten ist, dann muß $\langle s\#\langle x,A(x)\rangle, y, a^*\rangle$ in W enthalten, also $\|ADV(p(n), s\#\langle x,A(x)\rangle)\| \leq (1/3)\, \|ADV(p(n),s)\|$ sein. Folglich wird die Anzahl der mit s konsistenten Strings durch die Aufnahme von x in s mindestens auf $1/3$ (und damit erst recht auf mindestens $7/8$) des vorigen Wertes reduziert.

Ist dagegen für jedes $x \in \{0,1\}^n$ das Tupel $\langle s\#\langle x,\overline{A}(x)\rangle, y, a^*\rangle$ in W, dann ist in y (zusammen mit s) die gesamte Information darüber enthalten, welche Eingaben der Länge n zu A gehören, und welche nicht: Für alle $x \in \{0,1\}^n$ gelten dann nämlich die beiden Äquivalenzen

$$\langle s\#\langle x,0\rangle, y, a^*\rangle \in W \Leftrightarrow x \in A \quad \text{und} \quad \langle s\#\langle x,1\rangle, y, a^*\rangle \in W \Leftrightarrow x \notin A.$$

Das heißt, M kann den kleinsten korrekten Advice-String w_{min} der Länge $p(n)$ durch eine Präfixsuche unter Befragung des folgenden Orakels V bestimmen.

$\langle s, 0^n, y, a, u\rangle \in V \quad \Leftrightarrow_{def}$

es gibt ein v mit $|uv| = p(n)$, so daß für alle $x \in \{0,1\}^n$

– $\langle x, uv\rangle \in B$ und $\langle s\#\langle x, 0\rangle, y, a\rangle \in W$ oder

– $\langle x, uv\rangle \notin B$ und $\langle s\#\langle x, 1\rangle, y, a\rangle \in W$ ist.

Es bleibt nur noch zu zeigen, daß der in Abb. 4.4 beschriebene Algorithmus in $\mathcal{FP}(\mathcal{NP}_1(A) \oplus \Sigma_2^{\mathrm{P}})$ implementiert werden kann. Wie zu Beginn des Beweises ausgeführt, durchläuft M die loop-Schleife nur eine polynomiale Anzahl von Runden. Man überzeugt sich auch leicht davon, daß ein Orakel in $\mathcal{NP}_1(A) \oplus \Sigma_2^{\mathrm{P}}$ ausreicht, alle in Abb. 4.4 vorkommenden Bedingungen auszuwerten und die entsprechenden Anweisungen auszuführen:

- Die Berechnung von g kann unter Befragung eines geeigneten Σ_2^{P} Orakels ausgeführt werden;
- die Auswertung der Bedingung in der ersten if-Anweisung sowie die sich evtl. anschließende Präfixsuche nach x können wegen $U \in \Sigma_2^{\mathrm{P}}$ ebenfalls unter Befragung eines geeigneten Σ_2^{P}-Orakels ausgeführt werden;
- ebenso die Präfixsuche nach y im else-Teil, da W in *co-*$\mathcal{NP}$ ist;
- für die Auswertung der Bedingung in der zweiten if-Anweisung sowie die sich eventuell anschließende Präfixsuche nach x genügt ein $\mathcal{NP}_1(A)$ Orakel (man beachte, daß hierbei die Frage ob $ADV(p(n), s\#\langle x, A(x)\rangle)$ weniger Elemente enthält als $ADV(p(n), s\#\langle x, \overline{A}(x)\rangle)$, als negative Frage in der Form $\langle s\#\langle x, \overline{A}(x)\rangle, y, a^*\rangle \notin W$ an das *co-*$\mathcal{NP}$ Orakel W formuliert werden kann);
- schließlich kann auch die Suche nach w_{min} durch Fragen an V bestimmt werden, das offensichtlich in Σ_2^{P} liegt (man beachte, daß hierzu die entsprechenden Fragen an W in positiver Form gestellt werden müssen).

Damit ist der Beweis erbracht. ∎

Der Beweis von Satz 4.17 zeigt etwas allgemeiner, daß jede Menge A in $\mathcal{C}/poly$ eine Advice-Funktion hat, die in $\mathcal{FP}(\mathcal{NP}(A \oplus \mathcal{C}) \oplus \Sigma_2^{\mathrm{P}}(\mathcal{C}))$ berechenbar ist, wobei das $\mathcal{NP}(A \oplus \mathcal{C})$ Orakel durch eine $\mathcal{NP}$ Orakelmaschine entscheidbar ist, die auf jedem Berechnungspfad höchstens eine Frage an das Orakel A stellt. Unter Beachtung der Tatsache, daß Fragen an ein $\mathcal{NP} \cap$ *co-*$\mathcal{NP}$ Orakel in nichtdeterministischen Rechnungen simuliert werden können (siehe Satz 2.3) erhalten wir hieraus unmittelbar die folgende Verallgemeinerung von Satz 4.17.

Satz 4.18 *Für jede Sprache A in $(\mathcal{NP} \cap co\text{-}\mathcal{NP})/poly$ kann eine Advice-Funktion in der Klasse $\mathcal{FP}(\mathcal{NP}_1(A) \oplus \Sigma_2^{\mathrm{P}})$ berechnet werden.*

Wir beenden diesen Abschnitt mit der Bemerkung, daß die Komplexität von Advice-Funktionen für eine Menge $A \in \mathcal{P}/poly$ stark mit der Komplexität gekoppelt ist, eine Menge T über einem einelementigen Alphabet (sog. *tally* Beschreibungen von A) zu entscheiden, auf die A Turing-reduzierbar ist. In der Tat ist leicht zu sehen, daß die beiden Probleme, eine Advice-Funktion für A zu berechnen und eine tally Beschreibung für A zu erkennen, truth-table äquivalent sind. Folglich führt Satz 4.17 auf die folgende obere Schranke für die Komplexität von tally Mengen, die eine „kompakte Beschreibung" von Sprachen in $\mathcal{P}/poly$ abgeben.

Korollar 4.19 *Für jede Sprache A in $\mathcal{P}/poly$ gibt es eine tally Menge T in $\mathcal{P}(\mathcal{NP}_1(A) \oplus \Sigma_2^{\mathrm{P}})$ mit $A \in \mathcal{P}(T)$.*

Diese obere Schranke ist unvergleichbar mit Gavaldàs $\mathcal{P}_{tt}(\mathcal{NP}(A \oplus \Sigma_2^{\mathrm{P}}))$ Schranke (siehe Satz 4.15; Gavaldàs Schranke impliziert übrigens, daß es ein geeignetes T' in $\mathcal{NP}(A \oplus \Sigma_2^{\mathrm{P}})$ gibt, da es genügt, die Orakelantworten der $\mathcal{P}_{tt}(\mathcal{NP}(A \oplus \Sigma_2^{\mathrm{P}}))$ Berechnung in eine tally Menge T' zu kodieren). Ein interessantes offenes Problem ist, ob es für jede Sprache A in $\mathcal{P}/poly$ eine tally Beschreibung T in $\mathcal{P}_{tt}(\mathcal{NP}(A \oplus \mathcal{NP}))$ gibt (was die Existenz einer solchen Menge in $\mathcal{NP}(A \oplus \mathcal{NP})$ implizieren würde).

4.4 Einordnung von $\mathcal{P}/poly$ innerhalb der Low-Hierarchie

In diesem Abschnitt beweisen wir das Hauptresultat dieses Kapitels, das eine exakte Einordnung von $\mathcal{P}/poly$ in die $E\mathcal{L}_3^{\Theta}$-Stufe der erweiterten $\mathcal{L}ow$-Hierarchie erlaubt. Wie bereits in Abschnitt 4.2 gezeigt, genügt es hierzu, für jede Sprache $A \in \mathcal{P}/poly$ eine (evtl. mehrwertige) Advice-Funktion h zu finden, deren Graph in $\mathcal{NP}(A \oplus \mathrm{SAT})/\mathcal{FP}_{tt}(\mathcal{NP}(A \oplus \mathrm{SAT}))$ entscheidbar ist. Diese obere Schranke, die mit der in Satz 4.17 bewiesenen unvergleichbar ist, erhalten wir durch Kombination der bisher entwickelten Techniken mit einem sogenannten *Zensus-Argument*. Genauer gesagt berechnet die $\mathcal{FP}_{tt}(\mathcal{NP}(A \oplus \mathrm{SAT}))$ Advice-Funktion lediglich die *Anzahl* der Paare einer geeigneten Beispielfolge s, mit deren Hilfe der Graph von h in $\mathcal{NP}(A \oplus \mathrm{SAT})$ entscheidbar ist.

Satz 4.20 *Für jede Menge A in $\mathcal{P}/poly$ gibt es eine Advice-Funktion h, deren Graph in $\mathcal{NP}(A \oplus \mathrm{SAT})/\mathcal{FP}_{tt}(\mathcal{NP}(A \oplus \mathrm{SAT}))$ entschieden werden kann.*

Beweis. Sei $A \in \mathcal{P}/poly$, und seien $B \in \mathcal{P}$ eine Interpreter-Menge für A, $\hat{h} \in poly$ eine Advice-Funktion für A, sowie $p(n)$ ein Polynom, das die Länge des von $\hat{h}$ berechneten Advice-Strings $\hat{h}(n)$ angibt. Wir werden zeigen, daß der Graph der folgenden Funktion h in $\mathcal{NP}(A \oplus \mathrm{SAT})/\mathcal{FP}_{tt}(\mathcal{NP}(A \oplus \mathrm{SAT}))$ liegt: h bildet jedes n auf die Menge

$$\textit{set-h}(n) \ =_{def} \ \{w \in \{0,1\}^{p(n)} \mid \text{für alle } x \in \{0,1\}^n \text{ gilt } x \in A \Leftrightarrow \langle x, w\rangle \in B\}$$

aller Advice-Strings (oder Programme) w ab, die unter der Interpretermenge B alle Eingaben der Länge n korrekt entscheiden.

Wir beschreiben zunächst die zugrundeliegende Beweisidee. Die im Beweis von Satz 4.17 konstruierte Beispielfolge s hatte die folgende Eigenschaft:

($\star$) Jede einzelne Eingabe $x \in \{0,1\}^n$ wird von einer großen Mehrheit aller mit s konsistenten Advice-Strings korrekt entschieden.

Die Begründung für die Existenz einer Beispielfolge polynomialer Größe, die diese Bedingung erfüllt, lieferte hierbei das folgende Argument: Falls s diese Eigenschaft noch nicht besitzt, muß es eine Eingabe $x \in \{0,1\}^n$ geben, die von einem (nicht zu kleinen) Prozentsatz α aller mit s konsistenten Advice-Strings inkorrekt entschieden wird. Folglich kann durch die Aufnahme von x in s die Anzahl aller konsistenten Advice-Strings um den Faktor $1 - \alpha$ verkleinert werden. Da die Anzahl aller betrachteten Advice-Strings exponentiell ist, läßt sich daraus leicht eine polynomiale Schranke für die Anzahl der Erweiterungen ableiten, die zur Konstruktion einer die Bedingung ($\star$) erfüllenden Beispielfolge s benötigt werden.

Wie wir weiterhin im Beweis von Satz 4.17 gesehen haben, ist bei Kenntnis einer die Bedingung ($\star$) erfüllenden Beispielfolge s, sowie einer geeigneten Abschätzung des Wertes $\|ADV(p(n), s)\|$ die Korrektheit jedes beliebigen Advice-Strings durch ein Σ_2^{P}-Prädikat entscheidbar. Allerdings können wir uns diese Tatsache nicht unmittelbar für den Entwurf eines $\Sigma_2^{\mathrm{P}}/\mathcal{FP}_{tt}(\mathcal{NP}(A \oplus \mathrm{SAT}))$-Algorithmus' für *graph-h* zunutze machen, da es nicht ersichtlich ist, wie eine solche Folge s in $\mathcal{FP}_{tt}(\mathcal{NP}(A \oplus \mathrm{SAT}))$ konstruiert werden könnte. Es stellt sich jedoch heraus, daß die Kenntnis der *Anzahl* der in einer geeigneten Beispielfolge s enthaltenen Beispielpaare bereits ausreicht, um die Korrektheit eines beliebigen Advice-Strings durch einen $\mathcal{NP}(A \oplus \mathrm{SAT})$-Algorithmus zu entscheiden; und diese Information *kann* durch eine Funktion $g(n)$ in $\mathcal{FP}_{tt}(\mathcal{NP}(A \oplus \mathrm{SAT}))$ bereitgestellt werden. Wie wir sehen werden, liegt $g(n)$ sogar in $\mathcal{F}\Theta_2^{\mathrm{P}}(A \oplus \mathrm{SAT})$.

Zur Bestimmung der Anzahl von Beispielpaaren, mit der eine die Bedingung $(\star)$ erfüllende Beispielfolge s realisiert werden kann, betrachten wir zunächst die folgende Funktion

$$c^*(n,k) \quad =_{def} \quad min_{s \in S(n,k)} \|ADV(p(n),s)\|.$$

Hierbei ist $S(n,k)$ die Menge aller Beispielfolgen $\langle x_1, A(x_1)\rangle \# \cdots \# \langle x_k, A(x_k)\rangle$, wobei sämtliche Eingaben x_i, $1 \leq i \leq k$, die Länge n haben. Die Funktion $c^*(n,k)$ gibt also den kleinstmöglichen Wert an, auf den die Anzahl aller mit s konsistenten Advice-Strings durch eine Beispielfolge s abgesenkt werden kann, die aus maximal k Paaren besteht, die alle mit der Sprache A konsistent sind.

Zwischen der Funktion c^* und der für uns interessanten Eigenschaft $(\star)$ besteht die folgende Beziehung: Ist der Wert von $c^*(n,k+1)$ im Verhältnis zu $c^*(n,k)$ nur unwesentlich kleiner, dann erfüllt jede Beispielfolge $s \in S(n,k)$ mit $\|ADV(p(n),s)\| = c^*(n,k)$ die Bedingung $(\star)$. Dies folgt unmittelbar aus der Tatsache, daß jede Erweiterung $s' = s\#\langle x, A(x)\rangle$ von s aufgrund der Definition von c^* die Ungleichung $\|ADV(p(n),s')\| \geq c^*(n,k+1)$ erfüllen muß. Folglich ist für jedes $x \in \{0,1\}^n$ der Anteil aller Advice-Strings in $ADV(p(n),s)$, die x korrekt entscheiden, mindestens

$$\frac{\|ADV(p(n), s\#\langle x, A(x)\rangle)\|}{\|ADV(p(n),s)\|} \quad \geq \quad \frac{c^*(n,k+1)}{c^*(n,k)}.$$

Das heißt, wir sind an solchen Werten für k interessiert, für die der Quotient $c^*(n,k+1)/c^*(n,k)$ nahe genug an 1 liegt.

Wie man leicht sieht, ist für die folgende Funktion

$$d(n,k) \quad =_{def} \quad 2^{p(n)} \cdot (8/9)^k$$

dieser Quotient konstant gleich 8/9. Vergleichen wir den Werteverlauf der Funktion $c^*(n,k)$ mit dem der Funktion $d(n,k)$, so stellen wir fest, daß beide Funktionen für $k=0$ denselben Wert $d(n,0) = c^*(n,0) = 2^{p(n)}$ annehmen. Da $\hat{h}(n)$ für alle $k \geq 0$ mit jeder Beispielfolge s in $S(n,k)$ konsistent ist, kann der Wert von $c^*(n,k)$ nicht unter 1 fallen. Im Gegensatz hierzu fällt der Wert von $d(n,k)$ für $k \geq 8p(n)$ sicherlich unter 1, so daß es mindestens ein $k \in \{0,1,\ldots,8p(n)\}$ geben muß, für das zwar $c^*(n,k) \leq d(n,k)$, aber nicht $c^*(n,k+1) \leq d(n,k+1)$ ist. Jedes solche k führt jedoch auf einen Quotienten von $c^*(n,k+1)/c^*(n,k) \geq d(n,k+1)/d(n,k) = 8/9$.

Wir beschreiben nun im einzelnen, wie für ein gegebenes n durch eine Funktion g in $\mathcal{F}\Theta_2^{\mathrm{P}}(A \oplus \mathrm{SAT})$ ein geeigneter Wert k berechnet werden kann, für den der Quotient $c^*(n,k+1)/c^*(n,k)$ größer oder gleich 2/3 ist.

Da die Funktion $f(0^n, s) =_{def} \|ADV(p(n), s)\|$ in $\#\mathcal{P}$ liegt, existiert nach Satz 3.11 eine Sprache $U \in \Sigma_2^{\mathrm{P}}$, so daß für alle $n, a \geq 0$ und alle Beispielfolgen s die folgenden Implikationen gelten (den in Satz 3.11 vorkommenden Eingabeparameter l haben wir hierbei auf den Wert $l = 8$ fixiert):

$$\langle 0^n, s, a\rangle \in U \;\Rightarrow\; \|ADV(p(n), s)\| \leq (9/8)\ a, \tag{4.6}$$

$$\langle 0^n, s, a\rangle \notin U \;\Rightarrow\; \|ADV(p(n), s)\| > a. \tag{4.7}$$

Betrachte nun die folgende Sprache V, definiert als

$$\langle 0^n, k\rangle \in V \quad \Leftrightarrow_{def} \quad \text{es gibt eine Beispielfolge } s \text{ in } S(n,k), \text{ für die das Tupel } \langle 0^n, s, d(n,k)\rangle \text{ in } U \text{ liegt.}$$

Es ist leicht zu sehen, daß V in $\mathcal{NP}(A \oplus \mathrm{SAT})$ entscheidbar ist. Außerdem ist aufgrund von $\|ADV(p(n), \lambda)\| = 2^{p(n)}$ das Tupel $\langle 0^n, \lambda, d(n, 0)\rangle$ in U enthalten. Dies wiederum impliziert, daß $\langle 0^n, 0\rangle$ sicherlich zu V gehört. Folglich ist die folgende Funktion

$$g(n) \quad =_{def} \quad max\{k \mid 0 \leq k \leq 8p(n) \text{ und } \langle 0^n, k\rangle \in V\}$$

wohldefiniert und ist durch binäre Suche unter Befragung des Orakels V in $\mathcal{F}\Theta_2^{\mathrm{P}}(A \oplus \mathrm{SAT})$ berechenbar.

Als nächstes zeigen wir, daß der Graph von h unter Kenntnis von $g(n)$ in $\mathcal{NP}(A \oplus \mathrm{SAT})$ entscheidbar ist. Zunächst bemerken wir, daß das Tupel $\langle 0^n, 8p(n) + 1\rangle$ sicher nicht in V enthalten ist, da sonst ein s in $S(n, 8p(n) + 1)$ existieren müßte mit $\|ADV(p(n), s)\| \leq (9/8)\, d(n, 8p(n) + 1) < 1$, was jedoch ausgeschlossen ist. Dies impliziert, daß das Paar $\langle 0^n, g(n) + 1\rangle$ nicht in V liegt, es also keine Beispielfolge $s \in S(n, g(n) + 1)$ geben kann, für die $\langle 0^n, s, d(n, g(n) + 1)\rangle$ in U ist. Unter Berücksichtigung von (4.7) erhalten wir hieraus für jede Beispielfolge $s \in S(n, g(n))$ und für jedes $x \in \{0,1\}^n$ die folgende untere Schranke für die (absolute) Anzahl der Advice-Strings in $ADV(p(n), s)$, die x korrekt entscheiden.

- Für alle $s \in S(n, g(n))$ und für alle $x \in \{0,1\}^n$ gilt:

$$\begin{aligned} \|ADV(p(n), s\#\langle x, A(x)\rangle)\| \;&>\; d(n, g(n) + 1) \\ &=\; (8/9)\ d(n, g(n)). \end{aligned} \tag{4.8}$$

Weiterhin ergibt sich aus den Eigenschaften von V, U und g für jede Beispielfolge $s \in S(n, g(n))$, für die das Tupel $\langle 0^n, s, g(n)\rangle$ in U enthalten ist, die folgende obere Schranke für die Anzahl aller mit s konsistenten Advice-Strings.

input $\langle n, w, k\rangle$
($\star$ k ist der von $g \in \mathcal{F}\Theta_2^{\mathrm{P}}(A \oplus \mathrm{SAT})$ zur Verfügung gestellte Advice $\star$)
guess $x_1, \ldots, x_k \in \{0,1\}^n$
$s := \langle x_1, A(x_1)\rangle \# \cdots \# \langle x_k, A(x_k)\rangle$
guess $y \in \{0,1\}^{q(n)}$
if für alle $x \in \{0,1\}^n$ gilt

- $\langle x, w\rangle \in B$ und $\langle s\#\langle x, 0\rangle, y, k\rangle \in W$ oder
- $\langle x, w\rangle \notin B$ und $\langle s\#\langle x, 1\rangle, y, k\rangle \in W$

then accept else reject end

Abbildung 4.5: Überprüfen der Korrektheit eines Advice-Strings in $\mathcal{NP}(\mathcal{NP} \oplus A)$.

- Für alle $s \in S(n, g(n))$ gilt:

$$\langle 0^n, s, g(n)\rangle \in U \quad \Rightarrow \quad \|ADV(p(n), s)\| \;\leq\; (9/8)\; d(n, g(n)).$$

Ist $s \in S(n, g(n))$ also eine Beispielfolge, für die das Tupel $\langle 0^n, s, g(n)\rangle$ in U ist, so kann es für jedes $x \in \{0,1\}^n$ nur eine kleine Minderheit von maximal

$$\begin{aligned} \|ADV(p(n), s\#\langle x, \overline{A}(x)\rangle)\| &= \|ADV(p(n), s) - ADV(p(n), s\#\langle x, A(x)\rangle)\| \\ &\leq (9/8 - 8/9)\, d(n, g(n)) \\ &\leq d(n, g(n))/4 \end{aligned}$$

vielen Advice-Strings geben, die x inkorrekt entscheiden. Aufgrund der Definition von g gibt es nun aber tatsächlich Beispielfolgen $s \in S(n, g(n))$, für die das Tupel $\langle 0^n, s, g(n)\rangle$ in U ist.

Das heißt, wir haben die folgende Behauptung bewiesen.

- Es gibt ein $s \in S(n, g(n))$, so daß für alle $x \in \{0,1\}^n$ gilt:

$$\|ADV(p(n), s\#\langle x, \overline{A}(x)\rangle)\| \;\leq\; d(n, g(n))/4. \tag{4.9}$$

Damit sind wir nun in der Lage, einen $\mathcal{NP}(A \oplus \mathrm{SAT})$-Algorithmus zur Erkennung von *graph-h* anzugeben. Die in Abb. 4.5 dargestellte Orakelmaschine M erhält als Eingabe ein Tripel $\langle n, w, k\rangle$, wobei w ein potentieller Advice-String für die Eingabelänge n ist, und k eine natürliche Zahl, von der angenommen werden kann, daß sie den Wert von g an der Stelle n widergibt. Um die Korrektheit von w zu prüfen, rät M eine Beispielfolge $s \in S(n, k)$ sowie einen String $y \in \{0,1\}^{q(n)}$ und führt für jede Eingabe $x \in \{0,1\}^n$ den folgenden Test durch:

- falls w die Eingabe x akzeptiert, testet M, ob $\langle s\#\langle x,0\rangle,y,k\rangle$ in W ist,
- falls w die Eingabe x verwirft, testet M, ob $\langle s\#\langle x,1\rangle,y,k\rangle$ in W ist.

