Effizient prüfbare Beweise für NP-Probleme

DIPLOMARBEIT im Rahmen des Studiengangs Wirtschaftsmathematik der Fakultät für Mathematik und Wirtschaftswissenschaften an der Fakultät für Informatik (Abteilung Theoretische Informatik) der Universität Ulm gemäß der Prüfungsordnung vom 2. Oktober 1991.

Betreuer:	Dr. Johannes Köbler
Hauptberichter:	Prof. Dr. Uwe Schöning
Berichter:	Prof. Dr. Franz Schweiggert

vorgelegt von:

Martin Scheifele

Neu-Ulm

28. September 1993

Inhaltsverzeichnis

Einführung

Das Hauptziel dieser Arbeit wird es sein, eine neue Charakterisierung der Klasse NP zu präsentieren. Seit Anfang 1992 ist nämlich bekannt, daß die Klasse NP genau die Sprachen L enthält, welche einen Zugehörigkeitsbeweis (das ist der Beweis für $x \in L$) haben, der **probabilistisch** in polynomieller Zeit verifiziert werden kann, und zwar, indem eine *logarithmische* Anzahl von Zufallsbits und eine *konstante* Anzahl von Beweisbits benutzt werden. Es handelt sich also um sehr effizient prüfbare Beweise für NP-Probleme. Der Begriff *Beweis* ist im probabilistischen Sinn anders zu verstehen als im nicht-probabilistischen. Der nicht-probabilistische Beweis eines Theorems, und sei er noch so kurz, kann nicht dadurch geprüft werden, nur einen kleinen Teil des Beweises einzublicken, auch wenn man durch eine geschickte Zufallswahl, die zu prüfenden Teile des Beweises auswählt. Dies kann auch so interpretiert werden: ein Fehler in diesem Beweis kann nur entdeckt werden, wenn man den Beweis auf Korrektheit *jedes einzelnen* Schrittes prüft.

Anders ist das bei den probabilistischen Zugehörigkeitsbeweisen zu einer bestimmten Sprache. Wir werden sehen, daß man diese Beweise durch ein geschicktes *Aufblähen* der Information prüfen kann. Dabei gibt es bei $x \in L$ immer einen Beweis, der immer, d. h. mit der Wahrscheinlichkeit 1, akzeptiert wird. Im Falle $x \notin L$ kann man mit sehr hoher Wahrscheinlichkeit, die Hypothese $x \in L$ verwerfen, denn alle notwendigerweise falschen Beweise, werden mit einer gewissen positiven Wahrscheinlichkeit nicht akzeptiert. Die Tatsache, daß der Beweis durch seine enorme Größe nicht meht gut zu fassen ist, wird sozusagen wettgemacht durch den verhältnismäßig sehr kleinen Ausschnitt[1], den man von diesem Beweis sehen muß, um die entsprechenden Schlüsse -natürlich nur mit einer gewissen Wahrscheinlichkeit- ziehen zu können. Daher spricht man auch von **transparenten Beweisen**.

[1]In [BFLS 91] wird bemerkt, daß der Logarithmus der Anzahl aller Partikel im sichtbaren Universum nur 300 beträgt!

Die Arbeit ist folgendermaßen unterteilt: Im ersten Kapitel finden sich grundlegende Definitionen und einfache mathematische Sätze, welche im Lauf der Arbeit mehrmals gebraucht werden. Darauf wird im einzelnen verwiesen. Der Rest der Arbeit beinhaltet drei voneinander unabhängige Ergebnisse.

Kapitel 2: Das *MIP*=*NEXP*-Resultat

Kapitel 3: Der Klein-Grad-Test

Kapitel 4: Das Hauptresultat: $NP = PCP(\log n, 1)$

MIP ist die Klasse von Sprachen, die mit mehreren Provern, und einem probabilistischen Verifier *interaktiv* in polynomieller Zeit erkannt wird. Dabei sind nicht mehr als zwei Prover nötig, wie wir sehen werden. Die Klasse *NEXP* wird von nichtdeterministischen Turingmaschinen in exponentieller Zeit erkannt. Das *MIP*=*NEXP*-Resultat von [BFL 91] wird als erstes behandelt, da hier wichtige Techniken vorgestellt werden, die im Hauptresultat wieder verwendet werden, z. B. das Summenprotokoll von [LFKN 90] oder der Multilinearitätstest aus [BFLS 91]. Dabei ist ein Aspekt wesentlich: die Prover die nötig sind, um eine Sprache in *MIP* zu erkennen, können durch ein Orakel ersetzt werden, die Klasse bleibt die gleiche. Wir gehen hier also einen Umweg über *MIP*=*POM*.

Der Klein-Grad-Test von Rubinfeld und Sudan [RS 92] testet eine Funktion auf den Totalgrad; dabei ist das Matrixtranspositionslemma von besonderer Bedeutung. Das Hauptresultat in Kapitel 4 von [AS* 92, AMLSS 92, FGLSS 91] enthält eine ganze Reihe von Techniken, deshalb sind die Protokolle und die benutzten Ideen sehr komplex.

Zum Erkennen einer Sprache in *PCP* wird wie bei *POM* ein Orakel und ein probabilistischer Verifier benutzt. Sie ist aber aufgrund zweier Parameter etwas komplizierter. Der erste Parameter gibt an, wieviele Zufallsbits verwendet werden, der zweite, wieviele Beweisbits gelesen werden. Die Klasse *POM* entspricht daher der Klasse *PCP*, falls die beiden Parameter polynomiell sind. Die Klasse *PCP* ist also eine Verallgemeinerung von *POM*.

Zunächst werden hier zwei voneinender unabhängige *PCP*-Protokolle für *NP* vorgestellt, die unterschiedliche Zufalls- und Kommunikationsbeschränkungen haben. Danach wird die Rekursionsidee von Arora und Safra erläutert. Diese Idee nennt sich genauer die **Komposition von Beweisen**; zwei *PCP*-Protokoll werden ineinandergeschachtelt.

Das Hauptresultat hat nun einige auf bestimmte Probleme sehr einschränkende Konsequenzen. Diese negativen Ergebnisse beziehen sich auf Approximationen *NP*-vollständiger Probleme. Dabei spielen die Optimierungsprobleme der Klasse *MAX-SNP* eine große Rolle. Das Finden einer Belegung, die die maximale Anzahl von Klauseln einer 3-CNF erfüllt, genannt MAX-3-SAT, ist ein vollständiges Problem in dieser Klasse. Mit unserem Hauptresultat $NP = PCP(\log n, 1)$ ist nun folgender Sachverhalt [AMLSS 92] leicht zu zeigen:

$$P \neq NP \Rightarrow \exists \text{ kein PTAS für } \mathit{MAX\text{-}SNP}\text{-vollständige Probleme}$$

Ein PTAS ist ein in polynomieller Zeit berechenbares Approximationsschema, das die relative Abweichung vom Optimum unter einer festgelegten Grenze läßt. Dabei ist wichtig, daß die Approximation für alle Problemeingaben aller Problemlängen funktionieren muß. Unter diese Klassen fallen wie schon gesagt MAX-3-SAT und viele andere mehr, wie z. B. MAX-2-SAT, Vertex-Cover, Dominating Set, Independent Set, Euclidian TSP, Shortest Superstring, Euclidian Steiner Tree. Für alle diese Probleme braucht also in Zukunft kein polynomieller Approximationsalgorithmus gesucht werden; gäbe es nämlich einen, würde dies *P*=*NP* beweisen, was einer gewaltigen Sensation gleich käme.

Kapitel 1

Grundlagen

Gute Einführungen in die Theoretische Informatik, bzw. in die Komplexitätstheorie findet man zum Beispiel in [AHU 74, GJ 79, CLR 90]; zunächst aber einige mathematische Definitionen und Grundlagen.

1.1 Mathematische Grundlagen

Auf die Inhalte dieser mathematischen Grundlagen wird zu späteren Zeitpunkten hingewiesen. Sie können daher beim ersten Durchgehen der Arbeit übergangen werden.

Definition 1.1 *$O(g(n))$ (sprich: groß O) ist eines der Landau'schen Symbole mit der Bedeutung:*

$$h(n) = O(g(n)) \iff \exists \text{ Konstanten } k, n_0 \in \mathbb{N},\ \forall n \geq n_0,\ n \in \mathbb{N} : h(n) \leq k \cdot g(n),$$

Man sagt auch h ist höchstens von Größenordnung g.

Polynome und Erweiterungen

Definition 1.2 ((Univariates) Polynom) *Ein univariates Polynom p (in einer Variable) ist ein Element aus dem Polynomring $F[x]$, (F ein beliebiger Körper), hat also folgende Form:*

$$p(x) = \sum_{i=0}^{k} a_i x^i\,, \quad a_i \in F$$

wobei x die Variable ist. Der Grad von p ist $\max_{a_i \neq 0} i \leq k$

Definition 1.3 ((Multivariates) Polynom, multilinear, Totalgrad) *Ein multivariates Polynom p (in mehreren Variablen) ist ein Element aus dem Polynomring $F[x_1, ..., x_n]$, (F ein beliebiger Körper), hat also folgende Form:*

$$p(x) = \sum_{i_1=0}^{k} \sum_{i_2=0}^{k} \cdots \sum_{i_n=0}^{k} \alpha_{i_1,...,i_n} x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n}\,, \quad \alpha_{i_1,...,i_n} \in F$$

wobei die $x_1, ..., x_n$ die Variable sind. Der Grad in einer bestimmten Variablen ist $\text{Grad}_p(x_j) := \max_{\alpha_{i_1,...,i_n} \neq 0} i_j \; (\leq k)$. Gilt $\text{Grad}_p(x_j) = 1$, für $x_1, ..., x_n$, so spricht man von einem multilinearen Polynom. Der Totalgrad des Polynoms ist
$\text{Totalgrad}(p) := \max_{\alpha_{i_1,...,i_n} \neq 0} \left\{ \sum_{j=1}^{n} i_j \right\} \quad \left(\leq \sum_{j=1}^{n} \text{Grad}_p(x_j) \right)$

Bemerkung: Wenn nichts anderes gesagt, wird aus dem Kontext klar, ob von einem uni- oder multivariaten Polynom gesprochen wird. Ebenso kann mit Grad eines Polynoms der Totalgrad gemeint sein, was auch aus dem Kontext ersichtlich ist. Man beachte: die Anzahl der Summanden ist *exponentiell* in der Anzahl der Variablen, nämlich k^n. Der Totalgrad ist nach oben durch $n \cdot \max_i \text{Grad}(x_i)$ beschränkt.

Beispiel: Sei $p(x_1, x_2, x_3) = 11 + x_1 x_2^5 + x_1^3 x_2^3 + x_2^5 x_3^4$, also $k = 5$. Dann gilt $\text{Grad}_p(x_1) = 3$, $\text{Grad}_p(x_2) = 5$, $\text{Grad}_p(x_3) = 4$, $\text{Totalgrad}(p) = \max\{0, 1, 5, 6, 9\} = 9 \leq 3 + 5 + 4$.

$\diamond$

($\diamond$ soll das Ende eines Beispiels anzeigen.)

Lemma 1.4 (multilineare Erweiterung) *Sei $A : \{0,1\}^s \to F$ (F ein beliebiger Körper) eine beliebige Funktion. Dann hat A eine eindeutige multilineare Erweiterung $\tilde{A} : F^s \to F$. Nimmt A nur ganzzahlige Werte an, so nimmt auch $\tilde{A}$ an ganz F^s nur ganzzahlige Werte an.*

Beweis: Sei $\tilde{A}$ wie folgt definiert:

$$\tilde{A}(x) := \sum_{b \in \{0,1\}^s} A(b) \prod_{i=1}^{s} l_{\beta_i}(\xi_i) \tag{1.1}$$

wobei $x = (\xi_1, ..., \xi_s)$; b ist der Bitstring $\beta_1 \cdots \beta_s$ und sei $l_0(\xi) = 1 - \xi$, $l_1(\xi) = \xi$. Damit ist $\tilde{A}$ multilinear.

Zur Eindeutigkeit: Angenommen, $\tilde{B} : F^s \to F$ ist auch ein multilineare Erweiterung von A. Damit ist $f = \tilde{A} - \tilde{B}$ auch multilinear. Für $x \in F^s$ sei $k(x)$ die Anzahl der Koordinaten, die nicht aus {0,1} sind.

Induktionsbehauptung: $\forall x \in F^s$ mit $k(x) \leq n \; : \; f(x) = 0$

Induktionsanfang: $k(x) = 0 \; \surd$

Induktionsschritt: $k(x) = k$, sei OBdA. $\xi_1 \notin \{0,1\}$. Wenn man ξ_1 durch 0 oder 1 ersetzt, erhält man Stellen, an denen f verschwindet (nach Induktionsvoraussetztung gilt nämlich $f(0, \xi_2, ..., \xi_s) = f(1, \xi_2, ..., \xi_s) = 0$). Aus der Linearität von f in ξ_1 folgt, daß f bei x verschwindet. $\surd$

□

Dieses Lemma kann man folgendermaßen verallgemeinern: Statt der Boole'schen Grundmenge {0,1} nimmt man $[0...h]$ als Grundmenge.

Lemma 1.5 (Erweiterungslemma) *Sei $F \supseteq [0...h]$ ein beliebiger Körper und $f : [0...h]^s \to F$ eine beliebige Funktion in s Variablen. Dann existiert ein eindeutiges Polynom $\hat{f}(y_1, ..., y_s) : F^s \to F$ in s Variablen, die sog. Klein-Grad-Erweiterung von f, mit*

$$\hat{f}\Big|_{[0...h]^s} = f \quad \textit{und } Grad_{\hat{f}}(y_i) = h, \; \forall 1 \leq i \leq s$$

Man beachte: der Grad von $\hat{f}$ ist in jeder Variablen um genau 1 kleiner als die Mächtigkeit der Ausgangsmenge von f, diese war ja $[0...h]$.

Beweis: Wie in Lemma 1.4, mit folgender Modifikation:

$$\hat{f}(y_1, ..., y_s) = \sum_{b \in [0...h]^s} f(b) \prod_{i=1}^{s} L_{\beta_i}(y_i), \quad \beta_i \in [0...h]$$

wobei die $L_{\beta_i}(y_i)$ so gewählt sind, daß

$$L_{\beta_i}(y_i) = \begin{cases} 1 & \text{für } y_i = \beta_i \\ 0 & \text{für alle } y_i \in [0..h] - \{\beta_i\} \end{cases}$$

Dabei sind die $L_0(\xi), ..., L_h(\xi)$ Polynome von $F \to F$ mit $\text{Grad}_{L_i} = h,\ \forall 1 \leq i \leq h$. Denn, sie enstehen aus folgendem Schema:

$$\begin{array}{rcl}
L_0(\xi) = (\xi - k_0)(\xi - 2)\cdots(\xi - h) & & \text{mit } L_0(\xi)\Big|_{\xi=0} \stackrel{!}{=} 1 \\
\vdots & & \vdots \\
L_i(\xi) = \xi(\xi - 2)\cdots(\xi - (i-1))(\xi - k_i)(\xi - (i+1))\cdots(\xi - h) & & \text{mit } L_i(\xi)\Big|_{\xi=i} \stackrel{!}{=} 1 \\
\vdots & & \vdots \\
L_h(\xi) = \xi(\xi - 2)\cdots(\xi - (h-1))(\xi - k_h) & & \text{mit } L_h(\xi)\Big|_{\xi=h} \stackrel{!}{=} 1
\end{array}$$

mit eindeutigen $k_0, ..., k_h \in F$. Eindeutigkeit wie in Lemma 1.4. $\square$

Beispiel: Damit man sieht, wie aus einer beliebigen Funktion A eine multilineare Funktion $\tilde{A}$ wird, hier ein Beispiel mit einem Polynom und einer exp-Funktion. Der benutzte Körper könnte hier auch $\mathbb{R}$ sein; s ist 2.

$$A(x) = A(\xi_1, \xi_2) = \xi_2 exp(\xi_1) + 2\xi_1^3 \quad \begin{array}{ll} 0 & \text{für } (0,0) \\ 1 & \text{für } (0,1) \\ 2 & \text{für } (1,0) \\ e+2 & \text{für } (1,1) \end{array}$$

Die Funktion A geht also von $\{0,1\}^2$ nach $\mathbb{R}$. Die mit Lemma 1.4 konstruierte multilineare Erweiterung $\tilde{A}$ sieht so aus:

$$\begin{aligned}
\tilde{A}(\xi_1, \xi_2) &= A(0,0) l_0(\xi_1) l_0(\xi_2) \\
&\quad + A(0,1) l_0(\xi_1) l_1(\xi_2) \\
&\quad + A(1,0) l_1(\xi_1) l_0(\xi_2) \\
&\quad + A(1,1) l_1(\xi_1) l_1(\xi_2) \\
\\
&= A(0,0)(1 - \xi_1)(1 - \xi_2) \\
&\quad + A(0,1)(1 - \xi_1)\xi_2 \\
&\quad + A(1,0)\xi_1(1 - \xi_2) \\
&\quad + A(1,1)\xi_1\xi_2 \\
&= 0 + (1 - \xi_1)(\xi_2) + 2\xi_1(1 - \xi_2) + (e+2)\xi_1\xi_2 \\
&= 2\xi_1 + \xi_2 + (e-1)\xi_1\xi_2
\end{aligned}$$

In der letzten Zeile steht eine multilineare Funktion in den Variablen ξ_1 und ξ_2. Es ist unsere gesuchte Funktion, die an Boole'schen Werten mit der Funktion A übereinstimmt. Dieses ist leicht nachzuprüfen. $\diamond$

Korollar 1.6 *Sei $I = \{0, 1, ..., N-1\}$ und $\tilde{A} : I^s \to Q$ eine multilineare Erweiterung einer Boole'schen Funktion. Dann gilt für alle $x \in I^s : |\tilde{A}(x)| < (2N)^s$*

Beweis: Aus Lemma 1.4 folgt:
$|\tilde{A}(x)| \leq \sum_{b \in \{0,1\}^s} |A(b)| N^s = N^s 2^s |A(b)| \leq (2N)^s$
Der letzte Schritt ist korrekt, denn $|A(b)| \leq 1$, da A eine Boole'sche Formel. $\square$

Boole'sche Funktionen und die Arithmetisierung

Zu Beginn einige grundlegende Begriffserklärungen und ein ebenso einfacher wie wichtiger Sachverhalt. Eine *Boole'sche Funktion* in m Variablen ist eine Funktion $\{0,1\}^m \to \{0,1\}$. Ein arithmetischer Ausdruck f über einem beliebigen Körper stellt eine *Arithmetisierung* einer Boole'schen Funktion $B(u_1, ..., u_m)$ dar, genau dann wenn für alle 2^m Argumente $x \in \{0,1\}^m$ der (Boole'sche) Wert von B und der (arithmetische) Wert von f an der Stelle x übereinstimmen.

Eine *Boole'sche Formel* ist ein wohldefinierter Ausdruck aus den Konstanten 0 und 1, Variablensymbolen, den 2-stelligen Funktoren $\vee$ und $\wedge$, dem einstelligen Funktor $\neg$ und den technischen Zeichen (,). Eine Boole'sche Formel verkörpert also eine Boole'sche Funktion. Eine *arithmetische Formel* ist ein wohldefinierter Ausdruck, der aus den Konstanten 0 und 1, Variablensymbolen, den 2-stelligen Operatoren $+, -, \times$ und den technischen Zeichen (,) zusammengestellt ist. Daher entspricht eine arithmetische Formel einem Polynom in mehreren Variablen über einem kommutativen Ring mit 1.

Lemma 1.7 *Gegeben sei eine Boole'sche Formel B. In linearer Zeit kann eine arithmetische Formel konstruiert werden, die eine Arithmetisierung von B über einen beliebigen Körper (und damit über jeden beliebigen Ring mit 1) darstellt.*

Beweis: Nach DeMorgan können in B alle $\vee$ durch $\wedge$ und $\neg$ ersetzt werden. Dann werden die Boole'schen durch arithmetische Variablensymbole ersetzt. Alle Ausdrücke der Form $\neg g$ ersetze man durch $(1 - g)$ und $\wedge$ durch $\times$.

□

Beispiel: $B(u_1, u_2, u_3) = (u_1 \vee u_2 \vee \neg u_3) \wedge (\neg u_1 \vee \neg u_2)$

$$\begin{aligned} &\leadsto \neg(\neg u_1 \wedge \neg u_2 \wedge u_3) \wedge \neg(u_1 \wedge u_2) \\ &\leadsto (1 - (1 - x_1) \times (1 - x_2) \times x_3)) \times (1 - x_1 \times x_2) =: f(x_1, x_2, x_3) \end{aligned}$$

Dieser Ausdruck kann als Polynom in 3 Variablen interpretiert werden : $f(x_1, x_2, x_3) = 1 - x_3 + x_1x_3 - x_1x_2 - x_1^2x_2x_3 + x_2x_3 + x_1x_2^2x_3 - x_1^2x_2^2x_3$.
Dieses Polynom ist äquivalent zu der Arithmetisierung. Man beachte, daß bei Boole'schen 3-CNF Formeln (unser B gehört dazu) der $\mathrm{Grad}_f(x_i) \leq \#$ der Klauseln von B ist und damit $\mathrm{Totalgrad}(f) \leq (\#$ der Variablen$)\cdot$ ($\#$ der Klauseln). Da der Input von B aber lediglich aus Boole'schen Variablen besteht, erfüllt das vereinfachte Polynom (ohne Potenzen) $1 - x_3 + x_1x_3 - x_1x_2 + x_2x_3 - x_1x_2x_3\Big|_{\{0,1\}^3} = f\Big|_{\{0,1\}^3}$ die Bedingung der Übereinstimmung an allen möglichen Boole'schen Werten genauso, und wir werden dieses weiterbenutzen. ◇

Körpererweiterungen

Sei F ein endlicher Körper.
Ziel: Finde einen Körper F' mit $|F'| = |F|^s$ für ein s.

Vorgehensweise: Sei a ein über F irreduzibles Polynom vom Grad s:

$$a(x) = a_0 + a_1x + ... + a_sx^s, \quad a_\nu \in F,\ a_s \neq 0.$$

Sei ϱ eine der s verschiedenen Wurzeln von a; natürlich $\varrho \notin F$. Sei

$$F' = \left\{f_0 + f_1\varrho + f_2\varrho^2 + ... + f_{s-1}\varrho^{s-1} \,|\, f_0, f_1, ..., f_{s-1} \in F\right\}$$

ein Vektorraum.

Bildet $\{1, \varrho, \varrho^2, ..., \varrho^{s-1}\}$ eine Basis? Ja!

> Man nehme das Gegenteil an. Dann existieren p, q, zwei Polynome in ϱ, die den selben Wert haben. Daher $p - q =: f'(\varrho) = f_0' + f_1'\varrho + f_2'\varrho^2 + ... + f_{s-1}'\varrho^{s-1} = 0$ und nicht alle f_i' 0 sind. $\Rightarrow$ ϱ ist eine Wurzel von f' und mit Grad von f' gleich $s - 1$ ist das ein Widerspruch zu ϱ ist Wurzel eines irreduziblen Polynoms vom Grad s.

Ist F' ein Körper? Ja!

Die Addition führt nicht aus F'. Die Multiplikation könnte Schwierigkeiten bereiten: Das Produkt zweier Polynome aus F' wäre ein Polynom vom Grad $2s-2$. Diese Polynom wird einfach $\mathrm{mod}(a(\varrho))$ genommen. Dies ist sicher in F'. Die Problematik der Existenz der Inversen löst man mit dem Extended-Euklid-Algorithmus, siehe z. B. [CLR 90].

Also haben wir mit F' einen Körper mit $|F|^s$ Elementen gefunden, was unser Ziel war. Zu beachten ist noch folgendes: F' hat auch die Interpretation eines Vektoraums über F der Dimension s.

1.2 Komplexitätstheorie

Die **Komplexitätstheorie** ist ein Teilgebiet der Theoretischen Informatik, welches sich mit den Ressourcen beschäftigt, die ein *Algorithmus*[1] benötigt. Die Ressourcen sind einerseits die Platzbedürfnisse eines Algorithmus (intuitiv: wieviel einzelne Speicherzellen der Algorithmus benötigt) und andererseits die Zeitbedürfnisse eines Algorithmus (die *Laufzeit*), d. h. die Anzahl der ausgeführten Rechenschritte. Dieser Bedarf an Rechenzeit und Speicherplatz, zusammengefaßt die **Komplexität** des Algorithmus, wird natürlich von der "Größe" der Eingabe (d. h. von der Eingabelänge) abhängen. (Die Addition zweier 10-stelliger Zahlen benötigt offensichtlich mehr Platz und Zeit als die Addition zweier 3-stelligen Zahlen, oder: Für das Problem des Handlungsreisenden kann für 3 Städte die kürzeste Tour "von Hand" berechnet werden, während für 100 Städte der exakte Algorithmus auf dem derzeit schnellsten Computer Jahre bräuchte).

Auf der einen Seite wird versucht für gegebene, konkrete Probleme einen möglichst schnellen Algorithmus zu (er-)finden (oder einen bereits gefundenen zu verbessern); also die *obere Schranke* der Problem-Komplexität zu verkleinern. Auf der anderen Seite wird versucht, *untere Schranken* dieser Komplexität zu finden, was wesentlich schwieriger ist. (Dies verdeutliche man sich folgendermaßen: Die Aussage "Das Rucksack-Problem[2] hat mindestens die Komplexität K" schließt sämtliche Algorith-

[1] Unter einem Algorithmus versteht man eine Verarbeitungsvorschrift, die so präzise formuliert ist, daß sie von einem mechanisch oder elektronisch arbeitenden Gerät durchgeführt werden kann. Ein Algorithmus beschreibt eine Abbildung $f: E \to A$ von der Menge der zulässigen Eingabedaten in die Menge der Ausgabedaten.

[2] Maximiere den Wert eines größenbeschränkten Rucksacks, in den Gegenstände mit gegebenem diversen Wert/Grösse-Quotienten eingepackt werden.

men für das Rucksackproblem ein, auch bisher noch gar nicht gefundene !).

Vollkommen klar ist, daß unter der großen Zahl mathematisch formulierbarer Probleme die verschiedensten Schwierigkeitsgrade und die verschiedensten Laufzeiten auftreten. Dabei teilt die Komplexitätstheorie die Probleme in sog. **Komplexitätsklassen** ein.

Automaten und Sprachen

Sei Σ ein *Alphabet*, d. h. eine endliche, mindestens zwei-elementige Menge von Zeichen (Buchstaben); in der Komplexitätstheorie wird meist $\Sigma = \{0, 1\}$, das Boole'sche oder binäre Alphabet, benutzt. Die Menge aller *Wörter* oder *Strings*, die sich durch Hintereinanderreihung (Konkatenation) endlich vieler, nicht notwendig verschiedener Zeichen aus dem vorgegebenen Alphabet Σ ergeben, wird mit Σ^* bezeichnet. Hierin ist auch das *leere Wort* ϵ enthalten. Die *Länge* eines Wortes $x \in \Sigma^*$ wird mit $|x|$ bezeichnet. Man definiert nun: Jede Teilmenge von Wörtern $L \subseteq \Sigma^*$ heißt **Sprache** über Σ. Es ist also $\{0,1\}^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, ...\}$. Die Menge L der Binärdarstellungen von ganzen Zahlen (ohne führende Nullen) ist eine Sprache über {0,1}, denn $L = \{0, 1, 10, 11, 100, 101, ...\} \subseteq \{0,1\}^*$. Die Menge aller Sprachen über einem gegebenen Alphabet Σ ist gerade die Potenzmenge von Σ^* : $\mathcal{P}(\Sigma^*) = \{A \mid A \subseteq \Sigma^*\}$. Das *Komplement* einer Sprache L ist die Sprache $\bar{L} = \{x \in \Sigma^* | x \notin L\}$. Unter einer *Sprachklasse* $\mathcal{A} \subseteq \mathcal{P}(\Sigma^*)$ versteht man eine Menge von Sprachen. Das Komplement dieser Sprachklasse $\mathcal{A}$ wird co-$\mathcal{A}$ bezeichnet: co-$\mathcal{A} = \{A \subseteq \Sigma^* \, | \bar{A} \in \mathcal{A}\}$.

Unter einer (formalen) Sprache versteht man eine Sprache L zusammen mit einer Definitionsvorschrift, die im allgemeinen konstruktiv ist, d. h. sie entsteht aus einer gegebenen Grammatik oder aus einem gegebenen **Automaten**. Auf den Begriff der Grammatik wollen wir hier nicht näher eingehen.

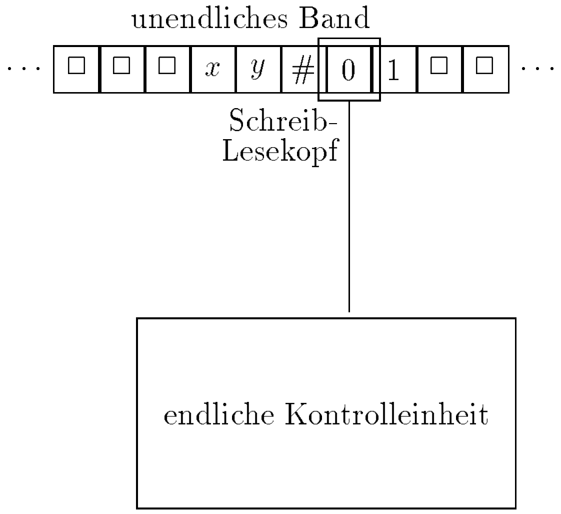
Sogenannte *Automaten* sind mathematische Modelle von Geräten, die Informationen verarbeiten und dabei Antworten auf Eingaben geben, wobei die Eingabe nur gelesen und nicht verändert werden darf. Automaten befinden sich immer in sog. Zuständen, die Informationen umfassen, die sich aus den bisherigen Eingaben ergeben, und die benötigt werden, um eine Reaktion des Modells auf noch folgende Eingaben zu bestimmen. Es gibt eine Vielzahl von verschiedenen Automatenmodellen, wir wollen hier aber nur auf den *endlichen Automaten*, der keinen Speicher hat, und speziell den *erkennenden, endlichen Automaten* (*Akzeptor*) näher eingehen.

Dieser wird auf ein Eingabewort angesetzt und *erkennt* (oder `akzeptiert`) dieses Wort schließlich, oder auch nicht; dann `verwirft` der Automat das Wort. Die Menge aller akzeptierten Wörter bildet die durch den Automaten M dargestellte oder definierte Sprache und wird mit $L(M)$ bezeichnet. Verallgemeinert man diese Automatentheorie, indem zusätzliche Speicher erlaubt werden, können ganz offensichtlich mehr Sprachen erkannt werden. Als Beispiel gilt der *Kellerautomat*, der einen last-in/first-out Speicher besitzt.

