Implementierung von Algorithmen zur Berechnung modularer Polynome und deren Anwendung im Algorithmus von Atkin

Diplomarbeit

von

Frank J. Lehmann

nach einem Thema von
Prof. Dr. Buchmann,
am Fachbereich 14 – Informatik –
der Universität des Saarlandes

Saarbrücken, den 29. Juli 1994

Hiermit erkläre ich an Eides Statt, daß ich diese Arbeit selbständig angefertigt und nur die im Literaturverzeichnis angegebenen Quellen verwendet habe.

Saarbrücken, den 29. Juli 1994

An dieser Stelle bedanke ich mich bei Herrn Prof. Dr. Buchmann für die Vergabe dieses sehr interessanten Themas.

Mein besonderer Dank gilt Volker Müller, der sich stets die Zeit nahm, alle meine Fragen ausführlich zu beantworten. Seine Hinweise und Erklärungen haben mir immer geholfen, einzelne Sachverhalte und Zusammenhänge besser zu verstehen.

Vielen Dank auch an Markus Maurer für die freundschaftliche Zusammenarbeit. Durch das angenehme Arbeitsklima konnten wir uns immer wieder motivieren.

Außerdem möchte ich mich bei Victor Shoup dafür bedanken, daß er uns die von ihm erstellte Software zur Verfügung gestellt hat. Auch durch seine Fragen und Anregungen hat er stets zu Verbesserungen unserer Implementierungen beigetragen.

Desweiteren bedanke ich mich bei allen Angehörigen des Lehrstuhls, die uns bei der Durchführung unserer Berechnungen unterstützt haben, insbesondere bei Franz-Dieter Berger, Thomas F. Denny, Andreas Müller, Thomas Papanikolaou, Thomas Sosnowski und Jörg Zayer.

Mein besonderer Dank gilt auch meinen Eltern, die mich immer unterstützt haben und es mir so ermöglichten, mein Studium nach meinen Vorstellungen zu gestalten.

Inhaltsverzeichnis

Einleitung

Seit einiger Zeit haben elliptische Kurven Anwendung gefunden in kryptographischen Algorithmen. So haben N. Koblitz in [Ko86] und V. Miller in [Mi86] einige Verfahren der Kryptographie auf die Gruppenstruktur elliptischer Kurven über endlichen Körpern übertragen. Die Sicherheit solcher Kryptosysteme beruht auf der Schwierigkeit, das *Diskrete Logarithmen Problem* über elliptischen Kurven zu lösen. Die besten Algorithmen zur Berechnung diskreter Logarithmen haben eine Laufzeit, die proportional zur Quadratwurzel des größten Primfaktors der „Gruppenordnung“ der jeweiligen elliptischen Kurve ist. Man kann damit also die Sicherheit von Kryptosystemen nachweisen, wenn man die Gruppenordnung (Punktanzahl) der verwendeten Kurve berechnen kann.

Erste Algorithmen zur Berechnung der Punktanzahl elliptischer Kurven über endlichen Körpern wurden von D. Schanks und R. Schoof entwickelt. Im Jahr 1988 wurde dann von A.O.L. Atkin ein Verfahren vorgeschlagen, mit dem es möglich ist, Gruppenordnungen elliptischer Kurven über „sehr großen“ endlichen Körpern, d.h. Körpern mit großer Charakteristik, zu berechnen (siehe [At88] und [At92]). Der *Algorithmus von Atkin* wird in der Dissertation [Mü94] von Volker Müller genau untersucht. Markus Maurer gibt in seiner Diplomarbeit [Ma94] eine Zusammenfassung dieses Algorithmus an und beschreibt eine Implementierung dieses Verfahrens.

Innerhalb des Algorithmus von Atkin erhält man Informationen über die Punktanzahl einer elliptischen Kurve durch die Untersuchung spezieller Polynome, sogenannter modularer Polynome. Ziel dieser Arbeit ist die Beschreibung und Implementierung von Algorithmen zur Berechnung von Polynomen, die in einem gewissen Sinne äquivalent zu den modularen Polynomen sind. Die Konstruktion solcher Polynome beschreibt Volker Müller in [Mü94] ausführlich, eine Zusammenfassung der Ideen werden wir hier angeben. Die Implementierung aller Algorithmen, die in dieser Arbeit beschrieben werden sowie der zugrunde liegenden Arithmetiken sind entstanden in Zusammenarbeit von Markus Maurer, Volker Müller, Victor Shoup und mir.

Die vorliegende Arbeit gliedert sich wie folgt:

- Wir geben zunächst einen kurzen Überblick über die mathematischen Grundlagen, die wir dann in den folgenden Kapiteln benötigen werden.