Fällt auch nur einer dieser Tests negativ aus, so verwirft M das Tripel $\langle n,w,k\rangle$, andernfalls wird es akzeptiert. Wie wir gleich sehen werden, ist W eine *co-*$\mathcal{NP}$-Sprache, weshalb die Frage ob w alle Tests besteht, durch ein geeignetes *co-*$\mathcal{NP}$-Orakel beantwortet werden kann. Folglich kann M in $\mathcal{NP}(\mathcal{NP}\oplus A)$ implementiert werden.

Ähnlich wie im Beweis von Satz 4.17 ist hierbei W eine von Satz 3.10 zur Verfügung gestellte Sprache in *co-*$\mathcal{NP}$ mit den folgenden Eigenschaften (für ein geeignetes Polynom q):

- Für alle Beispielfolgen s, $x\in\{0,1\}^n$, $b\in\{0,1\}$ und für alle $y\in\{0,1\}^{q(n)}$ gilt:
$$\langle s\#\langle x,b\rangle,y,k\rangle\in W \quad\Rightarrow\quad \|ADV(p(n),s\#\langle x,b\rangle,y,k)\|\leq(4/3)\,d(n,k)/4.$$
- Für jede Beispielfolge s gibt es ein $y\in\{0,1\}^{q(n)}$, so daß für alle $x\in\{0,1\}^n$ und $b\in\{0,1\}$ gilt:
$$\langle s\#\langle x,b\rangle,y,k\rangle\notin W \quad\Rightarrow\quad \|ADV(p(n),s\#\langle x,b\rangle)\|>d(n,k)/4.$$

Aufgrund von (4.9) ist sichergestellt, daß es ein $s\in S(n,g(n))$ gibt, so daß für alle $x\in\{0,1\}^n$ eines der beiden Tupel $\langle s\#\langle x,0\rangle,y,g(n)\rangle$ und $\langle s\#\langle x,1\rangle,y,g(n)\rangle$ in W ist. Auf der anderen Seite garantiert uns (4.8) in diesem Fall, daß für jedes solche s die beiden Äquivalenzen
$$\langle s\#\langle x,0\rangle,y,g(n)\rangle\in W\Leftrightarrow x\in A \quad\text{und}\quad \langle s\#\langle x,1\rangle,y,g(n)\rangle\in W\Leftrightarrow x\notin A.$$
gelten. Folglich werden unter der Bedingung $k=g(n)$ genau die korrekten Advice-Strings akzeptiert, womit der Satz bewiesen ist. ■

Der Beweis von Satz 4.20 zeigt sogar allgemeiner, daß jede Menge A in $\mathcal{C}/poly$ eine Advice-Funktion hat, deren Graph in $\mathcal{NP}(\mathcal{NP}(\mathcal{C})\oplus A)/\mathrm{F}\Theta_2^{\mathrm{P}}(\mathcal{NP}(\mathcal{C})\oplus A)$ entscheidbar ist. Daraus ergibt sich zusammen mit Lemma 4.8 das Hauptresultat dieses Kapitels.

Korollar 4.21

(i) $(\mathcal{NP}\cap co\text{-}\mathcal{NP})/poly\subseteq E\mathcal{L}_3^{\Theta}$,

(ii) $(\mathcal{NP} \cap co\text{-}\mathcal{NP})/poly \cap \mathcal{NP} \subseteq \mathcal{L}ow_3^{\Theta}$.

Da alle $\leq_T^{SN}$-vollständigen Sprachen für $\mathcal{NP}$ in $\mathcal{H}igh_1^{\Sigma} \subseteq \mathcal{H}igh_3^{\Theta}$ enthalten sind, erhalten wir unter Beachtung von Proposition 4.3 *(vii)* das folgende Korollar.

Korollar 4.22 *Keine $\leq_T^{SN}$-vollständige Sprache für $\mathcal{NP}$ kann Schaltkreise polynomialer Größe haben, außer wenn die Polynomialzeit-Hierarchie auf die Stufe Θ_3^{P} kollabiert.*

Obiges Korollar ist sowohl eine Verbesserung des Resultats von Ko und Schöning [KS85], daß es keine $\leq_T^{SN}$-vollständige Sprachen für $\mathcal{NP}$ in $\mathcal{P}/poly$ geben kann, außer wenn die Polynomialzeit-Hierarchie auf die Stufe Σ_3^{P} kollabiert, als auch des Ergebnisses von Kadin [Kad88], daß keine $\leq_T^{SN}$-vollständigen Sprachen für $\mathcal{NP}$ existieren, deren Komplement spärlich ist, außer wenn die Polynomialzeit-Hierarchie auf die Stufe Θ_3^{P} kollabiert.

Wir werden im nächsten Kapitel unter der stärkeren Voraussetzung, daß alle $\leq_T^{SN}$-vollständigen Sprachen für $\mathcal{NP}$ in $\mathcal{P}/poly$ enthalten sind, einen Kollaps der Polynomialzeit-Hierarchie auf $\mathcal{ZPP}(\mathcal{NP})$ herbeiführen. Dieser (möglicherweise) tiefere Kollaps beruht darauf, daß für die Teilklasse aller selbstreduzierbaren Sprachen in $\mathcal{P}/poly$ in der Tat stärkere Lowness-Resultate erzielbar sind: All diese Sprachen sind low für die Klasse $\mathcal{ZPP}(\mathcal{NP})$.[1] Sind folglich alle $\leq_T^{SN}$-vollständigen Sprachen für $\mathcal{NP}$ in $\mathcal{P}/poly$ enthalten, so befinden sich darunter sicherlich auch selbstreduzierbare $\leq_T^{SN}$-vollständige Sprachen (etwa SAT), was den Kollaps der Polynomialzeit-Hierarchie auf $\mathcal{ZPP}(\mathcal{NP})$ impliziert.

4.5 Untere Lowness-Schranken

Im letzten Abschnitt konnten wir die Klasse $\mathcal{P}/poly$ in der dritten Theta-Stufe $E\mathcal{L}_3^{\Theta}$ der erweiterten $\mathcal{L}ow$-Hierarchie lokalisieren. Wir befassen uns in diesem Abschnitt mit der Frage, ob diese Einordnung optimal ist. Daß nicht alle Sprachen in $\mathcal{P}/poly$ bereits in der nächstkleineren Stufe $E\mathcal{L}_2^{\Sigma}$ enthalten sind, haben Allender und Hemachandra durch Konstruktion einer spärlichen Menge S bewiesen, die nicht in $E\mathcal{L}_2^{\Sigma}$ enthalten ist, d.h. es gilt $\Sigma_2^{\mathrm{P}}(S) \not\subseteq \mathcal{NP}(\mathcal{NP} \oplus S)$.

Satz 4.23 [AH92] *Es gibt spärliche Mengen außerhalb von $E\mathcal{L}_2^{\Sigma}$.*

[1] Wie wir im nächsten Abschnitt zeigen werden, ist dagegen für die Klasse aller Sprachen in $\mathcal{P}/poly$ keine Verbesserung mehr möglich.

Diese untere Schranke wurde von Long und Sheu unter Verwendung von Techniken, die auf Kadin [Kad89] zurückgehen, noch weiter verbessert.

Für eine Sprachklasse $\mathcal{C}$ bezeichne $\mathcal{P}^{\mathcal{C}[f(n)]}$ die Klasse aller Sprachen, die in Polynomialzeit von einer Orakelmaschine erkannt werden können, die bei Eingaben der Länge n nicht mehr als $f(n)$ (adaptive) Fragen an ein Orakel $C \in \mathcal{C}$ stellt. Allgemeiner sei für eine Klasse $\mathcal{F}$ von Funktionen $\mathcal{P}^{\mathcal{C}[\mathcal{F}]}$ die Vereinigung der Klassen $\mathcal{P}^{\mathcal{C}[f(n)]}$ über alle Funktionen $f \in \mathcal{F}$.

Satz 4.24 [LS91]

(i) Sei $f(n) : \mathcal{N} \to \mathcal{N}$ eine Funktion, die für eine geeignete Konstante $c > 0$ die Ungleichung $f(n) < c \cdot \log n$ für unendlich viele n erfüllt. Dann gibt es eine spärliche Menge S mit der Eigenschaft

$$\Theta_2^{\mathrm{P}}(S) \quad \not\subseteq \quad \mathcal{P}^{\mathcal{NP}(\mathrm{SAT}\oplus S)[f(n)]}.$$

(ii) Sei $f(n) : \mathcal{N} \to \mathcal{N}$ eine Funktion, die für jede Konstante $c > 0$ die Ungleichung $f(n) < c \cdot \log n$ für unendlich viele n erfüllt. Dann gibt es eine spärliche Menge S mit der Eigenschaft

$$\Theta_2^{\mathrm{P}}(S) \quad \not\subseteq \quad \mathcal{P}^{\mathcal{NP}(\mathrm{SAT}\oplus S)[O(f(n))]}.$$

Aus diesen Resultaten lassen sich verbesserte untere Lowness-Schranken ableiten. Hierzu müssen wir allerdings die erweiterte $\mathcal{L}ow$-Hierarchie weiter verfeinern. In Anlehnung an [LS91] sei für eine Klasse $\mathcal{F}$ von Funktionen von $\mathcal{N}$ nach $\mathcal{N}$ und jedes $k \geq 2$,

$$E\mathcal{L}_k^{\mathcal{F}} \quad = \quad \left\{ A \;\middle|\; \Sigma_{k-1}^{\mathrm{P}}(A) \subseteq \mathcal{P}^{\Sigma_{k-2}^{\mathrm{P}}(\mathrm{SAT}\oplus A)[\mathcal{F}]} \right\}.$$

Da die Θ-Stufen der Polynomialzeit-Hierarchie unter dem Θ-Operator abgeschlossen sind, folgt unmittelbar, daß $E\mathcal{L}_k^{\Theta} = E\mathcal{L}_k^{O(log(n))}$ für jedes $k \geq 2$ ist.

Satz 4.24 impliziert nun die folgenden Lowness-Schranken für die Klasse der spärlichen Mengen.

Korollar 4.25 [LS91]

(i) Für jede Funktion f, die für eine Konstante $c > 0$ die Ungleichung $f(n) < c \cdot \log n$ für unendlich viele n erfüllt, gibt es eine spärliche Menge $S \notin E\mathcal{L}_3^{\{f\}}$.

(ii) Für jede Funktion f, die für jede Konstante $c > 0$ die Ungleichung $f(n) < c \cdot \log n$ für unendlich viele n erfüllt, gibt es eine spärliche Menge $S \notin E\mathcal{L}_3^{O(f(n))}$.

Das obige Korollar läßt immer noch die Frage offen, ob jede spärliche Menge (oder sogar jede Sprache in $\mathcal{P}/poly$) in der Vereinigung $\bigcup_{f \in O(\log n)} E\mathcal{L}_3^{\{f\}}$ aller $E\mathcal{L}_3^{\{f\}}$ Stufen mit $f \in O(\log n)$ enthalten ist. Es ist jedoch nicht sehr schwer, den Beweis von Satz 4.24 dahingehend zu modifizieren, daß er diese Möglichkeit ebenfalls ausschließt.

Satz 4.26 *Es gibt eine spärliche Menge S, so daß $\Theta_2^{\mathrm{P}}(S)$ für kein einzelnes $f \in O(\log n)$ in $\mathcal{P}^{\mathcal{NP}(\mathrm{SAT} \oplus S)[f(n)]}$ enthalten ist, das heißt, $S \notin \bigcup_{f \in O(\log n)} E\mathcal{L}_3^{\{f\}}$.*

Diese Schranken zeigen, daß die Einordnung von $\mathcal{P}/poly$ in $E\mathcal{L}_3^{\Theta} = E\mathcal{L}_3^{O(log(n))}$ optimal ist, da noch nicht einmal die Klasse aller spärlichen Mengen in der Klasse $\bigcup_{f \in O(\log n)} E\mathcal{L}_3^{\{f\}}$ enthalten ist.

Kapitel 5

Selbstreduzierbarkeit und Lowness

In diesem Kapitel untersuchen wir Lowness-Eigenschaften von selbstreduzierbaren Sprachen in Advice-Klassen. Als ein Hauptresultat erhalten wir, daß alle selbstreduzierbaren Sprachen in $\mathcal{P}/poly$ low sind für $\mathcal{ZPP}(\mathcal{NP})$. Daraus ergibt sich eine Vielfalt von Konsequenzen. Diese betreffen einerseits einen tieferen Kollaps der Polynomialzeit-Hierarchie (bzw. der Counting-Hierarchie) auf $\mathcal{ZPP}(\mathcal{NP})$ unter der Voraussetzung, daß vollständige Sprachen für $\mathcal{NP}$ (bzw. für $\mathcal{C}_=\mathcal{P}$) Schalkreise polynomialer Größe haben. Andererseits ergeben sich verbesserte untere Schranken für die Schaltkreiskomplexität von Sprachen in $\mathcal{ZPP}(\mathcal{NP})$.

Die in diesem Kapitel dargestellten Resultate werden in [KW95] erscheinen.

5.1 Selbstreduktionen

Selbstreduktionen spielen eine sehr wichtige Rolle in der Komplexitätstheorie. Aus intuitiver Sicht ist eine Sprache A selbstreduzierbar, falls es ein rekursives Programm für A gibt, das zumindest dann effizient ist, wenn man die Rechenzeit außer acht läßt, die zur Abarbeitung der rekursiven Aufrufe benötigt wird. Wir beginnen damit, eine möglichst weit gefaßte Definition der Selbstreduzierbarkeit zu geben. Diese ist zwar etwas technisch, andererseits macht sie jedoch die Struktur deutlich, deren Vorhandensein bei Sprachen in Advice-Klassen auf stärkere Lowness-Eigenschaften schließen läßt.

Definition 5.1

a) *Sei $\succ$ eine irreflexive und transitive Ordnungsrelation auf Σ^*. Wir nennen eine Folge $x_0, x_1, \ldots, x_k$ von Wörtern aus Σ^* eine* **$\succ$-Kette** *(der Länge k), falls $x_0 \succ x_1 \succ \cdots \succ x_k$ gilt. Die Relation $\succ$ heißt* **längen-prüfbar**, *falls es ein Polynom q gibt, so daß*

- *für alle $x, y \in \Sigma^*$ gilt: $y \prec x \Rightarrow |y| \leq q(|x|)$, und*
- *die Sprache $\{\langle x, y, k\rangle \mid$ es gibt eine $\succ$-Kette der Länge k von x nach $y\}$ in $\mathcal{NP}$ ist.*

b) *Eine Menge A heißt* **selbstreduzierbar**, *falls es eine polynomial zeitbeschränkte Orakelmaschine M_{self} und eine längen-prüfbare Ordnungsrelation $\succ$ gibt, so daß $A = L(M_{\mathit{self}}, A)$ ist, und M_{self} bei jeder Eingabe x nur Fragen $y \prec x$ an das Orakel stellt.*

Die Orakelmaschine M_{self} wird in diesem Kontext auch *Selbstreduktions-Maschine* genannt. Jede Selbstreduktions-Maschine spannt für eine gegebene Eingabe x einen sogenannten (geordneten) *Selbstreduktions-Baum* auf, dessen Wurzel mit x markiert ist, und in dem ein mit dem Wort y markierter Knoten eine (polynomial beschränkte) Anzahl von Nachfolge-Knoten hat, die mit den Orakelfragen bei Eingabe y (geordnet entsprechend der Reihenfolge, in der sie gestellt werden) markiert sind. An den Blättern dieses Baumes befinden sich demnach solche Eingaben, die M_{self} entscheiden kann, ohne das Orakel zu befragen. Man beachte, daß ein Selbstreduktions-Baum maximal exponentielle Tiefe hat, da jeder Pfad von der Wurzel x bis zu einem Blatt einer $\succ$-Kette entspricht, die nur aus Wörtern y mit $|y| \leq q(|x|)$ gebildet wird, und in der aufgrund der Irreflexivität von $\succ$ kein Wort doppelt vorkommen kann. Man überlegt sich auch leicht, daß alle selbstreduzierbaren Sprachen in $\mathcal{EXP}$ enthalten sind (Stichwort: Dynamisches Programmieren bzw. Tiefendurchlauf des Selbstreduktions-Baumes, ohne in Teilbäume mit derselben Wurzel mehrfach abzusteigen).

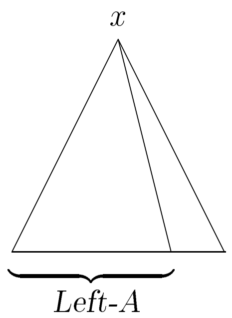
Die Klasse aller gemäß Definition 5.1 selbstreduzierbaren Sprachen umfaßt sowohl die im Sinne von Ko [Ko83] (polynomially related) selbstreduzierbaren Sprachen als auch die von Balcázar [Bal90] eingeführten (length-decreasing und word-decreasing) selbstreduzierbaren Sprachen. Außerdem haben alle bekannten Klassen zwischen $\mathcal{P}$ und $\mathcal{EXP}$, die überhaupt many-one vollständige Sprachen haben (wie zum Beispiel $\mathcal{NP}$, Σ_k^{P}, Π_k^{P}, $k \geq 1$, $\mathcal{PP}$, $\mathcal{C}_{=}\mathcal{P}$, $\mathcal{M}od_m\mathcal{P}$, $m \geq 2$, $\mathcal{PSPACE}$ und $\mathcal{EXP}$) auch selbstreduzierbare many-one vollständige Sprachen [BDG95, BDG90, Bal90, OL93].

Ein wichtiges Beispiel für selbstreduzierbare Sprachen bilden die sogenannten **Left-Mengen** von $\mathcal{NP}$ Sprachen. Diese spielen in der von Ogiwara und Watanabe [OW91] erzielten Erweiterung von Mahaneys Theorem (siehe Einleitung) von many-one Reduktionen auf bounded-truth-table Reduktionen[1] eine entscheidende Rolle. Das Resultat von Ogiwara und Watanabe liefert nebenbei bemerkt ein weiteres interessantes Beispiel dafür, welche Auswirkungen das Vorhandensein von strukturellen Eigenschaften für eine Sprache haben kann: Während es spärliche Sprachen gibt, die nicht in $E\mathcal{L}_2^\Sigma$ enthalten sind (siehe Abschnitt 4.5), konnten Ogiwara und Watanabe nachweisen, daß alle auf eine spärliche Menge bounded-truth-table reduzierbaren Sprachen, sofern sie die Struktur einer Left-Menge besitzen, in $\mathcal{P} = \mathcal{L}ow_0^\Sigma$ enthalten sind.

Beispiel 4 *Ist A eine $\mathcal{NP}$ Sprache, die von einer $\mathcal{NP}$-Maschine M mit Zeitschranke p erkannt wird, so ist die Left-Menge[2] von A definiert als*

$$\begin{array}{lll} \textit{Left-}A & =_{\text{def}} & \{\langle x, y\rangle \mid y \in \{0,1\}^{p(|x|)} \textit{ kodiert einen Berechnungspfad} \\ & & \textit{von } M \textit{ bei Eingabe } x \textit{ und es gibt einen Berechnungs-} \\ & & \textit{pfad } y' \geq y, \textit{ auf dem } M \textit{ die Eingabe } x \textit{ akzeptiert } \}. \end{array}$$

Hierbei wird jedem String y der Länge $p(n)$ diejenige Rechnung von M bei Eingabe x zugewiesen, bei der die nichtdeterministischen Entscheidungen der Reihe nach gemäß der Folge der Bits in y getroffen werden. Anschaulich gesprochen, werden also in einer Left-Menge alle Kodierungen von Berechnungspfaden „aufgefüllt", die links von dem lexikographisch größten akzeptierenden Pfad liegen.



Es ist klar, daß A mittels der Funktion $f(x) =_{\text{def}} \langle x, 0^{p(|x|)}\rangle$ auf ihre Left-Menge many-one reduzierbar ist. Offensichtlich liegt weiterhin die Left-Menge einer $\mathcal{NP}$

[1] Eine bounded-truth-table Reduktion ist eine Turing-Reduktion, bei der nur eine konstante Anzahl von Orakelfragen gestellt werden.

[2] Man beachte, daß es zu einer Sprache A verschiedene Left-Mengen geben kann, je nachdem welche $\mathcal{NP}$-Maschine M man ihrer Definition zugrundelegt.

Sprache ebenfalls in $\mathcal{NP}$. Interessant ist außerdem, daß für eine $\mathcal{UP}$ (bzw. $\mathcal{F}\text{ew}\mathcal{P}$) Sprache A auch Left-A in $\mathcal{UP}$ (bzw. $\mathcal{F}\text{ew}\mathcal{P}$) enthalten ist. Ferner ist leicht zu sehen, daß Left-A mittels folgender Maschine M_{self} selbstreduzierbar ist:

> *Bei Eingabe $\langle x, y\rangle$ teste zunächst, ob $|y| = p(|x|)$ ist und verwerfe, falls dies nicht der Fall ist. Falls y selbst einen akzeptierenden Berechnungspfad von M bei Eingabe x kodiert, dann akzeptiere. Ist $y \neq 1^{p(|x|)}$, dann stelle die Orakelfrage $\langle x, y^+\rangle$, wobei y^+ den lexikographischen Nachfolger von y bezeichnet. Falls die Antwort positiv ist, akzeptiere. Ist sie hingegen negativ oder ist $y = 1^{p(|x|)}$, dann verwerfe.*

Der von M_{self} bei einer Eingabe $\langle x, y\rangle$ aufgespannte Selbstreduktionsbaum ist hier also zu einem (möglicherweise exponentiell langen) Pfad entartet.

Die wichtigste Eigenschaft einer selbstreduzierbaren Menge A ist, daß jedes Programm M relativ effizient daraufhin getestet werden kann, ob es alle Eingaben für A aus einem vorgegebenen Bereich korrekt entscheidet. Die entsprechende Testprozedur kann hierbei so implementiert werden, daß sie gegebenenfalls ein Gegenbeispiel x zurückliefert, das von M falsch entschieden wird. Die Hauptidee für die Durchführung eines solchen Tests ist, M auf Konsistenz mit der Selbstreduktionsmaschine M_{self} für A zu überprüfen. Konkret heißt dies, daß M_{self} und M auf alle Eingaben x (in einem bestimmten Bereich) angesetzt werden, wobei zur Beantwortung aller von M_{self} gestellten Orakelfragen das Programm M verwendet wird. Falls hierbei Unstimmigkeiten auftreten, so kann die von M erkannte Sprache offensichtlich nicht A sein. Erstaunlicherweise ist es auch umgekehrt für kein fehlerhaftes Programm möglich, diesen Test zu passieren. Sei nämlich x eine bezüglich $\prec$ minimale Eingabe, die von M inkorrekt entschieden wird. Dann liefert die Simulation von M_{self} bei Eingabe x die richtige Antwort, da alle Orakelfragen aufgrund der Minimalität von x durch M richtig beantwortet werden. Dadurch ist garantiert, daß die Fehlerhaftigkeit von M von der Testprozedur zumindest bei Eingabe x erkannt wird.

In folgendem Lemma werden die zu überprüfenden Programme von einer $\mathcal{NP}$ Interpreter-Sprache D ausgewertet. Desweiteren wird vorausgesetzt, daß ein korrektes Programm eine Sprache der Form $A \oplus \overline{A}$ akzeptiert. Das heißt, die Eingaben für D haben die Form $\langle x, b, w\rangle$, $b \in \{0, 1\}$. Das Tupel $\langle x, 1, w\rangle$ ist hierbei in der Interpretersprache D enthalten, wenn das Programm w die Eingabe x akzeptiert, während die Zugehörigkeit von $\langle x, w, 0\rangle$ zu D signalisiert, daß das Programm w die Eingabe x verwirft. Wir schließen nicht von vornherein aus, daß für ein (inkorrektes) Programm beide Tupel in D enthalten sind (in diesem Fall wird x von w akzeptiert und gleichzeitig verworfen), oder keines der beiden Tupel in D enthalten ist (dies ist eine zweite Art von indifferentem Verhalten von w bei Eingabe x, das wir allerdings in Lemma 5.2 explizit ausschließen werden, da es von unserer Testprozedur unter Umständen unentdeckt bleibt).