1.3 Turingmaschine

Um einen Algorithmus in irgendeiner Weise beschreiben zu können, benötigt man ein (Maschinen-)Modell. Wir sprachen oben von Speicherzellen, ohne genau zu wissen, was diese darstellen und zu welchem Modell diese gehören. Alan M. Turing[3] hat ein solches Modell entwickelt, das nur wenig "berechnungsstärker" als ein Kellerautomat ist. Turings Intention war noch viel weitreichender, nämlich eine mathematisch klar beschreibbare Maschine anzugeben, die allgemein genug ist, um stellvertretend für *jeden beliebigen* algorithmischen Berechnungsprozeß zu stehen.

Anschaulich beschrieben besteht eine **Turingmaschine** aus einem (potentiell) unendlichen Band, das in Felder eingeteilt ist. Jedes Feld kann ein einzelnes Zeichen des sog. Bandalphabets (oder: Arbeitsalphabet) der Maschine enthalten. Auf dem Band kann sich ein Schreib-Lesekopf bewegen. Nur solche Zeichen, auf denen sich der Kopf gerade befindet, können in dem momentanen Rechenschritt verändert werden. Der Kopf kann in einem Rechenschritt dann um eine Position nach links oder rechts bewegt werden.



[3] 1912-1954, englischer Mathematiker, Kryptoanalytiker und Computerkonstrukteur hat mit seinem 1937 erschienenen Werk "On computable numbers, with an application to the Entscheidungsproblem" die Theorie der Informatik begründet.

Definition 1.8 (Turing-Maschine (TM)) *Eine Turing-Maschine ist gegeben durch ein 7-Tupel*

$$M = (Z, \Sigma, \Gamma, \delta, z_0, \Box, E)$$

Hierbei sind

- Z die endliche *Zustandsmenge,*
- Σ das *Eingabealphabet,*
- $\Gamma \supset \Sigma$ das *Bandalphabet,*
- $\delta: Z \times \Gamma \to Z \times \Gamma \times \{L, R, N\}$ im deterministischen Fall (bzw. $\delta: Z \times \Gamma \to \mathcal{P}(Z \times \Gamma \times \{L, R, N\})$ im nicht-deterministischen Fall) die *Überführungsfunktion,*
- $z_0 \in Z$ der *Startzustand ,*
- $\Box \in \Gamma - \Sigma$ das *Blank,*
- $E \subseteq Z$ die Menge der *Endzustände.*

Informal erklärt bedeutet

$$\delta(z, a) = (z', b, x) \text{ bzw. } \delta(z, a) \ni (z', b, x)$$

folgendes: Wenn sich M im Zustand z befindet und unter dem Schreib-Lesekopf das Zeichen a steht, so geht M im nächsten Schritt in den Zustand z' über, schreibt (statt a) b auf das Band und führt danach die Kopfbewegung $x \in \{L, R, N\}$ aus. (Hierbei bedeute L=links, R=rechts, und N=neutral, also Stehenbleiben). Die *Konfiguration* ist der momentane Gesamtzustand einer Maschine; bei der Turingmaschine besteht eine Konfiguration also aus dem Feld, auf dem sich momentan den Schreib-Lesekopf befindet, dem Zeichen, das auf diesem Feld steht, und dem momentanen Zustand der TM. Eine Folge von Konfigurationen -diese sind durch δ festgelegt- nennt man *Berechnung.* Eine *akzeptierende Rechnung* ist eine zulässige Folge von Konfigurationen (beginnend bei der Startkonfiguration und endend bei einer Endkonfiguration) bei einer nicht-deterministischen Maschine. Das hat folgende Interpretation: Eine nicht-deterministischen Maschine darf eine Lösung *raten,* oder: falls bei allen möglichen Rechnungen eine akzeptierende darunter ist, wird akzeptiert.

Orakelmaschinen

Eine **Orakelmaschine** M ist eine mit einem speziellen Orakelband ausgestattete Turingmaschine, (sonst wie in Definition 1.8) deren Zustandsmenge Z zusätzlich die drei Zustände $z_?$ (*Fragezustand*), z_j (*ja-Zustand*) und z_n *(nein-Zustand)* enthält. Eine Orakelmaschine M kann an eine Orakelmenge $\mathcal{O} \subseteq \Sigma^*$ Fragen stellen, indem sie ein Wort y auf das Orabelband schreibt und anschließend in den Fragezustand $z_?$ übergeht. Die Frage wird nun dadurch beantwortet, daß das Orakelband gelöscht wird, und M im Fall $y \in \mathcal{O}$ in den Zustand z_j und andernfalls in den Zustand z_n übergeht. Um die Abhängigkeit von einem bestimmten Orakel auszudrücken, schreiben wir für die Maschine, die mit dem Orakel $\mathcal{O}$ wie eben beschrieben ausgestattet ist, $M^{\mathcal{O}}$, und für die erkannte Sprache $L(M, \mathcal{O})$, die von M *relativ* zur Orakelmenge $\mathcal{O}$ akzeptierte Sprache.

Probabilistische Turingmaschine

Bei dieser Modifikation der Turingmaschine wird die Folge der Konfigurationen durch ein Zufallsexperiment (Werfen einer Münze) mit zwei Ausgängen bestimmt. Dadurch ensteht ein (vollständiger) Berechnungsbaum, dessen Höhe die Laufzeit ist. Man definiert: Ein Wort x wird genau dann von einer **probabilistischen Turingmaschine** M `akzeptiert`, falls mehr als die Hälfte der Berechnungen von M, angesetzt auf x, im akzeptierenden Zustand enden.

1.4 Komplexitätsklassen

Wir sprachen vorher im Beispiel des Rucksackproblems von "Komplexität K", ohne darauf näher einzugehen; dies soll nun geschehen. Die Komplexität eines Algorithmus besteht im wesentlichen aus dessen Laufzeit bei Eingabe (Input) x (x ist ein String über $\Sigma = \{0, 1\}$) entweder im schlechtesten Fall (worst case Betrachtungen; im allg. beschränken wir uns auf diese) oder im mittleren Fall (average case). Diese verschiedenen Komplexitätsbetrachtungen müssen also alle möglichen Problemeingaben (instances) einbeziehen, auch eventuell sehr ungünstige, die eine lange Laufzeit des Algorithmus verursachen.

Eine sehr wichtige Frage, die sich nun stellt, ist: Hängen die Betrachtungen der

Laufzeit vom benutzten Modell ab? Die Antwort ist nein. Jedes natürliche Maschinenmodell (oder Berechnungsmodell), darunter fallen die Turingmaschine, die random access machine (RAM) und jedes andere Modell, mit dem die Berechnungsschritte formalisiert werden können, z. B. die Programmiersprache Modula-2, kann von jedem anderen effizient simuliert werden. Das bedeutet, der Begriff *polynomielle Laufzeit* hängt also nicht vom gewählten Modell ab, er ist davon unabhängig.

Wie schon bemerkt wurde ist die Turingmaschine so konzipiert, daß sie jeden beliebigen algorithmischen Prozess darstellen kann. In dieser Arbeit wird daher ausschließlich das Maschinenmodell der Turingmaschine gewählt. Um ein Entscheidungsproblem (Problem mit 2 Antwortmöglichkeiten: *ja/nein*) von einer Turingmaschine lösen zu lassen, d. h. bei einer bestimmten Problemeingabe soll die Turingmaschine `akzeptieren` oder `verwerfen`"entscheiden", muß die Problemeingabe zunächst in eine geeignete Kodierung (Bezeichnung:$\langle$instance$\rangle$) in Wörter über einem Alphabet Σ überführt werden. (Das Entscheidungsproblem E ist also äquivalent in eine Sprache L_E übersetzt worden: $L_E = \{\langle \text{instance} \rangle \mid \text{instance hat die geeignete Eigenschaft}\}$)

Diverse Laufzeiten

Bei den nun folgenden Betrachtungen ist zu beachten, daß der Algorithmus, dessen Laufzeit, durch $f : \mathbb{N} \to \mathbb{N}$ beschränkt ist, für alle n und für alle Inputs der Länge n höchstens $f(n)$ Berechnungsschritte benötigt.

Ist die *Laufzeit* f für alle Eingaben der Länge n bis auf einen konstanten Faktor c gleich n, d. h.

$$f(n) \leq c \cdot n = O(n),$$

so spricht man von *linearer Laufzeit.* Entsprechend bei

$$f(n) \leq c \cdot n^2 = O(n^2),$$

von *quadratischer Laufzeit*, bei

$$f(n) = p(n) = O(n^k) \quad p \text{ ein Polynom, } k \in \mathbb{N},\ c \cdot n^k \text{ sei das führende Glied von } p,$$

von *polynomieller Laufzeit* und bei

$$f(n) \leq c \cdot 2^{p(n)} = 2^{n^{O(1)}},$$

von *exponentieller Laufzeit*. Für eine Turingmaschine ist die Laufzeit als die Anzahl der Zustandsübergänge im Laufe der Berechnung definiert. Man sagt: Ein Algorithmus ist polynomial zeitbeschränkt, wenn er von einer deterministischen Turingmaschine in polynomieller Laufzeit berechnet werden kann. P ist die Komplexitätsklasse aller Sprachen (Probleme), die durch einen polynomial zeitbeschränkten Algorithmus entschieden werden können. Zur Vereinfachung werden zunächst zwei Basisklassen definiert.

Definition 1.9 (*TIME*, *NTIME*) *Sei $f : \mathbb{N} \to \mathbb{N}$ eine Funktion. Die Klasse TIME(f(n)) [bzw. NTIME(f(n))] besteht aus allen Sprachen, die von einer deterministischen [bzw. nicht-deterministischen] Turingmaschine M akzpetiert werden können mit $time_M(x) \leq f(|x|)$.*
Hierbei bedeute

$$time_M(x) : \Sigma^* \to \mathbb{N} \textit{die Anzahl der Rechenschritte von } M \textit{ bei Eingabe } x$$

im deterministischen Fall [bzw.

$$time_M(x) = \begin{cases} \min[\textit{Länge einer akzeptierenden} & \\ \quad \textit{Rechnung von } M \textit{ auf } x] & \textit{für } x \in T(M) \\ 0 & \textit{für } x \notin T(M) \end{cases}$$

im nicht-deterministischen Fall.]

Definition 1.10 (*P* (deterministic) polynomial time)

$$\begin{aligned} P &= \{A \mid \textit{es gibt eine det.Turingmaschine } M \textit{ und ein Polynom } p \textit{ mit} \\ &\qquad T(M) = A \textit{ und } time_M(x) \leq p(|x|)\} \\ &= \bigcup_{p \text{ ein Polynom}} \mathit{TIME}(p(n)) \end{aligned}$$

Algorithmen mit Laufzeiten, die schneller wachsen als Polynome werden nicht mehr als "schnell" angesehen, auch solche mit einem großen polynomiellen Grad (> 10) nicht, diese kommen in der Praxis aber nicht vor. Aus diesem Grund spricht man bei P auch von der Sprachklasse, die *effizient* berchnet werden kann.

Definition 1.11 (*NP* nondeterministic polynomial time)

$$\begin{aligned} NP &= \{A \mid \textit{es gibt eine nicht-det. Turingmaschine } M \textit{ und ein Polynom } p \textit{ mit} \\ &\qquad T(M) = A \textit{ und } time_M(x) \leq p(|x|)\} \\ &= \bigcup_{p \text{ ein Polynom}} \mathit{NTIME}(p(n)) \end{aligned}$$

Für diese Arbeit werden noch weitere Klassen benötigt:

Definition 1.12 (*EXP*, *NEXP*)

$$
\begin{array}{lcl}
EXP & = & \bigcup_{k\geq 1} TIME(2^{n^k}) \\
NEXP & = & \bigcup_{k\geq 1} NTIME(2^{n^k})
\end{array}
$$

Die *Polynomielle Hierarchie PH* eingeführt von Meyer und Stockmeyer [MS 72, Stock 77] besteht aus den folgenden Sprachklassen.

$$
\begin{array}{lcl}
\Sigma_0^p & := & \Pi_0^p := \Delta_0^p := P; \\
\Delta_{k+1}^p & := & P^{\Sigma_k^p},\ k \geq 0; \\
\Sigma_{k+1}^p & := & NP^{\Sigma_k^p},\ k \geq 0; \\
\Pi_{k+1}^p & := & \text{co-}NP^{\Sigma_k^p},\ k \geq 0; \\
PH & := & \bigcup\limits_{k\geq 0} \Sigma_k^p
\end{array}
$$

Definition 1.13 (*PSPACE*) *PSPACE ist die Menge aller Sprachen, die von einer polynomiell platzbeschränkten, deterministischen Turingmaschine erkannt wird.*

1.5 $P\stackrel{?}{=}NP$ -Problem

Es ist klar, daß $P \subseteq NP$ gilt; ob aber die beiden Komplexitätsklassen gleich oder echt ineinander enthalten sind, ist unbekannt. Diese Fragestellung ist seit 1971 bekannt und wird *$P\stackrel{?}{=}NP$ -Problem* genannt. Dies wird oft als die wichtigste ungelöste Frage der Theoretischen Informatik angesehen.

Die Bedeutung dieser Frage erklärt sich daraus, daß es viele für die Praxis wichtige Aufgabestellungen gibt, von denen man leicht nachweist, daß sie in *NP* liegen (z. B. Problem des Handlungsreisenden, gewisse Reihenfolgeprobleme, das Erfüllbarkeitsproblem der Aussagenlogik (SAT) u.v.a.). Polynomiale Algorithmen für diese Probleme sind aber nicht bekannt. Daher könnte es sich um mögliche Kandidaten für die Mitgliedschaft in $NP - P$ handeln, (was $P \neq NP$ bedeuten würde). Andererseits ist vielleicht noch niemand der richtige Algorithmus mit polynomialer Komplexität eingefallen, obwohl er existiert.

NP-Vollständigkeit

Ein zweiter Grund für die Bedeutung und das Interesse an dem $P\stackrel{?}{=}NP$ -Problem ist, daß es - insbesondere aufgrund der Arbeiten S. Cook [Cook 71] und R. Karp [Karp 72] - gelungen ist, eine sehr schöne "Strukturtheorie" für das $P\stackrel{?}{=}NP$ -Problem aufzubauen. Zentral ist dabei der Begriff der *NP-Vollständigkeit*. Dieses sind die "schwierigsten" Probleme in *NP*. Eine Sprache (Problem) ist *NP*-vollständig, falls es in *NP* liegt und alle anderen Sprachen (Probleme) in *NP* auf dieses polynomiell reduziert werden können.

Definition 1.14 (Reduktion) *Die polynomielle Reduktion (many-one-Reduktion) ist die Überführung (Transformation) einer Sprache L_1 in eine Sprache L_2, und zwar mittels einer in polynomieller Zeit berechenbarer Funktion f. Es gilt also symbolisch $L_1 \leq_m^p L_2$, falls $x \in L_1 \iff f(x) \in L_2$ für alle $x \in \Sigma^*$. Eine schwächere Form von Reduktion ist die Turing-Reduktion ($L_1 \leq_T^p L_2$), die durch $A \in P^B$ definiert ist.*

Es konnte gezeigt werden, daß alle (bis auf wenige Ausnahmen) der oben angesprochenen *NP*-Probleme, für die kein polynomialer Algorithmus bekannt ist, in einer Weise miteinander verknüpft sind, so daß entweder *alle* diese Probleme polynomiale Algorithmen besitzen (nämlich falls $P=NP$) oder *keines* (falls $P\neq NP$). Anders ausgedrückt: alle *NP*-vollständigen Probleme sind gleich schwierig zu lösen.

Da (z. T. seit Jahrhunderten) von verschiedenen Forschern unabhängig voneinander erfolglos versucht wurde, polynomiale Algorithmen für *NP*-Probleme zu finden, wird dies allgemein als starkes Indiz für $P \neq NP$ angesehen.

Um festzustellen, ob eine unbekannte *NP*-Sprache (Problem) L *NP*-vollständig ist, gibt es eine relativ einfache Beweistechnik: Kennt man eine *NP*-vollständige Sprache (und die gibt es wirklich, nämlich SAT, Satz von Cook [Cook 71]), so zeigt man einfach eine polynomielle Reduktion von SAT auf L.

1.6 Interaktive Beweise

Motivation

Um die **interaktiven Beweise** zu motivieren, werden zwei Beispiel gebracht, die, ohne auf die wichtigen Begriffe und Definitionen (siehe unten) näher einzugehen, trotzdem einen kleinen Einblick in die Interaktion geben.

Für das *Perfect Matching*-Problem (*NP*-vollständig) gibt es eine schöne Interpretation. Am Hofe **König Arthurs** stellte sich irgendwann im Lauf der Geschichte eine gewisse Unzufriedenheit ein. Grund dieser Mißstimmung war wie so oft die leidliche Frage der Mann-Frau-Beziehungen, speziell die Heiratsfrage. 150 unverheiraten Rittern standen 150 ledigen Burgfräuleins gegenüber, was ja eigentlich gar kein Problem ist. Die Schwierigkeit kam erst auf, als König Arthur erfuhr, daß Spezialwünsche bestanden, das heißt jeder Ritter und jedes Burgfräulein nur einen, zwei, oder gar mehrere ganz bestimmte Partner im Sinn hatten, was nach modernen sozialpsychologischen Aspekten auch völlig in Ordnung geht. Das Glück, das Arthur beschienen war, bestand zum einen aus der Royal Secret Agency, die in Erfahrung bringen kann, wer wen heiraten will, und zum andern aus dem Zauberer **Merlin** (kann *NP*-vollständige Probleme in der Zeit $O(1)$ lösen!), der mit den Informationen der Royal Secret Agency eine Zusammenstellung von 150 Paaren ausfindig machen kann, sofern dies die oben genannten Wünsche überhaupt zuließen. Wie es der Zufall wollte, hatte Merlin schon nach kurzer Zeit das Ergebnis parat, welches den König gar nicht froh stimmen sollte. Merlin behauptete nämlich, es gebe keine passende Zusammenstellung, er habe alle 150! Möglichkeiten durchprobiert. Der König konnte dieses in seinem Zorn natürlich gar nicht glauben und wollte einen Beweis sehen, den er schnell nachvollziehen könne. Daraufhin präsentierte Merlin dem König eine Menge von N Burgfräuleins, deren Herz aber nur für $N-1$ Ritter brannte, was König Arthur akzeptieren mußte und ihn zwang über neue Gesellschaftssysteme nachzudenken.

Dies war ein sehr einfaches Beispiel eines interaktiven Beweises, da nur eine Botschaft vom Verifier (König Arthur) zum Prover (Merlin) und umgekehrt gebraucht wurde und Wahrscheinlichkeiten keine Rolle spielten.

Ein weiteres Problem am Hofe König Arthurs stellte sich als alle 150 Ritter zur gleichen Zeit anwesend waren und am größten runden Tisch gemeinsam tafeln wollten. Einige der Herren trugen ihre Fehden bis in die Festmahle, was gewisse Schwierigkeiten für den Platzanweiser mit sich brachte, denn zwei Ritter, die sich nicht vertragen, sitzen besser nicht nebeneinander. Das Problem ist isomorph zum Finden eines *Hamilton*-Kreises in einem gegebenem ungerichteten Graphen, oder anders gesagt zum Entscheidungsproblem "Besitzt ein Graph einen Hamilton-Kreis?"; dies ist ein weiteres *NP*-vollständiges Problem. Wieder hat Merlin schnell eine Lösung gefunden (er hat ja unglaubliche Schnellrechenkräfte): die Ritter können nicht reibungsfrei am Tisch plaziert werden. Wieder muß Merlin dem König beweisen, was er herausgefunden hat. Doch diesmal ist dies nicht so einfach, denn niemand weiß bis heute, wie das in polynomieller Zeit geschehen könnte, der König hat ja nur wenig Zeit, wie schon bemerkt. Deswegen muß Merlin auf die nächstbeste Vorgehensweise zurückgreifen, nämlich Arthur wenigstens zu überzeugen, daß *wahrscheinlich* keine geeignete Sitzordnung existiert.

Nun wird das Problem als Graph dargestellt. Die Ritter sind die Knoten, die möglichen Sitzpartner sind mit Kanten verbunden, d. h. nur zwischen nicht verfeindeten Rittern eine Kante. Nun muß man wissen. daß die Permanente der Adjazenzmatrix dieses Graphen (mit 150 Knoten) $2^{43\,200\,000} \times$ (# der Hamilton-Kreise in diesem Graphen) ergibt, deshalb reduziert sich das Problem auf das Berechnen der Permanente (Determinante ohne alternierendes Vorzeichens bei der Summenberechnung) einer Matrix. Um die Berechnungsschwierigkeit der Determinante oder Permanente einzusehen, führe man sich nochmals die Definition vor Augen: $\mathrm{Per}(A) = \sum\limits_{\sigma} \prod\limits_{i=1}^{n} a_{i,\sigma(i)}$, σ sind die Permutationen von $\{1, ..., n\}$. Dabei ist zu beachten: Es gibt $n! \approx 2^{n(\log n - 1)} 2^{\frac{1}{2}\log n} = 2^{n \log n - n + \frac{1}{2}\log n} = 2^{O(n^2)}$. In diesem Fall also $150! \approx 2^{330} \approx 10^{100}$ Permutationen.

Um zu zeigen, wie nun Merlin versucht, Arthur von dem Wert einer Permanenten einer im allg. $n \times n$- Matrix zu überzeugen, folgt ein Zahlenbeispiel und zwar eine 3×3-Matrix M mit Elementen aus $\mathbb{N}$.

$$\text{Sei} \quad M = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}$$

Merlin rechnet blitzschnell aus: $\mathrm{per} M = 15$.

> Darauf Arthur: Soso, 15 ist das Ergebnis, aber wie soll ich wissen, daß dies richtig ist?

Merlin berechnet nun eine 2×2 Matrix $D(x)$ mit Elementen aus $\mathbb{Q}[x]$ (Polynomring, Variable ist x). Er weiß nämlich folgendes: Nach *Laplace* gilt für $A_{n\times n}$: $\mathrm{Det}(A) = \sum\limits_{i=1}^{n} a_{ik} A_{ik}$ wobei A_{ik} die Adjunkte. Also gilt: $\mathrm{Per}(A) = \sum\limits_{i=1}^{n} a_{ik}|A_{ik}| = \sum\limits_{i=1}^{n} a_{ki}|A_{ki}|$ für ein $k \in \{1, ..., n\}$. (Entwickeln nach der k-ten Spalte). Für $k = 1$ ergibt sich also $\mathrm{Per}(M) = \sum\limits_{i=1}^{3} m_{i,1}|M_{i1}| = \sum\limits_{i=0}^{2} m_{i+1,1} M_{i+1,1}$, wobei $M_{i+1,1}$ eine 2×2-Matrix ist. (Im allg. eine $(n-1) \times (n-1)$-Matrix).

Mit Hilfe der *Lagrange-Interpolation* läßt sich leicht die Matrix $D(x) : \mathbb{Q}[x]^{2\times 2}$ angeben, sodaß $D(i) = M_{i+1,1}$ für $i = 0, 1, 2$ gilt.

Zum Beispiel muß $p_{1,2} = D_{(1,2)}$ folgendes erfüllen:
$$\begin{array}{rclcl} p_{1,2}(0) & \stackrel{!}{=} & (M_{1,1})_{(1,2)} & = & 1 \\ p_{1,2}(1) & \stackrel{!}{=} & (M_{2,1})_{(1,2)} & = & 5 \\ p_{1,2}(2) & \stackrel{!}{=} & (M_{3,1})_{(1,2)} & = & 5 \end{array}$$

$$\stackrel{\text{Lagr.-Int.}}{\Longrightarrow} p_{1,2}(x) = \sum_{i=0}^{2} y_i \frac{\prod_{k=0}^{2} (x_i - x_k)}{\prod_{\substack{j=0\\ j\neq i}}^{2} (x_i - x_j)} = 1\frac{(x-1)(x-2)}{0-1)(0-2)} + 5\frac{x(x-2)}{(1-0)(1-2)} + 5\frac{x(x-1)}{(2-0)(2-1)}$$

$$= \frac{1}{2}(x^2 - 3x + 2) - 5(x^2 - 2x) + \frac{5}{2}(x^2 - x) = 1 + 6x - 2x^2 \quad \in \mathbb{Q}[x]$$

Zusammenfassend haben wir nun

$$\mathrm{Per}(M) = \sum_{i=0}^{2} m_{i+1,1} \cdot \mathrm{Per}(D(i)) \ , \tag{1.2}$$

und mit $p(x) := \mathrm{Per}(D(x)) = 1 - \frac{5}{2}x - \frac{7}{2}x^2 + 4x^3 - x^4$

$$\mathrm{Per}(M) = p(0) + 2p(1) + p(2) \tag{1.3}$$

Zu beachten ist, daß der Grad von $p(x)$ höchstens $(n-1)^2$ ist; im Bsp. ist 4 der Grad von p. Merlin schickt nun

$$D(x) = \begin{pmatrix} 1 - 3x - x^2 & 1 + 6x - 2x^2 \\ -\frac{1}{2}x + \frac{1}{2}x^2 & 1 \end{pmatrix} \quad \text{und} \quad p(x) = 1 - \frac{5}{2}x - \frac{7}{2}x^2 + 4x^3 - x^4$$

an Arthur.

> Schön und gut, meinte Arthur, nachdem er verifiziert hatte, daß dieses Polynom p tatsächlich $p(0) + 2p(1) + p(2) = 1 + 6 + 8 = 15$ ergab. Woher soll ich wissen, daß Merlin nicht lügt und gar nicht das richtige Polynom geschickt hat? Denn es ist nicht so schwierig, ein Polynom, welches obiges erfüllt, zu finden. Arthur macht nun folgendes: er zieht eine Zufallszahl $r \in \{1, ..., N\} \subseteq \mathbb{N}$ und bittet Merlin, $\mathrm{Per}(D(r))$ auszurechnen.

Merlin bekam jetzt also die gleiche Aufgabe wie zuvor, nur ist die Matrix, deren Permanente er ausrechnen soll, nicht mehr eine $n \times n$ sondern eine $(n-1) \times (n-1)$ Matrix; im Zahlenbeispiel eine 2×2 Matrix. Dies erledigt er ebenso wie zuvor und schickt wieder $D(x)$ und $p(x)$ (p ist im Bsp. nun eine lineare Funktion) an Arthur.

> Arthur bekommt nach n sog. *Runden* die letzte Nachricht von Merlin, im Beispiel ist dies jetzt der Fall. $D(x)$ ist nun eine 1×1 Matrix, deren Permanente selbst er blitzschnell hat und damit auch feststellen kann, ob Merlin das richtige Polynom geschickt hat.

Da $p(x)$ ein Polynom höchstens vom Grad $(n-1)^2$ für eine $n \times n$ Matrix ist, kann das von Merlin geschickte Polynom, sollte es falsch sein, mit dem richtigen an höchstens $(n-1)^2$ Stellen übereinstimmen. Wenn r aus einer Menge der Größe N, mit N genügend groß, gewählt wird, gibt das Arthur eine große Gewißheit, daß Merlin mit dem Polynom die Wahrheit gesagt hat. Falls Merlin in irgendeiner Runde lügen sollte, wäre der Wert der Permanente der 1×1 Matrix am Schluß der Unterhaltung falsch. Sollte auch am Schluß ein richtiger Wert herauskommen, hat Merlin Arthur von dem Wert der Permanente einer bestimmten Matrix mit einer gewissen Wahrscheinlichkeit überzeugt. Dabei ist klar, daß diese Wahrscheinlichkeit von n und N abhängt. In unserer netten Geschichte weiß Arthur jetzt also, daß für seine Ritter sehr wahrscheinlich keine streitlose Sitzordnung existiert; er muß also den Streit schlichten oder einen neuen Tisch bauen lassen.

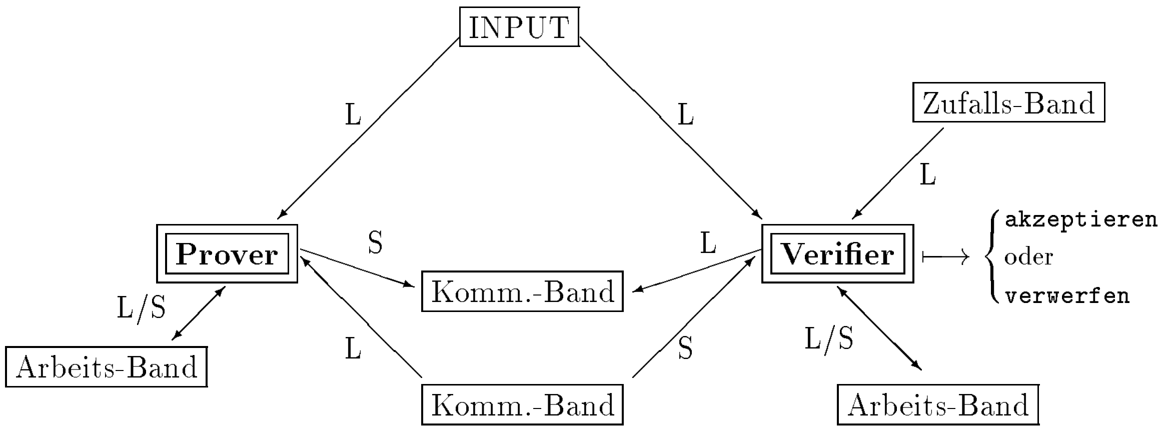
Interaktive Turingmaschinen, Protokolle und Beweissysteme

Hier sollen die exakten Definitionen der Interaktive Turingmaschinen und der Interaktiven Protokolle gebracht werden, da sie in der Arbeit sehr häufig benutzt werden.

Definition 1.15 (Interaktive Turingmaschine (ITM)) *Eine ITM ist eine Turingmaschine mit Zugriff auf ein read-only Input-Band, ein Zufalls-Band, ein Arbeitsband, ein read-only Kommunikations-Band und ein write-only Kommunikationsband. Das Zufalls-Band enthält eine unendliche Folge von Zufalls-Bits und kann nur von links nach rechts gelesen werden. Man sagt die Maschine "wirft eine Münze", was soviel heißt, daß sie das nächste Bit auf dem Zufalls-Band liest.*

Definition 1.16 (Interaktives Protokoll) *Ein Interaktives Protokoll ist ein, wie in folgendem Schaubild, geordnetes Paar von ITM's, genannt* **Prover** *und* **Verifier**. *Der Prover hat unbegrenzte Rechenzeit, während die des Verifiers durch ein Polynom in der Länge des Inputs beschränkt ist. Die beiden Maschinen sind jeweils nacheinander aktiv, der Verifier beginnt. In der Aktivphase einer Maschine werden von dieser*

Maschine alle zur Verfügung stehenden Bänder benutzt; am Ende der Aktivphase wird eine Nachricht auf das Schreibe-Kommunikations-Band geschrieben. Während diese Maschine mit dem "Abschicken" der Nachricht in die Passivphase übergeht, tritt die andere in den Aktivphase und darf die soeben geschriebene Nachricht auf seinem Lese-Kommunikationsband mitbenutzen. Jede Maschine darf den Prozess beenden, indem sie keine Nachricht mehr schickt. Schließlich `akzeptiert` *oder* `verwirft` *der Verifier den Input x.*



Dies war die allgemeine Definition eines Interaktiven Protokolls. Ein Interaktives Beweissystem (Interaktives Protokoll für eine Sprache L) wird im nächsten Kapitel definiert, Definition 2.2. Das Akzeptanzverhalten hängt selbstverständlich von den Münzwürfen ab. Als solche können die Einträge auf dem Zufallsband nämlich angesehen werden.