Sei $\{0,1\}^{\preceq n} =_{def} \{x \mid \exists y \in \Sigma^{\leq n} : x \preceq y\}$ die Menge aller potentiellen Orakelfragen x, die in einem Selbstreduktionsbaum von M_{self} vorkommen können, der von einer Eingabe y der Länge kleiner gleich n aufgespannt wird. Dann leistet die durch Lemma 5.2 zur Verfügung gestellte Testprozedur T bei Eingabe eines Paares $\langle 0^n, w\rangle$ unter der Voraussetzung, daß w jede Eingabe x in der Menge $\{0,1\}^{\preceq n}$ akzeptiert und/oder verwirft, folgendes:

- Falls das Programm w zumindest eine Eingabe x in $\{0,1\}^{\preceq n}$ falsch entscheidet, das heißt, $\langle x, \overline{A}(x), w\rangle$ ist in D, so resultiert der von T ausgeführte Test in der Ausgabe einer solchen Eingabe x zusammen mit der Information, ob x in A enthalten ist, oder nicht.
- Trifft w hingegen für alle Eingaben x in $\{0,1\}^{\preceq n}$ die richtige Entscheidung, so gibt die Testprozedur T das Sonderzeichen $\perp$ aus (was etwa durch Halten in einem speziellen Endzustand signalisiert wird).

Lemma 5.2 *Sei A selbstreduzierbar und sei D eine Sprache in $\mathcal{NP}$. Dann gibt es einen $\mathcal{FP}(\mathcal{NP})$-Transducer T_A, der bei jeder Eingabe $\langle 0^n, w\rangle$ unter der Voraussetzung, daß für alle $x \in \{0,1\}^{\preceq n}$ mindestens eines der beiden Tupel $\langle x, 0, w\rangle$ und $\langle x, 1, w\rangle$ in D ist, eine Ausgabe $T_A(0^n, w) \in \{\langle x, A(x)\rangle \mid x \in \{0,1\}^{\preceq n}\} \cup \{\perp\}$ erzeugt, so daß gilt:*

$$\begin{aligned} T_A(0^n, w) = \langle x, A(x)\rangle \quad &\Rightarrow \quad \langle x, \overline{A}(x), w\rangle \in D, \\ T_A(0^n, w) = \quad \perp \quad\quad &\Rightarrow \quad \textit{für kein } x \in \{0,1\}^{\preceq n} \textit{ ist } \langle x, \overline{A}(x), w\rangle \in D. \end{aligned}$$

Beweis. Die von T_A verfolgte Strategie ist grob gesagt die folgende: Zuerst testet T_A, ob es eine Eingabe $x \in \{0,1\}^{\preceq n}$ gibt, bei deren Eingabe M_{self} ein anderes Ergebnis liefert als das Programm w. Falls nicht, gibt T_A das Zeichen $\perp$ aus. Andernfalls bestimmt T_A die maximale Länge k_{max} einer Kette, die ein Wort $y \in \{0,1\}^{\preceq n}$ mit einer Eingabe x verbindet, die von w falsch entschieden wird. Danach ermittelt T_A durch Präfixsuche die lexikographisch kleinste Eingabe x_{min}, die von w falsch entschieden wird, und in die eine von einem Wort $y \in \{0,1\}^{\preceq n}$ ausgehende Kette der Länge k_{max} mündet. Da w alle Orakelfragen, die von M_{self} bei Eingabe x_{min} gestellt werden, korrekt entscheidet, ist T_A in der Lage, die Zugehörigkeit von x_{min} zu A zu entscheiden, und kann das Paar $\langle x_{min}, A(x_{min})\rangle$ ausgeben.

input $\langle 0^n, w\rangle$
if $\langle 0^n, 1^{q(n)}, w, 0, 0\rangle \in E$ **then**
$k_{max} := max\{k \leq 2^{q(n)+1} \mid \langle 0^n, 1^{q(n)}, w, k, 0\rangle \in E\}$
bestimme durch Präfixsuche die lexikographisch kleinste Eingabe x_{min},
für die das Tupel $\langle 0^n, x_{min}, w, k_{max}, 0\rangle$ in E ist.
if $\langle 0^n, x_{min}, w, k_{max}, 1\rangle \in E$ **then** $d := 1$
else $d := 0$ **end**
output $\langle x_{min}, d\rangle$
else output $\perp$ **end**

Abbildung 5.1: Testen eines Programms w für eine selbstreduzierbare Sprache.

Es folgt eine formale Beschreibung von T_A (siehe Abb. 5.1). T_A verwendet als Orakel die folgende Sprache

$E =_{def}$ $\{\langle 0^n, z, w, k, b\rangle \mid$ es gibt eine $\succ$-Kette der Länge k von einem String $y \in \Sigma^{\preceq n}$ zu einem String $x \leq z$ mit der Eigenschaft, daß M_{self} bei Eingabe x eine Rechnung ausführt, die die folgenden Bedingungen erfüllt:

- für jede Orakelfrage q, die mit „JA“ („NEIN“) beantwortet wird, ist das Tupel $\langle q, 1, w\rangle$ (bzw. $\langle q, 0, w\rangle$) in D enthalten,
- falls M_{self} bei dieser Rechnung akzeptiert (verwirft), dann ist das Tupel $\langle x, 0, w\rangle$ (bzw. $\langle x, 1, w\rangle$) in D enthalten,
- falls $b = 1$ ist, dann akzeptiert M_{self} bei dieser Rechnung $\}$.

T_A gibt genau dann das Sonderzeichen $\perp$ aus, wenn das Tupel $\langle 0^n, 1^{q(n)}, w, 0, 0\rangle$ nicht in E enthalten ist. Wir zeigen durch Kontraposition, daß es in diesem Fall kein x in $\{0,1\}^{\preceq n}$ geben kann, für das das Tupel $\langle x, \overline{A}(x), w\rangle$ in D ist. Sei also $\langle x, \overline{A}(x), w\rangle$ in D für ein x in $\{0,1\}^{\preceq n}$. Ohne Beschränkung der Allgemeinheit können wir ein bezüglich $\prec$ minimales x in $\{0,1\}^{\preceq n}$ mit dieser Eigenschaft wählen. Wir zeigen, daß M_{self} bei Eingabe x eine Rechnung ausführt (nämlich diejenige, bei der alle Orakelfragen gemäß A beantwortet werden), die alle drei in der Definition von E verlangten Bedingungen erfüllt. Dies impliziert bereits die Zugehörigkeit des Tupels $\langle 0^n, 1^{q(n)}, w, 0, 0\rangle$ zu E, da $x \in \{0,1\}^{\preceq n}$ liegt (es also für $y = x$ eine $\succ$-Kette der Länge 0 von einem String $y \in \Sigma^{\preceq n}$ zu x gibt) und auch $x \leq 1^{q(n)}$ ist.

- Zur ersten Bedingung: Aufgrund der Minimalität von x trifft das Programm

w unter der Interpretersprache D für alle $q \prec x$ die richtige Entscheidung, das heißt, es gilt

$$\begin{aligned} \langle q, 1, w\rangle \in D &\Leftrightarrow q \in A \\ &\Leftrightarrow \langle q, 0, w\rangle \notin D. \end{aligned}$$

Da während der von M_{self} bei Eingabe x ausgeführten Rechnung nur Orakelfragen $q \prec x$ gestellt werden, liegt also für jedes solche q das Tupel $\langle q, A(q), w\rangle$ in D. Man beachte, daß wir an dieser Stelle (und nur hier) die Voraussetzung benötigen, daß für alle $x \in \{0,1\}^{\preceq n}$ mindestens eines der beiden Tupel $\langle x, 0, w\rangle$ und $\langle x, 1, w\rangle$ in D enthalten ist.

- Die Gültigkeit der zweiten Bedingung ergibt sich unmittelbar aus der Tatsache, daß die von M_{self} bei Eingabe x unter Orakel A durchgeführte Rechnung genau dann akzeptiert, wenn x zu A gehört, und aufgrund der Wahl von x das Tupel $\langle x, \overline{A}(x), w\rangle$ in D enthalten ist.
- Die dritte Bedingung schließlich ist trivialerweise erfüllt, da $b = 0$ ist.

Als nächstes zeigen wir, daß jede von T_A erzeugte Ausgabe der Form $\langle x_{min}, d\rangle$ korrekt ist. Das heißt, für jede solche Ausgabe ist das Tupel $\langle x_{min}, \overline{A}(x_{min}), w\rangle$ in D enthalten und es gilt $d = A(x_{min})$. Wie in Abb. 5.1 dargestellt, wird eine solche Ausgabe nur dann erzeugt, wenn das Tupel $\langle 0^n, 1^{q(n)}, w, 0, 0\rangle$ in E enthalten ist. T_A bestimmt in diesem Fall k_{max} als den größten Wert $k \leq 2^{q(n)+1}$, für den das Tupel $\langle 0^n, 1^{q(n)}, w, k, 0\rangle$ noch in E ist. Ferner ist x_{min} die zugehörige lexikographisch kleinste Eingabe x, für die $\langle 0^n, x, w, k_{max}, 0\rangle$ in E ist. Aus den folgenden Überlegungen wird deutlich, daß es nur eine Rechnung von M_{self} geben kann, die die Zugehörigkeit von $\langle 0^n, x_{min}, w, k_{max}, 0\rangle$ in E bezeugt.

- Zunächst ist klar, daß die Eingabe dieser Rechnung das Wort x_{min} sein muß, da andernfalls eine Eingabe $x < x_{min}$ existieren würde, für die das Tupel $\langle 0^n, x, w, k_{max}, 0\rangle$ auch noch in E ist.
- Zweitens sorgt die Wahl von k_{max} dafür, daß alle bei dieser Rechnung gestellten Orakelfragen q gemäß A beantwortet werden. Andernfalls wäre für das entsprechende q das Tupel $\langle q, \overline{A}(q), w\rangle$ in D enthalten. Das heißt, es würde ein bezüglich $\prec$ minimales $x' \preceq q$ geben, für das das Tupel $\langle x', \overline{A}(x'), w\rangle$ in D enthalten ist. Wie im ersten Teil des Beweises gezeigt, gibt es bei Eingabe von x' eine Rechnung von M_{self}, die alle drei in der Definition von E verlangten Bedingungen erfüllt. Folglich ließe sich also die von einem String y zu x_{min} führende Kette der Länge k_{max} um mindestens ein Element zu einer

Kette verlängern, die von y über x_{min} zu x' verläuft. Dies steht jedoch im Widerspruch zur Maximalität von k_{max}.

Demnach ist also die von M_{self} bei Eingabe x_{min} unter dem Orakel A durchgeführte Rechnung die einzige, die die Zugehörigkeit von $\langle 0^n, x_{min}, w, k_{max}, 0\rangle$ zu E bezeugt. Aufgrund der zweiten Bedingung gilt daher $\langle x_{min}, \overline{A}(x_{min}), w\rangle \in D$. Ferner ist aufgrund der dritten Bedingung das Tupel $\langle 0^n, x_{min}, w, k_{max}, 1\rangle$ genau dann in E enthalten, wenn x_{min} zu A gehört. Der von T_A berechnete Wert d stimmt daher tatsächlich mit $A(x_{min})$ überein.

Diese Überlegungen beweisen die Korrektheit der Testprozedur T_A. Da E offensichtlich in $\mathcal{NP}$ liegt, und T_A in $\mathcal{FP}(E)$ implementiert werden kann, ist damit das Lemma bewiesen. ∎

5.2 Lowness-Eigenschaften von selbstreduzierbaren Mengen

In diesem Abschnitt beweisen wir das Hauptresultat dieses Kapitels. Wir zeigen daß alle selbstreduzierbaren Mengen A, die Schalkreise polynomialer Größe haben, low für $\mathcal{ZPP}(\mathcal{NP})$ sind. Wir zeigen zunächst, daß es hierfür genügt, für die Sprache $A \oplus \overline{A}$ die Existenz einer Advice-Funktion in $\mathcal{FZPP}(NP)$ und einer zugehörigen Interpreter-Sprache in $\mathcal{NP}$ nachzuweisen.

Lemma 5.3 *Sei A eine Menge. Ist $A \oplus \overline{A}$ in $\mathcal{NP}/poly$ mittels einer Advice-Funktion h, die in $\mathcal{FZPP}(NP)$ berechnet werden kann, dann ist A low für $\mathcal{ZPP}(\mathcal{NP})$.*

Beweis. Da $\mathcal{ZPP}(\mathcal{NP})$ unter dem $\mathcal{ZPP}(\cdot)$ Operator abgeschlossen ist [Zac82], genügt es zu zeigen, daß $\mathcal{NP}(A)$ in $\mathcal{ZPP}(\mathcal{NP})$ enthalten ist. Sei also L eine beliebige Sprache in $\mathcal{NP}(A)$. Dann gibt es gemäß Lemma 4.12 ein Polynom q und eine Sprache $L' \in \mathcal{NP}$, die für alle Tupel der Form $\langle x, w_0, \ldots, w_{q(n)}\rangle$, $|x| = n$, die folgende Bedingung erfüllt:

$$\{\langle 0, w_0\rangle, \ldots, \langle q(n), w_{q(n)}\rangle\} \subseteq \textit{graph-h} \quad\Rightarrow\quad (x \in L \Leftrightarrow \langle x, w_0, \ldots, w_{q(n)}\rangle \in L').$$

Sei T ein probabilistischer Transducer, der h berechnet. Wir können annehmen, daß T bei Eingabe 0^n mit einer Wahrscheinlichkeit von mindestens 2^{-n-2} eine Ausgabe w produziert, und daß jedes von $T(0^n)$ ausgegebene w in *set-h*(n) enthalten ist. L kann nun in $\mathcal{ZPP}(\mathcal{NP})$ wie folgt entschieden werden:

> Bei Eingabe x simuliere zuerst den probabilistischen Transducer T bei Eingabe 0^i für $i = 0, 1, \ldots, q(|x|)$. Falls hierbei nicht jedesmal eine Ausgabe w_i erzeugt wird, dann gib „?" aus. Andernfalls stelle an das Orakel L' die Frage $\langle x, w_0, \ldots, w_{q(|x|)} \rangle$ und akzeptiere (verwerfe), wenn diese Frage mit „JA" (bzw. „NEIN") beantwortet wird.

Da die Simulation von T mit einer Wahrscheinlichkeit von mindestens $1 - \sum_{i=0}^{q(|x|)} 2^{-i-2} \geq 1/2$ bei allen Eingaben 0^i, $i = 0, 1, \ldots, q(|x|)$, erfolgreich ist, und da es keine inkorrekten Berechnungen gibt, ist dies tatsächlich ein $\mathcal{ZPP}(\mathcal{NP})$ Algorithmus für L. ■

Nun beweisen wir, daß für selbstreduzierbare Mengen $A \in (\mathcal{NP} \cap co\text{-}\mathcal{NP})/poly$ eine geeignete Advice-Information durch eine $\mathcal{FZPP}(NP)$-Berechnung bereitgestellt werden kann, mit deren Hilfe die Sprache $A \oplus \overline{A}$ durch eine Interpreter-Sprache aus $\mathcal{NP}$ entschieden werden kann.

Lemma 5.4 *Sei A eine selbstreduzierbare Menge in $(\mathcal{NP} \cap co\text{-}\mathcal{NP})/poly$. Dann ist $A \oplus \overline{A}$ in $\mathcal{NP}/poly$ mittels einer (mehrwertigen) Advice-Funktion, die in $\mathcal{FZPP}(NP)$ berechnet werden kann.*

Beweis. Die Beweisstrategie ist ähnlich wie im Beweis von Theorem 4.17. Es gibt allerdings zwei gewichtige Unterschiede. Zum einen wird nun die in A enthaltene Information durch eine probabilistische Berechnung aus A gewonnen, und zum anderen werden die im Beweis von Satz 4.17 hierbei an A gestellten Fragen durch einen Aufruf einer geeigneten Testprozedur (vgl. Lemma 5.2) ersetzt, so daß auf eine Befragung des Orakels A verzichtet werden kann.

Sei also A selbstreduzierbar und in $(\mathcal{NP} \cap co\text{-}\mathcal{NP})/poly$, und sei $\prec$ die der Selbstreduktion zugrundeliegende Ordnungsrelation auf Σ^*. Dann gibt es eine Folge $w_0, w_1, \ldots$ von Advice-Strings und eine Sprache $B \in \mathcal{NP} \cap co\text{-}\mathcal{NP}$, so daß für alle n und alle Eingaben x in $\{0,1\}^{\preceq n}$ die Äquivalenz

$$x \in A \quad \Leftrightarrow \quad \langle x, w_n \rangle \in B$$

gilt. Man beachte, daß w_n nicht nur alle Eingaben der Länge n, sondern alle Eingaben x in $\{0,1\}^{\preceq n} = \{x \mid \exists y \in \Sigma^{\leq n} : x \preceq y\}$ korrekt entscheiden muß. Da die Menge $\{0,1\}^{\preceq n}$ für ein geeignetes Polynom q in $\{0,1\}^{\leq q(n)}$ enthalten ist, kann dies jedoch leicht durch Kombination der ersten $q(n)+1$ Advice-Strings und einer entsprechenden Modifikation der Interpreter-Sprache erreicht werden. Sei weiterhin p ein Polynom mit $|w_n| = p(n)$ für alle $n \geq 0$, und sei $t(n) =_{def} 2(p(n) + q(n) + 1)$.

```
input n
s := λ
loop
  for k := 1 to p(n) do
    choose randomly H_k aus Lin[t(n); p(n); k]
  end
  w := ⟨0^n, s, H_{k_max}⟩, wobei k_max der maximale Index k, 0 ≤ k ≤ p(n),
  ist, so daß für jedes i = 1, ..., k die Kollektion H_i eine Kollision auf
  ADV(p(n), s) verursacht (im Falle k_max = 0 sei H_0 = λ)
  if T_A(0^n, w) = ⟨x, d⟩ then
    s := s#⟨x, d⟩
  else exit(loop) end
end loop
output w
```

Abbildung 5.2: Probabilistische Konstruktion eines Programms w für eine selbstreduzierbare Sprache.

Der in $\mathcal{FZPP}(\mathcal{NP})$ berechenbare Advice für die Eingabelänge n besteht aus einer Beispielfolge s und einer Kollektion H von linearen Hashfunktionen. Dabei hat H die Eigenschaft, für jedes x in $\{0,1\}^{\preceq n}$ eine (1/2)-Kollision zwischen $ADV(p(n), s)$ und $ADV(p(n), s\#\langle x, A(x)\rangle)$ herbeizuführen, nicht jedoch zwischen $ADV(p(n), s)$ und $ADV(p(n), s\#\langle x, \overline{A}(x)\rangle)$. Demnach stellt das Tripel $w = \langle 0^n, s, H\rangle$ ein korrektes Programm für die Sprache $A \oplus \overline{A}$ (für alle Eingaben bis zur Länge n) dar, falls es durch die folgende Interpreter-Sprache

$$D =_{def} \{ \langle x, b, w\rangle \mid w = \langle 0^n, s, H\rangle, \text{ wobei } s \text{ eine Beispielfolge ist und entweder } H = \lambda \text{ und } ADV(p(n), s\#\langle x, b\rangle) \neq \emptyset \text{ ist, oder } H \text{ eine Kollektion in } \mathcal{L}in[t(n); p(n); k] \text{ ist, die eine (1/2)-Kollision zwischen der Menge } ADV(p(n), s) \text{ und der Menge } ADV(p(n), s\#\langle x, b\rangle) \text{ verursacht } \}$$

ausgewertet wird. Hierbei lassen wir den Randfall zu, daß $H = \lambda$ ist. Die „Kollektion" $H = \lambda$ führt sozusagen immer zu einer (1/2)-Kollision zwischen den beiden Mengen $ADV(p(n), s)$ und $ADV(p(n), s\#\langle x, b\rangle)$, außer wenn letztere die leere Menge ist. Da D in $\mathcal{NP}$ liegt und A selbstreduzierbar ist, gibt es nach Lemma 5.2 einen $\mathcal{FP}(\mathcal{NP})$-Transducer T_A, der bei jeder Eingabe $\langle 0^n, w\rangle$ unter der Voraussetzung, daß für alle $x \in \{0,1\}^{\preceq n}$ mindestens eines der beiden Tupel $\langle x, 0, w\rangle$ und $\langle x, 1, w\rangle$ in D enthalten ist, eine Ausgabe $T_A(0^n, w)$ erzeugt, die die folgenden Bedingungen erfüllt:

$$T_A(0^n, w) = \langle x, d\rangle \quad \Rightarrow \quad d = A(x) \text{ und } \langle x, \overline{A}(x), w\rangle \in D,$$

$$T_A(0^n, w) = \quad \bot \qquad \Rightarrow \quad \text{für kein } x \in \{0,1\}^{\preceq n} \text{ ist } \langle x, \overline{A}(x), w\rangle \in D.$$

Wir geben nun eine formale Beschreibung eines probabilistischen Transducers T (siehe Abb. 5.2), der bei Eingabe n mit hoher Wahrscheinlichkeit nach einer polynomialen Anzahl von Rechenschritten stoppt und am Ende jeder Rechnung ein Programm w ausgibt, das unter obiger Interpreter-Sprache D alle Eingaben x für die Sprache $A \oplus \overline{A}$ bis zur Länge n korrekt entscheidet.

T betritt bei Eingabe x eine loop-Schleife, in der die zu Beginn mit dem leeren Wort initialisierte Beispielfolge s solange erweitert wird, bis die Testprozedur T_A die Korrektheit des konstruierten Programms w signalisiert. Dabei setzt sich w jeweils aus der momentanen Beispielfolge s und einer zufällig gewählten Kollektion $H_{k_{max}}$ von Hashfunktionen zusammen. Da k_{max} so gewählt wird, daß $H_{k_{max}}$ eine Kollision auf $ADV(p(n), s)$ verursacht, ist nach Lemma 3.14 gewährleistet, daß durch $H_{k_{max}}$ für jede Eingabe x in $\{0,1\}^{\preceq n}$ eine (1/2)-Kollision zwischen den beiden Mengen

- $ADV(p(n), s)$ und $ADV(p(n), s\#\langle x, 0\rangle)$

oder zwischen

- $ADV(p(n), s)$ und $ADV(p(n), s\#\langle x, 1\rangle)$

verursacht wird. Dies bedeutet, daß die Testprozedur T_A nur mit solchen Programmen w aufgerufen wird, die alle Eingaben x in der Menge $\{0,1\}^{\preceq n}$ (positiv oder negativ) entscheiden, für die also mindestens eines der beiden Tupel $\langle x, 0, w\rangle$ und $\langle x, 1, w\rangle$ in D enthalten ist. Wie man sich leicht überlegt, trifft dies auch für den Randfall zu, daß $H_0 = \lambda$ zur Bildung von w verwendet wurde. Dadurch ist bereits die Korrektheit von T gesichert, da T nur solche Programme w ausgibt, die den von T_A durchgeführten Test bestanden haben, und folglich auf allen Eingaben x bis zur Länge n korrekt sind.