Kapitel 2

Das MIP=NEXP -Resultat

2.1 Vorbemerkungen

Seit den frühen 70er Jahren ist bekannt, daß Sprachen $L \in NP$ einen effizienten Zugehörigkeitsbeweis besitzen. Zu jedem Wort x einer Sprache L gibt es ein Zertifikat y, mit welchem *effizient*, also in polynomieller Zeit, die Zugehörigkeit ($x \in L$) verifiziert werden kann. In anderen Worten: es existiert ein effizienter Algorithmus, der mit der Eingabe (x, y) die Zugehörigkeit von x zu L bestätigt. [Cook 71].

Dieses Konzept wurde 1985 von Babai [Babai 85] und Goldwasser et al. [GMR 85] zu interaktiven Beweissystemen (interactive proofs) erweitert, die zugehörige Klasse heißt *IP* (exakte Definition auf Seite 28). Ein Prover-Verifier Paar (Prover und Verifier sind idealisierte Namen für Turingmaschinen, mit der Besonderheit von gemeinsamen Bändern, siehe Grundlagen) kommuniziert interaktiv, um einen Zugehörigkeitsbeweis zu verifizieren. Dabei hat der Prover unbeschränkte Rechenzeit, diejenige des Verifiers und die Anzahl der Kommunikationsschritte sind durch ein Polynom in der Länge der Eingabe beschränkt. Die Sprachen, welche ein solches Prover-Verifier Paar akzeptiert, gehören also zu *IP*.

Ein erstes wichtiges Resultat von Lund et al. [LFKN 90] war die Inklusion $PH \subseteq IP$; die Polynomielle Hierarchie ist also vollständig in *IP* enthalten, was die enorme Mächtigkeit von *IP* zeigt. Feldman schloß 1986 [Feldm 86], daß ein polynomiell platzbeschränkter Prover für *IP* ausreicht, formal $IP \subseteq PSPACE$; ein Ergebnis, das von Shamir nach Vorarbeit von Lund et al. [LFKN 90] 1990 zu $IP = PSPACE$ vervollständigt wurde [Shami 90].

In diesem Kapitel wird die von Ben-Or et al. [BGKW 88] eingeführte Klasse *MIP* genau untersucht und überraschende Ergebnisse dargestellt und bewiesen. *MIP* bedeutet multi-prover interactive proofs, also eine Erweiterung der Klasse *IP*, insofern, als daß nun nicht einer, sondern mehrere Prover zu Verfügung stehen. Diese Prover dürfen aber nicht miteinander kommunizieren und auch die Interaktionen des Verifiers mit den anderen Provern nicht einsehen. Fortnow zeigte in seiner Dissertation 1989 [Fortn 89], daß lediglich zwei Prover für eine Sprache in *MIP* ausreichen, formal 2-*IP* = *MIP*.

Die beste bekannte obere Grenze ist nicht-deterministische exponentielle Zeit (Fortnow et al. 1988 [FRS 88]). Hier wird nun gezeigt, daß diese obere Grenze scharf ist: *MIP* = *NEXP*. Dieses gravierende Resultat (als Babai-Fortnow-Lund Theorem bezeichnet) von Babai, Fortnow und Lund aus dem Jahre 1990 [BFL 91] benötigt in seinem Beweis viele verschiedene Techniken und ist daher sehr aufwendig zu beweisen. Es legt nach dem *IP* = $PSPACE$ Resultat die ungemein große Kraft der Randomisierung und der Interaktion dar.

Das Resultat sei hier als Theorem vorangestellt.

THEOREM 2.1 (Babai-Fortnow-Lund Theorem) *MIP = NEXP. In anderen Worten: Die Menge von Sprachen mit 2-Prover-Beweissystemen ist identisch mit der Menge der in nicht-deterministischer exponentieller Zeit berechenbaren Sprachen.*

Es folgen die nötigen Vorarbeiten:

- Einführen der Sprachklassen, 2.2,
- die Arbeitsweise des LFKN-Protokolls, 2.3.1, und seine Orakelversion,
- wir finden eine *NEXP*-vollständige Sprache in 2.3.3,
- Vorstellung eines Multilinearitätstests 2.4.2,
- der eigentliche Beweis mittels eines BFL-Protokolls in 2.4.3.

2.2 Multi-Prover-Protokolle und Probabilistische Orakelmaschinen

In diesem Abschnitt werden grundlegende Definitionen und Eigenschaften der Multi-Prover-Protokolle und der Probabilistischen Orakelmaschinen (ausgehend von Ben-Or et al. bzw. Fortnow et al. [BGKW 88, FRS 88]) dargelegt. Zunächst wird das *MIP*-Konzept vorgestellt:

Definition 2.2 (*IP*, *MIP*) *Seien $P_1, ..., P_k$*[1] *ITM's mit unbegrenzter Rechenzeit und V eine probabilistische polynomiell-zeitbeschränkte ITM, die sämtlich auf ein Eingabeband (read-only) zugreifen. Der Verifier V unterhält mit jedem einzelnen Prover ein Kommunikationsband; verschiedene Prover haben keinen Zugriff auf gemeinsame Bänder, außer auf das Eingabeband.*

$P_1, ..., P_k$ und V bilden ein Multi-Prover-Interaktivprotokoll für die Sprache L, wenn ein $\delta \in (0,1)$ existiert, sodaß

1. *für $x \in L$ folgt :* $\Pr$[2]$[P_1, ..., P_k$ *überzeugen V zu* `akzeptieren`$] = 1$

2. *für $x \notin L$ folgt für alle Prover $P_1^*, ..., P_k^*$:* $\Pr[P_1^*, ..., P_k^*$ *überzeugen V zu* `akzeptieren`$] \leq 1 - \delta$.

MIP ist die Klasse von Sprachen, die ein solches Protokoll besitzten. Für k=1 ergibt sich die Klasse IP.

Formal kann jeder Prover P_i als Funktion angesehen werden, die aus dem Input x und der bisherigen Konversation mit dem Verifier einen Output an V zurückschickt. An die Komplexität der Funktion P_i werden keine Anforderungen gestellt, die Ausgabe muß allerdings durch $p(|x|)$ beschränkt sein.

Die Wahrscheinlichkeiten sind willkürlich gewählt. Eine äquivalente Definition wäre eine solche mit den Wahrscheinlichkeiten $\frac{1}{2} + \epsilon$ statt 1 und $\frac{1}{2} - \epsilon$ statt $1 - \delta$ für ein $\epsilon \in (0, \frac{1}{2})$, denn aufgrund der Wahrscheinlichkeitsverstärkung folgt, daß mit entsprechend vielen Wiederholungen und anschließender Majoritäts-Entscheidung jede

[1] k darf ein Polynom $p(n)$ nicht überschreiten, wobei $|x| = n$.

[2] Die Münzwürfe des Verifiers (bzw. die Strings auf dem Zufallsband) bilden den hier benützten Wahrscheinlichkeitsraum.

beliebige Wahrscheinlichkeit erreicht werden kann. Dies gilt im allgemeinen bei allen probabilistischen Turingmaschinen, also auch im Rest der Arbeit.

Bemerkung: Zur sprachlichen Handhabe wird folgende (spieltheoretische) Terminologie vereinbart: $P_1, ..., P_k$ seien *ehrliche* Prover, jede Zusammenstellung von Provern aus 2. von Definition 2.2 wird *unehrlich* genannt; diese Prover versuchen zu *betrügen*. Im Fall $x \notin L$ sollen die ehrlichen Prover nicht versuchen, V von x zu überzeugen, sondern einfach ein Spezialsymbol zur sofortigen Ablehnung senden. Der Ausdruck, die Prover *gewinnen*, soll besagen, daß der Verifier akzeptiert, sollte er dieses nicht tun, *verlieren* die Prover. Unehrlichen Provern wird im Falle $x \in L$ auch erlaubt zu gewinnen, sie können aber auch mit hoher Wahrscheinlichkeit trotz $x \in L$ verlieren.

Für die beiden wichtigen Theoreme in diesem Kapitel wird noch folgende Definition benötigt.

Definition 2.3 (*POM*) *Sei M eine probabilistische, polynomiell-zeitbeschränkte Turingmaschine mit Zugriff auf ein Orakel $\mathcal{O}$. Die Sprache L wird von M* `akzeptiert` *genau dann, wenn ein $\delta \in (0,1)$ existiert, sodaß*

1. *für jedes $x \in L$ ein Orakel $\mathcal{O}$ existiert, sodaß $M^{\mathcal{O}}$ mit einer Wahrscheinlichkeit 1 akzeptiert.*
2. *für jedes $x \notin L$ und für jedes Orakel $\mathcal{O}'$ akzeptiert $M^{\mathcal{O}'}$ mit einer Wahrscheinlichkeit $\leq 1 - \delta$.*

Die zugehörige Klasse sei mit POM bezeichnet.

Dieses Modell kann so interpretiert werden: Im Gegensatz zum gewöhnlichen Modell des interaktiven Protokolls, wo der Prover seine Antworten von früheren Fragen abhängig machen kann, darf hier das Orakel seine Antworten nicht von früheren Fragen abhängig machen. Man könnte sagen das Orakel muß alle möglichen Antworten des Provers aus dem obigen Modell auflisten, bevor das Protokoll beginnt! Dieser scheinbar kleine Unterschied ist für den offensichtlich riesigen Mächtigkeitsanstieg von *IP* = *PSPACE* zu *MIP* = *NEXP* verantwortlich. Das Orakel kann also als sehr langer Beweis eines Theorems angesehen werden, den der Verifier schnell checken kann.

Für den Beweis des Babai-Fortnow-Lund Theorems benötigen wir die zu *MIP* äquivalente Klasse *POM*; diese Äquivalenz ist ein Ergebnis von [FRS 88].

THEOREM 2.4 (Fortnow, Rompel, Sipser; 1988) *POM=MIP, oder: L wird von einer probabilistischen Orakelmaschine akzeptiert genau dann, wenn sie ein Multi-Prover-Interaktivprotokoll besitzt.*

Ein weiteres wichtiges Resultat sagt aus, daß zwei Prover immer ausreichen :

THEOREM 2.5 (Ben-Or, Goldwasser, Kilian, Widgerson; 1988) *2-IP=MIP, oder: jede Sprache L, die von einem Multi-Prover-Interaktivprotokoll akzeptiert wird, besitzt ein 2-Prover-Interaktivprotokoll.*

Die Beweise beider Theoreme können sehr elegant zu einem zusammengefaßt werden.

Beweis: (von 2.4 und 2.5)

1. Zuerst wird $POM \subseteq 2\text{-}IP$ (1) gezeigt, indem eine probabilistische Orakelmaschine von einer interaktiven 2-Provermaschine simuliert wird.

Sei $L \in POM$ und M eine probabilistische Orakelmaschine, die in der Zeit n^c läuft. (Für eine Konstante c). Alle Orakelfragen werden vom Verifier an Prover 1 gestellt, davon eine per Zufall herausgepickt. Diese Frage wird dann an Prover 2 gestellt und danach `akzeptiert`, falls die beiden Antworten von Prover 1 und Prover 2 identisch sind und der Verifier mit den Antworten von Prover 1, anstatt den Antworten des Orakels akzeptiert hätte, ansonsten `verworfen`. Folgendes gilt für das Akzeptanzverhalten in beiden Fällen:

(i) Für jedes $x \in L$ existiert ein Orakel $\mathcal{O}$, sodaß $\Pr[M^{\mathcal{O}}$ akzeptiert $x] = 1$
Die ehrlichen Prover werden antworten wie das ehrliche Orakel, die Wahrscheinlichkeit für Akzeptanz ist also ebenso gleich 1.

(ii) Sei nun $x \notin L$.Eine der höchstens n^c Orakelfragen, die an Prover 1 gestellt wurden, wird nun an Prover 2 gestellt. Akzeptiert wird nur, falls beide Antworten übereinstimmen. Falls *alle* Orakelfragen von beiden Provern identisch beantwortet werden, ist die Wahrscheinlichkeit für `akzeptieren` höchstens δ. (Nach Definition POM)

Angenommen, die beiden Prover antworten bei mindestens einer Frage unterschiedlich. Die Wahrscheinlichkeit, daß eine dieser Fragen ausgewählt wird, ist mindestens n^{-c}.

Die Wahrscheinlichkeit, daß Prover 1 abweicht und der Verifier dies nicht merkt, ist also $1 - n^{-c}$. Wiederholt man diesen Prozess n^c (also polynomiell) mal, wird die Irrtumswahrscheinlichkeit kleiner als $(1-n^{-c})^{n^c}$. Nun gilt für genügend große $|x|$:

$$(1 - \frac{1}{n^c})^{n^c} \approx \frac{1}{e} \leq \delta^* := \frac{1}{2}$$

Aus **(i)** und **(ii)** folgt: Definition 2.2 ist erfüllt.

2. Hier wird $MIP \subseteq POM$ (2) gezeigt. Angenommen, L wird von einem Multi-Prover-Interaktivbeweissystem mit Verifier V akzeptiert. Dann wird M wie folgt definiert: M simuliert V indem alle Botschaften vermerkt werden. Wenn V die j-te Botschaft an den i-ten Prover schickt, stellt M an das Orakel die Fragen $(x, i, j, l, \beta_{i1}, ..., \beta_{ij})$, $1 \leq l \leq$ Länge der j-ten Botschaft, an P_i, geeignet codiert. Die Antwort des Orakels ergibt dann das l-te Bit der j-ten Botschaft von Prover i, wobei x der Input ist und $\beta_{i1}, ..., \beta_{ij}$ die ersten j Botschaften des Verifiers für den Prover i darstellen. M akzeptiert genau dann, wenn V akzeptiert. Nun gilt

(i) Seien $P_1, ..., P_k$ ehrliche Prover. Wenn $\mathcal{O}$ das Orakel ist, welches in oben angegebener Weise die Botschaften der Prover $P_1, ..., P_k$ codiert, dann wird $M^{\mathcal{O}}$ jedes $x \in L$ mit derselben Wahrscheinlichkeit akzeptieren.

(ii) Angenommen, $x \notin L$ und es existiert ein Orakel $\mathcal{O}$' mit $\Pr[M^{\mathcal{O}}$ akzeptiert $x] \geq 1 - \delta$, für ein $\delta > 0$. Dann könnten $P'_1, ..., P'_k$ konstruiert werden, die V mit derselben Wahrscheinlichkeit überzeugen, indem einfach $\mathcal{O}$' benützt wird (im obigen Sinne), um ihre Botschaften zu kreieren. Da nach Definition keine solchen Prover $P'_1, ..., P'_k$ existieren, kann es ein solches Orakel $\mathcal{O}$' auch nicht geben.

Aus **1.** und **2.** folgt $POM = 2\text{-}IP = MIP$. □

2.3 Das LFKN-Protokoll und eine *NEXP*-vollständige Sprache

Das Zentrum dieses Abschnitts steht das LFKN-Protokoll, das zur Verifizierung exponentiell großer Summen entwickelt wurde. Dazu benötigt wird die **Arithmetisierung Boole'scher Formeln**, d. h. die Umwandlung einer Boole'schen Formel in einen arithmetischen Ausdruck, was für den Beweis des Babai-Fortnow-Lund Theorems, Abschnitt 2.4, von grundlegender Bedeutung ist. Folgender Trick liegt dabei zugrunde: die Ausgangs- und die Zielmenge werden "aufgebläht", die Summe kann über einen weitaus größeren Bereich betrachtet werden. Da über Arithmetisierungen summiert wird, und da diese Arithmetisierungen Polynomen entsprechen, kann die entsprechende Theorie angewandt werden.

Das interaktive LFKN-Protokoll (siehe Beweis zu Lemma 2.6) wird im Lauf der Arbeit noch mehrmals verwendet. Hier speziell für ein Problem aus *co-NP*. Es folgt die Orakel-Version des LFKN-Protokolls, siehe Lemma 2.9, welches direkt im BFL-Protokoll verwendet wird.

Danach wird noch eine *NEXP*-vollständige Sprache vorgestellt, welche im Beweis des Babai-Fortnow-Lund Theorems, Abschnitt 2.4 nach üblichem Beweismuster verwendet wird.

2.3.1 Eine Variante des LFKN-Protokolls

LFKN steht für Lund, Fortnow, Karloff und Nisan [LFKN 90], deren Protokoll von Babai und Fortnow [BF 91] bzw. Shamir [Shami 90] weiterentwickelt wurde. Hier wird eine Variante benützt.

In diesem Unterkapitel steht das Lemma 2.6 im Zentrum, die ein interaktives Protokoll für ein Problem in *co-NP* beschreibt. Der Prover muß den Verifier von der Nicht-Erfüllbarkeit einer gegebenen Boole'schen Formel B überzeugen.

Sei nun f die Arithmetisierung von B, deren Existenz von Lemma 1.7 (siehe Grundlagen) garantiert wird. Wir sehen hier f als Erweiterung eines Polynoms in einen unendlichen Körper an. Der Prover muß den Verifier überzeugen, daß f bei allen (0,1)-Folgen verschwindet. Diese Bedingung für 2^m verschiedene Möglichkeiten der Belegung kann kurz so zusammengefaßt werden:

$$\sum_{x_1=0}^{1} \sum_{x_2=0}^{1} \cdots \sum_{x_m=0}^{1} (f(x_1, ..., x_m))^2 = 0 \tag{2.1}$$

Beachte die Quadrate, da sich +1 und -1 sonst gegenseitig zu 0 auslöschen könnten.

Eine allgemeine Gleichung dieser Art (siehe (2.2)) wird nun durch das LFKN-Protokoll verifiziert, wobei zu beachten ist, daß das Polynom h explizit dem Verifier und dem Prover gegeben ist. Sei $\mathrm{Grad}_h(x_i) = d,\ x_1, ..., x_m$, d. h. der Grad in jeder Variablen ist d, der Totalgrad höchstens md.

$$\sum_{x_1=0}^{1}\sum_{x_2=0}^{1}\cdots\sum_{x_m=0}^{1} h(x_1,...,x_m) = a \tag{2.2}$$

Diese Gleichung sagt aus, daß, falls h die Arithmetisierung einer Boole'sche Formel B ist, B genau a erfüllende Belegungen besitzt. (Im Falle $a = 0$ also ein *co-NP* Problem). Für das Schließen des Beweises beachte man, daß das Protokoll über einen beliebigen Körper F arbeitet.

Lemma 2.6 *Sei h eine arithmetische Formel und $a \in \mathbb{N}_0$. Das Entscheidungsproblem: "Erfüllt das Paar (h, a) die Gleichung (2.2) " gehört zu IP.*

Beweis: Sei $I \subseteq F$ mit $|I| \geq 2dm$ eine genügend große Teilmenge eines Körpers F. Für $i = 1, ..., m$ werden folgende Partialsummen betrachtet:

$$h_i(x_1,...,x_i) := \sum_{x_{i+1}=0}^{1}\cdots\sum_{x_m=0}^{1} h(x_1,...,x_m) = 0 \tag{2.3}$$

h_i ist ein Polynom in den ersten i Variablen, h_m sei h. Folgende Rückwärts-Rekursion ergibt sich direkt aus (2.3)

$$h_{i-1} = h_i(x_i = 0) + h_i(x_i = 1) \qquad i = 1, ..., m \tag{2.4}$$

Das Protokoll das (2.2) verifiziert durchläuft m Runden. r_i, $i = 1, ..., m$, seien Zufallszahlen aus I. $g_i(x)$, $i = 1, ..., m$ sind Polynome in einer Variablen (die i-te von h), wobei in h die ersten $i-1$ Variable mit Zufallszahlen belegt sind, die Variablen $i+1$ bis m alle (0,1)-Folgen durchlaufen und diese 2^{m-i-1} Polynome in x aufsummiert werden.
$\tilde{g}_i(x)$ sind Polynome, die der Prover liefert, wenn vom Verifier nach $g_i(x)$ gefragt wird. b_i sind Fixwerte, die sich aus den r_i und $\tilde{g}_i(x)$ ergeben.

Jetzt folgt das *Protokoll*:

Beginn des *Protokolls* mit **Eingabe:** $\boldsymbol{h(x_1,...,x_m),\ a}$

1\. Runde P: $g_1(x) = h_1(x) = \sum_{x_2=0}^{1} \cdots \sum_{x_m=0}^{1} h(x, x_2, ..., x_m)$

Wählt (die Koeffizienten von) $\tilde{g}_1$

$P \to V$: $\boxed{\tilde{g}_1}$

V: KONSISTENZTEST: $a = b_0 \stackrel{?}{=} \tilde{g}_1(0) + \tilde{g}_1(1)$

rät r_1 und berechnet damit $b_1 = \tilde{g}_1(r_1)$

$V \to P$: $\boxed{r_1}$

2\. Runde P: $g_2(x) = h_2(r_1, x) = \sum_{x_3=0}^{1} \cdots \sum_{x_m=0}^{1} h(r_1, x, x_3, ..., x_m)$

Wählt (die Koeffizienten von) $\tilde{g}_2$

$P \to V$: $\boxed{\tilde{g}_2}$

V: KONSISTENZTEST: $b_1 \stackrel{?}{=} \tilde{g}_2(0) + \tilde{g}_2(1)$

rät r_2 und berechnet damit $b_2 = \tilde{g}_2(r_2)$

$V \to P$: $\boxed{r_2}$

$\vdots$

j. Runde P:

$$\begin{aligned} g_j(x) &= h_j(r_1, ..., r_{j-1}, x) \\ &= \sum_{x_{j+1}=0}^{1} \cdots \sum_{x_m=0}^{1} h(r_1, ..., r_{j-1}, x, x_{j+1}, ..., x_m) \end{aligned}$$

Wählt (die Koeffizienten von) $\tilde{g}_j$

$P \to V$: $\boxed{\tilde{g}_j}$

V: KONSISTENZTEST: $b_{j-1} \stackrel{?}{=} \tilde{g}_j(0) + \tilde{g}_j(1)$

rät r_j und berechnet damit $b_j = \tilde{g}_j(r_j)$

$V \to P$: $\boxed{r_j}$

$\vdots$

m. Runde P: $g_m(x) = h_m(r_1, ..., r_{m-1}, x) = h(r_1, ..., r_{m-1}, x)$

Wählt (die Koeffizienten von) $\tilde{g}_m$

$P \to V$: $\boxed{\tilde{g}_m}$

V: KONSISTENZTEST: $b_{m-1} \stackrel{?}{=} \tilde{g}_m(0) + \tilde{g}_m(1)$

rät r_m und berechnet damit $b_m = \tilde{g}_m(r_m)$

FINAL TEST: $b_m \stackrel{?}{=} h(r_1, ..., r_m)$

Ausgabe: Der Verifier `akzeptiert`, wenn alle KONSISTENZTESTS und der FINAL TEST bestanden sind; andernfalls `verwirft` er.

Ende des Protokolls.

Beweis der Korrektheit des Protokolls

1. Angenommen, (2.2) gilt. Dann wird der ehrliche Prover immer $\tilde{g}_i = g_i$ zurückschicken können und immer gewinnen. (D. h. 1. aus der *IP*-Definition 2.2 ist erfüllt).

2. Zunächst beachte man folgendes: ein unehrlicher Prover möge in einer bestimmten Runde betrügen, d. h. $\tilde{g}_{i-1} \neq g_{i-1}$. Dann gilt $\Pr_{r_{i-1} \in I}[\tilde{g}_{i-1}(r_{i-1}) \neq g_{i-1}(r_{i-1})] \geq 1 - \frac{d}{|I|}$, da Polynome vom Grad d an nicht mehr als als d Stellen den gleichen Wert haben können, wenn sie nicht identisch sind. Wenn $\tilde{g}_i$ den Konsistenztest $b_{i-1} = \tilde{g}_i(0) + \tilde{g}_i(1)$ bestehen soll, muß der Prover in der nächsten Runde mit der Wahrscheinlichkeit von mindestens $1 - \frac{d}{|I|}$ auch betrügen; also $\tilde{g}_i \neq g_i$.

   Falls nun (2.2) nicht gilt, unterscheidet sich a von h_0, d. h. der Prover muß in der Runde 0 betrügen: $\tilde{g}_0 = b_0 = a \neq h_0 = g_0$. (Alles Konstanten). Damit folgt: der Prover muß in jeder Runde mit Gesamtwahrscheinlichkeit $\geq 1 - \frac{md}{|I|} \geq \frac{1}{2}$ betrügen. In der letzten Runde wird er aber durch den FINAL TEST entlarvt. Es gilt also Pr[unehrlicher Prover überzeugt den Verifier] $\leq \frac{1}{2} = \delta$, also ist 2. aus der *IP*-Definition 2.2 auch erfüllt. □

Bemerkung: Der Trick in diesem Protokoll bestand darin, daß der betrügende Prover, sehr wahrscheinlich in jeder Runde betrügen muß und zwar mit einem Polynom, das immer weniger Variable hat, bis am Ende eine Konstante erreicht wird.

Nun sollen Beispiele folgen, welche die Arbeitsweise des LFKN-Protokolls verdeutlichen. Dazu wird das multilineare Polynom des Beispiels auf Seite 11 verwendet.

Beispiele: Im ersten Beispiel soll eine richtige Eingabe zum Verifizieren führen.
Eingabe[3]**:** $\boldsymbol{h(x_1, x_2, x_3) = 1 - x_3 + x_1x_3 - x_1x_2 + x_2x_3 - x_1x_2x_3,\ a = b_0 = 5.}$

1.Runde	P:	$g_1(x) = h_1(x) = \sum\limits_{x_2=0}^{1}\sum\limits_{x_3=0}^{1} 1 - x_3 + xx_3 - xx_2 + x_2x_3 - xx_2x_3$
		$= \sum\limits_{x_2=0}^{1} (1 - xx_2) + (x + x_2 - xx_2 - xx_2)$
		$= 1 + x + (1 - x) + x + 1 - x - x = \ 3 - x$
	P $\rightarrow$ V:	$\boxed{3 - x}$
	V:	KONSISTENZTEST: $5 \stackrel{?}{=} 3 - 0 + 3 - 1\ \surd$
		$r_1 = 7, b_1 = -4$
	V $\rightarrow$ P:	$\boxed{7}$
2.Runde	P:	$g_2(x) = h_2(7, x) = \sum\limits_{x_3=0}^{1} (1 + 7x_3 + xx_3 - 7xx_3 - 7x - x_3)$
		$= 1 - 7x + 7 - 13x = \ 8 - 20x$
	P $\rightarrow$ V:	$\boxed{8 - 20x}$
	V:	KONSISTENZTEST: $-4 \stackrel{?}{=} 8 + 8 - 20\ \surd$
		$r_2 = 3, b_2 = -52$
	V $\rightarrow$ P:	$\boxed{3}$
3.Runde	P:	$g_3(x) = h_3(7, 3, x) = 1 + 7x + 3x - 21 - 21x - x = \ -12x - 20$
	P $\rightarrow$ V:	$\boxed{-12x - 20}$
	V:	$-52 \stackrel{?}{=} -20 - 12 - 20\ \surd$
		$r_3 = -2, b_3 = 4$
		FINAL TEST: $b_3 \stackrel{?}{=} h(7, 3, -2) = 4\ \surd \quad \rightarrow$ `akzeptieren`

Kommentar: Der Prover hatte keine Probleme; er konnte immer das richtige Polynom schicken, also $g_i \equiv \tilde{g}_i,\ 1 \leq i \leq 3.$ $\diamond$

Jetzt noch ein interessanteres Protokoll, das mit einer falschen Eingabe startet. Die Eingabe sei jetzt manipuliert: statt wie oben den korrekten Wert für b_0 zu nehmen, wird nun b_0 auf 1 gesetzt.

[3]Also $d = 1,\ m = 3,\ 2dm = 6$, es sei $\mathbb{N} \supseteq I = [-9, ..., 10] \supseteq \{1, ..., 6\}$. Bei der Vereinfachung von Arithmetisierungen Boole'scher Funktionen gilt immer $d = 1$

Eingabe[4]**:** $\boldsymbol{h(x_1, x_2, x_3) = 1 - x_3 + x_1x_3 - x_1x_2 + x_2x_3 - x_1x_2x_3,\ a = b_0 = 1}$.

1.Runde	$P \to V$:	$\boxed{1-x}$
	V:	$1 \stackrel{?}{=} 1+1-1\surd$
		$r_1 = 7, b_1 = -6$
	$V \to P$:	$\boxed{7}$
2.Runde	$P \to V$:	$\boxed{2x-4}$
	V:	$-6 \stackrel{?}{=} 2-4-4\surd$
		$r_2 = 3, b_2 = 2$
	$V \to P$:	$\boxed{3}$
3.Runde	$P \to V$:	$\boxed{6x-2}$
	V:	$2 \stackrel{?}{=} 6-2-2\surd$
		$r_3 = -2, b_3 = -14$
		FINAL TEST $-14 \stackrel{?}{=} h(7,3,-2) = 4 \quad \to$ `verwerfen`.

Kommentar:

1. Runde Der Prover kann jetzt mit der Wahrscheinlichkeit von mindestens $\frac{19}{20}$ nicht mehr das korrekte Polynom $3-x$ schicken, da lineare Polynome an nicht mehr als als einer Stelle den gleichen Wert haben können. Er schickt nun einfach ein anderes, das den KONSISTENZTEST besteht, also

$$a_1 + b_1x\,|_{x=0} + a_1 + b_1x\,|_{x=1} = 2a_1 + b_1 \stackrel{!}{=} 1$$

erfüllt, zum Beispiel $1-x$ oder x. Dies ist für den Prover kein Problem; Nun wählt der Verifier seine erste Zufallszahl: $r_1 = 7$. Das richtige g_1 hatte an r_1 den Wert -4 Das hier gelogene $1-x$ besteht zwar den KONSISTENZTEST gibt aber bei r_1 nicht -4 sondern -6. Die Wahrscheinlichkeit, daß der Verifier die Zufallszahl wählt, die bei $1-x$ den Wert -6 ergibt, ist gleich $\frac{1}{20}$, denn nur die $5 \in [-9, ..., 10]$ würde dies erfüllen. Genau eine Zahl aus 20.

2. Runde Wie in der ersten Runde, muß der Prover hier mit der Wahrscheinlichkeit von mindestens $\frac{19}{20}$ lügen; in unserem Beispiel schickt er das falsche $2x-4$, welches den Konsistenztest natürlich besteht.

3. Runde wie in zweiter Runde.

[4] I wie oben.

Resumee: Da der Prover in jeder Runde mit der Wahrscheinlichkeit von höchstens $\frac{1}{20}$ nicht betrügen muß, ist die Gesamtwahrscheinlichkeit, daß er nicht betrügen muß höchstens $\frac{3}{20}$. Da er bei Betrug sowieso durch den FINAL TEST entlarvt wird, ist die Wahrscheinlichkeit für einen unehrlichen Prover, der nicht entlarvt wird, $\leq \frac{3}{20} = \delta$.

$\diamond$

2.3.2 Orakel-Polynome und Orakel-Protokolle

In Lemma 2.6 soll die Voraussetzung, daß h explizit in einer arithmetischen Formel gegeben ist, nun abgeschwächt werden, danach wird das LFKN-Protokoll in modifizierter Form nochmals durchlaufen. Die Werte von h soll der Verifier stattdessen durch Orakelbefragung erhalten. Um dies zu formalisieren wird das Konzept der *interaktiven Orakelprotokolle* entwickelt.

Definition 2.7 (Interaktives Orakelprotokoll) *Ausgehend vom 2-Prover-Interaktivprotokoll (siehe Definition 2.2) wird lediglich der zweite Prover durch ein Orakel ersetzt.*

Der zweite Prover ist also nichtadaptiv, d. h. er kann vorhergehende Kommunikation nicht verwerten. Das Protokoll ist somit ein Ein-Prover-Protokoll, bei dem der Verifier auf ein Orakel zugreifen kann. Der Prover will den Verifier in polynomieller Zeit von einer bestimmten Eigenschaft des Orakels überzeugen.

Geringe Veränderungen des Orakels können vom Verifier nicht bemerkt werden, da der Verifier nur auf polynomiell viele Bits des Orakels zugreifen kann, das Orakel aber exponentiell viele Bits zur Verfügung hat. Wenn der Verifier allerdings "gutartige" Approximationen von h ermittelt, können diese für das Protokoll verwendet werden. Gutartig soll heißen: Polynome mit kleinen Graden in jeder Variablen. Die Schlüsselidee besteht darin, daß solche Polynome einen error-correcting code bilden. (Vgl. Bemerkung).

Definition 2.8 (δ-Approximation) *Seien f und g Funktionen, die auf einer endlichen Menge X definiert sind, und sei $\delta \in [0,1]$. f δ-approximiert g, wenn die Anzahl der $x \in X$ mit $f(x) \neq g(x)$ kleiner ist als $\delta|X|$.*

Lemma 2.9 *Seien $d, m \geq 0, k \geq 3$, ein Körper F und eine Teilmenge $I \subseteq F$ mit $|I| = kdm$ (Input) gegeben.*
Angenommen das Orakel akzeptiert Fragen der Form $(r_1, ..., r_m)(r_i \in I)$ und antwortet mit einem Element $h(r_1, ..., r_m) \in F$ von polynomieller Länge. Dann existiert ein interaktives Orakelprotokoll mit folgenden Eigenschaften:

(i) *Wenn h ein Polynom mit $Grad_h = d$ in allen Variablen ist, und (2.2) gilt, dann akzeptiert der Verifier mit Wahrscheinlichkeit 1*

(ii) *Wenn h von f auf I^m δ -approximiert wird (überdies gelte $Grad_f = d$ in allen Variablen und $\delta \leq \frac{1}{4}$) und ein Prover existiert, der mit Wahrscheinlichkeit $\geq \delta + \frac{1}{k}$ akzeptiert wird, dann gilt (2.2) mit f statt h.*

Bemerkungen:

- Die Eigenschaft (ii) könnte auch so lauten: Falls (2.2) mit einer δ-Approximation von h nicht gilt, kann man folgern, daß der Test (siehe Beweis) mit einer Wahrscheinlichkeit von mindestens $1 - \delta - \frac{1}{k}$ verwirft.

- Das Lemma sagt aus, daß die Gültigkeit von (2.2) in polynomieller Zeit und mit hoher Sicherheit verifiziert werden kann, wenn h in jeder Variablen klein-gradig ist; und selbst wenn h von einem Polynom f, das in jeder Variablen klein-gradig ist, nur angenähert[5] wird.

- Dies hat Bezug zur sog. error-correcting-Eigenschaft, d. h. Fehler in h werden automatisch verbessert, oder anders ausgedrückt, die Fehler in h werden als solche nicht wahrgenommen, der Test verläuft so, als wären sie nicht vorhanden. Die Grundlage bildet ein Lemma von Jakob Schwartz (1980):
  Ein multivariates Polynom ($\not\equiv 0$) mit Totalgrad $\leq d^*$ verschwindet höchstens an einem $\frac{d^*}{|I|}$ Bruchteil der Definitionsmenge I^m (m=Anzahl der Variablen).

- Es folgt, daß eine klein-gradige Korrektur von h eindeutig ist, falls sie existiert:

  Beweis: Seien f_1 und f_2 δ-Approximationen von h mit $\mathrm{Grad}_{f_1}, \mathrm{Grad}_{f_2} \leq d$ in allen Variablen; weiter sei $f = f_1 - f_2$. Dann gilt $\mathrm{Grad}_f = d$ an allen Variablen und f verschwindet nur an einem 2δ Bruchteil von I^m nicht.

  Daraus folgt mit Schwartz $f \equiv 0$ außer $1 - 2\delta \leq \frac{dm}{|I|}$ (dm ist der Totalgrad von f). Für $k > 2$ ergibt sich $1 - 2\delta < \frac{1}{2}$, was für $\delta \leq \frac{1}{4}$ unmöglich wird,

[5] f ist auf mindestens $\frac{3}{4}$ der Definitionsmenge mit h identisch, oder f hat ein Viertel der Wertemenge einfach ausgetauscht

d. h. $f \equiv 0 \Leftrightarrow f_1 \equiv f_2$ und somit gilt
für $\delta \leq \frac{1}{4}$ ist eine klein-gradige δ-Approximation von h eindeutig.

Beweis : [von Lemma 2.9] Das LFKN-Protokoll im Beweis von Lemma 2.6 wird folgendermaßen modifiziert: Im FINAL TEST fragt der Verifier das Orakel anstatt $h(r_1, ..., h_m)$ selbst zu berechnen.

(i) gilt nach 1. im Beweis von Lemma 2.6. Der ehrliche Prover gewinnt immer.

(ii) Sei f die (eindeutige) δ-Approximation von h mit $\text{Grad}_f = d$ in allen Variablen. Angenommen (2.2) wird von f nicht erfüllt. Der Konsistenztest aus dem Beweis von Lemma 2.6 wird nun den Prover zwingen in jeder Runde zu betrügen; die Gesamtwahrscheinlichkeit dafür ist $\geq 1 - \frac{dm}{|I|} = 1 - \frac{1}{k}$. Wenn der Prover in jeder Runde betrügen muß, bleibt ihm nur noch die Möglichkeit, daß sich f und h bei $(r_1, ..., r_m)$ unterscheiden. Dafür ist die Wahrscheinlichkeit höchstens δ. Also wird der Prover mit Wahrscheinlichkeit $\geq 1 - (\delta + \frac{1}{k})$ entlarvt. Die Definition des Orakelprotokolls ist also erfüllt. □

Die starke Annahme, daß h in jeder Variablen einen kleinen Grad hat oder zumindest eine gute Approximation eines solchen Polynoms ist, wird mit einem modifizierten Multilinearitätstest siehe Lemma 4.8 in Kapitel 4.3 geprüft werden.

2.3.3 Eine *NEXP*-vollständige Sprache

Zunächst einige Konventionen bezüglich der Notation. Kleinbuchstaben x, b, f, t, z, w werden für Variablenstrings, kleine, griechische Buchstaben $\xi_i, \beta_i, \varphi_i, \tau_i, \zeta_i, \omega_i$ für die dazugehörigen individuellen Variablen benutzt. Eine typische Variable im String b_j ist also β_{ji}. Diese Variablen sind entweder Boole'sch oder nehmen Werte aus einem Körper F an. Die Boole'sche Menge {0,1} wird als Teilmenge von F angesehen.

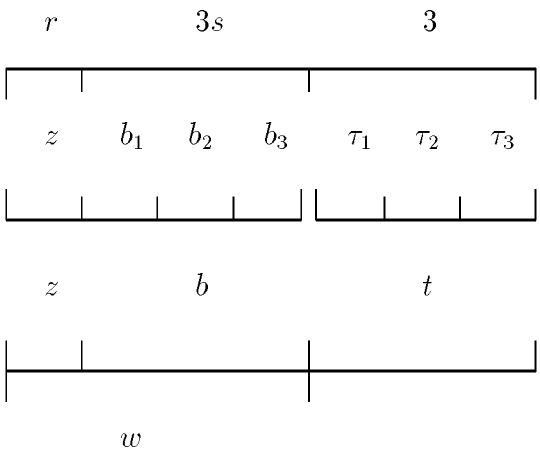
Definition 2.10 (3-SAT-Orakel, Orakel-3-erfüllbar, Orakel-3-SAT)
Seien $r, s \in \mathbb{N}, z$ und b_i (i=1,2,3) Variablenstrings mit $|z| = r$ und $|b_i| = s$. Zur Verkürzung: $b = b_1 b_2 b_3$ und $w = zb$. Desweiteren sei $t = \tau_1\tau_2\tau_3$ ein String mit 3 Variablen und $B(w, t)$ eine Boole'sche Formel (mit r+3s+3 Variablen).

(i) *Eine Boole'sche Funktion A in s Variablen ist ein 3-SAT-Orakel für B, wenn $B(w, A(b_1), A(b_2), A(b_3)) = 1$ für alle 2^{r+3s} Belegungen von w.*

(ii) *B ist Orakel-3-erfüllbar, wenn eine solche Funktion A existiert.*

(iii) *Orakel-3-SAT ist folgendes Entscheidungs-Problem: Ist B (wie oben) Orakel-3-erfüllbar?*

- Zum besseren Überblick über die Variablen folgende Skizze:



- Die Boole'sche Formel B könnte also wie folgt aussehen:
  $B(w,t) = (\zeta_2 \vee \beta_{2,5}) \wedge (\tau_3 \vee ...$
  mit $z = \zeta_1, ..., \zeta_r$ und $b_i = \beta_{i1}, ..., \beta_{is}$.

- Die Variablen haben folgende Interpretation: Die b_1, b_2, b_3 sind Strings der Länge s, die Variablen interpretieren. Die τ_1, τ_2, τ_3 geben das Vorzeichen dieser Variablen an: negiert oder nicht negiert. Das z der Länge r ist die Interpretation eines Klauselnamens.

  Das Orakel A soll nun für alle b_1, b_2, b_3 aus 2^{3s} , also für alle Belegungen, und für alle z aus 2^r, also für alle Klauseln die Variablen so belegen, daß B wahr wird.

- Intuitiv ist Orakel-3-SAT *NEXP*-vollständig, da das Orakel $A()$ die Berechnung einer *NEXP*-Turingmaschine repräsentiert.

Lemma 2.11 (*NEXP*-Version von Cook-Levin)
Orakel-3-SAT ist NEXP-vollständig.

Beweis: Orakel-3-SAT $\in$ *NEXP*. klar!

Bleibt die *NEXP*-Härte zu zeigen, d. h. $L \leq_m^p$ Orakel-3-SAT für alle $L \in NEXP$. Sei x eine Eingabe der Länge n. Nun wird in polynomieller Zeit in n eine Problemeingabe (B, r, s) von Orakel-3-SAT konstruiert, welche genau dann akzeptiert wird, wenn $x \in L$, also eine Reduktion von L (einer beliebigen Sprache in *NEXP*) auf Orakel-3-SAT (symbolische Abkürzung dafür: $L \leq_m^p$ Orakel-3-SAT).

Sei M eine *NEXP*-Turing Maschine, welche L akzeptiert. Wie im Beweis des Cook-Levin-Theorems [Cook 71, Levin 73] wird nun eine 3-CNF Φ_x[6] angegeben, für die gilt:

$$x \in L \iff \Phi_x \text{ erfüllbar.}$$

Die Anzahl N_v der Variablen sowie die der Klauseln N_c sind exponentiell. ObdA. sei $N_v = 2^s (s = n^c$ für eine Konstante c). Daher können wir die 2^s Variablen mit Binärstrings der Länge s beschreiben, und zwar mittels einer bijektiven Funktion X: $X : \{0,1\}^s \to \{\text{Variablen von } \Phi_x\}$.

Es gibt maximal 2^{3s+3} mögliche Klauseln mit 3 Literalen (ein Literal ist entweder x oder $\neg x$). Eine typische Klausel dieser Form ist

$$C(b, f, X) = (\varphi_1 \oplus X(b_1)) \vee (\varphi_2 \oplus X(b_2)) \vee (\varphi_3 \oplus X(b_3)) \tag{2.5}$$

wobei der 3-Bit String $f = \varphi_1\varphi_2\varphi_3$ den Zustand der Literale ($\varphi = 1$ für negierte Variable und $\varphi = 0$ für Variable) und der $3s$-Bit String $b = b_1b_2b_3$ die Variablen an sich codiert. ($\oplus$ bedeute Addition mod2).

Die Klauseln von Φ_x sind in polynomieller Zeit erkennbar; d. h. es gibt ein in polynomieller Zeit berechenbares Prädikat p, sodaß $C(b, f, X)$ eine Klausel von Φ_x g. d. w. $p(x, b, f) = 1$. Man folgert, daß $x \in L$ g. d. w. eine Boole'sche Funktion A existiert (erfüllende Belegung), sodaß für jedes $b \in \{0,1\}^{3s}$ und $t \in \{0,1\}^3$

$$D(x, b, f, A) := C(b, f, A) \vee \neg p(x, b, f) \tag{2.6}$$

wahr ist. (In anderen Worten: die Klauseln, welche gemäß p zu Φ_x gehören, müssen erfüllt sein, wenn die Variablen $X(b_i)$ durch die Boole'schen Werte $A(b_i)$ ersetzt werden).
Man beachte, daß $C(b, f, A)$ aus der expliziten Boole'schen Formel

$$B_1(f, t) := (\varphi_1 \oplus \tau_1) \vee (\varphi_2 \oplus \tau_2) \vee (\varphi_3 \oplus \tau_3) \tag{2.7}$$

[6] In dieser Boole'schen Formel stehen Variablen, die die Rand-, Anfangs-, Übergangs- und Endbedingungen beschreiben. Diese Bedingungen beziehen sich auf die Turingmaschine M und ihre Übergangsfunktion δ (von einer Bandinschrift zur nächsten). Diese Formel hängt selbstverständlich von der Eingabe x ab!

durch Substitution von τ_i durch $A(b_i)$ entsteht. In (2.6) stört nur noch $\neg p(x, b, f)$, welches in *NP* berechenbar ist und somit nach Cook-Levin eine äquivalente 3-CNF Formel B_2 besitzt. Diese 3-CNF Formel (in polynomieller Zeit in $|x|$- berechenbar) enthält die Variablenstrings b, f und u; der String u spielt die Rolle des Zeugen (von polynomieller Länge). Für feste Werte von b und f ist B_2 erfüllbar g. d. w. $p(x, b, f) = 1$.

Sei letztlich

$$B(u, b, f, t) := B_1(f, t) \vee \neg B_2(u, b, f) \tag{2.8}$$

Dies ist die Boole'sche Formel, die wir gesucht haben. (Zur Übereinstimmung mit Definition 2.10 sei $z = uf$ und r die Länge von z).
Wann gilt nun: $A : \{0,1\}^s \to \{0,1\}$ ist ein 3-SAT-Orakel für B ? Genau dann, wenn B für jedes $b = b_1 b_2 b_3$ von $\tau_i = A(b_i)$ -für alle möglichen Werte von z- erfüllt wird. Aber dies ist genau die Äquivalenz zu $x \in L$. □

2.4 Der Beweis des Babai-Fortnow-Lund Theorems

Dieses Kapitel widmet sich ausschließlich dem Beweis des Babai-Fortnow-Lund Theorems (THEOREM 2.1) aus [BFL 91]. Wie schon in der Einführung bemerkt, ist seit 1988 *MIP* $\subseteq$ *NEXP* bekannt. (Ergebnis von Fortnow, Rompel,Sipser [FRS 88]). Also muß hier noch die Inklusion *NEXP* $\subseteq$ *MIP* bewiesen werden. Dies gelingt mit einem *MIP*-Protokoll für eine *NEXP*-vollständige Sprache. Diese Sprache haben wir mit Orakel-3-SAT, siehe Lemma 2.11, bereits gefunden.

2.4.1 Die Arithmetisierung von *NEXP*

In Unterkapitel 2.3.3 wurde die *NEXP*-vollständige Sprache Orakel-3-SAT bereits vorgestellt Eine Eingabe (B, r, s), bzw. die darin vorkommende Boole'sche Formel B, wird nunzunächst arithmetisiert werden. Der Begiff Arithmetisierung ist so zu verstehen wie in 1.1 definiert. Es gelten die gleichen Variablensymbole wie in 2.3.3.

Lemma 2.12 *Gegeben eine Orakel-3-SAT Eingabe (B, r, s) (wobei $B = B(w, t)$ eine Boole'sche Formel mit r+3s+3 Variablen ist). In polynomieller Zeit kann ein Polynom mit ganzzahligen Koeffizienten berechnet werden, das mit dem arithmetischen*

Ausdruck zu identifizieren ist, welcher aus derselben Menge von $r + 3s + 3$ Variablen besteht und eine Arithmetisierung von B darstellt, sodaß folgendes gilt:
$\exists$ eine Funktion A mit: $A : \{0,1\}^s \to F$ bildet ein 3-SAT-Orakel für B g. d. w.

$$\sum_{w \in \{0,1\}^{r+3s}} g(w, A(b_1), A(b_2), A(b_3)) = 0 \tag{2.9}$$

Beweis Mit Lemma 1.7 erhält man Arithmetisierung von B. Das dazugehörige Polynoms sei f. Dann setze man

$$g(w,t) := (1 - f(w,t))^2 + (\tau_1(\tau_1 - 1))^2 \tag{2.10}$$

Die Summe in (2.9) verschwindet g. d. w. alle Terme in (2.10) verschwinden. Das Verschwinden aller zweiten Terme in (2.10) korrespondiert mit $A(b_1)(A(b_1)-1) = 0$. (Bedeutung: Alle Werte von A sind aus {0,1} und Quadrieren schließt eine Auslöschung gleicher positiver und negativer Werte aus). Angenommen, dies ist der Fall, bedeutet das Verschwinden aller ersten Terme in (2.10) genau $B(w, A(b_1), A(b_2), A(b_3)) = 1$.

□

2.4.2 Multilinearitätstest

Sei $A : \{0,1\}^s \to F$ die Funktion, die im Orakel gespeichert ist und
$h(w) := g(w, A(b_1), A(b_2), A(b_3))$. Die Aufgabe besteht darin, (2.9) zu verifizieren. Da wir das LFKN-Protokoll aus 2.3.1 zur Verfügung haben, muß A, wie in angegeben, zunächst in ein multilineares Polynom $\tilde{A}$ umgewandelt werden.

Diese multilineare Erweiterung bringt uns das Lemma 1.4 in den Grundlagen, Seite 7. Es ist zu beachten, daß $|F|$ groß genug ist, damit die Teilmenge von F den Quantitätsansprüchen des LFKN-Protokolls 2.6 genügt. Wir erinnern uns $|I| \geq 2dm$.

Multilinearität von A wird garantieren, daß die Funktion
$h(w) = g(w, \tilde{A}(b_1), \tilde{A}(b_2), \tilde{A}(b_3))$ ein Polynom mit kleinen Graden in allen Variablen ist und damit (2.9) verifiziert werden kann. Die Schwierigkeit dieses Verfahrens liegt darin, daß zunächst ein Multilinearitätstest durchgeführt werden muß.

Dazu sind folgende Definitionen und Notationen nötig. Sei $I = \{0, 1, ..., N-1\}$ und $\alpha_1, \alpha_2, ..., \alpha_{i-1}, \alpha_{i+1}, ..., \alpha_s \in I$. Dann ist

$$\ell := \{(\alpha_1, \alpha_2, ..., \alpha_{i-1}, \beta, \alpha_{i+1}, ..., \alpha_s) | \beta \in I\} \subseteq I^s$$

die durch die $\alpha_1, \alpha_2, ..., \alpha_{i-1}, \alpha_{i+1}, ..., \alpha_s$ festgelegte Gerade in I^s.

Die folgende Definition ist gleichwertig zur ersten Definition von *multilinear* in den Grundlagen.

Definition 2.13 (multilinear) *Sei $f\ : I^n \to F$ eine Funktion. F ist multilinear, falls*

$$\forall \ell \in I^n,\ f\,|_\ell \text{ linear ist.}$$

Definition 2.14 (δ -approximativ multilinear) *Sei $f\ : I^n \to F$ eine Funktion. Für $\delta \in [0,1]$ heißt f δ -approximativ multilinear, falls ein multilineares g existiert, welches f δ -approximiert.*

Multilinearitätstest:
Wähle m_2 zufällige Geraden L_i für alle i. Falls der Subtest alle diese Geraden akzeptiert, wird `akzeptiert`, sonst `verworfen`.

Subtest: (testet eine Gerade l)
Wähle zufällige $m_1 + 2$ Punkte auf l. `Akzeptiere`, falls A an diesen Punkten mit einer linearen Funktion übereinstimmt; andernfalls `verwerfen`.

Die Korrektheit dieses Tests, mit konstanten m_1, m_2, faßt folgendes Theorem (ohne Beweis) zusammen.

THEOREM 2.15 (BFL 91) *Seien $d, k \geq 1$ Konstanten und $N \in \mathbb{N}$, $s^{d+3} < N \leq 2^{s^k}$ für ein s. I sei die ganzzahlige Menge {0,1,...,N-1}. Desweiteren sei $A(\xi_1, ..., \xi_s)$ eine beliebige Funktion von $I^s \to F$. Dann gilt:*
Für eine Konstante k' existiert eine POM-Maschine M^A mit

1. *Wenn A multilinear ist und keine größern Werte als $2^{s^{k'}}$ annimmt, wird M^A immer akzeptieren.*

2. *Wenn A nicht s^{-d}-approximativ multilinear ist, verwirft M^A mit hoher Wahrscheinlichkeit.*

Bemerkungen:

- Es existiert auch ein modifizierter Multilinearitätstest, bei dem die Multilinearität von A durch "$\text{Grad}_A(x_i) \leq k_i$ für die Variablen $x_1, ..., x_n$ und $k_1, ..., k_n \in \mathbb{N}$ und kleiner als n^c" ersetzt wird. Dies werden wir später siehe Lemma 4.8 noch einmal brauchen.

- Die obere Grenze, die die multilinare Funktion nicht überschreiten darf, ist mit Korollar 1.6, siehe Grundlagen, zu erklären.
- In [FGLSS 91] werden neue effektivere Multilinearitätstests behandelt, die weniger Zufallsbits benötigen.

2.4.3 Das BFL-Protokoll

In diesem Unterkapitel steckt der eigentliche Beweis des Babai-Fortnow-Lund Theorems. Hier wird das *MIP*-Protokoll für die *NEXP*-vollständige Sprache formuliert, die wir ja bereits gefunden haben.

Sei $L \in NEXP$. Man muß nun ein *MIP*-Protokoll entwerfen, um $x \in L$ zu verifizieren. Wie in 2.3.2 beschrieben wird nun ein interaktives Orakel-Protokoll konstruiert werden.

Zur Verfügung stehen also ein Prover und ein Orakel. Dem Beweis von Lemma 2.11 entsprechend konstruiert der Verifier eine Eigabe (B, r, s) des Orakel-3-SAT Problems, die akzeptiert wird g. d. w. $x \in L$. Nach Lemma 2.12 kreiert dann der Verifier ein Polynoms g in $r + 3s + 3$ Variablen mit ganzzahligen Koeffizienten, sodaß $x \in L \iff$ Es existiert eine Boole'sche Funktion $A : \{0,1\}^s \to \{0,1\}$, die die Gleichung (2.9) erfüllt.

Nun werden Zuverlässigkeitsparameter gewählt: $\delta \leq \frac{1}{4}$ und $k \geq 4$, sodaß $\frac{1}{\delta} + k \leq n^{O(1)}$.
Man setzt $I = \{0, 1, ..., N-1\}$, wobei $N = kdm$, $d = \max_{i \in 1,...,r+3s+3} \mathrm{Grad}_g(x_i)$ und $m = r + 3s$. (Selbstverständlich sind diese Parameter von der Ordnung $n^{O(1)}$).

Das Orakel soll nun die Funktion A zu einer multilinearen Funktion $A' : I^s \to \mathbb{Z}$, erweitern. Das Orakel-Prover Paar will nun den Verifier davon überzeugen, daß die multilinearen Funktion $A' : I^s \to \mathbb{Z}$, eine multilineare Erweiterung eines 3-SAT Orakels für B ist.

Phase 1: Der MULTILINEARITÄTSTEST: Diese Phase erfordert keinen Prover; der Verifier stellt an das Orakel eine polynomiell begrenzte Anzahl von randomisierten Fragen entsprechend dem Multilinearitätstest. Verwirft der Verifier in dieser Phase wird $x \in L$ verworfen und das Protokoll ist damit beendet.

Phase 2: Falls der Multilinearitätstest mit Akzeptanz endet, geht das Protokoll in diese Phase. Sie beabsichtigt (2.9) zu verifizieren. Dies wird mittels des interaktiven Orakelprotokolls aus Lemma 2.9 (hier steckt das LFKN-Protokoll) erreicht und zwar mit der Funktion $h(w) = g(w, A(b_1), A(b_2), A(b_3))$.

Im FINAL TEST des LFKN-Protokolls wird $h(w)$ an einem einzigen zufälligen $w \in I^m$ ausgewertet. Der Verifier bewerkstelligt dies, indem er 3 Fragen an das Orakel stellt: $A(b_1), A(b_2)$ und $A(b_3)$ (in Phase 2 sind dies die einzigen Orakelfragen). Man beachte, daß diese 3 Stellen vom Verifier zufällig aus I^s ausgewählt werden. Gemäß des Ergebnisses von Phase 2 wird $x \in L$ `akzeptiert` oder `verworfen`.

Beweis der Korrektheit des Protokolls:
Im Falle $x \in L$ wird das ehrliche Prover-Orakel Paar natürlich immer gewinnen. Angenommen das Prover-Orakel Paar hat eine Gewinnchance von größer als $\delta + \frac{1}{k}$. Wir werden zeigen, daß daraus zwingend $x \in L$ folgt.

Zunächst gilt, das Orakel muß mindestens $\frac{\delta}{3}$-approximiert multilinear sein. Andernfalls wird schon in Phase 1 mit hoher Wahrscheinlichkeit verworfen. Sei A' eine multilineare Funktion, welche A $\frac{\delta}{3}$-approximiert.

Desweiteren sei $h(w) = g(w, A(b_1), A(b_2), A(b_3))$ und
$h'(w) = g(w, A'(b_1), A'(b_2), A'(b_3))$. Es ist klar, daß h' eine δ-Approximation von h ist und daß $\text{Grad}_{h'} \leq d$ in allen Variablen ist. Deshalb gilt nach Lemma 2.9, daß, wenn der Prover eine Gewinnchance von größer als $\delta + \frac{1}{k}$ hat, (2.9) mit A' statt A gilt. Das bedeutet, daß $A'\big|_{\{0,1\}}$ ein 3-SAT-Orakel für B ist. Damit ist $x \in L$ bewiesen.

Beweis: (von *MIP=NEXP*, Babai-Fortnow-Lund Theorem 2.1)
Orakel-3-SAT $\in$ *MIP* $\Rightarrow$ *NEXP* $\subseteq$ *MIP*. Mit dem Ergebnis *MIP* $\subseteq$ *NEXP* aus [FRS 88] folgt *MIP=NEXP*. □

Kapitel 3

Der Klein-Grad-Test

Der Klein-Grad-Test von Rubinfeld und Sudan [RS 92] steht innerhalb dieser Arbeit völlig unabhängig zu den den beiden anderen Ergebnissen. Ziel ist es, eine Funktion P, die durch eine Black Box gegeben ist, zu klassifizieren. Uns interssiert, ob sie zu einer bestimmte Klasse gehört, genauer, zu den Polynomen, welche einen kleinen Totalgrad haben.

Rubinfeld und Sudan entwickelten einen Test für P, der, falls er mit `verwerfen` endet, folgenden Schluß zuläßt: M it hoher Wahrscheinlichkeit ist P nicht mal dicht zu einem Klein-Grad-Polynom. Die Mittel die wir für diesen Test benötigen, sind zum einen die Self-Corrector Funktion, die P an einer Stelle berechnet, ohne die Black Box an dieser Stelle zu benützen; zum anderen das Matrix-Transpositionslemma, welches aussagt, daß man die "Grad d Polynom" -Eigenschaft der Matrix sozusagen transponieren kann

3.1 Die Testprozedur

Eine Funktion $P : F^n \to F$ (F ein Körper) soll getestet werden, und zwar, ob sie ein Polynom vom Totalgrad d (hier immer mit "Grad d Polynom" bezeichnet) ist, oder zumindest dicht zu einem. Man kann ohnehin nicht 100%-ig konstatieren daß die Funktion P ein Grad d Polynom P^* ist , denn dann müßte man das Polynom wirklich angeben können und alle Punkte aus F^n (also $|F|^n$ viele) in P und P^* eingesetzt müßten identisch sein. Diese Prozedur ist ob der Größe von $|F|$ nicht praktikabel. Offensichtlich muß man die Anzahl der Punkte, an denen getestet wird, reduzieren.

Daraus folgt sofort, daß nicht geschlossen werden kann :"P ist ein Polynom vom Totalgrad d", sondern allenfalls "P ist mit hoher Wahrscheinlichkeit δ^*-dicht zu einem P^* (mit Totalgrad$_{P^*} = d$)". Diese Schlußfolgerung bedeutet nach Definition

$$\Pr_{x \in F^n}[P(x) = P^*(x)] \geq 1 - \delta^*. \tag{3.1}$$

Die Fragen, die sich nun stellen, sind: Wieviele Punkte wählt man aus und wo liegen diese Punkte ? Nur $d+1$ Punkte auszuwählen wäre fatal, denn durch diese Punkte gibt es immer ein Polynom vom Grad d. (Dazu muß man nur ein lineares Gleichungssystem mit $(d+1)^n$ Variablen lösen. Die dazugehörige $(d+1)^n$-spaltige Matrix hat aber nur $d+1$ Zeilen $\Rightarrow$ für $n=1$ eindeutige Lösung, für $n>1$ keine eindeutigen Lösungen, also Variable sogar frei wählbar). Aus diesem Grund wählt man die Anzahl der Punkte entspechend groß: $10d+1$.