Es bleibt also nur noch, die Laufzeit von T abzuschätzen. Bei jedem Schleifendurchlauf (außer dem letzten) wird aufgrund der Korrektheit der Testprozedur T_A ein Paar $\langle x, d\rangle$ mit $d = A(x)$ und $x \in \{0,1\}^{\preceq n}$ zurückgegeben. Ferner ist gemäß Lemma 5.2 gewährleistet, daß in diesem Fall das Tupel $\langle x, \overline{A}(x), w\rangle$ in D enthalten ist. Durch die Aufnahme von $\langle x, A(x)\rangle$ in die Beispielfolge s werden alle Advice-Strings in $ADV(p(n), s\#\langle x, \overline{A}(x)\rangle)$ aus der Menge $ADV(p(n), s)$ entfernt, da diese die Eingabe

x inkorrekt entscheiden. Wir zeigen nun, daß die Größe von $ADV(p(n), s)$ bei jeder einzelnen Erweiterung von s mit einer Wahrscheinlichkeit von mindestens $1 - 2^{-t(n)}$ um einen Faktor von mindestens $1 - 1/2^7 t(n)$ verkleinert wird.

Wir beginnen damit, den Wert von k_{max} nach unten abzuschätzen. Sei

$$k_0 \quad =_{def} \quad min\{k \geq 0 \mid t(n) \cdot 2^{k+1} \geq \|ADV(p(n), s)\|\}.$$

Da entweder $k_{max} = p(n)$ ist oder aber die zufällig gewählte Kollektion $H_{k_{max}+1}$ keine Kollision auf der Menge $ADV(p(n), s)$ verursacht, gilt sicherlich $\|ADV(p(n), s)\| \leq t(n) \cdot 2^{k_{max}+1}$, was

$$k_{max} \quad \geq \quad k_0$$

impliziert. Aufgrund der Definition von k_0 ergibt sich weiterhin die folgende Ungleichung:

$$\frac{\|ADV(p(n), s)\|}{t(n)} \quad \leq \quad 2^{k_0+1}.$$

Sei nun $I(n, s)$ die Menge aller Eingaben $x \in \{0,1\}^{\preceq n}$, die – sofern sie zu s hinzugefügt werden – die Größe von $ADV(p(n), s)$ nicht um mindestens den Faktor $1 - 1/2^7 t(n)$ verkleinern:

$$\begin{aligned} x \in I(n, s) \quad \Leftrightarrow_{def} \quad & x \in \{0,1\}^{\preceq n} \text{ und} \\ & \|ADV(p(n), s\#\langle x, \overline{A}(x)\rangle)\| < \|ADV(p(n), s)\|/2^7 t(n). \end{aligned}$$

Offensichtlich kann $I(n, s)$ nicht mehr als $2^{q(n)+1}$ Wörter enthalten. Des weiteren ist

$$\begin{aligned} \beta_k \quad =_{def} \quad & max_{x \in I(n,s)} \frac{\|ADV(p(n), s\#\langle x, \overline{A}(x)\rangle)\|}{2^k} \\ < \quad & \frac{\|ADV(p(n), s)\|}{2^{k+7} t(n)} \\ \leq \quad & 2^{k_0-k-6}. \end{aligned}$$

Daher erhalten wir für jedes $k \geq k_0$ mit Lemma 3.15 die folgende Abschätzung für die Wahrscheinlichkeit, daß eine zufällig aus $\mathcal{L}in[t(n); p(n); k]$ gewählte Kollektion H eine (1/2)-Kollision zwischen $ADV(p(n), s)$ und mindestens einer der Mengen $ADV(p(n), s\#\langle x, \overline{A}(x)\rangle)$, $x \in I(n, s)$, verursacht:

$$\begin{aligned} & \Pr[\, \exists x \in I(n, s) : (1/2)\text{-}Kollision(ADV(p(n), s), ADV(p(n), s\#\langle x, \overline{A}(x)\rangle), H)] \\ & \quad \leq \quad \|I(n, s)\| \cdot \|ADV(p(n), s)\| \cdot (4\beta_k(1 - \beta_k))^{t(n)/2} \\ & \quad \leq \quad 2^{q(n)+1+p(n)} \cdot (2^{k_0-k-4})^{t(n)/2} \\ & \quad \leq \quad 2^{(k_0-k-3)(q(n)+p(n)+1)}. \end{aligned}$$

Da die Testprozedur T_A bei Eingabe von $\langle 0^n, s, H_{k_{max}}\rangle$ nur solche Paare $\langle x, A(x)\rangle$ zurückgibt, für die das Tupel $\langle x, \overline{A}(x), w\rangle$ in D enthalten ist, werden von T nur solche Paare $\langle x, A(x)\rangle$ an die Beispielfolge s angehängt, für die $H_{k_{max}}$ eine (1/2)-Kollision zwischen $ADV(p(n), s)$ und der Menge $ADV(p(n), s\#\langle x, \overline{A}(x)\rangle)$ verursacht. Folglich kann die Wahrscheinlichkeit, daß das von T_A zurückgegebene Paar $\langle x, A(x)\rangle$ die Anzahl der Advice-Strings nicht mindestens auf

$$(1 - 1/2^7 t(n))\,\|ADV(p(n), s)\|$$

verkleinert, durch die Wahrscheinlichkeit abgeschätzt werden, daß die Kollektion $H_{k_{max}}$ eine (1/2)-Kollision zwischen $ADV(p(n), s)$ und einer der Mengen $ADV(p(n), s\#\langle x, \overline{A}(x)\rangle)$, $x \in I(n, s)$, verursacht. Da, wie oben gezeigt, $k_{max} \geq k_0$ ist, ist diese Wahrscheinlichkeit nicht größer als

$$\begin{aligned} \sum_{k=k_0}^{p(n)} 2^{(k_0-k-3)(q(n)+p(n)+1)} &< 2^{(k_0-k-2)(q(n)+p(n)+1)} \\ &\leq 2^{-t(n)}. \end{aligned}$$

Die loop-Schleife muß spätestens dann abbrechen, wenn in $ADV(p(n), s)$ nur noch korrekte Advice-Strings enthalten sind. Um die anfängliche Größe von $ADV(p(n), s)$ soweit zu verkleinern, sind höchstens

$$\lceil p(n)/\log(2^7 t(n)/(2^7 t(n) - 1))\rceil \;\leq\; 2^7 t(n) p(n)$$

Erweiterungen von s nötig, sofern diese die Größe von $ADV(p(n), s)$ jeweils um mindestens den Faktor $1 - 1/2^7 t(n)$ verkleinern. Da der Verkleinerungsfaktor $1 - 1/2^7 t(n)$ in jedem einzelnen Schleifendurchlauf nur mit einer Wahrscheinlichkeit von höchstens $2^{-t(n)}$ nicht erreicht wird, ist die Wahrscheinlichkeit, daß bei $2^7 t(n) p(n)$ Schleifendurchläufen dieser Faktor nicht jedesmal erreicht wird, kleiner als

$$2^7 t(n) p(n) \cdot 2^{-t(n)} \;=\; o(1).$$

Damit ist gezeigt, daß T für fast alle n mit einer Wahrscheinlichkeit von mindestens 1/2 nach spätestens $2^7 t(n) p(n)$ Schleifendurchläufen stoppt, woraus sich die Existenz einer geeigneten Advice-Funktion in $\mathcal{FZPP}(NP)$ für $A \oplus \overline{A}$ in bezug auf die $\mathcal{NP}$ Interpreter-Sprache D unmittelbar ergibt. ∎

Durch Kombination von Lemma 5.3 und Lemma 5.4 erhalten wir nun unmittelbar das Hauptresultat dieses Kapitels.

Satz 5.5 *Jede selbstreduzierbare Menge in $(\mathcal{NP} \cap co\text{-}\mathcal{NP})/poly$ ist low für die Klasse $\mathcal{ZPP}(\mathcal{NP})$.*

5.3 Kollaps-Konsequenzen

Satz 5.5 impliziert einen tieferen Kollaps der Polynomialzeit-Hierarchie unter der Voraussetzung, daß alle Sprachen in $\mathcal{NP}$ Schaltkreise polynomialer Größe haben. Ähnliche Kollaps-Konsequenzen ergeben sich für eine ganze Reihe von weiteren Komplexitätsklassen wie zum Beispiel $\mathcal{C}_=\mathcal{P}$, $\mathcal{PP}$, $\mathcal{M}od_k\mathcal{P}$, $\mathcal{PSPACE}$ und $\mathcal{EXP}$.

Korollar 5.6 *Falls $\mathcal{NP}$ in $(\mathcal{NP} \cap co\text{-}\mathcal{NP})/poly$ enthalten ist, kollabiert die Polynomialzeit-Hierarchie auf $\mathcal{ZPP}(\mathcal{NP})$.*

Beweis. Da SAT selbstreduzierbar ist, folgt aus der Annahme, daß SAT in $(\mathcal{NP} \cap co\text{-}\mathcal{NP})/poly$ enthalten ist, mit Satz 5.5, daß SAT low ist für $\mathcal{ZPP}(\mathcal{NP})$. Dies impliziert jedoch wegen $\Sigma_2^{\mathrm{P}} = \mathcal{NP}(\mathcal{NP}) \subseteq \mathcal{ZPP}(\mathcal{NP}(\mathrm{SAT})) \subseteq \mathcal{ZPP}(\mathcal{NP}) \subseteq \Pi_2^{\mathrm{P}}$ den Kollaps der Polynomialzeit-Hierarchie auf $\mathcal{ZPP}(\mathcal{NP})$. ■

Korollar 5.6 verbessert das von Karp, Lipton und Sipser in [KL80] erzielte Resultat, daß $\mathcal{NP}$ nicht in $\mathcal{P}/poly$ enthalten ist außer wenn die Polynomialzeit-Hierarchie auf Σ_2^{P} kollabiert. Eine weitere Verbesserung des Kollapspunktes auf die Δ_2^{P} Stufe kann höchstens mit nicht-relativierbaren Beweistechniken erzielt werden, da es Orakel gibt, relativ zu denen $\mathcal{NP}$ zwar in $\mathcal{P}/poly$ enthalten ist, die Polynomialzeit-Hierarchie jedoch nicht auf $\Delta_2^{\mathrm{P}} = \mathcal{P}(\mathcal{NP})$ kollabiert [Hel86, Wil85].

Als nächstes betrachten wir Erweiterungen von Satz 5.5, die sich unmittelbar aus der Tatsache herleiten lassen, daß er relativ zu einem beliebigem Orakel gültig bleibt. Präzise formuliert erhalten wir aus dem Beweis von Satz 5.5 die folgende relativierte Fassung.

Satz 5.7 *Sei B eine beliebige Sprache. Dann gilt für jede selbstreduzierbare Menge A in $(\mathcal{NP}(B) \cap co\text{-}\mathcal{NP}(B))/poly$ die Inklusion $\mathcal{NP}(A) \subseteq \mathcal{ZPP}(\mathcal{NP}(B))$.*

Setzen wir hierbei für B eine Sprache ein, die vollständig für die $\Sigma_{k-1}^{\mathrm{P}}$-Stufe der Polynomialzeit-Hierarchie ist, so ergibt sich unmittelbar die folgende Verallgemeinerung von Satz 5.5.

Satz 5.8 *Für jede selbstreduzierbare Menge A in $(\Sigma_k^{\mathrm{P}} \cap \Pi_k^{\mathrm{P}})/poly$ ist $\mathcal{NP}(A)$ in $\mathcal{ZPP}(\Sigma_k^{\mathrm{P}})$ enthalten.*

Da alle Sigma-Stufen der Polynomialzeit-Hierarchie vollständige selbstreduzierbare Sprachen enthalten, führt Satz 5.8 unmittelbar auf die folgende Verbesserung von Resultaten in [AFK89, Käm91], in denen unter denselben Voraussetzungen wie in Korollar 5.9 ein Kollaps der Polynomialzeit-Hierarchie auf $\Sigma_{k+1}^{\mathrm{P}}$ herbeigeführt wird.

Korollar 5.9 *Sei $k \geq 1$. Falls Σ_k^{P} in $(\Sigma_k^{\mathrm{P}} \cap \Pi_k^{\mathrm{P}})/poly$ enthalten ist, kollabiert die Polynomialzeit-Hierarchie auf $\mathcal{ZPP}(\Sigma_k^{\mathrm{P}})$.*

Beweis. Da Σ_k^{P} vollständige selbstreduzierbare Sprachen enthält, impliziert die Voraussetzung, daß Σ_k^{P} in $(\Sigma_k^{\mathrm{P}} \cap \Pi_k^{\mathrm{P}})/poly$ enthalten ist, daß $\Sigma_{k+1}^{\mathrm{P}} = \mathcal{NP}(\Sigma_k^{\mathrm{P}})$ (und damit auch die Polynomialzeit-Hierarchie) auf $\mathcal{ZPP}(\Sigma_k^{\mathrm{P}})$ herunterfällt. ■

Als eine weitere Folgerung aus Satz 5.8 erhalten wir folgende Verbesserung eines Resultats von Yap [Yap83], in dem ein Kollaps der Polynomialzeit-Hierarchie auf $\Sigma_{k+2}^{\mathrm{P}}$ unter der Voraussetzung, daß Π_k^{P} in $\Sigma_k^{\mathrm{P}}/poly$ enthalten ist, nachgewiesen wird.

Korollar 5.10 *Sei $k \geq 1$. Falls Π_k^{P} in $\Sigma_k^{\mathrm{P}}/poly$ enthalten ist, kollabiert die Polynomialzeit-Hierarchie auf $\mathcal{ZPP}(\Sigma_{k+1}^{\mathrm{P}})$.*

Beweis. Aus der Annahme, daß Π_k^{P} in $\Sigma_k^{\mathrm{P}}/poly$ enthalten ist, folgt die Inklusion von $\Sigma_{k+1}^{\mathrm{P}}$ in $\Sigma_k^{\mathrm{P}}/poly$. Da $\Sigma_k^{\mathrm{P}}/poly$ eine Teilklasse von $(\Sigma_{k+1}^{\mathrm{P}} \cap \Pi_{k+1}^{\mathrm{P}})/poly$ ist, können wir Korollar 5.9 anwenden, um den gewünschten Kollaps der Polynomialzeit-Hierarchie auf $\mathcal{ZPP}(\Sigma_{k+1}^{\mathrm{P}})$ herbeizuführen. ■

Interessanterweise läßt sich Satz 5.8 nicht nur auf Komplexitätsklassen anwenden, die vollständige selbstreduzierbare Sprachen enthalten. Wie das folgende Korollar am Beispiel der Klassen $\mathcal{UP}$ und $\mathcal{F}ew\mathcal{P}$ zeigt, genügt es bereits, wenn jede beliebige Sprache A der betrachteten Klasse auf eine (von A abhängige) selbstreduzierbare Sprache innerhalb der Klasse reduziert werden kann.

Korollar 5.11 *$\mathcal{UP}$ ($\mathcal{F}ew\mathcal{P}$) ist nicht in $(\mathcal{NP} \cap co\text{-}\mathcal{NP})/poly$ enthalten, außer wenn $\mathcal{UP}$ (bzw. $\mathcal{F}ew\mathcal{P}$) low für $\mathcal{ZPP}(\mathcal{NP})$ ist.*

Beweis. Sei A eine beliebige Sprache in $\mathcal{UP}$ ($\mathcal{F}ew\mathcal{P}$). Es ist leicht zu sehen, daß dann auch die Left-Menge *Left-A* von A in $\mathcal{UP}$ (bzw. $\mathcal{F}ew\mathcal{P}$) liegt (siehe Beispiel 4). Da *Left-A* selbstreduzierbar ist, impliziert die Voraussetzung, daß $\mathcal{UP}$ (bzw. $\mathcal{F}ew\mathcal{P}$) in $(\mathcal{NP} \cap co\text{-}\mathcal{NP})/poly$ enthalten ist, daß *Left-A* low für $\mathcal{ZPP}(\mathcal{NP})$ ist. Da A auf *Left-A* many-one reduzierbar ist, ist dann aber auch A selbst low für $\mathcal{ZPP}(\mathcal{NP})$. ■

Das folgende Korollar zeigt Kollaps-Konsequenzen für weitere Komplexitätsklassen auf. Für die Klassen $\mathcal{PP}$, $\mathcal{M}od_m\mathcal{P}$, $\mathcal{PSPACE}$ und $\mathcal{EXP}$ (nicht jedoch für $\mathcal{NP}$ oder $\mathcal{C}_=\mathcal{P}$) konnte unter der Annahme, daß sie in $\mathcal{P}/poly$ enthalten sind, mit nicht-relativierbaren Beweistechniken, die auf dem Konzept interaktiver Beweissysteme basieren, ein vermutlich tieferer Kollaps auf $\mathcal{MA} \cap co\text{-}\mathcal{MA} \subseteq \mathcal{ZPP}(\mathcal{NP})$ erzielt werden (siehe Satz 5.13).

Korollar 5.12 *Sei $k \geq 1$.*

(i) Die Klasse $\mathcal{C}_=\mathcal{P}$ ist nicht in $(\Sigma_k^{\mathrm{P}} \cap \Pi_k^{\mathrm{P}})/poly$ enthalten, außer wenn die Counting-Hierarchie auf $\mathcal{ZPP}(\Sigma_k^{\mathrm{P}})$ kollabiert.

(ii) Keine der Klassen $\mathcal{M}od_m\mathcal{P}$, $m \geq 2$, ist in $(\Sigma_k^{\mathrm{P}} \cap \Pi_k^{\mathrm{P}})/poly$ enthalten, außer wenn die Polynomialzeit-Hierarchie (und $\mathcal{M}od_m\mathcal{P}$) auf $\mathcal{ZPP}(\Sigma_k^{\mathrm{P}})$ kollabiert.

(iii) Die Klassen $\mathcal{PSPACE}$ und $\mathcal{EXP}$ sind nicht in $(\Sigma_k^{\mathrm{P}} \cap \Pi_k^{\mathrm{P}})/poly$ enthalten, außer wenn sie auf $\mathcal{ZPP}(\Sigma_k^{\mathrm{P}})$ herunterfallen.

Beweis.

(i) Zunächst läßt sich aufgrund der Tatsache, daß $\mathcal{C}_=\mathcal{P}$ vollständige selbstreduzierbare Sprachen enthält, aus der Annahme, daß $\mathcal{C}_=\mathcal{P}$ in $(\Sigma_k^{\mathrm{P}} \cap \Pi_k^{\mathrm{P}})/poly$ enthalten ist, mit Satz 5.8 die Inklusion $\mathcal{C}_=\mathcal{P} \subseteq \mathcal{ZPP}(\Sigma_k^{\mathrm{P}})$ folgern.

Da weiterhin $\mathcal{PH}$ in $\mathcal{BPP}(\mathcal{C}_=\mathcal{P})$ enthalten ist [TO92, Tar93], impliziert die Annahme, daß $\mathcal{C}_=\mathcal{P}$ in $(\Sigma_k^{\mathrm{P}} \cap \Pi_k^{\mathrm{P}})/poly$ enthalten ist, auch die Inklusion $\mathcal{PH} \subseteq (\Sigma_k^{\mathrm{P}} \cap \Pi_k^{\mathrm{P}})/poly$, was mit Korollar 5.9 einen Kollaps von $\mathcal{PH}$ auf $\mathcal{ZPP}(\Sigma_k^{\mathrm{P}})$ bewirkt.

Da sich schließlich die Counting-Hierarchie mithilfe des $\mathcal{C}_=\mathcal{P}(\cdot)$-Operators aufbauen läßt [Tor91], das heißt $\mathcal{CH} = \mathcal{C}_=\mathcal{P} \cup \mathcal{C}_=\mathcal{P}(\mathcal{C}_=\mathcal{P}) \cup \ldots$, erhalten wir unter Verwendung von $\mathcal{C}_=\mathcal{P}(\mathcal{PH}) \subseteq \mathcal{BPP}(\mathcal{C}_=\mathcal{P})$ [TO92] $\subseteq \mathcal{PH}$ induktiv (über die Anzahl der Anwendungen des $\mathcal{C}_=\mathcal{P}(\cdot)$-Operators), daß die Counting-Hierarchie auf die Polynomialzeit-Hierarchie (und folglich auf $\mathcal{ZPP}(\Sigma_k^{\mathrm{P}})$) kollabiert.

(ii) Der Beweis verläuft vollkommen analog wie in Teil *i*) unter Berücksichtigung der Tatsache, daß $\mathcal{M}od_m\mathcal{P}$ vollständige selbstreduzierbare Sprachen enthält [OL93], und daß $\mathcal{PH}$ in $\mathcal{BPP}(\mathcal{M}od_m\mathcal{P})$ enthalten ist [TO92, Tar93].

(iii) ist eine unmittelbare Folgerung aus Satz 5.8, da $\mathcal{PSPACE}$ und $\mathcal{EXP}$ vollständige selbstreduzierbare Sprachen besitzen.

■

Die folgenden in [LFKN92, BFL91, BF91] bewiesenen Kollaps-Konsequenzen beruhen darauf, daß es für jede vollständige Sprache A in den Klassen $\mathcal{PP}$, $\mathcal{M}od_m\mathcal{P}$, $\mathcal{PSPACE}$ oder $\mathcal{EXP}$ ein interaktives Beweissystem gibt, bei dem der ehrliche (bzw. die ehrlichen) Prover durch einen $\mathcal{FP}(A)$ Transducer implementiert werden können. Unter der Annahme, daß A in $\mathcal{P}/poly$ enthalten ist, läßt sich dieses Beweissystem leicht in ein $\mathcal{MA}$ Protokoll verwandeln, bei dem Merlin lediglich die Aufgabe hat, in der ersten Runde geeignete Schaltkreise (oder Advice-Strings) für A an Arthur zu senden, der fortan die Rolle sowohl der Prover (unter Verwendung der Schaltkreise) als auch des Verifiers übernehmen kann.

Satz 5.13 [LFKN92, BFL91, BF91] *Die Klassen $\mathcal{PP}$, $\mathcal{M}od_m\mathcal{P}$, $\mathcal{PSPACE}$ und $\mathcal{EXP}$ besitzen keine Schaltkreise polynomialer Größe, außer wenn sie in $\mathcal{MA}$ enthalten sind.*

5.4 Superpolynomiale Schaltkreis-Komplexität

Die im letzten Abschnitt aufgezeigten Kollaps-Konsequenzen können dazu verwendet werden, verbesserte untere Schranken für die Schaltkreiskomplexität von Sprachen in bestimmten probabilistischen Komplexitätsklassen zu beweisen. Unser Ausgangspunkt ist das folgende Lemma, dessen Beweis im wesentlichen auf einer Diagonalisierungs-Technik beruht, die in ähnlicher Form von Kannan [Kan82] verwendet wurde.