Zur zweiten Frage: Die $10d+1$ Punkte werden mit bestimmten konstanten Abständen auf einer Gerade gewählt, die "zufällig" im Raum liegt. Zufällig bedeute durch einen zufällig gewählten Aufpunkt $x \in F^n$ und eine zufällig gewählte Richtung $t \in F^n$. Man beachte, daß x und t unter Gleichverteilungsannahme und unabhängig gewählt werden. Durch die Körpereigenschaften von F ist garantiert, daß $x, x+t, ..., x+10dt \in F^n$. Mit dieser Auswahl von Punkten ist die (3.1) in einem globalen Sinn sehr gut angenähert; die Wahrscheinlichkeitsbetrachtung kann deswegen konkretisiert werden:

$$\text{Für } \mathcal{T} := \{x, x+t, ..., x+10dt\} \text{ sei} \tag{3.2}$$
$$\Pr_{x,t \in F^n}[\nexists \text{ Polynom } P^* \text{ mit } grad(P^*) = d, \text{ das an } \mathcal{T} \text{ mit } P \text{ übereinstimmt}] = \delta.$$

Man beachte, daß δ nicht bekannt ist. Der Klein-Grad-Tester für P ist ein Self-Tester, da nur P benutzt; er sieht so aus:

Klein-Grad-Tester für P:
Setze b :=falsch
Wiederhole m mal
  Wähle $x, t \in_{Gl} F^n$
  Falls $\nexists$ Grad d Polynom h, sodaß $\forall j \in \{0, 1, ..., 10d\}$,
  $h(x+jt) = P(x+jt)$ gilt, setze b:=wahr
Ende der Schleife
`Verwerfen`, falls b wahr ist, ansonsten `akzeptieren`

Die Konstante m hängt von der gewünschten Irrtumswahrscheinlichkeit ab, die mittels Chernoff's bound[1] so gewählt wird, daß δ kleiner oder gleich der Irrtumswar-

[1]abgeschätzte Grenze für bestimmte Bereiche der Binomialverteilung

scheinlichkeit ist. Der Test, ob eine Funktion ein Polynom auf äquidistanten Punkten auf einer Gerade ist, gelingt aufgrund folgender Beobachtung von Van Der Waerden [VanDe 70]: *Sei $f^1(x) = f(x) - f(x-t)$ und $f^i(x) = f^{i-1}(x) - f^{i-1}(x-t)$. f stimmt mit einem Grad d Polynom an den Punkten $x_1, ..., x_k$ dann und nur dann überein, falls die $(d+1)$-te Differenz $f^{d+1}(x_j) = 0$, für $j \in \{d+2, ..., k\}$.* Nach dieser Beobachtung muß man also lediglich noch f^i für $i \in [1, ..., d+1]$ berechnen, d. h. für diesen Test benötigt man $O(kd) = O(d^2)$ Subtraktionen.

Insgesamt hat der Test folgende Komplexitität: $O((\log |F|)^n d^2)$

Von nun an wird eine obere Grenze für δ angenommen, oder anders interpretiert, alle Funktionen, welche ein größeres als in 3.3 angegebenes δ haben, werden nicht mehr betrachtet.

$$\delta \leq \frac{1}{100(d+1)} \tag{3.3}$$

THEOREM 3.1 (Rubinfeld und Sudan) *Aus (3.2) und dem δ aus (3.3) folgt: Wird der Klein-Grad-Tester für P verworfen, kann man mit hoher Wahscheinlichkeit schließen, daß P nicht dicht zu einem Grad d Polynom ist.*

Der ganze Rest des Kapitels stellt den Beweis dieses Theorems dar.

3.2 Die Self-Corrector Funktion

Der Self-Corrector, den wir nun konstruieren, genauer die Self-Corrector Funktion für P, wird so genannt, weil wieder, wie beim Self-Tester, nur Werte von P verwendet werden.

Nun zu den Einzelheiten. Man kann zeigen, daß es möglich ist, ein Grad d Polynom g zu definieren mit $g(x) = P(x)$ für fast alle $x \in F^n$. Wie gesagt würde g berechnet werden, indem nur Werte von P benutzt werden. Falls P wirklich ein Programm ist, welches eine Polynom-Funktion berechnet und welches meistens genau ist, dann würde g die präzise Antwort sein, da g ein Polynom ist und mit P fast überall übereinstimmt. Die Prozedur, die nun beschrieben wird, benötigt nicht das Wissen, welches Grad d Polynom P vorgibt zu berechnen. Damit ist es so, als hätte P sich selbst korrigiert.

Angenommen, $x, t \in F^n$ seien gegeben. (Aufpunkt und Richtung). Sei nun $\phi(j) :=$

$P(x+jt)$. Die Self-Corrector Funktion sei definiert durch

$$SC^P(x,t) = \begin{cases} h(0) & \text{falls für ein Grad } d \text{ Polynom } h : h(j) = \phi(j) \\ & \forall j = \{1,2,...,10d\} \text{ erfüllt ist} \\ \texttt{error} & \text{sonst} \end{cases}$$

Also berechnet sich der Wert von SC^P, indem $10d$ Werte von P verwendet werden. Diese Werte liegen auf einer Geraden, der Aufpunkt x ist ausgenommen. Falls P wirklich ein Polynom auf den $10d$ Punkten dieser Geraden ist, würde gelten: $SC^P(x,t) = P(x)$.

Angenommen, der Wert von P an der Stelle x wäre verfälscht, aber alle anderen Werte von P erhalten. Dann würde $SC^P(x,t)$ trotzdem den originalen Wert bekommen, da es in seiner Berechnung $P(x)$ nicht ins Kalkül zieht. Gewissermaßen ist $SC^P(x,t)$ der korrigierte Wert von $P(x)$. Falls P bei mehreren Werten verfälscht ist aber dennoch (3.2) erfüllt ist, kann man die Original-Funktion zurückerhalten,indem

$$g(x) := \text{majority}_t\{SC^P(x,t)\}$$

gesetzt wird, da in den meisten Richtungen keine verfälschten Punkte liegen. So wird also für jeden Punkt x, $g(x)$ aus der Menge aller $SC^P(x,.)$-Werte genommen, und zwar jenen mit der größten Häufigkeit. (Bei gleicher Häufigkeit suche man einen beliebigen aus).

Somit ist die Methode g aus P zu berechnen offensichtlich. In der Praxis wird majority$\{.\}$ natürlich nur an konstant vielen Zufalls-Werten von t berechnet. Für den Self-Test muß man g nicht überall berechnen, es genügt, daß man g theoretisch berechnen könnte.

3.3 Das Matrix-Transpositionslemma und seine Anwendung

Es folgen das wichtige Matrix-Transpositionslemma (MTL) und ein weiteres Lemma aus [RS 92, GLRSW 91]. Die Anwendungen des MTL für den Zweck des Klein-Grad-Tests von Polynomen sind vielfältig und folgen im Anschluß.

Lemma 3.2 (Matrix Transpositions Lemma (MTL)) *Angenommen, man hat 2 Teilmengen eines Körpers $X = \{x_0, ..., x_{d+1}\}$ und $Y = \{y_0, ..., y_{d+1}\}$ gegeben, wobei*

die x_i und die y_j jeweils verschieden sind. X und Y bilden ein Gitter, wobei der Gitterpunkt (i,j) das 2-Tupel (x_i, y_j) ist. Auf diesem Gitter ist eine Funktion $\phi(x,y)$ definiert. Weiter sei für alle $d+2$ Zeilen $\phi(x_i, y)$ ein Grad d Polynom in y und für $d+1$ Spalten (allen außer einer) $\phi(x, y_j)$ ein Grad d Polynom in x.
$\Rightarrow$ *ϕ ist auch auf der vorher ausgenommenen Spalte ein Grad d Polynom.*

Beweis:

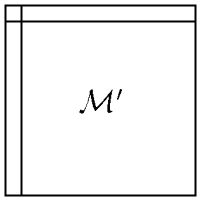
1. Man kann das Gitter auch folgendermaßen interpretieren: $(i,j) = x + x_i t_1 + y_j t_2 = m_{ij}$ mit festen aber zufälligen x, t_1, t_2 jeweils aus dem Körper. Das ergibt eine Matrix $\mathcal{M}_{(d+2)\times(d+2)}$ mit den Matrixelementen m_{ij}, für die folgende Voraussetzungen erfüllt sind:

$\forall i \in \{0, ..., d+1\} \exists$ ein Grad d Polynom $r_i : \phi(m_{ij}) = r_i(m_{ij}),\ \forall j \in \{0, ..., d+1\}$
und
$\forall j \in \{1, ..., d+1\} \exists$ ein Grad d Polynom $c_j : \phi(m_{ij}) = c_j(m_{ij}),\ \forall i \in \{0, ..., d+1\}$.
Es wird OBdA. die Spalte 0 ausgenommen.

2. Hier wird folgende Feststellung gemacht: Ist $\mathcal{M}'$ eine $(d+1)\times(d+1)$ Matrix aus Elementen wie oben, so existiert ein eindeutiges Polynom $p(x,y)$ vom Grad d in jeder Variablen, welches mit ϕ übereinstimmt, d. h. $p(x_i, y_j) = \phi(m_{ij})$ für $1 \le i \le d+1,\ 1 \le j \le d+1$.
Beweis: Es gibt aufgrund dieser $(d+1)\times(d+1)$ Matrix natürlich $(d+1)^2$ Tripel der Form $(x_i, y_j, \phi(x_i, y_j))$. Dies ist genau die Anzahl von Gleichungen der Form $p(x_i, y_j) \stackrel{!}{=} \phi(m_{ij})$. Andererseits hat das Polynom p folgende Anzahl von Koeffizienten: $\underbrace{1}_{\text{Absolut-gl.}} + \underbrace{d}_{\text{Potenzen in } x} + \underbrace{d}_{\text{Potenzen in } y} + \underbrace{d^2}_{\text{gemischte Potenzen}} = 1 + 2d + d^2 = (1+d)^2$. Diese Koeffizienten von p sind also durch ein lineares Gleichungssystem, dessen Koeffizientenmatrix durch die $x_i, y_j,\ 0 \le i,j \le d$ bestimmt sind, festgelegt. Dieses Gleichungssystem ist eindeutig lösbar, da die x_i und die y_j jeweils verschieden sind. Also ist das Polynom p eindeutig. Es gilt $p(x_i, y_j) = \phi(m_{ij})$. □

3. Nach **2.** existiert ein eindeutiges Polynom $p(x,y)$ vom Grad d in jeder Variablen, sodaß $p(x_i, y_j) = \phi(m_{ij})$ für $1 \le i,j \le d+1$.

Die c_j, $1 \le j \le d+1$ sind jeweils Grad d Polynome in x und stimmen jeweils an $d+1$ Stellen mit dem Grad d Polynom p überein, müssen also alle identisch sein. Für die r_i, $1 \le i \le d+1$ gilt dies analog. (Siehe Skizze: Matrix $\mathcal{M}$ mit dem Übereistimmungsbereich $\mathcal{M}'$).



Betrachte nun die j-Spalte. Nach Voraussetzung existiert ein Grad d Polynom c_j in einer Variablen, welches mit ϕ an allen $d+2$ Stellen der j-ten Spalte von M übereinstimmt. Daher muß c_j eindeutig sein, also mit $p(x_i, y)$ identisch sein. Daraus folgt: $p(x_0, y_j) = \phi(m_{0j})$ $\forall j \in \{1, ..., d+1\}$. Aus dem selben Grund gilt natürlich $p(x_i, y_0) = \phi(m_{i0})$ $\forall i \in \{1, ..., d+1\}$.

Betrachte nun die 0-te Zeile. Nach Vorausetzung gilt $\phi(m_{0j}) = r_0(m_{0j})$ mit eindeutigem Grad d Polynom r_0. Auf $d+1$ Punkten stimmt damit r_0 mit $p(x_0, y_j)$ überein, sie sind damit identisch.

Der Rest ist klar: An allen Punkten der 0-Spalte stimmen nun auch p mit ϕ überein. Insgesamt haben wir $p(x_i, y_j) = \phi(m_{ij})$, $\forall 0 \le i, j \le d+1$, insbesondere ist p das Grad d Polynom in der vorher ausgenommenen Spalte 0.

□

Den Nutzen der Self-Corrector Funktion $SC^P(x,t)$ für den Test wird durch folgendes Lemma von Gemmell et al. (hier ohne Beweis) richtig klar.

Lemma 3.3 (GLRSW 91) *Sei $g(x)$ wie oben definiert und δ wie in (3.3). Dann gilt*

$$\Pr_{x \in F^n}[g(x) = P(x)] \ge 1 - 2\delta$$

Bemerkung: Das letzte Lemma sagt aus, daß g mit P auf einem großen Teil von F^n übereinstimmt. Deswegen müssen wir nur noch zeigen, daß g wirklich ein Grad d Polynom ist. (Damit ist dann P dicht zu einem Grad d Polynom !).

Der ganze Rest dieses Kapitels (einschließlich des Schlußlemmas) wird dafür aufgewendet, zu zeigen, daß g wirklich ein Grad d Polynom ist. Dabei wird mehrmals3.3 benutzt und einige Male das MTL angewandt.

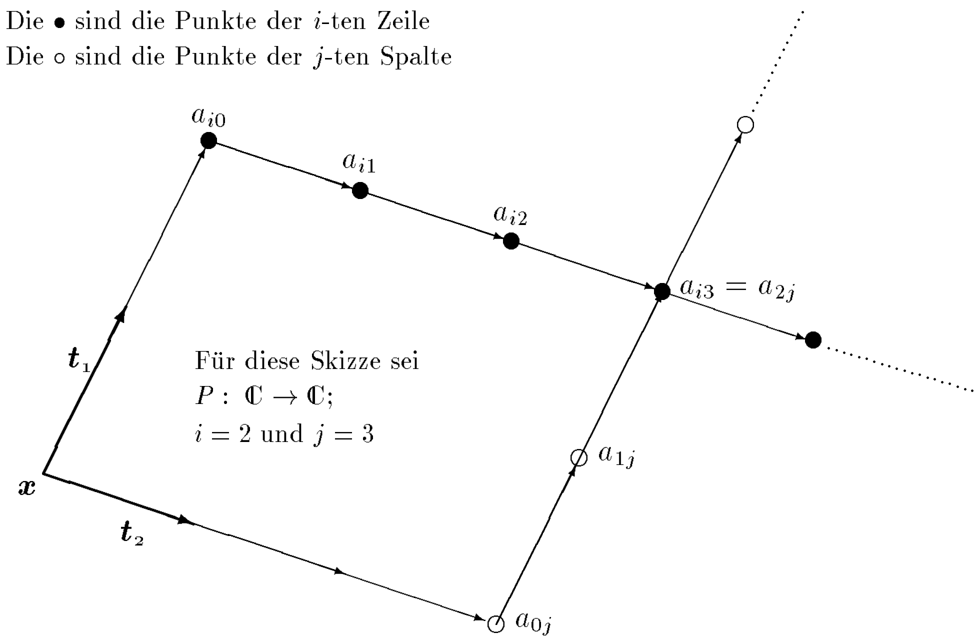
Im nächsten Lemma kommt Lemma 3.2, das MTL, zur Anwendung. Und zwar in der Behauptung, daß in der Definition von $g(x)$ die Majorität in den Richtungen eindeutig ist, ja daß sogar für die meisten Richtungen ein Wert $\neq$`error` auftaucht, d. h. $g(x)$ wohldefiniert ist.

Lemma 3.4 *$SC^P(x,t)$ sei wie oben definiert. Dann gilt mit (3.1) und (3.2)*

$$\forall x \Pr_{t_1,t_2 \in F^n}[SC^P(x,t_1) = SC^P(x,t_2) \neq \texttt{error}] \geq \frac{4}{5}$$

Beweis:

(I) Sei x fest. Die quadratische Matrix $\mathcal{A}$ sei so definiert: $\mathcal{A} = (a_{ij})_{i,j \in 0,1,\ldots,10d}$ mit $a_{ij} = x + it_1 + jt_2$. Bezeichne die i-te Zeile als "gut" g. d. w. ein Grad d Polynom r_i existiert, sodaß $\forall j,\ 0 \leq j \leq 10d,\ r_i(a_{ij}) = P(a_{ij})$. Analog spaltenweise: Für eine gute Spalte gilt: $\exists c_i$ (ein Grad d Polynom) mit $\forall i,\ 0 \leq j \leq 10d,\ r_i(a_{ij}) = P(a_{ij})$. Eine Zeile oder Spalte heißt "schlecht" g. d. w. sie nicht gut



ist.

Aus der Definition (3.3) folgt

$$\begin{aligned} \Pr_{t_1,t_2 \in F^n}[\text{Zeile } i \text{ ist gut}] &\geq 1 - \delta \quad (i \geq 1) \\ \Pr_{t_1,t_2 \in F^n}[\text{Spalte } j \text{ ist gut}] &\geq 1 - \delta \quad (j \geq 1) \end{aligned} \tag{3.4}$$

Dies gilt nur für $i, j \geq 1$, weil x fest und nicht mehr zufällig gewählt ist wie in 3.1 gefordert. Sämtliche anderen Zeilen und Spalten, außer der jeweils 0-ten, sind aber in Aufpunkt und Richtung zufällig und erfüllen (3.1).

Nun wird die *Ungleichung von Markov* angewendet, die für eine nicht negative Zufallsvariable X aussagt, daß $\forall a\ \Pr[X \geq a] \leq \frac{1}{a}E(X)$ gilt. Die Zufallsvariable S_i, die nun eingeführt wird, benötigen wir zum Aufsummieren der schlechten Zeilen und Spalten, deshalb

$$S_i = \begin{cases} 1 & \text{falls Zeile } i \text{ schlecht ist} \\ 0 & \text{falls Zeile } i \text{ gut ist} \end{cases}$$

Aus (3.4) folgt $\mathrm{E}(S_i) \leq \delta$ $(*)$. Die 0-te Zeile und Spalte werden nachher separat betrachtet, d. h. "Anzahl" (=#) bedeute: Anzahl ohne 0-te Zeile und Spalte.

$$\begin{aligned} \Pr_{t_1,t_2 \in F^n}[\#\text{ der schlechten Zeilen} \geq 100\delta d] &= \Pr_{t_1,t_2 \in F^n}\left[\sum_{1=1}^{10d} S_i \geq 100\delta d\right] \\ &\overset{\text{Markov-UG}}{\leq} \frac{1}{100\delta d}\,\mathrm{E}\left(\sum_{i=1}^{10d} S_i\right) \\ &= \frac{1}{100\delta d}\sum_{i=1}^{10d} \mathrm{E}\,(S_i) \\ &\overset{(*)}{\leq} \frac{1}{100\delta d} 10\delta d = \frac{1}{10} \end{aligned}$$

Analog gilt für die Spalten

$$\Pr_{t_1,t_2 \in F^n}[\#\text{ der schlechten Spalten} \leq 100\delta d] \leq \frac{1}{10}$$

und deshalb

$$\Pr_{t_1,t_2 \in F^n}\left[\#\text{ der schlechten Z.} \leq 100\delta d \bigvee \#\text{ der schlechten S.} \leq 100\delta d\right] \leq \frac{1}{5}$$

nach wie vor jeweils ohne die 0-te Zeile und Spalte. Nach unserer Vereinbarung (3.3) folgt unmittelbar $100\delta d < 1$. Demnach gilt (wieder ohne Berücksichtigung der 0-te Zeile und Spalte):

$$\Pr_{t_1,t_2 \in F^n}[\text{alle Zeilen und Spalten sind gut}] \geq \frac{4}{5}$$

(II) Für den Rest des Beweises fixieren wir (von den über 80% möglichen) t_1 und t_2, sodaß alle Zeilen und Spalten außer der jeweils 0-ten in der Tat gut sind.

Nun wird auf $\mathcal{A}$ eine Funktion ϕ definiert, sodaß $\phi(i,j) = P(a_{ij})$ für $(i,j) \in \{(i,j) :\ 0 \leq i,j \leq 10\delta d\} - \{(0,0)\}$. Wähle man nun $d+2$ Spalten inklusive der 0-ten aus, d. h. $Y = \{0\} \cup \{(d+1) \text{ Elemente aus } \{1,2,...,10d\}\}$. Wir können leicht ein Grad d Polynom r_0 finden, sodaß $r_0(a_{0j}) = \phi(0,j)$ für $0 \neq j \in Y$, (denn dies sind ja $d+1$ Punkte !). Man beachte, daß ϕ ausgenommen dem Nullpunkt des Gitters definiert war und jetzt dem Zeilen-Polynom r_0 "angepaßt" wird: $\phi(0,0) := r_0(a_{00})$.

Es sei die Menge $X = \{0\} \cup \{(d+1)$ Elemente aus $\{1,2,...,10d\}\}$. X und Y formen ein $(d+2)\times(d+2)$-Gitter G wie im MTL gefordert. Da ϕ schon an die 0-te Zeile angepaßt wurde, gibt es jetzt in allen gewählten Zeilen und Spalten außer der 0-ten Spalte Grad d Polynome. Das MTL kann angewendet werden, womit ϕ in der 0-ten Spalte auch ein Grad d Polynom ist. Wir finden also ein eindeutiges Grad d Polynom c_0 mit

$$c_o(a_{i0}) = \phi(i,0) \quad \text{für } i \in X \tag{3.5}$$

Nun treffen wir eine andere Zeilen-Auswahl X' mit $X' = (X - \{i_1\}) \cup \{i_2\}$ für ein $0 \neq i_1 \in X$ und $i_2 \in \{1,2,...,10d\} - X$. Das heißt X' ist mit X identisch bis auf ein ausgetauschtes Element $\neq 0$). Daraus wird ein neues Gitter G' gebildet und zwar mit X' und dem alten Y. Das MTL nochmals angewandt ergibt, daß ein Grad d Polynom mit ϕ an allen Punkten $\{(i,0):\ i \in X'\}$ übereinstimmt. Nach 3.5 gilt aber $c_0(a_{i0}) = \phi(i,0)$ für $i \in X \cap X'$. Dies bedeutet diese beiden Grad d Polynome stimmen an $d+1$ Stellen überein, ergo sind sie identisch:

$$c_0(a_{i0}) = \phi(i,0) \quad \text{für } i \in X'$$

Man kann X' mit jedem $i \in \{0,1,...,10d\}$ erweitern. Das ergibt dann immer ein Grad d Polynom c_0:

$$c_0(a_{i0}) = \phi(i,0) \quad \text{für alle } i \in \{0,1,...,10d\}$$

Statt eine andere Zeilen- kann selbstverständlich auch eine andere Spalten-Auswahl getroffen werden. Die Beobachtung gilt dann analog, d. h. es gibt ein eindeutiges Grad d Polynom r_0, das in der ersten Zeile mit ϕ übereinstimmt. Das bedeutet insbesondere, daß die erste Auswahl, von Spalten Y die Definition von $\phi(0,0)$ nicht beeiflußt hat.

In **(II)** wurde also gezeigt:

$$SC^P(x,t_1) = \underbrace{c_0(a_{00})}_{c_0(x)} = \phi(0,0) = \underbrace{r_0(a_{00})}_{r_0(x)} = SC^P(x,t_2)$$

$SC^P(x,t_1)$ kann also insbesondere nicht den Wert `error` haben, da c_0 an allen $10d$ betrachteten Punkten ein Grad d Polynom ist.

Da wir in **(II)** von vornherein $\frac{1}{5}$ der Richtungs-Tupel (t_1, t_2) eliminiert hatten, folgt die Behauptung.

□

Das nächste Lemma ist eine einfache Anwendung von Lemma 3.4 und sagt aus, daß die Funktion $g(.)$ mit $SC^P(x,t)$ mit hoher Wahrscheinlichkeit, d. h. in den meisten Richtungen, übereinstimmt, und dieser Wert nicht `error` ist.

Lemma 3.5

$$\forall x \Pr_{t \in F^n}[g(x) = SC^P(x,t) \neq \texttt{error}] \geq \frac{4}{5}$$

Beweis: Fixiere $x \in F^n$. Dann gilt

$$\begin{aligned}
\tfrac{4}{5} \overset{\text{Lemma 3.4}}{\leq} am \quad & \Pr_{t_1,t_2 \in F^n}\left[SC^P(x,t_1) = SC^P(x,t_2) \neq \texttt{error}\right] \\
\leq \quad & \max_{t_1 \in F^n}\left\{\Pr_{t_2 \in F^n}\left[SC^P(x,t_1) = SC^P(x,t_2) \neq \texttt{error}\right]\right\} \\
= \quad & \Pr_{t_2 \in F^n}\left[\text{majority}_{t_1}\{SC^P(x,t_1)\} = SC^P(x,t_2) \neq \texttt{error}\right] \\
= \quad & \Pr_{t_2 \in F^n}\left[g(x) = SC^P(x,t_2) \neq \texttt{error}\right]
\end{aligned}$$

□

Was ist bisher erreicht? Mit dem Lemma 3.5 haben wir die 80%-ige Gewissheit, daß die Funktion g mit dem Self-Corrector für P übereinstimmt. Genauer, in mindestens 80% aller Richtungen t stimmt g mit $h(0)$ überein und h ist ein Grad d Polynom. Ist aber g wirklich auch ein Grad d Polynom? Dieses werden wir im Rest dieses Abschnitts beweisen.

Man benötigt eine kleine Definition: Das *dichteste Grad d Polynom zu ϕ auf einer Menge S*, ist dasjenige Grad d Polynom h, welches $|\{x \in S\} : \ h(x) = \phi(x)|$ maximiert. Das dichteste Grad d Polynom zu ϕ ist nicht immer eindeutig; man wähle ein beliebiges, falls mehrere existieren.

Sei $h_{x,s}$ das dichteste Grad d Polynom zu $g(x)$ auf der Menge $S = \{x+s, x+2s, ..., x+(3d+3)s\}$. Gewissermaßen ist $h_{x,s}$ das Polynom, das g von x in Richtung s korrigiert. Das Ziel ist, wie schon gesagt, zu zeigen, daß g selbst ein Grad d Polynom ist. Zuerst werden wir zeigen, daß die Funktion g notwendigerweise mit $h_{x,s}$ übereinstimmt; dann werden wir im Schlußlemma sehen, daß dies ausreicht, um zu schließen: "g ist Grad d Polynom".

Lemma 3.6

$$\forall x, s \textit{ gilt: } g(x) = h_{x,s}(x) \textit{ Dabei ist } h_{x,s} \textit{ eindeutig.}$$

Beweis:

(I) Fixiere den Aufpunkt x und die Richtung s. Konstruiere eine Matrix $\mathcal{A} = (a_{ij})$ mit $a_{ij} = (x + is) + j(t_1 + it_2) = (x + jt_1) + i(s + jt_2)$ mit zufällig gewählten t_1, t_2. Sei $0 \leq i \leq 3d + 3$ und $0 \leq j \leq 10d$; $\mathcal{A}$ hat also $3d + 4$ Zeilen und $10d + 1$ Spalten, $SC^P(x + is, t_1 + it_2)$ hängt von den Elementen der i-ten Zeile und $SC^P(x + jt_1, s + jt_2)$ von den Elementen der j-ten Spalte ab.

Übersicht über das Array $\mathcal{A}_{(3d+4)\times(10d+1)}$

x	$x + t_1$	$x + 2t_1$	$\cdots$		$\cdots$	$x + 10dt_1$
$x + s$	$(x + s)+$ $(t_1 + t_2)$	$(x + s)+$ $2(t_1 + t_2)$				
$x + 2s$	$(x + 2s)+$ $(t_1 + 2t_2)$	$(x + 2s)+$ $2(t_1 + 2t_2)$				
$\vdots$						
				$(x + is)+$ $j(t_1 + it_2)$		
$\vdots$						
$x + (3d + 3)s$						

Bezeichne die i-te Zeile als "gut", wenn $g(x + is) = SC^P(x + is, t_1 + it_2)$. Selbst für $i = 0$ ist der Ausdruck $t_1 + it_2$ über F^n gleichverteilt, falls t_1 und t_2 gleichverteilt sind. Also kann Lemma 3.5 angewendet werden:

$$\Pr_{t_1,t_2 \in F^n}[\text{Zeile } i \text{ ist gut}] \geq \frac{4}{5} \quad \Longleftrightarrow \quad \Pr_{t_1,t_2 \in F^n}[\text{Zeile } i \text{ ist schlecht}] \leq \frac{1}{5}$$

Wir wollen nun die Ungleichung von Markov anwenden, um eine positive Wahrscheinlichkeit für einen Zeilenanteil, der gut ist, zu bekommen. Wir behandeln die Zeile 0 separat, um nachher schließen zu können, daß sie gut ist.

$$\Pr_{t_1,t_2 \in F^n}\left[\geq \frac{1}{3} \text{ aller Z. (außer Z. 0) sind schlecht}\right] \overset{\text{Markov-UG}}{\leq} \frac{1}{\frac{(3d+3)}{3}} \cdot (3d+3) \cdot \frac{1}{5} = \frac{3}{5}$$

und daher

$$\Pr_{t_1,t_2 \in F^n}\left[\geq \frac{2}{3} \text{ aller Zeilen (außer Z. 0) sind gut}\right] \geq \frac{1}{\frac{2(3d+3)}{3}} \cdot (3d + 3) \cdot \frac{4}{5} = \frac{2}{5}$$

und da $\Pr\limits_{t_1,t_2\in F^n}[\text{Zeile 0 ist gut}] \geq \frac{4}{5}$ ist, können wir schließen

$$\Pr_{t_1,t_2\in F^n}\left[\text{Z. 0 ist gut} \bigwedge \geq \frac{2}{3} \text{ aller Z. (außer Z. 0) sind gut}\right] \geq \frac{8}{25} > \frac{1}{5} \quad (3.6)$$

Definiere nun Spalte j als "gut", falls ein Grad d Polynom c_j existiert, sodaß

$c_i(a_{ij}) = P(a_{ij})$, $\forall\, 0 \leq i \leq 3d+3$. (Andernfalls als schlecht). Selbst wenn s, x und j fest sind, haben $x + jt_1$ und $s + jt_2$ für $j \geq 1$ einen gleichverteilten Zufalls-Startpunkt und eine gleichverteilte Zufalls-Richtung. Nach der Defintion von δ in (3.2) gilt:

$$\Pr_{t_1,t_2\in F^n}[\text{Spalte } j \text{ ist schlecht}] \leq \delta \quad (\delta \geq 1)$$

(Hier steht ein $\leq$ statt eines $=$, da jede Spalte nur $3d+3$ Punkte lang ist und damit kürzer als in (3.2) mit $10d$). Nun gilt

$$\begin{aligned}&\Pr_{t_1,t_2\in F^n}[\geq 10\delta \text{ aller Spalten (außer Spalte 0) sind schlecht}]^{2}\\ &\overset{\text{Markov-UG}}{\leq} \frac{1}{10\delta 10d} \cdot 10\delta d = \frac{1}{10}\end{aligned} \quad (3.7)$$

Aus (3.6) und (3.7) folgt

$$\begin{aligned}&\Pr_{t_1,t_2\in F^n}\left[\begin{array}{lll}\text{Zeile 0 ist gut} & \bigwedge & \geq \frac{2}{3} \text{ aller Zeilen (außer Zeile 0) sind gut}\\ & \bigwedge & \text{alle Spalten (außer Spalte 0) sind gut}\end{array}\right]\\ &\geq \frac{1}{5} \cdot \frac{9}{10} = \frac{9}{50} > \frac{1}{10}\end{aligned} \quad (3.8)$$

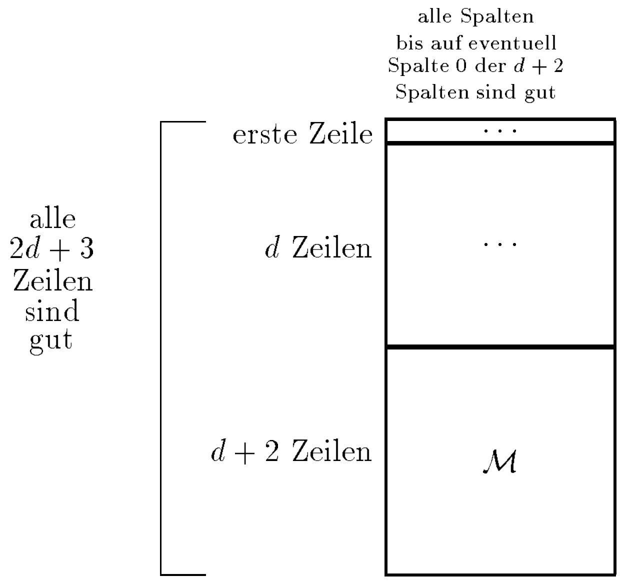
Da die Wahrscheinlichkeit echt größer Null ist, muß es ein 2-Tupel (t_1, t_2) geben, bei dem alle 3 Aussagen wahr sind. Von nun an halten wir so ein Tupel fest und betrachten den bisherigen Beweis als einen Existenznachweis dieses Tupels.