Lemma 5.14 *Für jedes feste Polynom s und jede Interpreter-Sprache $B \in \mathcal{P}$ gibt es eine $\mathcal{P}(\Sigma_2^{\mathrm{P}})$-auflistbare Sprache A, die nur solche Advice-Funktionen h hat, deren Länge für fast alle n größer als $s(n)$ ist. Desweiteren enthält A nur Wörter der Bauart $x = 0^{|x|-\lceil c \log |x| \rceil}y$ für eine geeignete (von s abhängige) Konstante c.*

Beweis. Die Konstruktion von A erfolgt separat für jede Eingabelänge n, unabhängig davon, welche Wörter der Länge ungleich n in A aufgenommen werden. Sei $x_1, x_2, \ldots, x_{2^n}$ eine Aufzählung aller Wörter der Länge n in lexikographischer Ordnung. Wir definieren Mengen *Have-Advice* und *Find-A* wie folgt:

$$\begin{aligned}&\langle n, a_1 \cdots a_{s(n)} \rangle \in \textit{Have-Advice} \iff \\ &\qquad \exists\, w \in \Sigma^{<s(n)}, \forall\, i, 1 \le i \le min(s(n), 2^n) : a_i = B(x_i, w),\end{aligned}$$

$$\begin{aligned}&\langle n, a_1 \cdots a_j 10^{s(n)-j} \rangle \in \textit{Find-A} \iff \\ &\qquad \exists\, a_{j+1} \cdots a_{s(n)} : \langle n, a_1 \cdots a_j a_{j+1} \cdots a_{s(n)} \rangle \notin \textit{Have-Advice}.\end{aligned}$$

Man beachte, daß es für alle n mit $s(n) \leq 2^n$ zumindest ein Paar der Form $\langle n, a_1 \cdots a_{s(n)} \rangle$ gibt, das nicht in *Have-Advice* enthalten ist. Jedes solche Tupel beschreibt also eine Teilmenge von $\{0,1\}^n$, die durch keinen Schaltkreis der Größe kleiner als $s(n)$ berechnet werden kann. Das heißt, keiner dieser Schaltkreise akzeptiert die Wörter $x_1, \ldots, x_{s(n)}$ in der durch die Bitfolge $a_1, \ldots, a_{s(n)}$ vorgegebenen Weise. Sei α_n das lexikographisch kleinste solche Paar. Dann können wir $A^{=n}$ wie folgt definieren.

> Ein String x_i der Länge n gehört genau dann zu A, wenn $i \leq s(n) \leq 2^n$ ist, und das i-te Bit von α_n gleich 1 ist.

Offensichtlich gibt es für fast alle n keinen Schaltkreis der Größe kleiner als $s(n)$, der $A^{=n}$ korrekt entscheidet. Da *Have-Advice* in $\mathcal{NP}$ ist, ist *Find-A* in $\mathcal{NP}(\mathcal{NP})$, und schließlich ist leicht zu sehen, daß A unter Befragung des Orakels *Find-A* in Polynomialzeit aufgelistet werden kann. ■

Da jeder Schaltkreis der Größe m in einen Advice-String der Länge $m \log m$ kodiert und als solcher von einer geeigneten Interpreter-Sprache I_{univ} ausgewertet werden kann, beinhaltet Lemma 5.14 als Spezialfall die Aussage, daß für jedes feste Polynom s eine $\mathcal{P}(\Sigma_2^{\mathrm{P}})$-auflistbare Sprache A existiert, die nur für endlich viele n Schaltkreise der Größe $s(n)$ hat.

Unter Verwendung von Korollar 5.6 sind wir nun in der Lage, das Resultat von Kannan zu verbessern, das für jedes Polynom s die Existenz einer Sprache in $\Sigma_2^{\mathrm{P}} \cap \Pi_2^{\mathrm{P}}$ garantiert, die keine Schaltkreise der Größe $O(s(n))$ hat.

Korollar 5.15 *Für jedes feste Polynom s gibt es eine Sprache A in $\mathcal{ZPP}(\mathcal{NP})$, die keine Schaltkreise der Größe $O(s(n))$ hat.*

Beweis. Falls $\mathcal{NP}$ nicht in $\mathcal{P}/poly$ enthalten ist, können wir $A = \mathrm{SAT}$ wählen. Andernfalls kollabiert die Polynomialzeit-Hierarchie gemäß Korollar 5.6 auf $\mathcal{ZPP}(\mathcal{NP})$, das heißt, es gibt unter Hinzuziehung von Lemma 5.14 sogar eine spärliche Menge in $\Delta_3^{\mathrm{P}} \subseteq \mathcal{ZPP}(\mathcal{NP})$, die keine Schaltkreise der Größe $s^2(n)$ hat. ■

Durch einen einfachen Padding-Trick (genau genommen handelt es sich um einen „Unpadding"-Trick) läßt sich nun die Existenz von Sprachen, die nur Schaltkreise superpolynomialer Größe haben, in Komplexitätsklassen nachweisen, die nur geringfügig größer sind als $\mathcal{ZPP}(\mathcal{NP})$.

Satz 5.16 *Sei $t(n)$ eine zeitkonstruierbare monotone Funktion, die nicht in $n^{O(1)}$ liegt. Dann gibt es in ZPTIME$[t(n)](\mathcal{NP})$ eine Sprache L, die nicht in $\mathcal{P}/poly$ enthalten ist.*

Beweis. Falls $\mathcal{NP}$ nicht in $\mathcal{P}/poly$ enthalten ist, können wir wieder für A die Sprache SAT wählen. Andernfalls sei I_{univ} eine universelle Interpreter-Sprache (siehe Abschnitt 2.3), das heißt, für jede Interpreter-Sprache $B \in \mathcal{P}$ gibt es ein Polynom p, so daß für jede Advice-Funktion h eine Advice-Funktion h' existiert mit

- $B/h = I_{univ}/h'$ und
- für alle n ist $|h'(n)| \leq p(n + |h(n)|)$.

Nach Lemma 5.14 gibt es in $\mathcal{ZPP}(\mathcal{NP})$ (also in ZPTIME$[n^k](\mathcal{NP})$, k fest) eine Sprache A, die bezüglich der universellen Interpreter-Sprache I_{univ} nur Advice-Funktionen h mit $|h(n)| \geq n$ für fast alle n hat. Außerdem können wir nach Lemma 5.14 voraussetzen, daß jedes Wort x der Länge n in A von der Bauart $x = 0^{n-\lceil c \log n \rceil} y$ für eine geeignete Konstante c ist. Das heißt, eine Eins kann nur an den $\lceil c \log n \rceil$ niederwertigen Bitstellen vorkommen. Betrachte nun die Sprache $L =_{def} f^{-1}(A)$, die durch folgende (injektive) Reduktionsfunktion f auf A reduziert wird.

$$f(x) \quad =_{def} \quad 0^{\lfloor t(n)^{1/k} \rfloor - |x|} x.$$

Offenbar liegt f nicht in $\mathcal{FP}$, wohl aber ist die (partielle) Umkehrfunktion f^{-1} in Polynomialzeit berechenbar (man beachte hierzu, daß t zeitkonstruierbar und monoton ist). Es ist auch leicht zu sehen, daß L in ZPTIME$[t(n)](\mathcal{NP})$ enthalten ist. Sei nun h eine beliebige Advice-Funktion für L in bezug auf I_{univ}. Wir zeigen, daß h nicht polynomial längen-beschränkt sein kann, und folglich L nicht in $\mathcal{P}/poly$ liegt. Betrachte die Interpreter-Sprache B, die wie folgt definiert ist.

$$\begin{aligned} \langle y, w \rangle \in B \quad \Leftrightarrow_{def} \quad & y \in range(f) \text{ (das heißt, } y \text{ besitzt ein Urbild bzgl. } f) \\ & \text{und } \langle f^{-1}(y), w \rangle \in I_{univ}. \end{aligned}$$

B ist offensichtlich eine Sprache in $\mathcal{P}$. Weiterhin gilt für alle $n \geq 0$ und für alle y der Länge $\lfloor t(n)^{1/k} \rfloor$,

$$\begin{aligned} y \in A \quad & \Leftrightarrow \quad y \in range(f) \text{ und } f^{-1}(y) \in L \\ & \Leftrightarrow \quad y \in range(f) \text{ und } \langle f^{-1}(y), h(|f^{-1}(y)|) \rangle \in I_{univ} \\ & \Leftrightarrow \quad \langle y, h(n) \rangle \in B. \end{aligned}$$

Folglich ist $h(n)$ ein Advice, der in bezug auf die Interpreter-Sprache B alle Eingaben der Länge $\lfloor t(n)^{1/k} \rfloor$ korrekt entscheidet. Aufgrund der Universalität von I_{univ} gibt es also eine Advice-Funktion $h'(n)$ und ein Polynom p mit $|h'(n)| \leq p(n+|h(n)|)$, so daß $h'(n)$ unter I_{univ} für alle n einen korrekten Advice für sämtliche Eingaben der Länge $\lfloor t(n)^{1/k} \rfloor$ für A darstellt. Da A für fast alle Eingabelängen keinen sublinearen Advice bzgl. der universellen Interpretersprache I_{univ} hat, erhalten wir die folgende Abschätzung für fast alle n:

$$p(n + |h(n)|) \;\geq\; |h'(n)| \;\geq\; \lfloor t(n)^{1/k} \rfloor.$$

Also muß die Länge von h superpolynomial sein. ∎

Insbesondere muß es also Sprachen in $\mathcal{ZPEXP}(\mathcal{NP})$ geben, die keine Schaltkreise polynomialer Größe haben. Seltsamerweise ist aus obigem Beweis nicht ersichtlich, welche Sprache in $\mathcal{ZPEXP}(\mathcal{NP})$ nun konkret keine Schaltkreise polynomialer Größe hat. Es wird zwar eine Sprache L konstruiert, die definitiv nicht in $\mathcal{P}/poly$ liegt. Von dieser Sprache kann allerdings nur unter der Annahme, daß SAT Schaltkreise polynomialer Größe hat, gezeigt werden, daß sie in $ZPTIME[t(n)](\mathcal{NP})$ liegt. Das heißt, wir wissen zwar, daß zumindest eine der beiden Sprachen SAT und L in $ZPTIME[t(n)](\mathcal{NP}) - \mathcal{P}/poly$ liegt, aber wir wissen nicht welche. Dies wirft die interessante Frage auf, ob es möglich ist, *eine* konkrete Sprache L anzugeben, die definitiv in $ZPTIME[t(n)](\mathcal{NP}) - \mathcal{P}/poly$ liegt. Mit den heutigen Techniken scheint dies nicht einmal innerhalb von $\mathcal{NEXP}(\mathcal{NP}) \cap co\text{-}\mathcal{NEXP}(\mathcal{NP}) - \mathcal{P}/poly$ möglich zu sein.

Kapitel 6

Erlernbarkeit von Booleschen Schaltkreisen

In diesem Kapitel zeigen wir, daß Boolesche Schaltkreise mithilfe von Äquivalenzfragen und unter zusätzlicher Verwendung eines Σ_2^p-Orakels in Polynomialzeit exakt erlernbar sind. Dies verbessert ein entsprechendes Resultat in [BCG$^+$95], das anstelle des Σ_2^p-Orakels ein Σ_3^p-Orakel erfordert. Diese Verbesserung wird unter anderem durch die in Abschnitt 3.3 weiterentwickelte Hashing-Technik ermöglicht, mit deren Hilfe die maximale Mächtigkeit von exponentiell vielen Mengen mit einer im Prinzip beliebig hohen Genauigkeit abgeschätzt werden kann.

6.1 Erlernen von Konzepten

Die Theorie des *algorithmischen Lernens* entwickelte sich aus dem Bestreben heraus, den intutiven Begriff des Lernens formal zu fassen. Hierzu wurden in der Vergangenheit verschiedenartige Lernmodelle vorgeschlagen, die auf Ideen einerseits aus dem Bereich der Berechenbarkeit und - in neuerer Zeit - auch der Komplexitätstheorie basieren. Einer Vielzahl dieser Modelle liegt (möglicherweise in leicht abgewandelter Form) die folgende Ausgangskonstellation zugrunde: ein *Lernalgorithmus* soll in Gegenwart eines *Lehrers* ein ihm anfangs vollkommen unbekanntes *Konzept* aus einer fest vorgegebenen *Konzeptklasse* erlernen. Diese Art von Lernen wird auch als „supervised learning“ bezeichnet.

Das Spiel „Mastermind“ stellt ein sehr anschauliches Beispiel für diese Art des Lernens dar, anhand dessen sich die erwähnten Begriffsbildungen sehr schön illustrieren lassen. In diesem Spiel wählt der eine der beiden Spieler (der Lehrer) zu Beginn eine Farbkombination aus, die der zweite Mitspieler (der Lernende) durch geschicktes Stellen von Fragen an den Lehrer herausfinden muß. Das Stellen einer Frage ist hierbei gleichbedeutend mit dem Aufstellen einer Hypothese, wie die durch den Lehrer ausgewählte Farbkombination aussehen könnte. In seiner Antwort teilt der Lehrer dem zweiten Spieler mit, inwieweit sich die vorgelegte Hypothese von der „richtigen“ Farbkombination noch unterscheidet. In diesem Beispiel besteht also die Konzeptklasse aus allen zulässigen Farbkombinationen, und der Lehrer bestimmt hieraus zu Beginn des Spiels willkürlich ein Konzept, das anschließend durch den Mitspieler in einer möglichst geringen Anzahl von Frage- und Antwortrunden „erlernt“ werden soll.

In der Terminologie des algorithmischen Lernens wird das zu erlernende Konzept mit dem Begriff *Zielkonzept* bezeichnet. Ein Konzept C entspricht hierbei einer Ja/Nein-Klassifikation von gewissen Objekten aus einem festgelegten Universum U. Das heißt, jedes Objekt in U erfährt durch ein Konzept C entweder eine positive oder negative Klassifizierung, je nachdem ob es als ein Repräsentant dieses Konzeptes angesehen wird oder nicht.[1]

Das soeben beschriebene Beispiel eines Lernvorgangs anhand des Unterhaltungsspiels „Mastermind“ könnte man etwa dadurch formalisieren, daß man das Universum U aus Paaren der Form (i, c) aufbaut, und zwar mit der Intention, eine vorgegebene Farbkombination $(c_1, \ldots, c_k)$ mit dem Konzept $\{(1, c_1), \ldots, (k, c_k)\} \subseteq U$ zu identifizieren. Um ein weiteres anschauliches Beispiel einer Konzeptklasse zu geben, könnten wir über dem Universum U aller an der Universität Ulm beschäftigten Personen Konzepte wie „Professor“ oder „Assistent“ usw. betrachten. Das Konzept „Professor“ würde sich in diesem Beispiel aus allen Professoren der Universität Ulm zusammensetzen.

Generell wird jedes Objekt des Universums durch ein Wort über einem geeigneten Alphabet Σ kodiert. Dies bedeutet, daß jedes Konzept eindeutig durch eine Sprache über Σ^* modelliert wird. Ein vorgegebenes Zielkonzept zu erlernen bedeutet für einen Lernalgorithmus, es entweder exakt oder zumindest näherungsweise zu identifizieren. Dies setzt voraus, daß jedes einzelne Konzept durch ein endliches Objekt beschreibbar ist.

Zur Spezifizierung eines *Lernproblems* gehört daher neben der Angabe der zu erlernenden Konzeptklasse auch immer ein geeignetes *Beschreibungsmodell*, das zur

[1] Seltener werden in der Literatur auch funktionale Konzepte betrachtet; wir beschränken uns jedoch in dieser Arbeit auf die Betrachtung von Sprachkonzepten.

Darstellung der Konzepte in dieser Klasse herangezogen wird. Ein typisches Beispiel für ein formales Lernproblem erhält man etwa, indem man als Konzeptklasse die Klasse der regulären Sprachen wählt, und vereinbart, zu ihrer Beschreibung reguläre Ausdrücke zu verwenden. Die Wahl des verwendeten Beschreibungsmodells hat im allgemeinen einen erheblichen Einfluß darauf, wie effizient eine Konzeptklasse in einem bestimmten Lernmodell erlernt werden kann.

Nachdem wir nun beschrieben haben, welche Art von Lernprobleme ein Lernalgorithmus zu bewältigen hat, diskutieren wir als nächstes, durch welche Rahmenbedingungen die Erlernbarkeit einer Konzeptklasse in verschiedenen Lernmodellen beeinflußt wird. Die Spezifikation eines formalen Lernmodells erfordert u.a. eine Festlegung der durch folgende Fragestellungen charakterisierten Parameter:

- *Wer* soll etwas erlernen, das heißt, welche Art von Lernalgorithmen soll untersucht werden, und welches *Maschinenmodell* liegt ihnen zugrunde?
- *Wie* soll gelernt werden, das heißt, nach welchen *Regeln* und unter welchen Bedingungen soll der Lernprozeß ablaufen?
- Welche *Gütekriterien* muß ein Lernalgorithmus erfüllen, bevor man von einer *erfolgreichen* Lösung eines gegebenen Lernproblems sprechen kann?
- Auf den sparsamen Verbrauch welcher *Ressourcen* wird dabei Wert gelegt?

Um ein vorgegebenes Zielkonzept $\hat{C}$ ausfindig machen zu können, müßen dem Lernalgorithmus im Verlauf des Lernprozesses Informationen über das Zielkonzept $\hat{C}$ zugeführt werden. In dieser Arbeit werden wir die beiden folgenden Lernmodelle diskutieren, die sich in erster Linie dadurch unterscheiden, nach welchen Regeln der Lernalgorithmus zum Zwecke der Informationsbeschaffung mit dem Lehrer (bzw. der Außenwelt) kommuniziert.

- Das Modell des PAC-Lernens, das von Valiant in seiner fundamentalen Arbeit [Val84] eingeführt wurde und zur Kategorie des „Lernens von Konzepten durch Beispiele“ gehört (siehe Abschnitt 6.4), sowie
- das von Angluin [Ang88] zur Beschreibung von *interaktiven* Lernprozessen eingeführte Modell des „exakten Lernens von Konzepten durch Fragen“ (siehe Abschnitt 6.2).

6.2 Exaktes Lernen durch Fragen

In diesem Abschnitt geben wir eine formale Beschreibung für das Modell des exakten Lernens durch Fragen. Wir verwenden im wesentlichen die von Angluin eingeführte Notation und verweisen den an einer weiterführenden Diskussion interessierten Leser auf die Arbeiten [Ang88, Wat94].

Die durch einen Lernalgorithmus zu lösende Aufgabe besteht in diesem Modell darin, eine Repräsentation r für ein beliebig vorgegebenes Zielkonzept $\hat{C}$ aus einer festen Konzeptklasse $\mathcal{C}$ unter Befragung eines Experten (Lehrers), der nur ganz bestimmte Arten von Fragen über das Zielkonzept zu beantworten weiß, exakt zu bestimmen.

VEREINBARUNG

- Jedes einzelne **Objekt** des Universums wird durch ein Wort u über dem Binäralphabet $\Sigma = \{0, 1\}$ beschrieben, das heißt, das **Universum** U besteht hier immer aus der Menge Σ^* aller Binärwörter.
- Ein **Konzept** ist folglich eine Sprache $C \subseteq \Sigma^*$, und eine **Konzeptklasse** ist eine Teilmenge der Potenzmenge von Σ^*.
- Jedes einzelne Konzept C wird durch eine oder meist mehrere Beschreibungen aus einer festen **Repräsentationssprache** R dargestellt. Eine **Repräsentation** $r \in R$ ist hierbei ein Wort über dem der Repräsentationssprache zugrundeliegenden Alphabet (von dem wir ebenfalls annehmen, daß es das Binäralphabet Σ ist).
- Das durch r dargestellte Konzept bezeichnen wir mit $\Phi(r)$. Das heißt, jeder Repräsentation $r \in R$ wird ein Konzept mittels einer surjektiven **Semantikfunktion** $\Phi : R \to \mathcal{C}$ zugeordnet. Da oftmals die zugrundeliegende Semantikfunktion Φ nur implizit definiert ist, werden wir gelegentlich auch die Bezeichnung C_r anstelle von $\Phi(r)$ verwenden.

Beispiel 5 *Jeder deterministische endliche Automat M repräsentiert eine reguläre Sprache $L(M) = \{x \mid M \text{ akzeptiert } x\}$. Die Repräsentationssprache R_{DFA} aller (Beschreibungen von) deterministischen endlichen Automaten bildet folglich eine unter einer Vielzahl von Möglichkeiten, die Konzeptklasse $\mathcal{C}_{\mathrm{REG}}$ aller regulären Sprachen zu repräsentieren.*

Genau genommen müßten wir in obigem Beispiel zwischen einem deterministischen endlichen Automaten M (als mathematischem Objekt) und seiner Kodierung (durch ein Wort über einem geeigneten Repräsentationsalphabet) unterscheiden. Wir setzen jedoch für alle hier betrachteten Beschreibungsmodelle (wie zum Beispiel reguläre Ausdrücke, endliche Automaten, Boolesche Schaltkreise, etc.) eine geeignete Kodierungsvorschrift voraus, ohne sie im einzelnen festzulegen. Man beachte, daß durch die verwendete Kodierung auch immer eine entsprechende Semantikfunktion implizit vorgegeben ist.

Prinzipiell kann eine Konzeptklasse $\mathcal{C}$ auf verschiedene Arten durch entsprechende Repräsentationssprachen beschrieben werden. Wie allgemein üblich, setzen wir jedoch generell voraus, daß jede zur Darstellung einer Konzeptklasse $\mathcal{C}$ verwendete Repräsentationssprache R die folgende **Uniformitätsbedingung** erfüllt.

VEREINBARUNG

Wir setzen voraus, daß für jede hier betrachtete Repräsentationssprache R ein Polynomialzeit-Algorithmus existiert, der für ein gegebenes Wort x und für eine gegebene Repräsentation r entscheidet, ob x zu dem durch r dargestellten Konzept $\Phi(r)$ gehört oder nicht.

Unter Beachtung dieser Uniformitätsbedingung sind grundsätzlich nur solche Konzeptklassen $\mathcal{C}$ repräsentierbar, für die es eine Konstante k gibt, so daß alle in $\mathcal{C}$ enthaltenen Sprachkonzepte in $DTIME(n^k)$ *entscheidbar* sind. Nichtsdestotrotz kann es sehr schwierig sein, die Konzeptklasse $\mathcal{C}$ zu *erlernen*, das heißt, eine Beschreibung r für ein beliebig durch einen Lehrer vorgegebenes Zielkonzept aus der Klasse $\mathcal{C}$ zu finden.

Den im Modell des exakten Lernens durch Fragen betrachteten Lernalgorithmen liegt das folgende Maschinenmodell zugrunde.

Definition 6.1 *Ein* **Lern-Algorithmus** *wird formal beschrieben durch eine Mehrband-Turingmaschine, die zusätzlich zu ihren Arbeitsbändern noch ein spezielles Ausgabe- und ein spezielles Orakelband besitzt.*

Das Orakelband wird zur Kommunikation mit einem Lehrer benutzt, und zwar sowohl, um dem Lehrer Fragen über das Zielkonzept zu stellen, als auch zur Entgegennahme der Gegenbeispiele (siehe nächsten Abschnitt). Das Ausgabeband dient lediglich dazu, am Ende des Lernvorgangs eine Beschreibung des ermittelten Zielkonzeptes auszugeben.

6.3 Interaktion mit der Außenwelt

Außer der Entscheidung, welche Repräsentationssprache R zur Darstellung einer gegebenen Konzeptklasse $\mathcal{C}$ gewählt wird, spielt es für die Erlernbarkeit von $\mathcal{C}$ eine mindestens ebenso wichtige Rolle, welche Arten von Fragen über das Zielkonzept zugelassen werden. Welche Typen von Fragen ein Lernalgorithmus stellen darf, wird durch ein sogenanntes *Protokoll* festgelegt, das mehr oder weniger flexibel sein kann. Jeder Fragetyp charakterisiert hierbei eine ganze Klasse von Fragen. Wir vereinbaren im einzelnen die folgenden vier Fragetypen, auf die sich unsere Diskussion beschränken wird („Mem" steht für „membership", „Equ" für „equivalence", „Sub" für „subset" und „Sup" für „superset").

Zugehörigkeitsfragen (Mem): Um eine Zugehörigkeitsfrage zu stellen, schreibt der Lern-Algorithmus ein Wort $x \in \{0,1\}^*$ auf das Orakelband und geht anschließend in einen speziellen (Mem-)Fragezustand über. Die Antwort wird der Maschine durch Versetzen in einen entsprechenden Antwortzustand mitgeteilt und ist entweder „JA" oder „NEIN", je nachdem, ob x ein Element des Zielkonzeptes $\hat{C}$ ist oder nicht.