(II) (a) Sei t_1, t_2 so gewählt, daß die 3 Ereignisse aus (3.8) wahr sind. Nun wählen wir eine Untermatrix von $\mathcal{A}$, die die Zeile 0 und die Spalte 0 enthält. Zusätzlich $2d+2$ Zeilen und $d+1$ Spalten, die jeweils gut sind; die Spalten sind ohnehin gut und die $2d+2$ Zeilen sind exakt $\frac{2}{3}$ aller Zeilen, welche nach (3.8) wählbar sind. Diese Untermatrix sei $\mathcal{B}$ genannt und hat nach Konstruktion außer eventuell Spalte 0 nur gute Zeilen und Spalten.

[2] Dies ist die Wahrscheinlichkeit, daß eine oder mehr Spalten schlecht sind (siehe Beweis zu Lemma 3.4). Das Komplementärereignis ist: "alle Spalten (außer Spalte 0) sind gut" und hat die Wahrscheinlichkeit $\geq \frac{9}{10}$.

(b) Hier werden wir zeigen, daß die 0-te Spalte auch gut sein muß. Die quadratische Matrix $\mathcal{M}_{(d+2,d+2)}$ besteht aus dem Unterteil von $\mathcal{B}$ (siehe Skizze) und hat selbstverständlich wie $\mathcal{B}$ nur gute Zeilen. Die Spalten (außer eventuell Spalte 0) sind auch gut, da durch die Punkte der j_k-ten Spalte genau das gleiche Polynom läuft, wie durch die j_k-te Spalte in $\mathcal{B}$, obwohl nur $d+2$ Punkte in dieser Spalte liegen und nicht mehr $3d+4$.

Die Matrix $\mathcal{B}$



Sei $\phi(i,j) := P(a_{ij})$. Wende nun das MTL auf die Matrix $\mathcal{M}$ an. Daraus folgt, daß auch die 0-te Spalte in $\mathcal{M}$ gut ist, d. h. das Fragment der 0-ten Spalte von $\mathcal{B}$, welches in $\mathcal{M}$ auftaucht, ist gut. Also existiert ein Grad d Polynom c_0 mit $c_0(a_{i0}) = \phi(i,0) = P(a_{i0})$ für alle $d+1 \le i \le 2d+2$.

Die Matrix $\mathcal{M}$ wird nun eine Zeile nach oben geschoben, d. h. es entsteht $\mathcal{M}'$ aus $\mathcal{M}$, indem eine weitere Zeile oben dazukommt und unten eine wegfällt. Für $\mathcal{M}'$ folgt analog zu $\mathcal{M}$, daß ein Grad d Polynom c_0' existiert mit $c_0'(a_{i0}) = \phi(i,0) = P(a_{i0})$ für alle $d+2$ Zeilen, die in $\mathcal{M}'$ vorkommen. Davon sind $d+1$ schon in $\mathcal{M}$ vorgekommen, deswegen müssen c_0 und c_0' identische Grad d Polynome sein. Nach diesem Strickmuster wird nun $\mathcal{M}'$ Zeile für Zeile (bis zur einschließlich ersten) nach oben geschoben und immer auf die selbe Weise argumentiert. Daraus folgt schließlich $c_0(a_{i0}) = P(a_{i0})$ für alle $0 \le i \le 2d+2$, d. h. palte 0 in $\mathcal{B}$ ist gut.

(c) In **(I)** wurde gezeigt, daß für spezielle Wahl von (t_1, t_2) $\frac{2}{3}$ aller Zeilen i aus $1 \le i \le 3d+4$ und Zeile 0 gut sind. Jetzt folgt zur besseren Übersicht nochmals die Matrix $\mathcal{B}$ mit den einzelnen Punkten.

Übersicht über das Array $\mathcal{B}_{(2d+3)\times(d+2)}$

x	$x+j_1t_1$	$\cdots$		$\cdots$	$x+j_{d+1}t_1$
$x+i_1s$	$(x+i_1s)+$ $j_1(t_1+t_2)$				
$x+i_2s$					
$\vdots$					
			$(x+i_ks)+$ $j_k(t_1+t_2)$		
$\vdots$					
$x+i_{2d+2}s$					

OBdA. $J: j_1 < j_2 < \ldots < j_{d+1} \in [1..10d]$. $I: i_1 < i_2 < \ldots < i_{2d+2} \in [1..3d+3]$. Nach Definition war nur eventuell Spalte 0 schlecht. $\stackrel{(II)(b)}{\longrightarrow}$ Alle Zeilen und Spalten sind gut.

Dies bedeutet für alle $i_k \in I$:

$$
\begin{aligned}
&g(x+i_ks) \stackrel{(I)}{=} SC^P(x+i_ks, t_1+i_kt_2) \stackrel{Def.}{=} h_{i_k}(0) = \phi_{i_k}(0) = P(x+i_ks)\\
&c_0(x+i_ks) \stackrel{(II)(b)}{=} P(x+i_ks)\\
&\Rightarrow g \text{ und } c_0 \text{ stimmen bei } x+i_ks,\ i_k \in I \text{ überein}
\end{aligned}
$$

Desweiteren folgt aus **(b)**:

$$
\begin{aligned}
&g(x) \stackrel{(I)}{=} SC^P(x, t_1) \stackrel{Def.}{=} h_0(0) = \phi_0(0) = P(x)\\
&c_0(x) \stackrel{(II)(b)}{=} P(x)\\
&\Rightarrow g \text{ und } c_0 \text{ stimmen auch bei } x \text{ überein}
\end{aligned}
$$

Das Grad d Polynom c_0 stimmt also in der kompletten 0-ten Spalte von $\mathcal{B}$ mit $h_{x,s}$ überein, da dieses Grad d Polynom ja das dichteste Grad d Polynom zu g auf einer Obermenge der Punkte in dieser Spalte ist. Speziell für die 0-te Zeile ergibt dies:

$$h_{x,s}(x) = c_0(x) = c_0(a_{00}) = g(a_{00}) = g(x)$$

Sei nun $S = \{x+s, x+2s, \ldots, x+(3d+3)s\}$ wie schon vorher bei der Definition von $h_{x,s}$ festgelegt. Wir haben gezeigt, daß $g = h_{x,s}$ auf $\frac{2}{3}$ der $3d+3$-elementigen Menge S. Sei nun k ein Grad d Polynom, welches auf mindestens $2d+2$ Punkten aus S mit g übereinstimmt. Dann stimmt aber k mit $h_{x,s}$ auf mindestens $d+1$ Punkten überein, was soviel heißt, daß die beiden Grad d Polynome identisch sind. Also ist $h_{x,s}$ das eindeutige, dichteste Grad d Polynom zu g auf S.

$\square$

Schlußlemma

Schließlich zeigen wir noch, daß das Lemma 3.6 uns die Eigenschaft von g gibt, welche wir brauchen, um in Lemma 3.7 zu schließen, "g ist ein Grad d Polynom".

Lemma 3.7 *Sei ϕ eine Funktion von $F^n \to F$ mit $\phi(x) = h_{x,s}(x)\ \forall x, s$, wobei $h_{x,s}$ das eindeutige dichteste Grad d Polynom zu ϕ auf $\{x + is : \ i = 1, 2, ..., 3d + 3\}$. Dann gilt: ϕ muß ein Polynom mit Totalgrad d sein.*

Beweis:

(I) Betrachte zuerst eine Gerade in F^n, die durch x geht und die Richtung s hat. Wir zeigen, daß ϕ ein Polynom "entlang" dieser Geraden ist. (Mit entlang ist gemeint, auf äquidistanten Punkten auf dieser Geraden). Wir haben eine Familie von Polynomen:

$$Q = \{h_{x+is,s} : \ i \in \mathbb{Z}\}.$$

Angenommen, Q enthält mehr als ein spezielles Polynom, dann $\exists j \in \mathbb{Z}$ mit $h_{x+js,s} \neq h_{x+(j-1)s,s}$. Sei

$$S = \{x + is : \ j + 1 \leq i \leq j + 3d + 3\}, \quad T = \{x + is : \ j \leq i \leq j + 3d + 2\}.$$

So ist S die Punktmenge, welche $h_{x+js,s}$ benötigt und T diejenige von $h_{x+(j-1)s,s}$. Nach Vorausetzung gilt aber $h_{x+js,s}(x + js) = \phi(x + js)$. Daher stimmt h_{x+js} mit ϕ auf T mindestens genauso oft überein wie auf S. Da T aus S entsteht, indem ein Element ausgetauscht wird, ist es unmöglich, daß ein Polynom q existiert, welches mit ϕ auf T in mehr Stellen als mit ϕ auf S übereinstimmt.

Da aber h_{x+js} und $h_{x+(j-1)s}$ eindeutig sind, also mit ϕ an wirklich mehr Stellen als alle anderen Grad d Polynome auf den jeweiligen Mengen übereinstimmen, müssen sie gleich sein. Dies ist ein Widerspruch!
Daraus folgt Q hat nur ein Element. Da $\phi(x + is) = h_{x+is,s}(x + is)$ für alle i gilt, muß ϕ entlang dieser Geraden ein Grad d Polynom sein.

(II) Als nächstes schließen wir, daß, wenn ϕ auf jeder Geraden ein Grad d Polynom ist, es als Polynom in n Variablen mit $\text{grad}(\phi) = d$ ausgedrückt werden kann. Sei $s_i = (0, ..., 0, 1, 0, ..., 0)$ der i-te Einheitsvektor. Wir wollen nun ein Polynom q definieren, sodaß

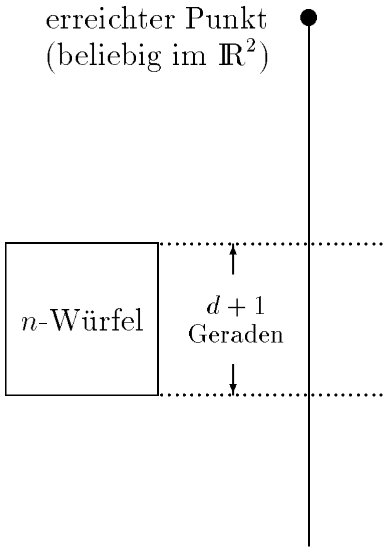
$$q(a_1, ..., a_n) = \phi(a_1 s_1 + a_2 s_2 + ... + a_n s_n)$$

Nach dem Standard-Interpolations Argument kann man die Koeffizienten von q eindeutig festlegen, wenn q mit ϕ auf einem n-dimensionalen (Gitter-) Würfel

$$\{(a_1, ..., a_n) :\ \forall i,\ 0 \leq a_i \leq d\} \quad (\text{mit} \quad (d+1) \times (d+1) \text{ Gitterpunkten})$$

übereinstimmen sollen.

Wenn man animmt, daß $|F| > d+1$, ist die Einbettung des Würfels in F^n gesichert. Jetzt ist es leicht zu sehen, daß q auch überall außerhalb des Würfels mit ϕ identisch ist: Sei U die Menge von Punkten auf denen q und ϕ übereinstimmen; der Würfel ist also in U. Da aber auf jeder Geraden, die parallel zu einer Achse des Würfel liegt, $d+1$ Gitterpunkte liegen, kann man U um diese Geraden erweitern. (Da sowohl q als auch ϕ den Grad d auf dieser Geraden haben, und sie in $d+1$ Punkten übereinstimmen).



Auf diese Weise wird U um alle $d+1$ Geraden erweitert. Wenn man auf jeder dieser Geraden eine bestimmte Entfernungsbetrag aus dem Gitter herausgeht, ist eine Gerade mit $d{+}1$ Punkten entstanden. Diese Gerade kann mit demselben Argument in alle Richtungen verlängert werden; die erreichten Punkte sind immer noch in U. Da aber der Entfernungsbetrag beliebig war, kann jeder beliebige Punkt in F^n auf diese Weise erreicht werden; also $U = F^n$. (In der Skizze der Fall $n = 2$).

Wir wissen, daß ϕ den Grad d in jeder Variable hat; der Totalgrad also kleiner gleich dn ist. Sei m der tatsächliche Totalgrad von ϕ. Schreibe nun ϕ wie folgt: $\phi(x) = q(x) + r(x)$, wobei $q(x)$ die Summe aller Grad d Monomiale in ϕ ist. Deswegen ist $r(x)$ von einem Totalgrad echt kleiner als m und $q \not\equiv 0$[3]. Da q in

[3] Im Falle $n = d = 2$ gilt:

$$q(x_1, x_2) = \underbrace{a_1 x_1^2 x_2^2}_{q(x_1,x_2)} + \underbrace{a_2 x_1^2 x_2 + a_3 x_1 x_2^2 + a_4 x_1 x_2 + a_5 x_1^2 + a_6 x_2^2 + a_7 x_1 + a_8 x_2 + a_9}_{r(x_1,x_2)}$$

Das Monom (Produkt aus ausschließlich quadrierten Variablen) ist in diesem Fall $a_1 x_1^2 x_2^2$ mit einer Konstanten a_1

jeder Richtung den Grad d hat und $|F| > d+1$ gibt es einen Punkt s in F^n, sodaß $q(s) \neq 0$. Sei $c = q(s)$.

Nun betrachte die Gerade vom Startpunkt $x = 0$ in die Richtung s. Ein Punkt auf dieser Geraden hat die Form is. Da q homogen vom Grad m ist, gilt:

$$\phi(is) = q(is) + r(is) = i^m q(s) + r(is) = i^m c + r(is)$$

Also hat ϕ den Grad m auf dieser Geraden. Daher gilt $m \leq d$ und wir haben das erwünschte Resultat, □

Lemma 3.6 zeigte, daß g die von ϕ in Lemma 3.7 geforderten Eigenschaften erfüllt. So kann also Lemma 3.7 auf g angewandt werden, was bedeutet, daß g in der Tat ein Grad d Polynom ist, was wir zeigen wollten.

Kapitel 4

Das Hauptresultat: $\mathrm{NP} = \mathrm{PCP}(\log n, 1)$

In diesem Kapitel wird die *NP*-Charakterisierung $NP = PCP(\log n, 1)$ bewiesen, die auf eine sehr aktuelle Idee von Arora und Safra [AS 92] zurückgeht. Dieses überraschende Ergebnis hat weitreichende Konsequenzen, die in dieser Arbeit nicht behandelt werden, siehe Einführung.

Im ersten Abschnitt werden die Klassen genau definiert und einfache oder schon länger bekannte Ergebnisse vorgestellt. Eine kurze intuitive Beschreibung einer *PCP*-Maschine lautet wie folgt : Eine Turingmaschine hat Zugriff auf einen Beweis (genauer: Zugehörigkeitsbeweis) und auf ein Zufallsband und darf jeweils nur eine bestimmte Größenordnung von Bits lesen, um den Beweis zu akzeptieren oder zu verwerfen.

Die Abschnitte zwei und drei laufen nach dem selben Muster ab.

1. Jeweils ein *PCP*-Protokoll für 3-SAT, was $NP \subseteq PCP(n^{O(1)}, 1)$ und $NP \subseteq PCP\left(\log n, (\log n)^{O(1)}\right)$ einbringt.

2. Die Kodierung des Inputs, zuzüglich der Rekursionsidee von Arora und Safra im zweiten Abschnitt, die die Kodierung benötigt, um

3. die obigen zwei Protokolle anzuwenden. Dieser Teil wird als Rekursionsfolgerung bezeichnet und bringt uns zwei aussagekräftige Inklusionen von *PCP*-Klassen.

Im vierten Abschnitt werden dann die Früchte aus den vorangehenden geerntet. Soll heißen, das Hauptresultat $NP = PCP(\log n, 1)$ kann hier in Kürze bewiesen werden.

4.1 Probabilistisch prüfbare Beweise

Als Grundlage folgen nun die exakte Definition der *PCP*-Klasse, sowie einfache und doch wichtige Eigenschaften, die auf unser Hauptziel hinsteuern.

4.1.1 Die Klassen *PCP* und *OPT*

Nun wird die Komplexitätsklasse *PCP* (steht für probabilistically checkable proofs) nach Arora und Safra [AS 92] definiert. Im Gegensatz zu anderen Komplexitätsklassen wie *NP*, *MIP*, *EXP* oder *PSPACE* hat die Klasse *PCP* Parameter, und zwar $r(n)$ und $q(n)$, jeweils Funktionen $\mathbb{N} \to \mathbb{R}_+$.

Definition 4.1 (*PCP*) *Eine Sprache L ist in PCP(r(n),q(n)), g. d. w. eine probabilistische, polynomiell zeitbeschränkte Turingmaschine M mit Zugriff auf einen Beweis Π existiert, sodaß M beim Input x, mit $|x| = n$:*

(i) *$O(r(n))$ Zufallsbits benützt,*

(ii) *$O(q(n))$ Bits des Beweises Π benützt,*

(iii) *alle Fragen nicht-adaptiv sind,*

(iv) *0 oder 1 ausgibt.*

Darüberhinaus existiere ein $\delta \in (0,1)$, sodaß für alle x gelte:

(1) *$x \in L \Rightarrow \exists$ ein Beweis Π sodaß*

$$\Pr_{\tau}[M^{\Pi}(x,\tau) = 1] = 1$$

(2) *$x \notin L \Rightarrow \forall$ Beweise Π gilt*

$$\Pr_{\tau}[M^{\Pi}(x,\tau) = 0] \quad \geq \quad \delta$$

Bemerkungen:

- Eine *PCP*-Maschine M ist also eine polynomiell zeitbeschränkte Turingmaschine, die auf einen String Π Zugriff hat. Dieser String stellt den Zugehörigkeitsbeweis dar, d. h. den Beweis für $x \in L$. Die Maschine M, die auch als Verifier bezeichnet wird, kann theoretisch auf jedes Bit von Π zugreifen, zusätzlich auf ein Zufallsband.

- Die 1. Bedingung könnte man auch so formulieren: Für jedes $x \in L$ gibt es einen Beweis, der für jeden beliebigen Zufallsstring M zum Akzeptieren zwingt. Es ist klar, daß dieses neue Modell eine Verallgemeinerung der probabilistischen Orakelmaschinen, bzw. der dazugehörigen Klasse *POM* darstellt.[1] Der Beweis Π kann als Orakel betrachtet werden, $q(n)$ wäre dann die Anzahl von Orakelfragen; der Unterschied zu *POM* besteht dann nur noch in der Beschränkung der verwendeten Bit-Anzahl.

- Wie in (iii) verlangt, werden in probabilistisch prüfbaren Beweisen nicht-adaptive Fragen gestellt, d. h. alle Fragen können als zu *einem* Zeitpunkt gestellt betrachtet werden, während im interaktiven Beweis die Fragen adaptiv sind, d. h. die Antworten jeweils von früheren Frage/Antwort-Paaren abhängig sein dürfen.

Die Klasse *OPT* (ebenfalls nach Arora und Safra [AS 92]) ist eine spezielle Form der Klasse *PCP*. Neben den Eigenschaften (i)-(iv) und (1),(2) gelten noch die Eigenschaften (v) und (vi).

Definition 4.2 (*OPT*) *Eine Sprache L ist in $OPT(r(n),q(n))$, g. d. w. eine probabilistische, polynomiell zeitbeschränkte Turingmaschine M mit Zugriff auf einen Beweis Π existiert, sodaß M beim Input x, mit $|x| = n$:*

(i) *$O(r(n))$ Zufallsbits benützt,*

(ii) *$O(q(n))$ Bits des Beweises Π benützt,*

(iii) *alle Fragen nicht-adaptiv sind,*

(iv) *0 oder 1 ausgibt,*

[1] Damit ist auch die Verwandtschaft zu *MIP* evident; man vergegenwärtige sich die Theoreme 2.4 und 2.5. Ausführlicher im nächsten Abschnitt.

(v) *und* Π *unterteilt ist in eine Menge von Segmenten* $\pi_1, \pi_2, ..., \pi_m$, *sodaß jedes Segment die Größe* $O(q(|x|))$ *hat.* M *kann auf alle Bits in einer konstanten Anzahl von Segmenten zugreifen.*

π_1	π_2	π_3	π_4	$\cdots$	π_m

(vi) *Nachdem* M *den Beweis gelesen hat produziert sie in der Zeit* $(q(|x|)^{O(1)}$ *den Output.*

Darüberhinaus existiere ein $\delta \in (0,1)$, *sodaß für alle* x *gelte:*

(1) $x \in L \Rightarrow \exists$ *ein Beweis* Π *sodaß*

$$\Pr_{\tau}[M^{\Pi}(x,\tau) = 1] = 1$$

(2) $x \notin L \Rightarrow \forall$ *Beweise* Π *gilt*

$$\Pr_{\tau}[M^{\Pi}(x,\tau) = 0] \quad \geq \quad \delta$$

Bemerkungen:

- Die Maschine M hat zwar für die Untersuchung der Bits des Beweises polynomial-Zeit, danach stehen ihr $(q(n))^{O(1)}$ Zeit zur Verfügung, bis sie den Output ausgibt, (für z. B. $q(n) = \log n$ ist dies viel weniger Zeit!).

- In den Definitionen von *PCP* und *OPT* ist nichts über die Länge von Π gesagt, OBdA. kann man also annehmen $|\Pi| = 2^{O(r(n))}q(n)$.

- In [AMLSS* 92] ist die Klasse *OPT* mittels Schaltkreisen definiert. Da Schaltkreise und Turingmaschinen sich gegenseitig effizient simulieren können, ist unsere Formulierung dazu äquivalent. Die Formilierung mittels Turingmaschinen stammt von [Beige 92].

- Unmittelbar klar ist

$$OPT(r(n), q(n)) \subseteq PCP(r(n), q(n))$$

Die Äquivalenz der beiden Klassen OPT und PCP wurde 1992 von Arora und Safra [AS 92] nachgewiesen; es ist bis heute noch kein einfacher Beweis gefunden worden, deswegen steht diese Aussage, als Lemma formuliert, hier ohne Beweis:

Lemma 4.3 (Arora-Safra)

$$PCP(r(n), q(n)) \subseteq OPT(r(n), q(n))$$

d. h. jeder nichtsegmentierte Beweis einer Sprache in $PCP(r(n), q(n))$ kann in einen segmentierten Beweis transformiert werden, wobei der Verifier nur auf eine konstante Anzahl von Segmenten zugreift.

4.1.2 Charakterisierung von *NP*

Um das Hauptresultat herauszuschälen, folgt nach einigen einfachen Sachverhalten eine chronologische Darstellung der Ergebnisse in Tabellenform. Die Parameter $r(n)$ und $q(n)$, speziell der zweite, wurden für NP in den letzten Jahren immer wieder verbessert, also verkleinert, was schließlich 1992 zu einem optimalen Ergebnis geführt hat. [AMLSS 92].

Die PCP-Notation erlaubt nun eine Verallgemeinerung von NP:

$$NP = PCP(0, n^{O(1)}) \tag{4.1}$$

Dies ist ganz klar, denn $r(n) = 0$ bedeutet, daß keine Zufallsbits gebraucht werden und $q(n) = n^{O(1)} \leq n^k$ für ein $k \in \mathbb{N}$ bedeutet, daß ein Beweis existiert, von dem polynomiell viele Bits ausreichen, um eine Sprache von einem wiederum polynomiell zeitbeschränkten Verifier zu akzeptieren. Dies ist (siehe Grundlagen) aber gerade die Klasse NP.

Was ergibt sich, wenn gar keine Bits des Beweises betrachtet werden, sondern nur polynomiell viele Zufallsbits ?

$$\text{co-}R = PCP(n^{O(1)}, 0) \tag{4.2}$$

Eine probabilistische Turingmaschine vom Typ R gehört zu den sog. Monte-Carlo-Algorithmen, welche nur bei Akzeptanz irren können (one-way-error).

Eine sehr wichtige und nicht augenscheinliche Inklusion ist die folgende

$$PCP(r(n), q(n)) \subseteq NTIME(n^{O(1)} + q(n)2^{r(n)}) \tag{4.3}$$

Begründung: Die *NTIME*-Maschine rät einen Beweis der Länge $q(n)$. Es existieren $2^{r(n)}$ verschiedene Strings der Länge $r(n)$, für die die *NTIME*-Maschine die Berechnung durchführen kann. sollte $q(n)2^{r(n)}$ noch nicht polynomiell groß sein, hat die *NTIME*-Maschine natürlich noch zusätzlich $n^{O(1)}$ Zeit.

Damit ist leicht zu zeigen $PCP(\log n, 1) \subseteq NTIME(n^{O(1)} + 1 \cdot 2^{\log n}) = NTIME(n^{O(1)}) = \bigcup_{p \text{ ein Polynom}} NTIME(p(n)) \stackrel{Def.}{=} NP$, also kurz

$$PCP(\log n, 1) \subseteq NP \tag{4.4}$$

Wie schon im vorangehenden Abschnitt angesprochen, ist das *PCP*-Modell eine Verallgemeinerung der probabilistischen Orakelmaschinen. Aus dem Beweis des Babai-Fortnow-Lund Theorems *MIP*=*NEXP* kann man folgern

$$MIP = PCP(n^{O(1)}, n^{O(1)}) \tag{4.5}$$

Mit diesem Ergebnis und mit (4.3) kann interessanterweise das von Fortnow, Rompel und Sipser 1988 gezeigte Resultat einfach hergeleitet werden:
$MIP = PCP(n^{O(1)}, n^{O(1)}) \subseteq NTIME(n^{O(1)} + n^{O(1)}2^{n^{O(1)}}) = NTIME(2^{n^{O(1)}}) = \bigcup_{k>0} NTIME(2^{n^k}) = NEXP$. Zusammenfassend $MIP \subseteq NEXP$ (siehe Seite 43). Nun folgt eine tabellarische Zusammenfassung der wichtigsten *PCP*-Ergebnisse. Man beachte, daß die Chronologie gleichbedeutend mit einer sukzessiven Verbesserung des ersten, zweiten oder beider Parameter ist.

Ergebnisse		
	Aussagen	*Quellenhinweise*
(1)	$NP \subseteq PCP(\log n^{O(1)}, \log n^{O(1)})$	[BFLS 91]
(2)	$NP \subseteq PCP(\log n \log\log n, \log n \log\log n)$	[FGLSS 91]
(3)	$NP = PCP(\log n, (\log n)^{O(1)})$	[FGLSS 91]
(4)	$NP = PCP(\log n, \log n \log\log n)$	[FGLSS 91]
(5)	$NP = PCP(\log n, \log n)$	[AS 92]
(6)	$NP = PCP(\log n, \sqrt{\log n})$	[AS 92]
(7)	$NP \subseteq PCP(n^3, 1)$	[AMLSS 92]
(8)	$NP \subseteq PCP(n^{O(1)}, 1)$	[AMLSS* 92]
(9)	$NP \subseteq PCP(\log n, \log\log n)$	[AS 92]
(10)	$NP = PCP(\log n, 1)$	[AMLSS 92]

Bemerkung: Schon in Aussage (3) ist der erste Parameter $r(n) = \log n$, also (siehe (10)) bereits nicht mehr zu verkleinern. Der zweite, $q(n)$ (die Anzahl der Bits von Π, die M liest) wird in mehreren Abschnitten von den genannten Autoren nach und nach verkleinert. Aussage (7) stellt eine Ausnahme dar; man sieht, daß der zweite Parameter auf 1 (d. h. $q(n) \leq k$ für ein $k \in \mathbb{N}$) geschrumpft, der erste allerdings auf n^3 gestiegen ist. Dieser erkaufte Ausgleich der beiden Ziele, Parameter 1 und 2 zu verkleinern, wird durch das Ergebnis von [AMLSS 92] (10) belanglos. Ihnen gelang der Beweis, daß die beiden Aussagen (7) und z. B. (3) in dem Sinne kombinierbar sind, von beiden Aussagen die jeweils optimale untere Grenze zu einem finalen Ergebnis zu verschmelzen.

In dieser Arbeit werden wir die Aussagen (3) und (8) und damit (10) (als Haupttheorem bezeichnet) beweisen. Diese neue Charakterisierung von *NP* (10) wird folgend noch als Theorem formuliert.

THEOREM 4.4 (Haupttheorem) $NP = PCP(\log n, 1)$*; oder: Zugehörigkeitsbeweise für alle NP- Sprachen können probabilistisch geprüft werden, indem -unter Verwendung von* $O(\log n)$ *vielen Zufallsbits- nur eine konstante Anzahl von Bits (des Beweises) gelesen werden.*

4.2 $NP \subseteq PCP(n^{O(1)}, 1)$

In diesem Abschnitt wird erstens ein $PCP(n^{O(1)}, 1)$-Protokoll für 3-SAT angegeben, was $NP \subseteq PCP(n^{O(1)}, 1)$ bestätigt. Zum zweiten eine Kodierung des Inputs vorgommen, die es erlaubt, nur noch kostant viele Bits des Inputs betrachten zu müssen. Drittens wird dann diese Kodierung des Inputs verwendet, um die Rekursionsidee (siehe Anfang von Abschnitt 4.2.3) auszuführen, und zwar mit dem Ergebnis aus 4.2.1, $NP \subseteq PCP(n^{O(1)}, 1)$, welches sozusagen in die Rekursion eingesetzt wird.

4.2.1 Das $NP \subseteq PCP(n^{O(1)}, 1)$ -Protokoll

Dieser Abschnitt enthält den Beweis, daß jede *NP*-Sprache einen probabilistischen Beweis besitzt, welcher transparent ist. Für das Aussuchen der zu untersuchenden Bits des Beweises sind polynomiell viele Zufallsbits nötig. Zunächst ein

Überblick über den Beweis:

- Arithmetisierung von 3-SAT
- Konstruktion des Beweises Π, der exponentielle Länge hat
- Erstellen einer Testprozedur aus 4 Einzeltests, die nur auf polynomiell viele Zufallsbits und konstant viele Bits von Π zugreifen.
- Beweis, daß die Testprozedur, wie angegeben, das erwünschte leistet, die *PCP*-Definition also erfüllt ist
- Der Schlüssel zu diesem Beweis liefert ein Lemma von Blum, Luby und Rubinfeld [BLR 90], welches in dieser Arbeit nicht bewiesen wird.

  Lemma 4.5 ([BLR 90]) *Sei $\tilde{g} : F \to F'$ (wobei F, F' Körper). Wenn für $x, y \in_{Gl} F$ gilt:*
  $$\Pr[\tilde{g}(x) + \tilde{g}(y) \neq \tilde{g}(x+y)] \;\leq\; \frac{\delta}{2}$$
  dann existiert eine lineare Funktion $g : F \to F'$, sodaß $(g, \tilde{g})$ δ-dicht sind.

THEOREM 4.6 *NP* $\subseteq$ *PCP*$(n^{O(1)}, 1)$

Beweis: Es reicht zu zeigen: 3-SAT $\in$ *PCP*$(n^{O(1)}, 1)$.

Zur Schreibweise: Wie im Beweis der *NEXP*-Version des Cook-Levin Theorems in Abschnitt 2.3.3 bedeute $\oplus, \bigoplus$ die Addition mod2 in der Menge $\mathcal{Z}_2 = \{0,1\} = {}^{\mathbb{Z}}/_{2\mathbb{Z}}$, dem Restklassenring mod2. (Auch Paritätsfunktion oder XOR-Funktion: $x_1 \oplus x_2 = (\overline{x}_1 \wedge x_2) \vee (x_1 \wedge \overline{x}_2)$). Im folgenden ist zu beachten, daß $y \oplus (x \oplus y) = x$ ergibt. (Denn $(\mathcal{Z}_2, \oplus)$ ist eine kommutative Gruppe, in der jedes Element sein eigenes Inverses ist). Ebenso ist der Begriff Summation von Elementen aus $\mathcal{Z}_2$ immer mod2 zu verstehen.

Sei f eine (Boole'sche) 3-CNF Formel; f ist genau dann erfüllbar, wenn eine erfüllende Belegung A existiert.

Zunächst wird jede Klausel C_j arithmetisiert zu $\hat{C}_j$, (siehe Abschnitt 1.1), d. h. wir konstruieren eine Menge von arithmetischen Ausdrücken in den Variablen von f. Bei der Arithmetisierung wird ein Trick angewendet: Sämtliche Variablen werden negiert, also wird x_i zu $\neg x_i$ und $\neg x_j$ zu x_j. Beispielsweise wird aus $C_j = (x_{j1} \vee \neg x_{j2} \vee x_{j3})$ die Arithmetisierung $\hat{C}_j = (1 \oplus x_{j1}) x_{j2} (1 \oplus x_{j3})$, was dem Polynom $p_j = x_{j2} + x_{j1}x_{j2} +$

$x_{j2}x_{j3} + x_{j1}x_{j2}x_{j3}$ entspricht. Werden in diesem Polynom die Variablen mit Werten aus $\mathcal{Z}_2$ belegt, sind die Additionen selbstverständlich mod 2 zu verstehen. Allgemein: $C_j(x) = \text{true} \iff \hat{C}_j(x) = 0$.

Danach müßte man also checken, ob alle arithmetisierten Klauseln, bzw. die entsprechenden Polynome vom Grad 3 bei der Belegung A den Wert Null haben.

Nun folgt eine Variablen-Transformation: neue Variablen (polynomiell viele in der Anzahl der alten) werden eingeführt und für jedes Paar von Variablen x, y definieren wir $w = xy$ und ersetzen xyz durch wz und fügen $xy + w$ $(= 0)$ hinzu. Dadurch haben wir zwar mehr Variablen, der Grad ist aber nur noch maximal 2.

Seien $x_1, x_2, ..., x_n$ die neue, erweiterte Menge von Variablen. Das Problem hat sich also reduziert auf das Checken, daß die Polynome $P_1, P_2, ..., P_l$, definiert über die neuen Variablen, bei A Null ergeben. Beachte $n = (\#\text{ alten Variablen})^{O(1)}$ und $l = n^{O(1)} = (\#\text{alten Variablen})^{O(1)}$, d. h. sowohl n als auch l sind mit der ursprünglichen Anzahl von 3-SAT Variablen polynomiell verknüpft.

Seien nun $\mathcal{P}_1$ die Menge aller Linearkombinationen der x_i und $\mathcal{P}_2$ die Menge aller Polynome vom Grad 2, mit Koeffizienten aus $\mathcal{Z}_2$:

$$\mathcal{P}_1 = \left\{ \sum_{i=1}^{n} c_i x_i \mid c_1, c_2, ... c_n \in \mathcal{Z}_2 \right\}$$

$$\mathcal{P}_2 = \left\{ c_0 + \sum_{i=1}^{n} c_i x_i + \sum_{1 \le i < j \le n} c_{ij} x_i x_j \mid c_i, c_{ij} \in \mathcal{Z}_2 \right\}$$

Wir konstuieren den Beweis Π, der alle Werte der Polynome aus $\mathcal{P}_2$ bei A enthält. Die Kodierungen von $\mathcal{P}_1$ und $\mathcal{P}_2$ haben die Längen 2^n und $\approx 2^{n^2}$. Um ein Element aus $\mathcal{P}_1$ zu wählen benötigen wir daher n Zufallsbits, für eines aus $\mathcal{P}_2$ $1 + n + \binom{n}{k} = O(n^2)$ Zufallsbits. Die Menge $\mathcal{P}_2$ sieht also folgendermaßen aus:

$$\{0, 1, x_1, ..., x_n, x_1 \oplus x_2, x_1 \oplus x_3, ..., x_4 \oplus x_{10}, ..., 1 \oplus x_1 \oplus x_3 x_4, ..., x_7 \oplus x_{16} x_{23}, ...\}$$

Beachte: $\mathcal{P}_1$ ist in $\mathcal{P}_2$ enthalten; setze alle $c_{ij} = 0$.

Wir prüfen nun probabilistisch, daß alle Polynome $P_1, P_2, ..., P_l$ bei A Null ergeben, indem zufällige Linearkombinationen von ihnen gewählt werden. Solch eine zufällige Linearkombination hat die Form:

$$P_\lambda = \sum_{i=1}^{l} \lambda_i P_i$$

wobei die λ_i's gleichverteilt und unabhängig aus $\mathcal{Z}_2$ gewählt werden; $\lambda = (\lambda_1, ..., \lambda_l)$. Sei $\mathcal{D}$ die Verteilung auf $\mathcal{P}_2$ bei dieser Wahl von P_λ. Beachte, daß die nach $\mathcal{D}$ verteilten Elemente aus $\mathcal{P}_2$ aus Linearkombinationen von Polynomen bestehen, die alle den Wert Null liefern "sollten".

Für alle $P \in \mathcal{P}_2$ sei y_P der Wert $P(A)$, der aus dem Beweis Π ersichtlich ist. Wenn man sich die Elemente in $\mathcal{P}_2$ durchnummeriert vorstellte, also z. B. Nummer(P^*)=17, wäre $P^*(A)$ das 17. Bit von Π. Wir identifizieren also $\mathcal{P}_2$ mit dem Index von Π. Falls nun Π tatsächlich eine erfüllende Belegung für f kodiert, wird jedes Polynom aus der Verteilung $\mathcal{D}$ Null ergeben.

Man beachte: Alle $\hat{C}_j(A)$, $1 \leq j \leq l$, sind jetzt irgendwo in Π enthalten. Dazu noch die Werte alle anderen möglichen Polynome in n Variablen vom Grad ≤ 2.

Nun werden folgende Tests (aus [BLR 90]) durchgeführt:

1. **Linearitätstest** $y_P \oplus y_Q = y_{P+Q}$, für P, Q, gleichverteilte Zufallselemente aus $\mathcal{P}_2$.

2. **Nachbartest** $y_Z \oplus y_{Z+1} = 1$, für Z, gleichverteiltes Zufallselement aus $\mathcal{P}_2$.

3. **Linearitätstest für die Multiplikation** $(y_R \oplus y_{R+L_1})(y_S \oplus y_{S+L_2}) = y_T \oplus y_{T+L_1L_2}$, für R, S, T gleichverteilte Zufallselemente aus $\mathcal{P}_2$ und L_1, L_2 gleichverteilte Zufallselemente aus $\mathcal{P}_1$.

4. **Eigentlicher Test** $y_U \oplus y_{U+V} = 0$, für U gleichverteiltes Zufallselement aus $\mathcal{P}_2$ und V Zufallselement aus $\mathcal{P}_2$ verteilt nach $\mathcal{D}$.

`Akzeptieren`, falls alle Tests gelingen, ansonsten `verwerfen`.

In den Tests 3 und 4 nützen wir die Idee des self-correcting von [BLR 90], um die Vertrauenswahrscheinlichkeit in die Werte aus dem Beweis zu erhöhen. Im speziellen prüfen wir im Test nicht direkt $y_{L_1}y_{L_2} = y_{L_1L_2}$, weil L_1 und L_2 aus einem sehr kleinen Teil von Π, der mit $\mathcal{P}_1$ korrespondiert, gewählt wird, wo der Beweis auch betrügen könnte. Stattdessen wenden wir auf y_{L_1}, y_{L_2} und $y_{L_1L_2}$ die self-correcting Methode an, um für den Test die korrigierten Werte zu erhalten.

In den obigen Tests brauchen wir 7 Polynome aus $\mathcal{P}_2$ und 2 aus $\mathcal{P}_1$. Dazu sind (siehe obige Feststellung) $O(n^2) + 2n = O(n^2)$ Zufallsbits nötig. Ingesamt haben wir 13 Fragen an den Beweis gestellt; das ist eine polynomielle Anzahl von Zufallsbits und eine konstante Anzahl von Fragen.

Die obige Konstruktion basiert auf der error-detecting/correcting Methode von [BLR 90]. Es ist offensichtlich, daß, wenn f erfüllbar ist, der Beweis Π die Maschine M immer zum Akzeptieren bringt. Es bleibt zu zeigen, daß für alle nicht erfüllbaren f jeder Beweis Π mit einer positiven Wahrscheinlichkeit von der Maschine verworfen wird.

Faktum 1: *Falls f nicht erfüllbar ist, verwirft M für alle Π mit Wahrscheinlichkeit $\geq 0,01$.*

Beweis von Faktum 1: Angenommen, das Gegenteil stimme. Dann gibt es ein f, welches nicht erfüllbar ist, aber die Tests 1-4 mit einer Warscheinlichkeit $\geq 0,99$ mit akzeptieren enden. Nach den Resultaten von [BLR 90], Lemma 4.5, ist der Beweis Π wirklich dicht zu einer linearen Funktion $L : \mathcal{P}_2 \rightarrow \mathcal{Z}_2$. Das bedeutet, es existiert ein L sodaß für alle $P, Q \in \mathcal{P}_2$,

$$L(P) \oplus L(Q) = L(P+Q) \tag{4.6}$$

und

$$\Pr_{P \in_{\mathrm{Gl}} \mathcal{P}_2}[y_P = L(P)] \geq 0,98 \tag{4.7}$$

Kombiniert man die Ungleichungen 4.6 und 4.7, erhält man: Für alle $P \in \mathcal{P}_2$,

$$\Pr_{R \in_{\mathrm{Gl}} \mathcal{P}_2}[L(P) = y_{R+P} \oplus y_R] \geq 0,96 \tag{4.8}$$

was aussagt, daß die self-correcting Methode für y bei beliebigem $P \in \mathcal{P}_2$ funktioniert. Aus Ungleichung 4.8 folgt

$$\Pr_{L_1, L_2 \in_{\mathrm{Gl}} \mathcal{P}_1}[L(L_1)L(L_2) = L(L_1 L_2)] \geq 0,87 \tag{4.9}$$

da

$$\underbrace{(y_R \oplus y_{R+L_1})}_{=\mathrm{L(L_1)\ mit\ Pr\ } \geq 0{,}96} \cdot \underbrace{(y_S \oplus y_{S+L_2})}_{=\mathrm{L(L_2)\ mit\ Pr\ } \geq 0{,}96} = \underbrace{(y_T \oplus y_{T+L_1 L_2})}_{=\mathrm{L(L_1 L_2)\ mit\ Pr\ } \geq 0{,}96} \quad \text{mit Pr } \geq 0,99$$

Aus dem Ausdruck 4.8 erhalten wir

$$\Pr_{R \in_{\mathrm{Gl}} \mathcal{P}_2}[L(1) = y_{R+1} \oplus y_R] \geq 0,96$$

oder

$$\Pr_{R \in_{\mathrm{Gl}} \mathcal{P}_2}[L(1) = 1] \geq 0,95 \tag{4.10}$$

da

$$\Pr_{R \in_{\mathrm{Gl}} \mathcal{P}_2}[y_{R+1} \oplus y_R = 1] \geq 0,99$$

Aber in 4.10 ist der Zufall eliminiert. Also $L(1) = 1$. Schließlich gilt

$$\Pr_{V \in_{\mathcal{D}} \mathcal{P}_2}[L(V) = 0] \geq 0,95 \tag{4.11}$$

da

$$\Pr_{\substack{U \in_{\mathrm{Gl}} \mathcal{P}_2 \\ V \in_{\mathcal{D}} \mathcal{P}_2}}[y_U \oplus y_{U+V} = 0] \geq 0,99$$

und

$$\Pr_{U \in_{\mathrm{Gl}} \mathcal{P}_2}[L(V) = y_U \oplus y_{U+V}] \geq 0,96$$

Um den Beweis von Faktum 1 fertigzustellen, benötigen wir noch einen kleinen Umweg über Faktum 2.

Faktum 2: $L(x_i)L(x_j) = L(x_i x_j)$ *für alle* $1 \leq i, j \leq n$

Beweis von Faktum 2: Für diesen Beweis wird eine einfache, aber verblüffende Tatsache benutzt, nämlich:

Faktum 3: *Sei v ein Vektor, der gleichverteilt und zufällig aus $\mathcal{Z}_2$ und w ein fester Vektor aus $\mathcal{Z}_2$, der nicht der Nullvektor ist. Daraus folgt* $\Pr_v[vw \neq 0] = \frac{1}{2}$.

Beweis von Faktum 3: Betrachte ein Bit von w, das nicht Null ist. Das Produkt mit dem korrespondierenden Bit in v ergibt 1, und zwar exakt mit der Wahrscheinlichkeit $\frac{1}{2}$. Egal was die anderen Terme aufsummiert ergeben, wenn diese zu dem Produkt addiert werden, erhält man 0 und 1 mit den gleichen Wahrscheinlichkeiten.

(F3)□

Definiere Matrizen Y und Z: $Y_{ij} = L(x_i)L(x_j)$ und $Z_{ij} = L(x_i x_j)$. Die Aussage von Faktum 2 kann nun so formuliert werden: $Y = Z$.

Betrachte $L_1, L_2 \in_{\mathrm{GL}} \mathcal{P}_1$. Sei v der Vektor der Koeffizienten von L_1, d. h. $L_1 = \sum v_i x_i$. Analog sei w der Vektor der Koeffizienten von L_2, also $L_2 = \sum w_i x_i$. Da L linear ist, haben wir:

$$L(L_1)L(L_2) = \sum_i v_i L(x_i) \sum_j w_j L(x_j) = \sum_{i,j} v_i w_j L(x_i)L(x_j) = \sum_{i,j} v_i w_j Y_{ij} = v^T Y w.$$

Analog gilt $L(L_1 L_2) = v^T Z w$.

Annahme: $Y \neq Z$.
Wir können zeigen: $\Pr_v[v^T Y \neq v^T Z] \geq \frac{1}{2}$. Betrachte $v^T(Y - Z)$, laut Annahme ist $Y - Z \neq 0$, wir können also mindestens eine Spalte finden, die nicht Null ist.

Dank dieser Spalte ist die Wahrscheinlichkeit, daß die entsprechende Komponente in $v^T(Y-Z)$ nicht Null ist, exakt $\frac{1}{2}$. Da es aber eventuell mehr solcher Spalten gibt, ist die Wahrscheinlichkeit, daß $v^T(Y-Z)$ nicht Null ergibt, mindestens $\frac{1}{2}$.

Nun sei ein v gegeben mit $v^TY \neq v^TZ$; wir zeigen $\Pr_w[v^TYw \neq v^TZw] = \frac{1}{2}$. Denn, betrachte $(v^TY - v^TZ)w$; für unabhängig gewählte u, v gilt $\Pr_{v,w}[v^TYw \neq v^TZw] \geq \frac{1}{4}$. Dieses umformuliert:

$$\Pr_{L_1, L_2 \in_{\mathrm{Gl}} \mathcal{P}_1}[L(L_1)L(L_2) \neq L(L_1L_2)] \geq \frac{1}{4}$$

Das steht im Widerspruch zu 4.9, da die Wahrscheinlichkeit kleiner als 0,13 sein müßte. $\Rightarrow$ Annahme war falsch, also $Y = Z$.

(F2)□

Faktum 2 hat folgende Interpretation: L ist eine Belegung der Variablen. Verbunden mit $L(1) = 1$ heißt das, um den Wert von L an einem Grad 2 Polynom zu bekommen, muß lediglich der Wert von L an den Variablen bekannt sein.

$L(1) = 1$ bedeutet, daß L bei konstanten Termen korrekt ist. Diese Tatsache und das Faktum 2 sagen dann aus, daß die Werte von $L(x_i)$ $1 \leq i \leq n$ den Wert von $L(P)$ für jedes $P \in \mathcal{P}_2$ festlegen.

Wir können nun den Widerspruch herleiten. Da f nicht erfüllbar ist, muß bei jeder Belegung mindestens eines der (Klausel-)Polynome $P_1, ..., P_l$ den Wert 1 haben.

Darum gilt nach Faktum 3, daß eine zufällige Linearkombination von $L(P_1), L(P_2), ..., L(P_l)$, also $\sum_{i=1}^{l} v_i L(P_i)$, $v = v_1 \cdots v_l$ Zufallsvektor aus $\mathcal{Z}_2$, mit einer Wahrscheinlichkeit $\frac{1}{2}$ nicht Null ist. Desweiteren gilt aufgrund der Eigenschaft von L $\Pr_{V \in \mathcal{D}}[L(V) = 0] = \frac{1}{2}$.
Dies ist ein Widerspruch zu 4.11 $\Rightarrow$ Annahme war falsch.

(F1)□

Mit konstant vielen Durchläufen kann die Irrtumswahrscheinlichkeit (d. h. die Tests 1-4 haben den Output=`Akzeptieren`, obwohl f nicht erfüllbar ist) unter jede beliebige Grenze gedrückt werden. $\Rightarrow$ 3-SAT $\in PCP(n^{O(1)}, 1)$ $\Rightarrow$ Behauptung. □

4.2.2 Kodierung des Inputs

Für die Rekursion im direkt anschließenden Protokoll müssen wir den Input auf eine solche Weise kodieren, daß wir ihn nur noch an $O(1)$ Stellen anschauen müssen.

Wir wenden im folgenden das Cook-Levin-Theorem an und beziehen uns direkt auf den Cook-Levin-Beweis. Die Aussage ist, jedes Problem in *NP* kann auf 3-SAT reduziert werden. Aus dem Beweis des Theorems ist ersichlich, daß für jede Sprache $L \in NP$ und für jedes n eine 3-CNF-Formel $\phi_n(x_1, x_2, ..., x_n, y_1, y_2, ..., y_{n^c})$ sodaß

$$x = (x_1 x_2 \cdots x_n) \in L \iff \exists y_1, y_2, ..., y_{n^c} \text{ mit } \phi_n(x_1, x_2, ..., x_n, y_1, y_2, ..., y_{n^c}) = 1$$

In anderen Worten, Cook-Levin gibt eine 3-CNF-Formel als eine Funktion der Inputbits, ohne zu spezifizieren, welches der Input tatsächlich ist (beachte: selbst falls wir eine 3-SAT-Formel gegeben haben, wenden wir Cook-Levin an, weil wir eine neue Formel brauchen, welche die alte als Input hat). Eine erfüllende Belegung ist nun von der Form $A: [1, n + n^c] \to \{0, 1\}$. Zwei Dinge sind zu verifizieren:

Erstens, daß A eine erfüllende Belegung für ϕ_n ist. Dafür benützen wir das vorangehende Protokoll. Da wir ϕ_n an beliebig vielen Stellen anschauen dürfen, können wir das Protokoll leicht durchführen.

Zweitens müssen wir verifizieren, daß der Input mit den Variablen in ϕ_n übereinstimmt; das heißt $A(i) = x_i, \ i = 1, 2, ..., n$ gelten muß. Um dies mittels einer konstanten Anzahl von Bits des Inputs und des Beweises zu bewerkstelligen, wird der Input wie folgt kodiert.

Der Inputstring sei x. Er wird nun kodiert als die Menge aller möglichen inneren Produkte von x. Also

$$\text{code}(x) = \{xu^T\}_{u \in \mathcal{Z}_2^{|x|}}$$

wobei die Zufallsstrings u den Index in dieser Menge bilden. Da $\mathcal{P}_2$ (unter anderem) alle Linearkombinationen von unseren neuen modifizierten Variablen (sie hießen auch $x_1, x_2, ..., x_n$, Vorsicht!) enthält, sind natürlich auch alle inneren Produkte von x auch irgendwo in $\mathcal{P}_2$ enthalten.

Um nun den Input zu prüfen, wählen wir eine konstante Anzahl von Zufallsstrings u, und checken für jeden, ob das korrespondierende innere Produkt das gleiche ist wie das innere Produkt, welches im Beweis Π steht. Da wir ja nicht wissen wie genau der Beweis ist, prüfen wir dies nicht direkt, sondern wenden den self-correcting Trick an: Für einen String u, der ja zu einem $P \in \mathcal{P}_2$ korrespondiert, wählen wir ein $Q \in_{\text{Gl}} \mathcal{P}_2$

und ckecken $y_Q \oplus y_{Q+P}$. Mit einer gewissen Wahrscheinlichkeit ergibt dies y_P, was mit xu^T übereinstimmen soll. Dies ergibt einen Test, welcher durch k-malige (k konstant) Wiederholung, die Vertrauenswahrscheinlichkeit zu jeder beliebigen erhöht.

4.2.3 Erste Rekursions-Folgerung

Wie kann es möglich sein, den Zugriff auf einen Beweis so einschränken zu können, daß nur noch eine konstante Anzahl von Bits untersucht wird? Die Idee der *Komposition von Beweisen*, genauer eine rekursive Beweisprüfung, liefert den Schlüssel dazu. Arora und Safra [AS* 92] entwickelten diese nicht ganz einfache Methode, deren Grundidee zunächst erläutert werden soll, bevor sie direkt im Anschluß und nochmals im Unterkapitel 4.3.3 unabhängig voneinander angewandt wird.

Die Rekursionsidee

Betrachte einen Verifier für eine Sprache in $OPT(r(n), q(n))$, der mit V_0 bezeichnet sei.

Ziel: Reduzieren der Anzahl von Bits des Beweises, die V_0 anschaut.

Vorgehensweise: Es existiere ferner ein weiters Protokoll. Dabei sei V_1 der Verifier, der eine Sprache in $OPT(r'(n), q'(n))$ erkennt.

Nach Definition betrachtet V_0 eine konstante Segmentanzahl des Beweises; jedes von der Größe $O(q(n))$. Anstatt nun V_0 tatsächlich auf die Segmente zugreifen zu lassen, führt V_1 sein Protokoll durch, um zu verifizieren, daß V_0 bei Zugriff auf diese Segmente akzeptieren würde.

Beachte, daß V_0 etwas in $OPT(r(n), q(n))$ verifiziert, und wir deshalb eine konstante Anzahl (die Konstante sei k) von Segmenten von jeweils beschränkter Länge betrachten können. Diese Segmente bilden den Input des rekursiven Aufrufs; deswegen muß einerseits die Anzahl der Segmente konstant sein, und andererseits jedes Segment beschränkt lang sein, sonst könnten wir ja nicht die Anzahl der Bits des neuen Beweises beschränken. Welches sind nun die neuen Grenzen für Zufalls-und Beweisbits? Klar, wir starten ein $OPT(r'(n), q'(n))$-Protokoll mit dem Input der Größe $k \cdot O(q(n)) = O(q(n))$ und dazu benutzen wir noch die Zufallsbits des V_0-Protokolls. Das ergibt ein $OPT(r(n) + r'(q(n)), q'(q(n)))$-Protokoll.

Probleme:

- Es ist nicht klar, wie $PCP(r(n), q(n)$-Protokolle in $OPT(r(n), q(n)$-Protokolle umgewandelt werden. Das bedeutet, daß es nicht klar ist, wie man den Zugriff auf eine konstante Anzahl von Segmenten der Größe $O(q(n))$ beschränken kann.

- Wenn V_1 den Beweis, "V_0 hätte eine gewisse Menge von Segmenten akzeptiert," rekursiv verifiziert, sieht er diese Segmente als Input an. Also jedesmal wenn V_1 auf seinen Input zugreift, greift er in Wirklichkeit auf einen Beweis zu, der ursprünglich gegeben war. Wenn wir also diesen Zugriff einschränken wollen, müssen wir die Inputbits von V_1 beschränken. Das bedeutet, falls V_1 eine $OPT(r'(n), q'(n))$-Maschine ist, müßten wir die Anzahl der Inputbits auf $O(q'(n))$ Bits beschränken.

Direkt im Anschluß folgt die erste Rekursionsanwendung, bei der die Anzahl der Inputbits auf $O(1)$ beschränkt werden muß, was bereits zuvor in 4.2.2 gemacht wurde. In diesem Fall ist das erste Problem auf triviale Art zu lösen: jedes Inputbit stellt ein eigenes Segment dar. In 4.3.2 wird dann ein weiteres Kodierungsschema angegeben, bei dem auf $O((\log n)^{O(1)})$ Inputbits limitiert wird. Diese Anzahl läßt keine triviale Segmentierung zu; das Lemma von Arora-Safra 4.3 muß herangezogen werden.

THEOREM 4.7

$$OPT(r(n), q(n)) \subseteq OPT(r(n) + q(n)^{O(1)}, 1)$$

Beweis: Sei L eine Sprache in $OPT(r(n), q(n))$. Betrachte eine $OPT(r(n), q(n))$-Maschine M für L. Beim Input x, mit $|x| = n$ und Beweis Π,

1. benötigt M $O(r(n))$ Zufallsbits, um $O(1)$ (also konstant viele) Segmente von Π zu lesen. Die Segmente seien $\pi_{i_1}, \pi_{i_2}, ..., \pi_{i_k}$, mit $|\pi_i| = O(q(n))$. Die Berechnungszeit von M ist polynomiell in der Länge von x;

2. verwendet M alle Bits in den Segmenten $\pi_{i_1}, \pi_{i_2}, ..., \pi_{i_k}$ und berechnet seinen Output in der Zeit $(q(n))^{O(1)}$.

Aus Theorem 4.6 und der Äquivalenz von OPT und PCP 4.3 folgt: $NP \subseteq OPT(n^{O(1)}, 1)$. Mit diesem Resultat konstruieren wir eine $OPT(r(n) + q(n)^{O(1)}, 1)$-Maschine M' für L, was den Beweis vervollständigt.

Betrachte den 2. Schritt von M. Um den Output mittels den Segmenten $\pi_{i_1}, \pi_{i_2}, ..., \pi_{i_k}$ zu berechnen, braucht M polynomielle Zeit in $q(n)$. Daher können wir diesen deterministischen Berechnungsprozess, der insbesonders ein *NP*-Problem darstellt, durch eine $OPT(n^{O(1)}, 1)$-Maschine lösen lassen.

Die neue Maschine M' arbeitet wie folgt:

1. M' benötigt $O(r(n))$ Zufallsbits, um $O(1)$ (also konstant viele) Segmente von Π zu lesen. Die Segmente seien $\pi_{i_1}, \pi_{i_2}, ..., \pi_{i_k}$, mit $|\pi_i| = O(q(n))$. Die Berechnungszeit von M' ist polynomiell in der Länge von x;

2. M' verwendet alle Bits in den Segmenten $\pi_{i_1}, \pi_{i_2}, ..., \pi_{i_k}$, zusätzlich $O(q(n)^{O(1)})$ Zufallsbits und $O(1)$ Beweisbits des neuen Beweises Π', um seine Entscheidung zu treffen.

Schritt 1 ist bei M und M' identisch. In Schritt 2 nutzt M' die π_{i_j}'s als zusätzlichen Input für das Protokoll aus Theorem 4.6, um zu entscheiden. M' braucht in Schritt 1 $O(r(n))$, und in Schritt 2 $O(q(n)^{O(1)})$ Zufallsbits, macht zusammen $O(r(n)+q(n)^{O(1)})$ Zufallsbits. Desweiteren braucht M' $O(1)$ Bits des neuen Beweises Π'. Insgesamt haben wir ein $OPT(r(n) + q(n)^{O(1)}, 1)$-Protokoll für L. $\Rightarrow$Behauptung. $\square$

4.3 $NP \subseteq PCP\left(\log n, (\log n)^{O(1)}\right)$

Völlig unabhängig vom $NP \subseteq PCP(n^{O(1)}, 1)$-Ergebnis, stellen wir nun ein weiteres *PCP*-Protokoll für *NP*-Sprachen vor. Die Zahl der Zufallsbits ist nur noch $\log n$, die der Beweisbits dagegen polylogarithmisch. Wie im vorangehenden Kapitel folgen nun 1. der Nachweis von *NP* in der bestimmten *PCP*-Klasse, 2. die Kodierung des Inputs, um eine weitere Rekursion durchzuführen und 3. die Rekursionsanwendung von $NP \subseteq PCP\left(\log n, (\log n)^{O(1)}\right)$ mittels der Kodierung des Inputs.