Äquivalenzfragen (Equ): Eine Äquivalenzfrage zu stellen bedeutet, eine Repräsentation $r \in R$ auf das Orakelband zu schreiben. Das durch r dargestellte Konzept C_r wird als **Hypothese** bezeichnet. Die Antwort ist „JA", falls die Hypothese C_r mit dem Zielkonzept $\hat{C}$ übereinstimmt. Andernfalls besteht die Antwort in der Angabe eines beliebigen Gegenbeispiels $x \in \hat{C} \Delta C_r$, das in einem Rechenschritt anstelle von r auf dem Orakelband erscheint, und von dort gelesen werden kann.

Teilmengenfragen (Sub): Die Antwort auf eine Teilmengenfrage $r \in R$ ist „JA", falls die durch r dargestellte Hypothese C_r im Zielkonzept $\hat{C}$ enthalten ist, andernfalls wird ein beliebiges Gegenbeispiel $x \in C_r - \hat{C}$ zurückgegeben.

Obermengenfragen (Sup): Die Antwort auf eine Obermengenfrage $r \in R$ ist „JA", falls die durch r dargestellte Hypothese C_r eine Obermenge des Zielkonzeptes $\hat{C}$ ist, andernfalls wird ein beliebiges Gegenbeispiel $x \in \hat{C} - C_r$ zurückgegeben.

Im bereits angesprochenen Beispiel des Lernens von regulären Sprachen würde das Protokoll $\alpha = (\text{Mem})$ vorschreiben, daß nur Fragen nach der Zugehörigkeit beliebiger Objekte zur Zielsprache erlaubt sind. Das etwas flexiblere Protokoll $\alpha =$ (Mem, Equ) würde hingegen auch das Stellen von Äquivalenzfragen zulassen. Das heißt, der Lernalgorithmus darf dem Lehrer (wiederholt) einen beliebigen regulären Ausdruck vorlegen. Solange die dadurch beschriebene Hypothese das Zielkonzept noch nicht korrekt beschreibt, ermittelt der Lehrer ein entsprechendes Gegenbeispiel und gibt dieses dem Lernalgorithmus als Antwort auf seine Frage zurück. Der Lernvorgang gelangt natürlich spätestens dann zu einem erfolgreichen Ende, sowie der Lernalgorithmus eine Hypothese als Äquivalenzfrage vorlegt, die mit „JA“ beantwortet wird, und folglich mit dem Zielkonzept exakt übereinstimmt.

VEREINBARUNG

Formal wird ein **Lehrer** für ein Zielkonzept $\hat{C}$ durch eine Funktion beschrieben, die in Abhängigkeit von der momentanen und den zuvor gestellten Fragen eine Antwort auf die aktuelle Frage erteilt, bzw. ein Gegenbeispiel liefert.

Man beachte, daß es zu einem festen Zielkonzept $\hat{C}$ unter einem Protokoll α – falls α nicht ausschließlich Mem-Fragen zuläßt – im allgemeinen mehrere verschiedene Lehrer geben kann. Das liegt daran, daß dem Lehrer bei der Wahl seiner Gegenbeispiele in keiner Weise eine Einschränkung auferlegt wird. Gewissermaßen ist der Lehrer sogar mit einem „Gedächtnis“ ausgestattet, das heißt, er kann (sofern es sich nicht um eine Mem-Frage handelt und auch nicht nur ein einziges Gegenbeispiel zur Auswahl steht) auf dieselbe Frage mit unterschiedlichen Gegenbeispielen antworten, je nachdem welche Fragen ihm bis zu diesem Zeitpunkt bereits gestellt worden sind.

Als Maß für die Effizienz eines Lernalgorithmus' wird in erster Linie sein asymptotisches Laufzeitverhalten angesehen. Daneben werden aber auch häufig andere Ressourcen wie zum Beispiel der Bedarf an Speicherplatz oder die Anzahl der gestellten Fragen als Effizienzkriterien in Betracht gezogen. Während bei Algorithmen zur Lösung von Entscheidungs- oder (funktionalen) Berechnungsproblemen der Ressourcenverbrauch üblicherweise in Abhängigkeit von der Länge der Eingabe gemessen wird, tritt an diese Stelle bei Lernalgorithmen in erster Linie die Größe des jeweiligen Zielkonzeptes. Unter der Größe eines Konzeptes C ist hierbei die Länge einer kürzesten Beschreibung für C zu verstehen. Um einem Lernalgorithmus die Analyse der erhaltenen Gegenbeispiele zu ermöglichen, fließt in die Laufzeitbetrachtung darüberhinaus die Länge der durch den Lehrer erteilten Gegenbeispiele mit ein.

Definition 6.2 *Ein Lern-Algorithmus M ist* polynomial zeitbeschränkt*, falls für ein Polynom t die Anzahl der von M ausgeführten Rechenschritte unter jedem Lehrer höchstens gleich $t(n+m)$ ist. Dabei ist n die Länge einer kürzesten Repräsentation* *für das jeweilige Zielkonzept, und m ist die Maximallänge aller durch den Lehrer bereitgestellten Gegenbeispiele.*

Eine Konzeptklasse $\mathcal{C}$, dargestellt durch eine Repräsentationssprache R, ist (in deterministischer Polynomialzeit) exakt α-erlernbar*, falls es einen polynomial zeitbeschränkten Lern-Algorithmus gibt, der für jedes Zielkonzept $\hat{C} \in \mathcal{C}$ und bei jedem Lehrer für $\hat{C}$ eine Repräsentation $r \in R$ mit $\Phi(r) = \hat{C}$ ausgibt. Dabei dürfen nur Fragen eines im Protokoll α enthaltenen Typs gestellt werden.*

Je nach Wahl der Gegenbeispiele durch den Lehrer kann die durch einen Lernalgorithmus beanspruchte Rechenzeit erheblich variieren. Gemäß obiger Definition der Erlernbarkeit muß der Lernalgorithmus jedoch für jedes Zielkonzept $\hat{C}$ der Größe n zumindest dann nach einer in n polynomial beschränkten Anzahl von Rechenschritten eine Beschreibung von $\hat{C}$ ausgeben, wenn die Antworten des Lehrers mit $\hat{C}$ konsistent sind und er nur Gegenbeispiele polynomialer Länge in n zurückgibt.

Gelegentlich werden wir auch sogenannte **schwache** Versionen der oben beschriebenen Fragetypen betrachten. Auf diese Fragen gibt der Lehrer kein Gegenbeispiel zurück, sondern beantwortet sie lediglich mit „JA“oder „NEIN“.

Das hier beschriebene Modell gehört zur Kategorie der sogenannten *echten Lernmodelle* (engl. proper learning). Das heißt, es dürfen neben den Zugehörigkeitsfragen nur solche Repräsentationen r als Hypothese formuliert werden, die Konzepte aus der zu erlernenden Klasse $\mathcal{C}$ darstellen. Demgegenüber wird in den als *unecht* bezeichneten Lernmodellen (engl. improper learning) außer einer Konzeptklasse $\mathcal{C}$ zusätzlich ein *Hypothesenraum* $\mathcal{H}$ und eine entsprechende Repräsentationssprache $R_{\mathcal{H}}$ explizit vorgegeben, wobei der Hypothesenraum $\mathcal{H}$ die Konzeptklasse $\mathcal{C}$ im allgemeinen echt umfaßt. Zur Formulierung der Ausgabe und der Hypothesen darf der Lernalgorithmus dann beliebige Repräsentationen r aus $R_{\mathcal{H}}$ verwenden, was unter Umständen eine erhebliche Vereinfachung der Lernaufgabe bedeuten kann.

Beispielsweise könnte man das Erlernen von regulären Ausdrücken für einen Lernalgorithmus dadurch erleichtern, daß man ihm erlaubt, beliebige kontextfreie Grammatiken (anstelle von regulären Ausdrücken) zur Formulierung von Hypothesen bei Equ-, Sub- oder Sup-Fragen und zur Repräsentation des Zielkonzeptes bei der Ausgabe zu verwenden.

Eine andere Möglichkeit, einem Lernalgorithmus die Lösung eines Lernproblems zu erleichtern, besteht darin, zusätzlich Fragen an eine Orakelsprache aus einer bestimmten Komplexitätsklasse (wie z.B. $\mathcal{NP}$) zuzulassen. Interessanterweise lassen sich auf diese Weise offene Fragen der Lern- bzw. Nichterlernbarkeit von bestimmten Konzeptklassen mit offenen Fragen der Komplexitätstheorie koppeln. Wir werden in Abschnitt 6.6 eine solche Querbeziehung zwischen der Komplexitätstheorie und der Theorie des algorithmischen Lernens aufzeigen.

Abhängig von der zu erlernenden Konzeptklasse ist es manchmal sogar möglich, die Antwort auf eine beliebige Frage an ein Orakel (wie zum Beispiel an SAT) durch geeignete Fragen an den Lehrer herauszufinden. Dieses Phänomen tritt beispielsweise beim Erlernen von Booleschen Schaltkreisen auf.

Beispiel 6 *Es gibt mehrere Möglichkeiten, einem Booleschen Schaltkreis c ein Konzept $C \subseteq \{0,1\}^*$ zuzuordnen. In der Literatur werden hauptsächlich die beiden folgenden betrachtet.*

1. *Ein Schaltkreis c mit n Eingängen repräsentiert die Menge C aller Binärstrings der Länge n, bei deren Eingabe er den Wert 1 liefert, das heißt,*

$$C = \{x \in \{0,1\}^n \mid c(x) = 1\}.$$

2. *Falls auch Binärstrings x der Länge ungleich n durch einen Schaltkreis c mit n Eingängen klassifiziert werden sollen, so wird meistens die folgende Vereinbarung getroffen. Falls die Länge von x größer als n ist, so verwendet man lediglich die ersten n Bits von x als Eingabe für c. Andernfalls wird x durch c dann positiv klassifiziert, falls c allen Erweiterungen von x zu einem Bitvektor der Länge n den Wert 1 zuteilt.*

Interessanterweise können nun beim exakten Erlernen von Booleschen Schaltkreisen alle hier betrachteten Fragetypen durch schwache Teil- und Obermengenfragen simuliert werden (vgl. [BCG$^+$95]): Es ist klar daß Mem- und Equ-Fragen durch Sub- und Sup-Fragen ersetzt werden können. Es genügt also zu zeigen, wie Sub-Fragen durch schwache Sub-Fragen simuliert werden können (die Simulation von Sup-Fragen durch schwache Sup-Fragen verläuft analog). Wird eine Sub-Frage mit „NEIN" beantwortet, dann ist es möglich, durch binäre Suche (genauer: durch wiederholtes Stellen von Sub-Fragen der Form $c'(x) = c(x) \wedge (x \leq y)$) die lexikographisch kleinste Eingabe y zu bestimmen, die zwar durch das von c repräsentierte Konzept, nicht aber durch das Zielkonzept positiv klassifiziert wird. Auf diese Art kann offenbar ein geeignetes Gegenbeispiel bestimmt werden.

Tatsächlich lassen sich auch Fragen an ein $\mathcal{NP}$ Orakel durch schwache Teil- und Obermengenfragen simulieren. Jede Frage an ein $\mathcal{NP}$ Orakel kann nämlich in Polynomialzeit in einen Booleschen Schaltkreis c transformiert werden, der genau dann allen Eingaben den Wert 0 zuweist, wenn die Antwort auf die $\mathcal{NP}$-Frage „NEIN" lautet. Wird nun das in Beispiel 6 unter *1.* beschriebene Modell verwendet, so genügt es offensichtlich, c als Sub-Frage zu stellen, sofern sichergestellt wird, daß c mehr Eingabegatter hat als der das Zielkonzept darstellende Schaltkreis. Wie in [BCG$^+$95] gezeigt, genügt es unter dem in Beispiel 6 unter *2.* beschriebenen Modell, c als Sub- und den negierten Schaltkreis $\overline{c}$ als Sup-Frage zu stellen, da c genau dann allen Eingaben den Wert 0 zuweist, wenn beide Fragen mit „JA“ beantwortet werden. Offensichtlich gilt dies vollkommen unabhängig davon, welches Zielkonzept vorliegt.

6.4 Ein Vergleich mit Valiants PAC-Lernmodell

Das wahrscheinlich populärste Lernmodell ist das von Valiant in seiner fundamentalen Arbeit [Val84] eingeführte Modell des PAC-Lernens. Die Abkürzung PAC steht hierbei für „probably approximately correct“ (identification under random sampling). Im Unterschied zu dem im vorigen Abschnitt beschriebenen Modell des exakten Lernens durch Fragen, bei dem ein Lernalgorithmus selbständig Fragen generiert und somit in einen interaktiven Kommunikationsprozeß mit dem Lehrer tritt, findet bei einem PAC-Lernprozeß nur eine „einseitige Interaktion“ des Lehrers mit dem Lernalgorithmus statt. Insofern ist das PAC-Lernen mit der auf Gold [Gol67] zurückgehenden Methode der *Induktiven Inferenz* verwandt, die – nebenbei bemerkt – einen wichtigen Grundpfeiler für die Lerntheorie bildet, ähnlich wie die Berechenbarkeitstheorie für die Komplexitätstheorie. Für eine ausführliche Darstellung der Theorie der Induktiven Inferenz sei der interessierte Leser auf die Arbeit [AS83] verwiesen.

Im Modell des PAC-Lernens werden dem Lernalgorithmus eine Reihe von Objekten vorgelegt, die unter einer beliebigen, dem Lernalgorithmus unbekannten Wahrscheinlichkeitsverteilung generiert werden. Zusammen mit diesen Beispielobjekten erhält der Lernalgorithmus jeweils die Information mitgeteilt, ob es sich um ein durch das Zielkonzept positiv oder negativ klassifiziertes Objekt handelt.

Ein PAC-Lernalgorithmus kann also keinerlei Einfluß darauf nehmen, welche Informationen ihm über das Zielkonzept zufließen. Ein weiterer Unterschied zum Modell des exakten Lernens besteht darin, daß ein PAC-Lernalgorithmus das Zielkonzept nicht exakt identifizieren muß, sondern anhand der Lehr-Beispiele lediglich mit einer genügend hohen Wahrscheinlichkeit eine Beschreibung eines Konzeptes auszugeben braucht, das dem Zielkonzept hinreichend ähnlich ist.

VEREINBARUNG

Ein **Lehr-Beispiel** ist ein Paar $\langle x, b\rangle$, bestehend aus einem Objekt $x \in \{0,1\}^*$ und einem Bit b, das die Information beinhaltet, ob x durch das Zielkonzept positiv ($b = 1$) oder negativ ($b = 0$) klassifiziert wird.
Eine Folge $s = \langle x_1, b_1\rangle \# \cdots \# \langle x_k, b_k\rangle$ von solchen Lehr-Beispielen bezeichnen wir kurz als **Beispielfolge** (in der englischen Literatur meist mit dem Begriff „training sample" oder „classified sample" bezeichnet).

Definition 6.3 *Ein* **PAC-Lernalgorithmus** *ist eine Orakel-Turingmaschine M, die in einen speziellen Fragezustand übergehen kann. Geschieht dies, so erscheint in einem Rechenschritt ein Lehr-Beispiel $\langle x, b\rangle$ auf dem Orakelband.*

Die Lehr-Beispiele werden im PAC-Modell stochastisch unabhängig voneinander gemäß einer beliebigen Wahrscheinlichkeitsverteilung P auf $\{0,1\}^*$ generiert. $P(x)$ ist also die Wahrscheinlichkeit dafür, daß das Objekt x als Komponente eines Lehr-Beispiels gewählt wird. In etwas suggestiverer Ausdrucksweise ist $P(x)$ die Wahrscheinlichkeit, daß das Vorliegen bzw. Nichtvorliegen der mit dem Zielkonzept verbundenen Eigenschaft anhand des Objektes x beobachtet werden kann. Obwohl M deterministisch[2] ist, ist die Ausgabe eines PAC-Lernalgorithmus' M eine Zufallsvariable, da sie im allgemeinen von den zufällig gewählten Lehr-Beispielen abhängt, die manchmal mehr, manchmal aber auch weniger repräsentative Objekte des Zielkonzepts bzw. des dazu komplementären Konzeptes darstellen können. Intuitiv gesprochen ist jedoch die Wahrscheinlichkeit für Objekte mit untypischen Eigenschaften, die den Lernalgorithmus in die Irre führen könnten, relativ gering, so daß es nicht unrealistisch erscheint, von einem guten Lernalgorithmus zu verlangen, daß er mit hoher Wahrscheinlichkeit eine Beschreibung eines solchen Konzeptes ausgibt, das dem Zielkonzept hinreichend ähnlich ist.

Die Ähnlichkeit des durch einen PAC-Lernalgorithmus ausgegebenen Konzeptes mit dem Zielkonzept wird hierbei ebenfalls auf der Grundlage der vorgegebenen Verteilung P gemessen, anhand derer die Lehr-Beispiele generiert werden. Dies ist plausibel, da eine fehlerhafte Klassifikation bei einem Objekt, das man relativ häufig zu Gesicht bekommt, schwerer wiegen sollte als bei einem Objekt, das relativ selten zu sehen ist.

Definition 6.4 *Zwei Konzepte C und C'* unterscheiden sich um höchstens ε, *falls ε eine obere Schranke für die Wahrscheinlichkeit $P(C \Delta C')$ der symmetrischen Differenz der beiden Konzepte C und C' ist.*

[2] In der Literatur werden mitunter auch probabilistische PAC-Lernalgorithmen betrachtet.

Mit anderen Worten, C und C' unterscheiden sich genau dann um höchstens ε, wenn ein zufällig gemäß P generiertes Lehr-Beispiel x von den beiden Konzepten mit einer Wahrscheinlichkeit von höchstens ε unterschiedlich klassifiziert wird.

Definition 6.5 *Eine Konzeptklasse $\mathcal{C}$, dargestellt durch eine Repräsentationssprache R, heißt* (in deterministischer Polynomialzeit) PAC-erlernbar, *falls es einen PAC-Lernalgorithmus gibt, der unter jeder auf $\{0,1\}^*$ definierten Wahrscheinlichkeitsverteilung P und für jedes Zielkonzept $\hat{C} \in \mathcal{C}$ bei Eingabe von $\varepsilon, \delta > 0$ nach spätestens $t(n+m+1/\varepsilon+1/\delta)$ Rechenschritten mit Wahrscheinlichkeit mindestens $1-\delta$ eine Repräsentation $r \in R$ für eine Hypothese C_r ausgibt, die sich von $\hat{C}$ um höchstens ε unterscheidet.*

Hierbei ist t ein Polynom, n die Länge einer kürzesten Repräsentation für $\hat{C}$, und m ist die Maximallänge der unabhängig unter P generierten und dem PAC-Lernalgorithmus übergebenen Lehr-Beispiele.

Ein PAC-Lernalgorithmus muß also die polynomiale Laufzeitschranke unabhängig davon einhalten, unter welcher Wahrscheinlichkeitsverteilung die Lehr-Beispiele generiert werden.

Interessanterweise ist jede durch Äquivalenzfragen exakt erlernbare Konzeptklasse auch schon PAC-erlernbar [Ang88]. Der Grund hierfür ist, daß ein PAC-Lernalgorithmus die Beantwortung jeder von einem exakten Lernalgorithmus in Form einer Hypothese C gestellten Äquivalenzfrage dadurch simulieren kann, daß er sich eine genügend große Anzahl von zufällig generierten Lehr-Beispielen geben läßt, und jedes einzelne daraufhin testet, ob es ein Gegenbeispiel für die Hypothese C darstellt. Ist dies für mindestens eines der getesteten Lehr-Beispiele der Fall, so hat der PAC-Lernalgorithmus ein geeignetes Gegenbeispiel auf die momentane Äquivalenzfrage gefunden und kann die Simulation des exakten Lernalgorithmus' fortsetzen. Werden auf diese Art im Laufe der Simulation Gegenbeispiele für alle der durch den exakten Lernalgorithmus aufgestellten Hypothesen erzeugt, so ist der PAC-Lernalgorithmus am Ende der Simulation sogar in der Lage, das Zielkonzept $\hat{C}$ exakt zu identifizieren und eine entsprechende Repräsentation auszugeben. Stellt der exakte Lernalgorithmus dagegen eine Hypothese C auf, so daß alle für den Test dieser Hypothese generierten Lehr-Beispiele durch C und das Zielkonzept $\hat{C}$ übereinstimmend klassifiziert werden, so bricht der PAC-Lernalgorithmus die Simulation einfach ab und gibt die von dem exakten Lernalgorithmus auf das Orakelband geschriebene Repräsentation dieser Hypothese aus.

Man kann sich leicht davon überzeugen, daß die Anzahl der zum Test einer Äquivalenzfrage benötigten Lehr-Beispiele in Abhängigkeit von ε und δ nicht allzu groß gewählt werden muß, um die Bedingungen der PAC-Erlernbarkeit gemäß

Definition 6.5 zu erfüllen: Angenommen der PAC-Lernalgorithmus verwendet für den Test der i-ten Äquivalenzfrage eine Folge von maximal $q_i = \lceil \frac{i - \log\delta}{\varepsilon} \rceil$ Lehr-Beispielen. Diese Anzahl ist offensichtlich polynomial in i, $1/\varepsilon$ und sogar $\log(1/\delta)$. Seien $C_1, \ldots, C_k$ die der Reihe nach im Laufe der Simulation von dem exakten Lernalgorithmus aufgestellten Hypothesen, und sei

$$I = \{1 \leq i \leq k \mid P(\hat{C} \Delta C_i) > \varepsilon\}$$

die Menge der Indizes derjenigen Hypothesen C_i, die vom Zielkonzept $\hat{C}$ um mehr als ε abweichen. Offenbar genügt es zu zeigen, daß mit hoher Wahrscheinlichkeit für jede dieser Hypothesen ein Gegenbeispiel gefunden wird.

Wir schätzen nun für jedes feste $i \in I$ die Wahrscheinlichkeit für das Ereignis ab, daß keines der für den Ähnlichkeitstest zwischen der Hypothese C_i und dem Zielkonzept $\hat{C}$ generierten Lehr-Beispiele $x_{i,1}, \ldots, x_{i,q_i}$ einen Unterschied zwischen den beiden Konzepten aufdeckt. Wegen $\log(1-\varepsilon) < -\varepsilon$ ist

$$\begin{aligned}
\Pr[\forall j = 1, \ldots, q_i : x_{i,j} \notin \hat{C} \Delta C_i\,] &= \prod_{j=1}^{q_i} \Pr[\, x_{i,j} \notin \hat{C} \Delta C_i\,] \\
&< (1-\varepsilon)^{q_i} \\
&\leq 2^{q_i \log(1-\varepsilon)} \\
&\leq 2^{(i - \log\delta) \cdot \log(1-\varepsilon)/\varepsilon} \\
&\leq \delta 2^{-i}.
\end{aligned}$$

Folglich gibt der beschriebene PAC-Lernalgorithmus mit einer Wahrscheinlichkeit von höchstens

$$\sum_{i \in I} \delta 2^{-i} \;<\; \delta$$

ein Konzept C_i aus, das vom Zielkonzept um mehr als ε abweicht.