4.3.1 Das $NP \subseteq PCP\left(\log n, (\log n)^{O(1)}\right)$ -Protokoll

Um dieses Resultat zu erhalten, werden wir ein *PCP*-Protokoll für 3-SAT entwickeln, welches von Feige et al. [FGLSS 91] gefunden wurde. Zunächst ist eine gewisse Vorarbeit nötig.

Sei f eine (Boole'sche) 3-CNF Formel; f ist genau dann erfüllbar, wenn eine erfüllende Belegung A existiert. Der Beweis Π, der die erfüllende Belegung, in der Form einer Funktion der Variablen nach $\{0,1\}$ enthält, erweitert diese Funktion zu einem Klein-Grad-Polynom über einen endlichen Körper.

Das erste was der Verifier checken muß, ist, daß der Beweis Π wirklich ein Polynom mit kleinem Grad in jeder Variablen enthält, oder zumindest dicht zu solch einem ist. Dies ist **Teil 1**. In **Teil 2** durchläuft der Verifier ein Summenprotokoll, basierend auf der Erweitererung in einen Körper, um zu prüfen, daß sie tatsächlich erfüllbar ist.

Der Hauptunterschied zu anderen Protokollen besteht darin, wie eine erfüllende Belegung repräsentiert wird, bevor sie zu einem Klein-Grad-Polynom erweitert wird. Eine Belegung A kann als Funktion interpretiert werden:

$$A: \{0,1\}^{\log n} \rightarrow \{0,1\},$$

da es eine Bijektion zwischen $\{0,1\}^n$ und allen möglichen Funktionen dieser Art gibt. Weiter kann man die Ausgangsmenge verkleinern und gleichzeitig die Variablenmenge vergrößern; man erhält wieder eine neue aber gleichwertige Interpretation von A:

$$A: \{0,1,...,\log n-1\}^{\frac{\log n}{\log\log n}} \rightarrow \{0,1\}. \tag{4.12}$$

Das ergibt die gleiche Anzahl von Variablen, denn $(\log n)^{\frac{\log n}{\log\log n}} = n$. Nun wählen wir einen Körper der Mächtigkeit von ungefähr $(\log n)^2$. Wir sagen ungefähr, da ein Körper $\mathcal{Z}_{p^k} = ^{\mathbb{Z}}/_{p^k\mathbb{Z}}$, p prim, mit $p^k \geq (\log n)^2$ geeignet ist, und wir diesen wählen. Der Grund für diese spezielle Größe wird später klar.

Nun wird obige Funktion zu einer Funktion A' erweitert:

$$A': F^{\frac{\log n}{\log\log n}} \rightarrow F. \tag{4.13}$$

$F^{\frac{\log n}{\log\log n}}$ kann auf zwei verschiedene Arten interpretiert werden, die wir beide brauchen. (Siehe Mathematische Grundlagen, 1.1 Abschnitt "Körpererweiterungen"). Zunächst kann es als Vektorraum angesehen werden. (Zur anderen Interpretation kommen wir später!). Dieser Vektorraum besitzt wegen $|F| \approx (\log n)^2$ ungefähr $((logn)^2)^{\frac{\log n}{\log\log n}} = n^2$ verschiedene Elemente.

Nach dem Erweiterungslemma 1.5 existiert eine Erweiterung A', deren Grad $\log n - 1$ in jeder ihrer $\frac{\log n}{\log\log n}$ Variablen beträgt.

Wir benötigen jetzt einen modifizierten Multilinearitätstest. Wie der ursprüngliche Multilinearitätstest aus [BFL 91] auf Seite 44 ist auch dieser in [BFL 91] zu finden.

Wir verzichten auf die Einzelheiten des Tests, formulieren lediglich das Lemma, welches die Existenz des Tests bestätigt. Die Werte h und m sind auf unseren Fall bereits zugeschnitten.

Lemma 4.8 ([BFL 91, BFLS 91]) *Sei F ein Körper, I eine Telmenge von F, h, m positiv und ganzzahlig. Sei $f : I^m \to F$ eine beliebige Funktion. Es existiert ein probabilistischer Algorithmus, der die Funktion f auf approximative Dichte zu einem Polynom vom Grad h in allen Variablen testet, d. h.*

(i) *falls f ein Polynom vom Grad h in allen Variablen ist,* `akzeptiert` *der Test immer,*

(ii) *falls der Test* `verwirft` *kann man mit hoher Wahrscheinlichkeit schließen, daß f nicht α-dicht zu einem Polynom vom Grad h in allen Variablen ist, wobei $\alpha = \frac{4m^2h}{|I|}$.*

Der Algorithmus benötigt $hm^{O(1)}$ Werte von f.

Die Anzahl der Beweisbits, die wir für diesen Test brauchen ist also

$$\log n \cdot \left(\frac{\log n}{\log\log n}\right)^{O(1)} = (\log n)^{O(1)}$$

Die Anzahl der Zufallsbits, die wir benötigen, ist nach [BFLS 91]

$$\begin{aligned} &\tfrac{\log n}{\log\log n} \cdot \log\left(\tfrac{\log n}{\log\log n}\right) = \tfrac{\log n}{\log\log n} \cdot (\log\log n - \log\log\log n) \\ &= \tfrac{\log n}{\log\log n} \cdot O(\log\log n) = O(\log n) \end{aligned}$$

Nun können wir also annehmen, daß der Beweis tatsächlich zu einer Funktion dicht ist, welche den Grad $\log n - 1$ in jeder Variablen hat. **Teil 1** ist damit erledigt.

Kommen wir nun zu **Teil 2**; hier verifiziert der Verifier, daß der Beweis eine erfüllende Belegung enthält.

Im einzelnen müssen wir prüfen, ob die Belegung alle Klauseln von f erfüllt.

Nun führen wir das clause-check-Polynom $CC(A, \vec{c})$ ein, das 0 ergibt, falls A die Klausel $\vec{c}$ erfüllt, und $+/-\ 1$ sonst. Zur Abkürzung sei $h = \log n - 1$, $m = \frac{\log n}{\log\log n}$.

$$CC(A, \vec{c}) = \sum_{\vec{v_1} \in [0..h]^m} \sum_{\vec{v_2} \in [0..h]^m} \sum_{\vec{v_3} \in [0..h]^m} \prod_{j=1}^{3} \chi_j(\vec{c}, \vec{v_j}) \cdot (S_j(\vec{c}) - A'(\vec{v_j}))$$

wobei

- $\vec{c}$ sei eine Klauselnummer ($0 \leq \vec{c} \leq (h+1)^m$), repräsentiert als ein String in $[0..h]^m$.
- $\vec{v}$ sei eine Variablennummer ($0 \leq \vec{c} \leq (h+1)^m$), repräsentiert als ein String in $[0..h]^m$.
- Für $j = 1, 2, 3$
  $\chi_j(\vec{c}, \vec{v}) = \begin{cases} 1 & \text{falls } \vec{v} \text{ die } j\text{-te Variable in der Klausel } \vec{c} \text{ ist} \\ 0 & \text{sonst} \end{cases}$
- Für $j = 1, 2, 3$
  $S_j(\vec{c}) = \begin{cases} 1 & \text{falls das } j\text{-te Literal in der Klausel } \vec{c} \text{ nicht negiert ist} \\ 0 & \text{sonst} \end{cases}$
- Beachte $\chi_1, \chi_2\chi_3$ sind Polynome von $[0..h]^{2m} \rightarrow \{0, 1\}$, und die S_1, S_2, S_3 sind Polynome von $[0..h]^m \rightarrow \{0, 1\}$. Diese beiden Funktionen sind multilinear, d. h. sie haben den Grad 1 in allen Variablen.

Damit bedeutet

$$CC(A, \vec{c}) = 0, \quad \forall \vec{c} \in [0...h]^m,$$

daß die Belegung alle Klauseln und somit f erfüllt. Dies muß nun verifiziert werden, wofür es mehrere Möglichkeiten gibt.

1. Möglichkeit: Wir könnten einfach alle Klauseln ausprobieren. Es sind nur polynomiell viele. Dies würde aber die Anzahl der Beweisbits, die wir benötigen dürfen, nämlich polylogarithmisch viele, bei weitem übersteigen.

2. Möglichkeit: Wir testen $\sum_{\vec{c} \in [0...h]^m} (CC(A, \vec{c}))^2 \stackrel{?}{=} 0$ mit einem LFKN-Protokoll. Dies funktioniert leider nicht wie in $\mathbb{R}$ oder $\mathbb{Q}$, da wir in einem endlichen Körper arbeiten. Die Summe könnte also 0 ergeben, obwohl die einzelnen Quadrate nicht alle 0 sind.

3. Möglichkeit: Wir testen, ob $CC(A, \vec{c})$ bei einer Zufalls-Linearkombinationen Null ergibt, was uns eine große Wahrscheinlichkeit für $CC(A, \vec{c}) = 0, \quad \forall \vec{c} \in [0...h]^m$ geben würde. Im Detail

$$\sum_{y \in [0...h]^m} R^*(y) CC(y) = 0, \quad \text{mit einer beliebigen Funktion } R^* : [0...h]^m \rightarrow F$$

Um diese völlig beliebige Zufallsfunktion zu bekommen, würden wir $O(n \cdot \log |F|) = O(n^2)$ Zufallsbits benötigen, wir dürfen aber nur $O(\log n)$ verwenden.

Die Klasse der Zufallsfunktion müßte irgenwie eingeschränkt werden. Damit genug der nicht ausführbaren Möglichkeiten. Kommen wir zur

4. Möglichkeit: Nun werden wir die oben angedeutete andere Interpretation von 4.13 annehmen. Siehe dazu Mathematische Grundlagen, 1.1 Abschnitt "Körpererweiterungen". Damit haben wir einen Körper der Mächtigkeit $|F|^m$, der mit F' benannt sei. Jedes Element $e \in F'$ läßt sich darstellen als

$$e = f_0 + f_1\varrho + f_2\varrho^2 + ... + f_{m-1}\varrho^{m-1} \text{ mit } f_0, f_1, ..., f_{m-1} \in F$$

Nun definieren wir $[e]_i$ als den Koeffizienten von ϱ^i in dieser Darstellung von e, also $[e]_i = f_i$. Sei jetzt x ein Zufallselement aus F' und i eine Zufallszahl aus $\{0, ..., m-1\}$. Aus $x \in F'$ folgt $x^{\vec{c}}$, mit $\vec{c}$ als natürliche Zahl interpretiert, ist in F', also darstellbar wie oben. Die

$$\left\{[x^{\vec{c}}]_i\right\}_{\substack{0 \leq \vec{c} \leq (h+1)^m \\ 0 \leq i \leq m-1}}$$

bilden eine Familie von Polynomen. Dies ist die eingeschränkte Klasse von Zufallsfunktionen, die sog. *Pseudozufallsfunktion*, die wir gesucht haben. Mit dieser Familie gelang Feige et al. der Beweis folgenden Lemmas (h und m wie oben):

Lemma 4.9 (FGLSS 91) *Es existiert eine Familie von $O(n^3)$ Polynomen $\{R_1, R_2, ...\}$, jedes ein Grad h Polynom in m Variablen, sodaß für eine beliebige Funktion $f : [0...h]^m \to F$, wobei $(f \not\equiv 0)$, folgendes gilt:*

$$\Pr_i\left[\sum_{y \in [0...h]^m} R_i(y) f(y) = 0\right] \leq \frac{1}{100},$$

wobei ein R_i zufällig aus dieser Familie zu wählen $O(\log n)$ Zufallsbits benötigt. Diese Familie ist in der Zeit $h^{O(m)} = O(n)$ konstruierbar.

Damit sind wir in der Lage, folgendes Theorem zu beweisen.

THEOREM 4.10

$$NP \subseteq PCP\left(\log n, (\log n)^{O(1)}\right)$$

Beweis: In dem Lemma von [FGLSS 91] identifizieren wir f mit CC. Das clause-check-Polynom CC kann der Verifier selbst erzeugen. Damit können wir folgenden Test durchführen:

FGLSS-Test

Wähle ein $x \in F'$ und ein $i \in \{0, ..., m-1\}$

$$\sum_{\vec{c} \in [0...h]^m} f(\vec{c})[x^{\vec{c}}]_i \stackrel{?}{=} 0 ,$$

Bei Gleichheit `akzeptieren`, sonst `verwerfen`.

Diese Summe aufgeschlüsselt sieht folgendermaßen aus:

$$\sum_{\vec{c} \in [0...h]^m} \sum_{\vec{v_1} \in [0..h]^m} \sum_{\vec{v_2} \in [0..h]^m} \sum_{\vec{v_3} \in [0..h]^m} \left(\prod_{j=1}^{3} \chi_j(\vec{c}, \vec{v_j}) \cdot (S_j(\vec{c}) - A'(\vec{v_j})) \right) [x^{\vec{c}}]_i$$

Um die Summe zu prüfen, benutzen wir das LFKN-Protokoll siehe Unterkapitel 2.3.1.

Wichtig ist nun, daß alle Rechenschritte im Körper F durchgeführt werden. Dies gelingt dadurch, daß der Verifier die Funktionen χ_j und S_j in den Körper F erweitert. Nach dem gelungenen modifizierten Multilinearitätstest kann man A' als Polynom mit dem Grad $\log n$ in jeder Variablen betrachten. Der gesamte Ausdruck nach den Summenzeichen, kann also nun als Polynom mit dem Grad $\log n$ in jeder Variablen betrachtet werden.

Die Anzahl der Runden ist gleich die Anzahl der Variablen, also in dem Fall $4\frac{(\log n)}{\log \log n}$. In jeder Runde checkt der Verifier $\tilde{g}_i(0) + \tilde{g}_i(1) + ... + \tilde{g}_i(h) \stackrel{?}{=} r_i$, indem er einen Zufallswert aus F wählt und das Polynoman an diesem Punkt prüft. Das Polynom hat den Grad $\log n$ in jeder Variablen, so folgt mit $|F| \approx (\log n)^2$, daß die Wahrscheinlichkeit, einen Betrug nicht zu entdecken, höchstens $\frac{1}{\log n}$ ist. In $\frac{(\log n)}{\log \log n}$ Runden ist die gesamte Irrtumswahrscheinlichkeit also $\leq \frac{\log n}{\log \log n} \cdot \frac{1}{\log n} = \frac{1}{\log \log n}$.

Wieviele Bits wurden benötigt?

- Wieviele Zufallsbits? In jeder Runde mußte ein Element aus F gewählt werden, was $\approx 2 \log \log n$ Bits verbraucht. Multipliziert mit der Anzahl der Runden ergibt das $\approx 8 \log n$ Zufallsbits. Im FINAL TEST werden keine Zufallsbits gebraucht.

- Wie groß ist die Anzahl der Bits des Beweises? In jeder Runde muß der Prover ein Polynom vom Grad $\frac{(\log n)}{\log\log n}$ über F liefern, was zur Darstellung $\left(\frac{(\log n)}{\log\log n}\right)(\log|F|) \approx 2\log n$ Bits braucht. Dies multipliziert mit der Rundenanzahl ergibt $\frac{2(\log n)^2}{\log\log n}$. Im FINAL TEST werden nur 3 Werte der Belegung eingesetzt, was $3 = log|F| \approx 6\log\log n$ Bits ausmacht.

Das Summenprotokoll benötigt also $O(\log n)$ Zufallsbits und $O\left(\frac{(\log n)^2}{\log\log n}\right)$ Bits des Beweises.

Zusammen mit dem modifizierten Multilinearitätstest ergibt das $O(\log n)$ Zufallsbits und $O\left((\log n)^2\right)$ Bits des Beweises.

Ein k-maliges Wiederholen (k konstant) des oben angegebenen Tests erlaubt es uns, die Irrtumswahrscheinlichkeit unter jede beliebige Grenze zu drücken. Zusammenfassend haben wir also ein $PCP\left(\log n, (\log n)^{O(1)}\right)$-Protokoll für 3-SAT angegeben. $\Rightarrow$ 3-SAT $\in PCP\left(\log n, (\log n)^{O(1)}\right) \Rightarrow$ Behauptung. $\square$

4.3.2 Kodierung des Inputs

Wie in 4.2.2 wird jetzt ein Verfahren angegeben, wie der Input kodiert werden muß, mit dem Ziel, die nachfolgende Rekursion auszuführen. Dazu muß die Anzahl der zu lesenden Inputbits auf $O((\log n)^{O(1)})$ beschränkt werden. Ein ähnlicher Trick wie in 4.2.2 führt zum Ergebnis.

Wir wenden im folgenden wieder das Cook-Levin-Theorem an und beziehen uns direkt auf den Cook-Levin-Beweis. Die Aussage ist, jedes Problem in NP kann auf 3-SAT reduziert werden. Aus dem Beweis des Theorems ist ersichlich, daß für jede Sprache $L \in NP$ und für jedes n eine 3-CNF-Formel $\phi_n(x_1, x_2, ..., x_n, y_1, y_2, ..., y_{n^c})$ existiert, sodaß

$$x = (x_1 x_2 \cdots x_n) \in L \iff \exists y_1, y_2, ..., y_{n^c} \text{ mit } \phi_n(x_1, x_2, ..., x_n, y_1, y_2, ..., y_{n^c}) = 1$$

Eine erfüllende Belegung ist nun von der Form $A : [1, n + n^c] \to \{0, 1\}$. Zwei Dinge sind zu verifizieren:

Erstens, daß A tatsächlich eine erfüllende Belegung für ϕ_n ist. Dafür benützen wir das vorangehende Protokoll.

Zweitens müssen wir mit nur $O((\log n)^{O(1)})$ Bits des Beweises und $O(\log n)$ Zufallsbits verifizieren, daß die ersten n Variablen in ϕ_n mit dem Inputstring übereinstimmen;

das heißt $A(i) = x_i,\ i = 1, 2, ..., n$ gelten muß. Um dies mittels dieser Anzahl von Bits des Inputs und des Beweises zu bewerkstelligen, wird der Input wie folgt kodiert.

Achtung ! Was jetzt folgt entspicht zum Teil genau dem vorangehenden Protokoll.

Sei $x \in \{0,1\}^n$ der Inputstring und $h = \log n - 1$ und $m = \frac{\log n}{\log \log n}$. Wir können äquivalent den Inputstring x als Funktion $[0...h]^m \to \{0,1\}$ interpretieren. Diese Funktion wird nun nach dem Erweiterungslemma 1.5 zu einem Polynom in m Variablen und Grad h in jeder Variablen über dem Körper F (aus vorangehendem Protokoll) erweitert. Dieses Polynom nennen wir x', es wird die kodierte Form des Inputs sein.

Um mit dieser kodierten Version zu arbeiten, müssen wir sicherstellen, daß es wirklich dicht zu einem Polynom mit kleinem Grad in allen Variablen ist. Dazu verwenden wir wie zuvor den modifizierten Multilinearitätstest.

Von nun an können wir also annehmen, daß x' in der Tat dicht zu einem Polynom mit kleinem Grad in allen Variablen ist. $A(i) = x_i,\ i = 1, 2, ..., n$ wird jetzt also über den Umweg der Klein-Grad-Erweiterungen x' und A' verifiziert. Zur Erinnerung: A' war als Funktion $F^{\frac{\log n}{\log \log n}} \to F$ definiert, ebenso wie x. Deswegen können wir nun verifieren, daß x' und A' auf $[0...h]^m$ identisch sind:

$$A'(\vec{i}) - x'(\vec{i}) = 0 \quad \forall \vec{i} \in [0...h]^m$$

Wir wollen also verifizieren, daß ein Polynom über einer gewissen Menge verschwindet, allerdings ohne zuviel Zufallsbits zu verbrauchen. Wie im vorangehenden Protokoll wird die Pseudo-Zufallsfunktion und das Lemma 4.9 von [FGLSS 91] benutzt. In diesem Fall wird $h(y)$ interpretiert als $A'(y) - x'(y)$. Der Rest läuft wie zuvor.

4.3.3 Zweite Rekursions-Folgerung

THEOREM 4.11

$$OPT\left(r\left(n\right), q\left(n\right)\right) \subseteq OPT\left(r\left(n\right) + \log\left(q\left(n\right)\right), \left(\log\left(q\left(n\right)\right)\right)^{O(1)}\right)$$

Beweis: Sei L eine Sprache in $OPT(r(n), q(n))$. Betrachte eine $OPT(r(n), q(n))$-Maschine M für L. Beim Input x, mit $|x| = n$ und Beweis Π,

1. benötigt M $O(r(n))$ Zufallsbits, um $O(1)$ (also konstant viele) Segmente von Π zu lesen. Die Segmente seien $\pi_{i_1}, \pi_{i_2}, ..., \pi_{i_k}$, mit $|\pi_i| = O(q(n))$. Die Berechnungszeit von M ist polynomiell in der Länge von x;

2. betrachtet M alle Bits in den Segmenten $\pi_{i_1}, \pi_{i_2}, ..., \pi_{i_k}$ und berechnet seinen Output in der Zeit $(q(n))^{O(1)}$.

Aus Theorem 4.10 und der Äquivalenz von OPT und PCP, siehe Lemma 4.3, folgt: $NP \subseteq OPT\left(\log n, (\log n)^{O(1)}\right)$. Mit diesem Resultat konstruieren wir eine $OPT\left(r(n) + \log(q(n)), (\log(q(n)))^{O(1)}\right)$ -Maschine M' für L, was den Beweis vervollständigt.

Betrachte den 2. Schritt von M. Um den Output mittels den Segmenten $\pi_{i_1}, \pi_{i_2}, ..., \pi_{i_k}$ zu berechnen, braucht M polynomielle Zeit in $q(n)$. Daher können wir diesen deterministischen, polynomiellen Berechnungsprozess, der insbesonders ein NP-Problem darstellt, durch eine $OPT\left(\log n, (\log n)^{O(1)}\right)$-Maschine lösen lassen.

Die neue Maschine M' arbeitet wie folgt:

1. M' benötigt $O(r(n))$ Zufallsbits, um $O(1)$ (also konstant viele) Segmente von Π zu lesen. Die Segmente seien $\pi_{i_1}, \pi_{i_2}, ..., \pi_{i_k}$, mit $|\pi_i| = O(q(n))$. Die Berechnungszeit von M' ist polynomiell in der Länge von x;

2. M' verwendet alle Bits in den Segmenten $\pi_{i_1}, \pi_{i_2}, ..., \pi_{i_k}$, zusätzlich $O(\log(q(n)))$ Zufallsbits und $O\left((\log(q(n)))^{O(1)}\right)$ Beweisbits des neuen Beweises Π', um seine Entscheidung zu treffen.

Schritt 1 ist bei M und M' identisch. In Schritt 2 nutzt M' die π_{i_j}'s als zusätzlichen Input für das Protokoll aus Theorem 4.10, um zu entscheiden. M' braucht in Schritt 1 $O(r(n))$, und in Schritt 2 $O(\log(q(n)))$ Zufallsbits, macht zusammen $O(r(n) + \log(q(n)))$ Zufallsbits. Desweiteren braucht M' $O\left((\log(q(n)))^{O(1)}\right)$ Bits des neuen Beweises Π'. Insgesamt haben wir ein $OPT\left(r(n) + \log(q(n)), (\log(q(n)))^{O(1)}\right)$-Protokoll für L. $\Rightarrow$ Behauptung. □

4.4 Zusammenfassende Folgerung

Der Beweis des Haupttheorems 4.4 kann nun als eine einfache Folgerung der Ergebnisse aus den letzten beiden Unterkapiteln erbracht werden.

THEOREM 4.12

$$NP = PCP(\log n, 1)$$

Beweis:

$$
\begin{aligned}
NP &\overset{4.2}{\subseteq} OPT\left(0, n^{O(1)}\right)\\
&\overset{4.7}{\subseteq} OPT\left(0 + \log\left(n^{O(1)}\right), \left(\log\left(n^{O(1)}\right)\right)^{O(1)}\right)\\
&= OPT\left(\log n, (\log n)^{O(1)}\right)\\
&\overset{4.7}{\subseteq} OPT\left(\log n + \log\left((\log n)^{O(1)}\right), \left(\log\left((\log n)^{O(1)}\right)\right)^{O(1)}\right)\\
&= OPT\left(\log n, (\log(\log n))^{O(1)}\right)\\
&\overset{4.11}{\subseteq} OPT\left(\log n + \left((\log(\log n))^{O(1)}\right)^{O(1)}, 1\right)\\
&= OPT(\log n, 1)\\
&\overset{4.3}{=} PCP(\log n, 1)
\end{aligned}
$$

Aus Inklusion (4.4) $PCP(\log n, 1) \subseteq NP$ folgt die Behauptung direkt. □

Literaturverzeichnis

[AHU 74] A.V. Aho, J.E. Hopcroft, J.D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, 1974.

[AMLSS 92] S. Arora, R. Motwani, C. Lund, M. Sudan and M. Szegedy, On the Intractability of Approximation Problems, early draft.

[AMLSS* 92] S. Arora, R. Motwani, C. Lund, M. Sudan and M. Szegedy, Proof Verification and Hardness of Approximation Problems, in *Proc. 33rd IEEE Symp. on Foundations of Computer Science*, 1992, 14-23.

[AS 92] S. Arora and S. Safra, Approximating clique is *NP*-comlete. Unpublished manuscript, 1992.

[AS* 92] S. Arora and S. Safra Probabilistic checking of proofs, in *Proc. 33rd IEEE Symp. on Foundations of Computer Science*, 1992.

[Babai 85] L. Babai, Trading group theory for randomness, in *Proc. 17st. Ann. ACM Symp. Theory of Computing*, 1985, 421-429.

[Beige 92] Beigel, Lecture Notes, Yale University, 1992.

[BF 91] L.Babai and L. Fortnow, Arithmetization: a new method in structural complexity theory, *Computational Complexity 1* (1991),41-66. (Preliminary version appeared as: A characterization of #P by arithmetize straight line programs, in *Proc. 31st. Ann. IEEE Symp. Foundations Comp. Sci.*, 1991, 26-34.)

[BFL 91] L. Babai, L. Fortnow and C. Lund, Non-deterministic exponential time has two-prover interactive protocols. In *Proc. 31rd Ann. IEEE Symp. on Foundations of Computer Science*, 1991, 16-25.

[BFLS 91] L. Babai, L. Fortnow, L. Levin, M. Szegedy, Checking computations in polylogarithmic time. In *Proc. 23rd ACM Symp. of Theory of Comp.* 1991, 21-31.

[BGKW 88] M. Ben-Or, S. Goldwasser, J. Kilian, A. Widgerson, Multiprover interactive proofs: How to remove the intractability assumptions, in *Proc. 17st. Ann. ACM Symp. Theory of Computing*, 1988, 113-131.

[BLR 90] M. Blum, M. Luby and R. Rubinfeld, Self-testing/correcting with applications to Numerical problems. In *Proc. 22rd Ann. ACM Symp. Theory of Computing*, 1990, 73-83.

[CLR 90] T.H. Cormen, C.E. Leiserson, R.L. Rivest, Introduction to Algorithms, MIT Press, 1990.

[Cook 71] S. A. Cook, The complexity of theorem proving procedures, in *Proc. 3rd Ann. ACM Symp. Theory of Computing,* 1971, 151-158.

[Feige 91] U. Feige, On the Success Probability of the Two Provers in One Round Proof Systems, *STRUCTURE*, 1991.

[Feldm 86] P. Feldman, The optimum prover lives in *PSPACE*, manuscript, 1986.

[FGLSS 91] U. Feige, S. Goldwasser, L. Lovasz, S. Safra and M. Szegedy, Approximating clique is almost *NP*-complete. In *Proc. 32rd Ann. IEEE Symp. Foundations Comp. Sci.*, 1991, 2-12.

[Fortn 89] L. Fortnow Complexity-Theoretic Aspects of Interactiv Proof Systems, Ph. D. Thesis, Mass. Inst. of Technology, Laboratory for Computer Science, Tech. Report, MIT/LCS/TR-447, 1988.

[Freiv 79] R. Freivalds, Fast probabilistic algorithms. In *Mathematical foundations of computer science*, Springer Verlag Lecture notes in Computer science, No. 74, 1979, 57-69.

[FRS 88] L. Fortnow, J. Rompel and M. Sipser, On the power of multiprover interactive protocols, *Proc. 3rd Structure in Complexity Theory Conf.*, 1988, 156-161.

[GJ 79] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of *NP*-Completeness, W.H. Freeman, 1979.

[GLRSW 91] P. Gemmell, R. Lipton, R. Rubinfeld, M. Sudan and A. Widgerson, Self-testing/correcting for polynomials and for approximate functions. *Proc. 23rd Ann. ACM Symp. Theory of Computing,* 1991, 32-42.

[GMR 85] S. Goldwasser, S. Micali and C. Rackoff, The knowledge complexity of interactive proofs, *SIAM J. Comput. 18* (1989), 186-208. (Preliminary version appeared in *Proc. 18rd Ann. ACM Symp. Theory of Computing,* 1985, 291-304).

[Karp 72] R. M. Karp, Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Complexity of Computer Computations*, Plenum Press, 1972, 85-103.

[Levin 73] L. Levin, Universal'nyĭe perebornyĭe zadachi (Universal search problems, in Russian), *Problemy Peredachi Informatsii 9* (1973), 265-266. A corrected english translation appears in an appendix to Trahtenbrot.

[LFKN 90] C. Lund, L.Fortnow, H. Karloff and N. Nisan, Algebraic methods for interactive proof systems, in *Proc. 31rd Ann. IEEE Symp. Foundations Comp. Sci.*, 1990,1-10.

[MS 72] A. R. Meyer, L. J. Stockmeyer, The equivalence problem for regular expressions with squaring requires exponential space. *Proceedings 13rd Annual Symposium on Switching and Automata Theory,* 1972, 125-129.

[RS 92] R. Rubinfeld and M. Sudan, Testing polynomial functions efficiently and over rational domains. In *Proc. 3rd Annual ACM-SIAM Symp. on Discrete Algorithms*, 1992, 23-32.

[Rubin 90] R. Rubinfeld, A mathematical theory of self-checking, self-testing and self-correcting programs. Ph.D. Thesis, Computer Science Department, U.C. Berkeley, 1990.

[Shami 90] Shamir, *IP=PSPACE*, in *Proc. 31rd Ann. IEEE Symp. Foundations Comp. Sci.*, 1990, 11-15.

[Stock 77] STOCKMEYER, The polynomial-time hierarchy. In *Theoretical Computer Science*, 1977, 1-22.

[VanDe 70] B. L. VAN DER WAERDEN, Algebra, Vol. 1, Frederick Ungar Publishing Co., Inc., 1970, 86-91.

Erklärung:

Hiermit erkläre ich, Martin Scheifele, geborem am 19. 9. 1966 in Ulm, vorliegende Diplomarbeit selbständig und nur mit den angegebenen Mitteln und Quellen angefertigt zu haben.