Tatsächlich sind umgekehrt viele Konzeptklassen, die PAC-erlernbar sind, auch exakt erlernbar mit Äquivalenzfragen. Unter der (plausiblen) Voraussetzung, daß Einwegfunktionen existieren, gibt es jedoch PAC-erlernbare Konzeptklassen, die nicht exakt (Equ)-erlernbar sind (siehe [Blu94]). Unter dieser Voraussetzung bilden also alle exakt(Equ)-erlernbaren Lernprobleme eine echte Teilklasse der Menge aller Lernprobleme, die PAC-erlernbar sind.

6.5 Die Strategie des Mehrheitsvotums

Bei vielen Lernproblemen liegt das Universum U in parametrisierter Form vor. Das heißt, U ist in eine Folge $U = \bigcup_{n\geq 1} U_n$ von endlichen Mengen U_n aufgesplittet, die etwa aus allen Objekten der Größe n bestehen. In diesem Fall ist dann auch die Konzeptklasse $\mathcal{C}$ in eine Folge von Klassen $\mathcal{C}_n$ unterteilt, wobei jede einzelne Klasse $\mathcal{C}_n$ typischerweise nur Konzepte $C \subseteq U_n$ enthält.

Beispiel 7 *Genau wie Boolesche Schaltkreise (siehe Beispiel 6) können Boolesche Funktionen $f : \{0,1\}^n \to \{0,1\}$ auf verschiedene Arten in Sprachkonzepte transformiert werden. Eine naheliegende Möglichkeit besteht darin, für U_n die Menge $\{0,1\}^n$ aller Binärstrings der Länge n zu wählen, und f mit der Sprache $\{x \in \{0,1\}^n \mid f(x) = 1\}$ zu identifizieren. In diesem Fall ist jedes $\mathcal{C}_n$ eine geeignete Teilmenge der Potenzmenge von $\{0,1\}^n$. An dieser Stelle seien beispielhaft nur die folgenden zwei Konzeptklassen von Booleschen Funktionen erwähnt.*

- *Die Klasse aller monotonen Booleschen Funktionen führt etwa auf die Konzeptklasse $\mathcal{M} = \bigcup_{n\geq 1} \mathcal{M}_n$, wobei $\mathcal{M}_n$ aus allen Konzepten $C \subseteq \{0,1\}^n$ besteht, deren charakteristische Funktion auf $\{0,1\}^n$ durch einen monotonen Schaltkreis berechnet werden kann.*
- *Auf eine weitere interessante Konzeptklasse führt die Klasse aller Booleschen Funktionen f, deren Schaltkreiskomplexität $C(f)$ polynomial beschränkt ist.*

Sei $\hat{C} \subseteq U_n$ ein vorgegebenes Zielkonzept. Wie beim Erlernen von parametrisierten Konzeptklassen üblich, vereinbaren wir, daß dem Lernalgorithmus der zum Zielkonzept gehörige Parameterwert n über die Eingabe mitgeteilt wird. Da $\mathcal{C}_n$ endlich ist, besteht für den Lernalgorithmus grundsätzlich die Möglichkeit, den Hypothesenraum $\mathcal{C}_n$ vollständig nach dem Zielkonzept $\hat{C}$ zu durchsuchen. Unter Verwendung von (schwachen) Äquivalenzfragen ist es beispielsweise möglich, durch jede Frage, die nicht mit „JA" beantwortet wird, mindestens eine Hypothese aus $\mathcal{C}_n$ zu eliminieren, so daß spätestens nach einer Anzahl von $\|\mathcal{C}_n\|$ Fragen das Zielkonzept feststeht. Diese Vorgehensweise ist allerdings nur dann praktikabel, wenn die Anzahl der in $\|\mathcal{C}_n\|$ enthaltenen Konzepte der Größe $\leq m$ polynomial in m ist, was jedoch in den allermeisten Fällen nicht zutrifft.

Die auf Angluin [Ang88] und noch weiter auf [BF72] zurückgehende und unter dem Begriff „Strategie des Mehrheitsvotums" bekannte Vorgehensweise beruht auf der folgenden Idee. Versuche durch geschickte Wahl der nächsten Äquivalenzfrage nicht nur eine, sondern mindestens die Hälfte aller noch als Kandidaten für das Zielkonzept in Frage kommenden Hypothesen zu eliminieren. Dies bedeutet, daß der Lernalgorithmus für die nächste Äquivalenzfrage jeweils eine Hypothese C mit der folgenden Eigenschaft wählen muß:

> Jedes Objekt $x \in U_n$ wird durch das Konzept C im Einklang mit dem Mehrheitsvotum der noch verbliebenen Hypothesen klassifiziert.

Stellt nämlich der Lernalgorithmus eine Hypothese C mit dieser Eigenschaft als Äquivalenzfrage, und stimmt diese nicht mit dem Zielkonzept $\hat{C}$ überein, so werden mindestens die Hälfte der noch verbliebenen Hypothesen eliminiert, und zwar vollkommen unabhängig davon, welches Gegenbeispiel x der Lehrer für seine Antwort auf diese Hypothese auswählt.

Die durch einen Lehrer an einen Lernalgorithmus im Verlauf eines Lernprozesses übermittelte Information kann vollständig durch eine Beispielfolge $s = \langle x_1, b_1\rangle\# \cdots \#\langle x_k, b_k\rangle$ beschrieben werden. Jedes einzelne Wort x_i entspricht hierbei einem Gegenbeispiel, das der Lehrer auf eine vorausgehende Äquivalenzfrage geliefert hat, während in dem dazugehörigen Bit b_i die Information festgehalten wird, ob das Zielkonzept das Gegenbeispiel x_i positiv ($b = 1$) oder negativ ($b = 0$) klassifiziert. Das Bit b_i kann der Lernalgorithmus aufgrund der in Abschnitt 6.2 vereinbarten Uniformitätsbedingung an die verwendete Repräsentationssprache in Polynomialzeit herausfinden, da x_i ein Gegenbeispiel auf eine entsprechende Hypothese H_i bildet und folglich von H_i und dem Zielkonzept unterschiedlich klassifiziert wird.

Durch eine Beispielfolge s werden all diejenigen Konzepte als Kandidaten für das Zielkonzept ausgeschlossen, die durch mindestens eines der in s festgehaltenen Gegenbeispiele widerlegt werden. Es kommen also nur noch solche Konzepte als Kandidaten für das Zielkonzept in Betracht, die alle bisher erhaltenen Gegenbeispiele im Einklang mit dem Zielkonzept klassifizieren. Dies führt auf den wichtigen Begriff der Konsistenz eines Konzeptes C mit einer Beispielfolge s.

Definition 6.6 *Sei $s = \langle x_1, b_1\rangle\# \cdots \#\langle x_k, b_k\rangle$ eine Folge von Lehr-Beispielen. Ein Konzept C heißt mit s* konsistent*, falls für alle $i = 1, \ldots, k$ die Äquivalenz*

$$x_i \in C \Leftrightarrow b_i = 1$$

gilt. Das heißt, jedes der in s vorkommenden Beispiele x_i wird durch C entsprechend dem Bit b_i klassifiziert.

input n
$s := \lambda$
loop
  bestimme eine Repräsentation r für eine Hypothese C,
  die für alle $x \in U_n$ die folgende Eigenschaft hat:

$$x \in C \quad \Leftrightarrow \quad x \text{ ist in mindestens der Hälfte aller Konzepte in } \mathcal{C}_n \text{ enthalten, die mit } s \text{ konsistent sind,} \tag{6.1}$$

  und stelle r als Äquivalenzfrage
  if Antwort besteht in der Angabe eines Gegenbeispiels x **then**
    $s := s\#\langle x, b\rangle$, wobei $b = 0$ ist, falls $x \in C$, und $b = 1$ sonst.
  else exit(**loop**)
end loop
output r

Abbildung 6.1: Ein exemplarischer Lernalgorithmus nach der Strategie des Mehrheitsvotums.

Der in Abb. 6.1 dargestellte Pseudo-Code zeigt exemplarisch die der Strategie des Mehrheitsvotums zugrundeliegende Vorgehensweise auf. Beginnend mit der leeren Folge $s = \lambda$ baut der Lernalgorithmus in mehreren Iterationen aus den erhaltenen Gegenbeispielen eine mit dem Zielkonzept konsistente Beispielfolge s auf. In jedem Schleifendurchlauf stellt der Lernalgorithmus hierzu eine Äquivalenzfrage, die aus der Repräsentation eines Konzeptes C besteht, das jedes Objekt x im Einklang mit dem Mehrheitsvotum aller Konzepte klassifiziert, die mit der bisher konstruierten Beispielfolge s konsistent sind.

Offensichtlich steht nach spätestens $\log(\|\mathcal{C}_n\|)$ vielen Fragerunden das Zielkonzept eindeutig fest. Das heißt, die Strategie des Mehrheitsvotums ist im Prinzip auch dann noch effizient, wenn die Anzahl aller in $\mathcal{C}_n$ enthaltenen Konzepte exponentiell in der dem Lernalgorithmus zur Verfügung stehenden Zeit ist. Durch eine leichte Modifikation der in Abb. 6.1 dargestellten Vorgehensweise ist es sogar möglich, das Zielkonzept in im Prinzip beliebig großen Konzeptklassen $\mathcal{C}_n$ ausfindig zu machen. Hierzu durchsucht der Lernalgorithmus schrittweise für $m = 1, 2$, usw. nacheinander die Teilklassen $\mathcal{C}_{n,m}$ von $\mathcal{C}_n$, die alle Konzepte der Größe m enthalten, bis das Zielkonzept gefunden ist (auf diese Weise werden wir im Beweis des Hauptresultats dieses Kapitels vorgehen; siehe Abschnitt 6.6).

Die Hauptschwierigkeit bei der Implementation eines Algorithmus, der die Strategie des Mehrheitsvotums verfolgt, besteht natürlich darin, eine Repräsentation für eine

Hypothese mit der in Abb. 6.1 beschriebenen Eigenschaft (6.1) zu ermitteln. In vielen Fällen wird es nicht einmal eine Hypothese in $\mathcal{C}_n$ geben, die das Mehrheitsvotum aller Konzepte zum Ausdruck bringt, die mit der bisher konstruierten Beispielfolge s konsistent sind. (Auch wenn für eine gegebene Beispielfolge s eine solche Hypothese H in $\mathcal{C}_n$ (oder – im Modell des unechten Lernens – in dem zugehörigen Hypothesenraum $\mathcal{H}_n$) existiert, ist damit zu rechnen, daß die dem Lernalgorithmus zur Verfügung stehende Zeit nicht ausreicht, eine Repräsentation für eine solche Hypothese zu finden.)

Eine Möglichkeit, dieses Problem in den Griff zu bekommen, besteht darin, die in Abb. 6.1 verlangte Eigenschaft (6.1) ein wenig abzuschwächen. Anstatt nämlich auf einer Hypothese H zu bestehen, die jedes Objekt x in U_n *exakt* gemäß der Mehrheit der verbliebenen Hypothesen klassifiziert, ist es schon vollkommen ausreichend, daß H die Eigenschaft hat, für kein x nur eine sehr schwache Minderheit der verbliebenen Hypothesen auf ihrer Seite zu haben. Wird nämlich auf eine solche Hypothese H eine negative Antwort in Form eines Gegenbeispiels x erteilt, so reduziert sich durch Aufnahme des daraus resultierenden Lehr-Beispiels in die Folge s die Menge der verbliebenen Hypothesen immerhin um die bezüglich x mit H konforme Minderheit. Indem der Lernalgorithmus eine entsprechend höhere Anzahl von Fragerunden durchführt, kann er daher immer noch sicherstellen, daß alle falschen Hypothesen in der zur Verfügung stehenden Zeit eliminiert werden.

Tatsächlich kann ein Konzept K, das ein derartiges „approximatives Mehrheitsvotum“ zum Ausdruck bringt, aus relativ wenigen Konzepten durch Mehrheitsbildung aufgebaut werden. Den Schlüssel zur Konstruktion eines solchen Konzeptes liefert das folgende auf Gesetzen der Statistik beruhende Verfahren:

> Um in einer bestimmten Frage die Mehrheitsverhältnisse in einer unüberschaubaren Gesamtheit näherungsweise zu bestimmen, ermittle man die Meinung einer kleinen, aber repräsentativen Gruppe von Individuen, und ziehe aus dem Ergebnis dieser „Strichprobe“ Schlüsse auf die Gesamtheit.

In der Tat genügt für unsere Zwecke schon ein Bit an Information aus einer solchen Strichprobe, nämlich das Mehrheitsvotum der befragten Individuen.

VEREINBARUNG

Für eine beliebige endliche Menge $\mathcal{K} = \{K_1, \ldots, K_l\}$ von Konzepten bezeichnen wir mit $maj(\mathcal{K}) = maj(K_1, \ldots, K_{2l})$ das Konzept, das jedes Objekt x in U entsprechend der Mehrheit der Konzepte $K_1, \ldots, K_l$ klassifiziert,

$$x \in maj(\mathcal{K}) \quad =_{def} \quad \|\{1 \leq i \leq l \mid x \in K_i\}\| \geq l/2.$$

Wir nennen das Konzept $maj(\mathcal{K})$ auch das **Mehrheitsvotum** aus $\mathcal{K}$.

Lemma 6.7 *Sei $\mathcal{K}$ eine beliebige endliche Menge von Konzepten und sei $u \in U$ ein festes Objekt. Angenommen, es werden $2l$ Konzepte $K_1, \ldots, K_{2l}$ unabhängig unter Gleichverteilung aus $\mathcal{K}$ ausgewählt. Dann ist die Wahrscheinlichkeit für das Ereignis E_u, daß sich Mehrheitsvotum $\mathrm{maj}(K_1, \ldots, K_{2l})$ in der Klassifikation des Objektes u mit mindestens einem Viertel aller in $\mathcal{K}$ enthaltenen Konzepte übereinstimmt, größer als $1 - (3/4)^l$.*

Beweis. Für $i = 1, \ldots, 2l$ betrachte die Zufallsvariable

$$X_i \quad = \quad \begin{cases} 1, & u \in K_i \Delta maj(\mathcal{K}), \\ 0, & \text{sonst}, \end{cases}$$

die anzeigt, ob das zufällig gewählte Konzept K_i das Objekt u anders als die überwiegende Mehrheit aller in $\mathcal{K}$ enthaltenen Konzepte klassifiziert wird oder nicht. Offensichtlich ist $p =_{def} E(X_i) \leq 1/2$. Gilt $p \geq 1/4$, so tritt das Ereignis E_u mit Sicherheit ein, gleichgültig wie U durch das Mehrheitsvotum der Konzepte $K_1, \ldots, K_{2l}$ klassifiziert wird. Wir können also $p < 1/4$ voraussetzen. Da das Objekt u nur dann durch die beiden Konzepte $maj(K_1, \ldots, K_{2l})$ und $maj(\mathcal{K})$ unterschiedlich klassifiziert wird, wenn $\sum_{i=1}^{2l} X_i \geq l$ ist, erhalten wir unter Verwendung von Lemma 2.1 die folgende Wahrscheinlichkeitsabschätzung dafür, daß das Ereignis E_u nicht eintritt:

$$\Pr\Big[u \in maj(K_1, \ldots, K_{2l}) \Delta maj(\mathcal{K})\Big] \quad \leq \quad \Pr\left[\sum_{i=1}^{2l} X_i \geq l\right] \quad < \quad (3/4)^l.$$

∎

Damit ergibt sich nun unmittelbar das folgende Resultat, das die Existenz einer Hypothese H sicherstellt, die in der Klassifizierung eines jeden Objektes $u \in U$ zumindest mit einem konstanten Anteil der noch vorhandenen Hypothesen übereinstimmt (vgl. [BCKT94]).

Lemma 6.8 *Sei $\mathcal{K}$ eine beliebige Konzeptklasse über einem Universum U und sei $U' \subseteq U$ mit $\|U'\| \leq 2^l$ für ein $l \geq 0$. Dann ist die Wahrscheinlichkeit dafür, daß es mindestens ein $u \in U'$ gibt, für das mehr als $3/4$ aller in $\mathcal{K}$ enthaltenen Konzepte nicht mit dem Mehrheitsvotum $\mathrm{maj}(K_1, \ldots, K_{6l+2k})$ von $6l + 2k$ unabhängig unter Gleichverteilung aus $\mathcal{K}$ gewählten Konzepten übereinstimmen, kleiner als $(3/4)^k$.*

Beweis. Nach Lemma 6.7 ist für ein festes $u \in U$ die Wahrscheinlichkeit, daß mehr als $3/4$ aller in $\mathcal{K}$ enthaltenen Konzepte nicht mit dem Mehrheitsvotum $maj(K_1, \ldots, K_{6l+2k})$ übereinstimmen, kleiner als $(3/4)^{3l+k}$. Folglich ist die Wahrscheinlichkeit, daß ein $u \in U'$ existiert, für das dieses Ereignis eintritt, höchstens

$$2^l \cdot (3/4)^{3l+k} \;<\; \left(2(3/4)^3\right)^l \cdot (3/4)^k \;<\; (3/4)^k.$$

■

6.6 Exaktes Lernen von Schaltkreisen

Bis heute ist es ein wichtiges offenes Problem in der Theorie des algorithmischen Lernens, ob Boolesche Schaltkreise durch Fragen exakt erlernbar sind. Erst vor kurzem konnte gezeigt werden, daß Boolesche Schaltkreise mithilfe von Äquivalenzfragen unter zusätzlicher Verwendung eines Σ_3^p-Orakels in Polynomialzeit exakt erlernbar sind [BCG+95]. Eine interessante Konsequenz hieraus ist, daß ein Beweis für die Nichterlernbarkeit von Booleschen Schaltkreisen durch Äquivalenzfragen mindestens so schwer zu erbringen ist, wie ein Beweis für $\mathcal{P} \neq \mathcal{NP}$.

Wir verbessern in diesem Abschnitt obiges Resultat, indem wir das Σ_3^p-Orakel durch ein Σ_2^p-Orakel ersetzen. Eine weitere Verbesserung von Σ_2^p auf $\mathcal{NP}$ (oder, gleichbedeutend damit, auf Δ_2^{P}) würde übrigens bedeuten, daß Boolesche Schaltkreise mithilfe von schwachen Teil- und Obermengenfragen ohne Verwendung eines zusätzlichen Orakels in Polynomialzeit exakt erlernbar sind. Dies folgt unmittelbar aus der in [BCG+95] beobachteten Tatsache, daß beim exakten Erlernen von Booleschen Schaltkreisen sowohl Äquivalenzfragen als auch Fragen an ein $\mathcal{NP}$ Orakel durch schwache Teil- und Obermengenfragen simuliert werden können (siehe auch Seite 114).

Vereinbarung

Sei $s = \langle x_1, b_1\rangle \# \cdots \# \langle x_k, b_k\rangle$ eine Beispielfolge mit $x_1, \ldots, x_k \in \{0,1\}^n$.

CIR(n) bezeichne die Menge der Kodierungen aller Schaltkreise mit n Eingängen,

CIR(n, m) bezeichne die Menge der Kodierungen aller Schaltkreise mit n Eingängen und m Gattern, und

CIR(n, m, s) bezeichne schließlich die Menge

$$\{c \in \mathit{CIR}(n,m) \mid \text{ für alle } i = 1, \ldots, k \text{ gilt } c(x_i) = b_i\}$$

aller Schaltkreise in $\mathit{CIR}(n,m)$, die mit der Beispielfolge s konsistent sind.

Wir setzen ein geeignetes Kodierungsverfahren für Schaltkreise durch Binärstrings voraus. Es ist nicht schwer (nur etwas mühsam), ein Kodierungsverfahren anzugeben, das jeden Schaltkreis der Größe m durch einen Binärstring der Länge $O(m \log m)$ kodiert. Für unsere Betrachtungen ist es allerdings nur von Bedeutung, daß die Länge der Kodierung polynomial in der Schaltkreisgröße ist, und daß aus einer Kodierung der zugehörige Schaltkreis in Polynomialzeit rekonstruiert werden kann. Sei also p ein festes Polynom, das die Länge der Kodierung eines Schaltkreises c angibt, d.h. wir setzen voraus, daß ein Schaltkreis mit m Gattern (inklusive Eingabegatter) durch einen Binärstring der Länge $p(m)$ kodiert wird. Der Einfachheit halber werden wir im folgenden nicht zwischen einer Kodierung und dem zugehörigen Schaltkreis unterscheiden, sondern für jedes $c \in \mathit{CIR}(n)$ einfach von dem „Schaltkreis“ c sprechen.

Satz 6.9 *Die Klasse der Booleschen Schaltkreise ist exakt erlernbar mit Äquivalenzfragen und einem Σ_2^p-Orakel.*

Beweis. Wir verwenden die approximative Form der im vorigen Abschnitt diskutierten Strategie des Mehrheitsvotums. Der in Abb. 6.2 dargestellte Lernalgorithmus M durchsucht die Mengen $\mathit{CIR}(n,m)$ für $m = n, n+1, n+2$ usw. nach einem Schaltkreis, der das unbekannte Zielkonzept repräsentiert.

Hierzu erfragt sich M für jedes $m = n, n+1, n+2$ usw. schrittweise (in der loop-Schleife) eine Beispielfolge s, die eine schnelle Verkleinerung der Menge $\mathit{CIR}(n, m, s)$

input n
$s := \lambda$
$m := n$
loop
  if $\mathit{CIR}(n,m,s) = \emptyset$ **then**
    $m := m + 1;$
    $s := \lambda$
  else
    bestimme durch Aufruf der Funktionsprozedur $\mathit{Hyp}(0^n, 0^m, s)$
    einen Schaltkreis $c \in \mathit{CIR}(n)$, so daß für alle $x \in \{0,1\}^n$

$$\|\mathit{CIR}(n,m,s\#\langle x, c(x)\rangle)\| \geq \frac{\|\mathit{CIR}(n,m,s)\|}{16} \tag{6.2}$$

    gilt und stelle c als Äquivalenzfrage
    if Lehrer liefert ein Gegenbeispiel $x \in \{0,1\}^n$ **then**
      $s := s\#\langle x, \overline{c}(x)\rangle$
    else exit(**loop**) **end**
  end
end loop
output c

Abbildung 6.2: Ein Lernalgorithmus für Boolesche Schaltkreise.

aller mit s konsistenten Schaltkreise in $\mathit{CIR}(n,m)$ bewirkt. In jedem Schleifendurchlauf produziert M die jeweils nächste Äquivalenzfrage unter Verwendung einer Funktionsprozedur *Hyp*. Diese stellt ihr bei jedem Aufruf eine Hypothese c zur Verfügung, die für alle $x \in \{0,1\}^n$ mit zumindest $1/16$ der noch verbliebenen Schaltkreise in $\mathit{CIR}(n,m,s)$ übereinstimmt. Wird für den aktuellen Wert von m festgestellt, daß der gesuchte Schaltkreis nicht in $\mathit{CIR}(n,m)$ enthalten sein kann (das heißt, ist $\mathit{CIR}(n,m,s) = \emptyset$), so wird der Wert von m um eins hochgezählt und s wieder mit der leeren Folge initialisiert. Dies wird solange wiederholt, bis der Lehrer eine an ihn gerichtete Äquivalenzfrage mit „JA“ beantwortet. In diesem Fall hat M eine Repräsentation für das Zielkonzept gefunden und kann es ausgeben. Damit ist bereits die Korrektheit von M gezeigt.

Bevor wir erklären, wie die Funktionsprozedur *Hyp* unter Befragung eines Σ_2^p-Orakels eine Hypothese c konstruiert, die die in Abb. 6.2 angegebene Bedingung (6.2) erfüllt, schätzen wir die Anzahl der von M benötigten Schleifendurchläufe nach oben ab.

```
input (0^n, 0^m, s)
k := n
c := λ
while ⟨0^n, 0^m, 0^k, s, c⟩ ∉ U do
  k := k + 1
end
while c ∉ CIR(n, k) do
  if ⟨0^n, 0^m, 0^k, s, c0⟩ ∈ U then
    c := c0
  else
    c := c1
  end
end
return c
```

Abbildung 6.3: Die Funktionsprozedur *Hyp*.

Sei $\hat{m}$ die minimale Anzahl von Gattern, die ein Schaltkreis $\hat{c}$ benötigt, der das Zielkonzept repräsentiert. Da immer nur solche Lehr-Beispiele in die Beispielfolge s aufgenommen werden, die mit dem Zielkonzept übereinstimmen, kann der Schaltkreis $\hat{c}$ nicht aus der Menge $CIR(n, \hat{m}, s)$ herausfallen. Folglich kann die Menge $CIR(n, \hat{m}, s)$ nicht leer werden, so daß die Variable m sicherlich nicht über den Wert $\hat{m}$ hinaus hochgezählt wird.

Da bei jedem Aufruf der Prozedur $Hyp(0^n, 0^m, s)$ ein Schaltkreis c übergeben wird, der die Bedingung (6.2) erfüllt, verliert die Menge $CIR(n, m, s)$ bei jeder Erweiterung von s mindestens $1/16$ der noch in ihr verbliebenen Schaltkreise. Sicherlich enthält für jedes $m = n, n+1, \ldots, \hat{m}$ die Menge $CIR(n, m, s)$ zum Zeitpunkt, wenn die Suche nach einem Schaltkreis mit m Gattern gestartet wird, nicht mehr als $2^{p(m)}$ Schaltkreise. Deshalb kann für jedes m die zu Beginn der Suche nach einem Schaltkreis in $CIR(n, m)$ mit der leeren Folge λ initialisierte Beispielfolge s höchstens auf eine Länge von $\lceil p(m)/\log(16/15)\rceil \leq 15p(m)$ Lehrbeispielen anwachsen, bevor entweder das Zielkonzept herausgefunden wird, oder alle Schaltkreise in $CIR(n, m)$ durch ein Gegenbeispiel widerlegt werden. Insgesamt ist somit die Anzahl der Schleifendurchläufe durch

$$15 \sum_{m=n}^{\hat{m}} p(m) = O(\hat{m} \cdot p(\hat{m}))$$

beschränkt.

Als nächstes zeigen wir, wie die Prozedur *Hyp* unter Befragung eines geeigneten

Σ_2^{P}-Orakels einen Schaltkreis c konstruieren kann, der die Bedingung (6.2) erfüllt. Aus beweistechnischen Gründen lassen wir die Prozedur *Hyp* dabei Fragen an ein Δ_3^{P}-Orakel U stellen, es ist aber klar, daß jede solche Frage durch eine polynomiale Anzahl von Fragen an ein geeignetes Σ_2^{P}-Orakel ersetzt werden kann. Die durch die Prozedur *Hyp* ausgenützte Eigenschaft des Orakels U halten wir in folgender Behauptung fest, deren Gültigkeit wir am Ende des Beweises zeigen werden ($\preceq$ bezeichnet hierbei die Präfixrelation, das heißt, es gilt $c \preceq \hat{c}$ genau dann, wenn c ein Präfix (der Kodierung) von $\hat{c}$ ist).

Behauptung 1 *Es gibt eine Sprache $U \in \Delta_3^{\mathrm{P}}$, so daß für alle $n, m, k \geq 0$, jede Beispielfolge s und jeden Binärstring c die folgenden Implikationen gelten:*

$\langle 0^n, 0^m, 0^k, s, c\rangle \in U \;\Rightarrow$

es gibt einen Schaltkreis $\hat{c} \in \mathit{CIR}(n,k)$ mit $c \preceq \hat{c}$, so daß für alle $x \in \{0,1\}^n$ gilt $\|\mathit{CIR}(n,m,s\#\langle x, \hat{c}(x)\rangle)\| \geq \left(\frac{1}{16}\right) \|\mathit{CIR}(n,m,s)\|$,

$\langle 0^n, 0^m, 0^k, s, c\rangle \notin U \;\Rightarrow$

für alle Schaltkreise $\hat{c} \in \mathit{CIR}(n,k)$ mit $c \preceq \hat{c}$ gibt es ein $x \in \{0,1\}^n$ mit $\|\mathit{CIR}(n,m,s\#\langle x, \hat{c}(x)\rangle)\| < \left(\frac{1}{4}\right) \|\mathit{CIR}(n,m,s)\|$.

Falls also U auf die Frage $\langle 0^n, 0^m, 0^k, s, c\rangle$ eine positive Antwort gibt, dann gibt es einen Schaltkreis in $\mathit{CIR}(n,k)$, dessen Kodierung $\hat{c}$ mit dem Präfix c beginnt, und der für jedes $x \in \{0,1\}^n$ mit mindestens $1/16$ aller Schaltkreise in $\mathit{CIR}(n,m,s)$ übereinstimmt. Das heißt, $\hat{c}$ hat die in Abb. 6.2 verlangte Eigenschaft (6.2) und erfüllt zusätzlich die Bedingung $c \preceq \hat{c}$. Andererseits wird eine Frage $\langle 0^n, 0^m, 0^k, s, c\rangle$ von U sicher dann positiv beantwortet, wenn es einen Schaltkreis $\hat{c} \in \mathit{CIR}(n,k)$ mit $c \preceq \hat{c}$ gibt, der für jedes $x \in \{0,1\}^n$ sogar mit mindestens einem Viertel aller Schaltkreise in $\mathit{CIR}(n,m,s)$ übereinstimmt.

Abb. 6.3 enthält eine Beschreibung der Funktionsprozedur *Hyp*. Durch inkrementelle Suche stellt *Hyp* zuerst unter Befragung des Orakels U die Anzahl der Gatter des von ihr zu konstruierenden Schaltkreises fest. Danach führt *Hyp* eine Präfixsuche nach dem gewünschten Schaltkreis durch. Aufgrund von Behauptung 1 ist sichergestellt, daß *Hyp* nur solche Schaltkreise zurückgibt, die die Bedingung (6.2) erfüllen.

Wir müssen uns noch davon überzeugen, daß die von *Hyp* berechneten Schaltkreise nicht zu groß sind. Gemäß Lemma 6.8 muß es in $\mathit{CIR}(n,m,s)$ $6n$ Schaltkreise

$c_1, \ldots, c_{6n}$ mit der Eigenschaft geben, daß das Mehrheitsvotum $maj(c_1, \ldots, c_{6n})$ dieser Schaltkreise für jedes $x \in \{0,1\}^n$ mit mindestens einem Viertel aller Schaltkreise in der Menge $CIR(n,m,s)$ übereinstimmt. Also können diese Schaltkreise zu einem einzigen Schaltkreis c zusammengebaut werden, der die Bedingung (6.2) erfüllt. Offensichtlich werden für c nicht mehr als $O(m \cdot n)$ viele Gatter benötigt.

Es bleibt jetzt nur noch, den Beweis für Behauptung 1 zu erbringen. Da die Funktion $h(0^n, 0^m, s, c) =_{def} \max_{|x|=n} \|CIR(n,m,s\#\langle x, \overline{c}(x)\rangle)\|$ in $max\text{-}\mathcal{P}$ liegt, gibt es nach Satz 3.11 eine Sprache $W \in \Sigma_2^p$ mit folgender Eigenschaft: Für alle $x \in \{0,1\}^*$, alle $n, m, a \geq 0$, jede Beispielfolge s und jeden Schaltkreis c gilt

$$\langle 0^n, 0^m, s, c, a\rangle \in W \;\Rightarrow\; \max_{|x|=n} \|CIR(n,m,s\#\langle x, \overline{c}(x)\rangle)\| \;\leq\; \left(1 + \frac{1}{9}\right) a,$$

$$\langle 0^n, 0^m, s, c, a\rangle \notin W \;\Rightarrow\; \max_{|x|=n} \|CIR(n,m,s\#\langle x, \overline{c}(x)\rangle)\| \;>\; a.$$

Sei nun $g(0^n, 0^m, s)$ eine Approximation an die $\#\mathcal{P}$-Funktion $f(0^n, 0^m, s) = \|CIR(n,m,s)\|$, die folgende Approximationsgüte aufweist:

$$\|CIR(n,m,s)\| \;\leq\; g(0^n, 0^m, s) \;\leq\; \left(1 + \tfrac{1}{9}\right) \|CIR(n,m,s)\|.$$

Nach Satz 3.8 ist es möglich, eine solche Funktion g in $\mathcal{FP}(\Sigma_2^{\mathrm{P}})$ zu berechnen. Dann liegt natürlich auch die Funktion

$$g'(0^n, 0^m, s) \;=_{def}\; \lfloor (3/4) g(0^n, 0^m, s) \rfloor$$

in $\mathcal{FP}(\Sigma_2^{\mathrm{P}})$. Außerdem erhalten wir die folgenden Implikationen:

$$\langle 0^n, 0^m, s, c, g'(0^n, 0^m, s)\rangle \in W$$
$$\Rightarrow \quad \max_{|x|=n} \|CIR(n,m,s\#\langle x, \overline{c}(x)\rangle)\| \;\leq\; \frac{15}{16} \|CIR(n,m,s)\|,$$

$$\langle 0^n, 0^m, s, c, g'(0^n, 0^m, s)\rangle \notin W$$
$$\Rightarrow \quad \max_{|x|=n} \|CIR(n,m,s\#\langle x, \overline{c}(x)\rangle)\| \;>\; \frac{3}{4} \|CIR(n,m,s)\|.$$

Dies impliziert jedoch unmittelbar die Gültigkeit von Behauptung 1 für die folgendermaßen definierte Orakelsprache U.

$$\langle 0^n, 0^m, 0^k, s, c\rangle \in U \;\Leftrightarrow_{def}\; \text{es gibt einen Schaltkreis } \hat{c} \in CIR(n,k) \text{ mit } c \preceq \hat{c},$$
$$\text{so daß } \langle 0^n, 0^m, s, c, g'(0^n, 0^m, s)\rangle \in W \text{ ist.}$$

Da U offensichtlich in Δ_3^{P} liegt, ist damit der Beweis des Satzes erbracht. ∎

Literaturverzeichnis

[ABG90] A. Amir, R. Beigel und W. Gasarch. Some connections between bounded query classes and non-uniform complexity. In *Proceedings of the 5th Structure in Complexity Theory Conference*, 232–243. IEEE Computer Society Press, 1990.

[AFK89] M. Abadi, J. Feigenbaum und J. Kilian. On hiding information from an oracle. *Journal of Computer and System Sciences*, **39**:21–30, 1989.

[AH87] L. Adleman und M. Huang. Recognizing primes in random polynomial time. In *Proceedings of the 19th ACM Symposium on Theory of Computing*, 462–469. ACM Press, 1987.

[AH92] E. Allender und L. A. Hemachandra. Lower bounds for the low hierarchy. *Journal of the ACM*, **39**:234–250, 1992.

[AKM92] V. Arvind, J. Köbler und M. Mundhenk. Lowness and the complexity of sparse and tally descriptions. In *Proceedings of the 3rd International Symposium on Algorithms and Computation*, Lecture Notes in Computer Science #650, 249–258. Springer-Verlag, 1992.

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

[AM77] L. Adleman und K. Manders. Reducibility, randomness, and intractibility. In *Proceedings of the 9th ACM Symposium on Theory of Computing*, 151–163. ACM Press, 1977.

[Ang88] D. Angluin. Queries and concept learning. *Machine Learning*, **2**:319–342, 1988.

[AS83] D. Angluin und C. Smith. Inductive inference: theory and methods. *ACM Computing Surveys*, **15**:237–269, 1983.

[Bab85] L. Babai. Trading group theory for randomness. In *Proceedings of the 17th ACM Symposium on Theory of Computing*, 421–429. ACM Press, 1985.

[Bal90] J. Balcázar. Self-reducibility. *Journal of Computer and System Sciences*, **41**:367–388, 1990.

[BB86] J.L. Balcázar und R. Book. Sets with small generalized Kolmogorov complexity. *Acta Informatica*, **23**:679–688, 1986.

[BBS86] J. Balcázar, R. Book und U. Schöning. Sparse sets, lowness and highness. *SIAM Journal on Computing*, **23**:679–688, 1986.

[BC94] D. Bovet und P. Crescenzi. *Introduction to the Theory of Complexity.* Prentice-Hall, 1994.

[BCG+95] N. Bshouty, R. Cleve, R. Gavaldà, S. Kannan und C. Tamon. Oracles and queries that are sufficient for exact learning. Technical Report TR 95-015, ECCC, 1995. To appear in Journal of Computer and System Sciences.

[BCKT94] N. Bshouty, R. Cleve, S. Kannan und C. Tamon. Oracles and queries that are sufficient for exact learning. In *Proceedings of the 7th ACM Conference on Computational Learning Theory*, 130–139. Morgan Kaufmann, 1994.

[BDG90] J. Balcázar, J. Díaz und J. Gabarró. *Structural Complexity II.* Springer-Verlag, 1990.

[BDG95] J. Balcázar, J. Díaz und J. Gabarró. *Structural Complexity I.* Springer-Verlag, second edition, 1995.

[BF72] J. Barzdin und R. Freivald. On the prediction of general recursive functions. *Soviet Mathematics Doklady*, **13**:1224–1228, 1972.

[BF91] L. Babai und L. Fortnow. Arithmetization: A new method in structural complexity. *Computational Complexity*, **1**:41–66, 1991.

[BFL91] L. Babai, L. Fortnow und C. Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational Complexity*, **1**:1–40, 1991.

[BG81] C.H. Bennett und J. Gill. Relative to a random oracle A, $P^A \neq NP^A \neq co\text{-}NP^A$ with probability 1. *SIAM Journal on Computing*, **10**:96–113, 1981.

[BH77] L. Berman und J. Hartmanis. On isomorphism and density of NP and other complexity classes. *SIAM Journal on Computing*, **6**:305–327, 1977.

[BH91] S. Buss und L. Hay. On truth-table reducibility to SAT. *Information and Computation*, **91**(1):86–102, 1991.

[Blu94] A. L. Blum. Separating distribution-free and mistake-bound learning models over the boolean domain. *SIAM Journal on Computing*, **23**(5):990–1000, 1994.

[BT76] I. Borosh und L.B. Treybig. Bounds on positive integral solutions of linear Diophantine equations. *pams*, **55**:299–304, 1976.

[Che52] H. Chernoff. A measure of the asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Mathematical Statistics*, **23**:493–509, 1952.

[CKS81] A. Chandra, D. Kozen und L. Stockmeyer. Alternation. *Journal of the ACM*, **28**:114–133, 1981.

[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.

[FK92] M. R. Fellows und N. Koblitz. Self-witnessing polynomial-time complexity and prime factorization. In *Proceedings of the 7th Structure in Complexity Theory Conference*, 107–110. IEEE Computer Society Press, 1992.

[Gav92] R. Gavaldà. Bounding the complexity of advice functions. In *Proceedings of the 7th Structure in Complexity Theory Conference*, 249–254. IEEE Computer Society Press, 1992.

[Gav93] R. Gavaldà. On the power of equivalence queries. In *Proceedings of the 1st European Conference on Computational Learning Theory*, 193–203, 1993.

[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.

[GKP89] R. Graham, D. Knuth und O. Patashnik. *Concrete Mathematics.* Addison-Wesley, 1989.

[Gol67] E. M. Gold. Language identification in the limit. *Information and Control*, **10**:447–474, 1967.

[GS89] S. Goldwasser und M. Sipser. Private coins versus public coins in interactive proof systems. In S. Micali, editor, *Randomness and Computation*, Advances in Computing Research #5, 73–90. JAI Press, 1989.

[GW93] R. Gavaldà und O. Watanabe. On the computational complexity of small descriptions. *SIAM Journal on Computing*, **22**:1257–1275, 1993.

[Har83] J. Hartmanis. On sparse sets in NP-P. *Information Processing Letters*, **16**:55–60, 1983.

[Hel86] H. Heller. On relativized exponential and probabilistic complexity classes. *Information and Control*, **71**:231–243, 1986.

[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.

[HHO+93] L. A. Hemachandra, A. Hoene, M. Ogiwara, A. Selman, T. Thierauf und J. Wang. Selectivity. In *Proceedings of the 5th ICCI*, 55–59. IEEE Computer Society Press, 1993.

[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.

[Kad88] J. Kadin. *Restricted Turing Reducibilities and the Structure of the Polynomial Time Hierarchy.* Dissertation, Cornell University, New York, 1988.

[Kad89] J. Kadin. $\mathrm{P}^{\mathrm{NP}[\log n]}$ and sparse Turing-complete sets for NP. *Journal of Computer and System Sciences*, **39**:282–298, 1989.

[Käm91] J. Kämper. Non-uniform proof systems: A new framework to describe non-uniform and probabilistic complexity classes. *Theoretical Computer Science*, **85**(2):305–331, 1991.

[Kan82] R. Kannan. Circuit-size lower bounds and non-reducibility to sparse sets. *Information and Control*, **55**:40–56, 1982.

[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.

[Kar84] N. KARMAKAR. A new polynomial algorithm for linear programming. *Combinatorica*, **4**:373–395, 1984.

[Kha79] L.G. KHACHIYAN. A polynomial algorithm in linear programming. *Soviet Mathematics Doklady*, **20**:191–194, 1979.

[KL80] R. M. KARP UND R. J. LIPTON. Some connections between nonuniform and uniform complexity classes. In *Proceedings of the 12th ACM Symposium on Theory of Computing*, 302–309. ACM Press, 1980.

[Ko83] K. KO. On self-reducibility and weak p-selectivity. *Journal of Computer and System Sciences*, **26**:209–221, 1983.

[Köb94] J. KÖBLER. Locating P/poly optimally in the extended low hierarchy. *Theoretical Computer Science*, **134**(2):263–285, 1994.

[KS85] K. KO UND U. SCHÖNING. On circuit-size complexity and the low hierarchy in NP. *SIAM Journal on Computing*, **14**:41–51, 1985.

[KST89] J. KÖBLER, U. SCHÖNING UND J. TORÁN. On counting and approximation. *Acta Informatica*, **26**(4):363–379, 1989.

[KW95] J. KÖBLER UND O. WATANABE. New collapse consequences of NP having small circuits. In *Proceedings of the 22nd International Colloquium on Automata, Languages, and Programming*, Lecture Notes in Computer Science #944, 196–207. Springer-Verlag, 1995.

[Lad75a] R. LADNER. The circuit value problem is log space complete for P. *SIGACT News*, **7**:18–20, 1975.

[Lad75b] R. LADNER. On the structure of polynomial-time reducibility. *Journal of the ACM*, **22**:155–171, 1975.

[Lau83] C. LAUTEMANN. BPP and the polynomial hierarchy. *Information Processing Letters*, **14**:215–217, 1983.

[Ler83] M. LERMAN. *Degrees of Unsolvability.* Springer-Verlag, 1983.

[LFKN92] C. LUND, L. FORTNOW, H. KARLOFF UND N. NISAN. Algebraic methods for interactive proof systems. *Journal of the ACM*, **39**(4):859–868, 1992.

[LLS75] R. LADNER, N. LYNCH UND A. SELMAN. A comparison of polynomial time reducibilities. *Theoretical Computer Science*, **1**:103–123, 1975.

[Lon82] T. Long. Strong nondeterministic polynomial-time reducibilities. *Theoretical Computer Science*, **21**:1–25, 1982.

[Lon85] T. Long. On restricting the size of oracles compared with restricting access to oracles. *SIAM Journal on Computing*, **14**:585–597, 1985.

[LS91] T. Long und M. Sheu. A refinement of the low and high hierarchies. Technical Report OSU-CISRC-2/91-TR6, The Ohio State University, 1991.

[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.

[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.

[OKSW94] P. Orponen, K. Ko, U. Schöning und O. Watanabe. Instance complexity. *Journal of the ACM*, **41**(1):96–121, 1994.

[OL93] M. Ogiwara und A. Lozano. On sparse hard sets for counting classes. *Theoretical Computer Science*, **112**:255–275, 1993.

[OW91] M. Ogiwara und O. Watanabe. On polynomial-time bounded truth-table reducibility of NP sets to sparse sets. *SIAM Journal on Computing*, **20**:471–483, 1991.

[Pap94] C. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[Pio91] M. Piotrów. On the complexity of counting in the polynomial hierarchy. *Theoretical Computer Science*, **81**(1):77–95, 1991.

[Pip79] N. Pippenger. On simultaneous resource bounds. In *Proceedings of the 20th IEEE Symposium on the Foundations of Computer Science*, 307–311. IEEE Computer Society Press, 1979.

[Rab76] M.O. Rabin. Probabilistic algorithms. In J.F. Traub, editor, *Algorithms and Complexity.* Academic Press, 1976.

[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.

[Sch92] U. Schöning. *Theoretische Informatik kurz gefaßt.* Bibl. Inst. Wissenschaftsverlag, 1992.

[Sch95] U. Schöning. *Perlen der Theoretischen Informatik.* Bibl. Inst. Wissenschaftsverlag, 1995.

[Sel78] A. Selman. Polynomial time enumeration reducibility. *SIAM Journal on Computing*, **7**:440–447, 1978.

[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.

[SL94] M.-J. Sheu und T. Long. The extended low hierarchy is an infinite hierarchy. *Journal of Computer and System Sciences*, **23**(3):488–509, 1994.

[Soa87] R. I. Soare. *Recursively Enumerable Sets and Degrees*, Perspectives in Mathematical Logic . Perspectives in Mathematical Logic. Springer-Verlag, 1987.

[SS77] R. Solovay und V. Strassen. A fast Monte-Carlo test for primality. *SIAM Journal on Computing*, **6**:84–85, 1977.

[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.

[Tar93] J. Tarui. Probabilistic polynomials, AC^0 functions, and the polynomial time hierarchy. *Theoretical Computer Science*, **113**:167–183, 1993.

[TO92] S. Toda und M. Ogiwara. Counting classes are at least as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, **21**:316–328, 1992.

[Tod91] S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, **20**:865–877, 1991.

[Tor91] J. Torán. Complexity classes defined by counting quantifiers. *Journal of the ACM*, **38**:753–774, 1991.

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

[Val79] L. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, **8**:189–201, 1979.

[Val84] L. Valiant. A theory of the learnable. *Communications of the ACM*, **27**(11):1134–1142, 1984.

[Vol94] H. Vollmer. *Komplexitätsklassen von Funktionen.* Dissertation, Universität Würzburg, 1994.

[Wag86] K.W. Wagner. The complexity of combinatorial problems with succinct input representation. *Acta Informatica*, **23**:325–356, 1986.

[Wag90] K.W. Wagner. Bounded query classes. *SIAM Journal on Computing*, **19**:833–846, 1990.

[Wat94] O. Watanabe. A framework for polynomial-time query learnability. *Mathematical Systems Theory*, **27**:211–229, 1994.

[Weg87] I. Wegener. *The Complexity of Boolean Functions.* Wiley-Teubner, 1987.

[Wil85] C. Wilson. Relativized circuit complexity. *Journal of Computer and System Sciences*, **31**(2):169–181, 1985.

[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.

[Yap83] C. Yap. Some consequences of non-uniform conditions on uniform classes. *Theoretical Computer Science*, **26**:287–300, 1983.

[Zac82] S. Zachos. Robustness of probabilistic computational complexity classes under definitional perturbations. *Information and Control*, **54**:143–154, 1982.

Stichwortverzeichnis

Z

Symbolverzeichnis