- Um die praktische Bedeutung modularer Polynome aufzuzeigen, beschreiben wir im zweiten Kapitel kurz den Algorithmus von Atkin. Dabei werden wir erklären, welche Informationen darin mit Hilfe modularer Polynome gewonnen werden können.

- Für die Herleitung der modularen Polynome werden wir bestimmte Funktionen über den komplexen Zahlen betrachten. Diese Funktionen haben die Eigenschaft, daß sie mit Hilfe sogenannter Fourier-Reihen, einer Verallgemeinerung gewöhnlicher Potenzreihen, dargestellt werden können. In den Algorithmen zur Berechnung der Polynome benötigen wir dann eine Arithmetik für solche Fourier-Reihen. Daher betrachten wir in Kapitel 3 einige Eigenschaften formaler Fourier-Reihen und stellen grundlegende Algorithmen vor, die das Rechnen mit diesen Reihen ermöglichen.

- In Kapitel 4 beschreiben wir die Struktur elliptischer Kurven über den komplexen Zahlen, und führen danach die Klasse der modularen Polynome ein. Wir werden dabei auch Polynome vorstellen, die im Algorithmus von Atkin dieselbe Aufgabe erfüllen wie die modularen Polynome, die man aber in der Praxis schneller berechnen kann.

- In den Kapiteln 5 und 6 werden wir dann Algorithmen angeben, mit deren Hilfe die zuvor eingeführten Polynome berechnet werden können. Dabei betrachten wir zwei unterschiedliche Typen von Polynomen, die anstelle der modularen Polynome verwendet werden können.

- Im letzten Kapitel beschreiben wir die Implementierung eines Programmes zur Berechnung der beschriebenen Polynome, welches verteilt auf einem Netz von Rechnern arbeitet.

Bezeichnungen

$\mathbb{N}$	die Menge der natürlichen Zahlen
$\mathbb{Z}$	die Menge der ganzen Zahlen
$\mathbb{Q}$	die Menge der rationalen Zahlen
$\mathbb{R}$	die Menge der reellen Zahlen
$\mathbb{C}$	die Menge der komplexen Zahlen
$\mathbb{F}_p$	ein Körper mit p Elementen
$\mathbb{P}$	die Menge der Primzahlen
$\mathbb{Z}/n\mathbb{Z}$	die Menge der Restklassen modulo n, $n \in \mathbb{Z}$
$R[X]$	der Polynomring über R
$\deg\,(f(X))$	der Grad des Polynoms $f(X)$
E	eine elliptische Kurve
$\mathrm{ggT}\,(a, b)$	größter gemeinsamer Teiler von a und b
$a \mid b$	a teilt b
$a \nmid b$	a teilt b nicht
$a \equiv b \pmod{n}$	$(a - b)$ ist durch n teilbar
l	Primzahl größer als 2
p	Primzahl größer als 3
$\Im(z)$	Imaginärteil einer komplexen Zahl z

Kapitel 1

Grundlagen

In diesem Abschnitt werden wir einige mathematische Grundlagen beschreiben sowie Vereinbarungen und Definitionen angeben, die wir in den folgenden Kapiteln benötigen werden.

1.1 Grundlagen der Körpertheorie

Wir werden im folgenden stets elliptische Kurven über *Körpern* betrachten. Daher werden wir zunächst einige grundlegende Definitionen und Aussagen aus der Körpertheorie angeben. Ausführliche Beschreibungen und Beweise zu diesem Sachgebiet können z.B. in [Mey2, Kap. 6] nachgelesen werden.

Unter einem **Körper** versteht man eine nichtleere Menge K mit zwei Verknüpfungen — einer Addition „+“ sowie einer Multiplikation „·“. Dabei sind sowohl $(K,+)$ als auch $(K \setminus \{0\}, \cdot)$ *abelsche Gruppen*, und zwischen Addition und Multiplikation gilt ein *Distributivgesetz*. Ein **Unterkörper** M von K ist eine nichtleere Teilmenge von K, die mit den in K definierten Operationen selbst ein Körper ist; K heißt dann auch **Erweiterungskörper** von M. Der (bzgl. Inklusion) kleinste Unterkörper von K wird als **Primkörper** von K bezeichnet. Der **algebraische Abschluß** $\overline{K}$ des Körpers K ist der minimale Erweiterungskörper von K, der die Eigenschaft besitzt, daß er die Nullstellen eines jeden Polynoms mit Koeffizienten aus K bereits enthält.

Die **Charakteristik** eines Körpers ist die kleinste natürliche Zahl n, für die $n \cdot 1 = (\underbrace{1 + \ldots + 1}_{n-mal}) = 0$; sie ist gleich 0, falls $k \cdot 1 \neq 0$ für alle k aus $\mathbb{N}$. Ein **endlicher Körper** besitzt damit also stets eine von Null verschiedene Charakteristik.

Für die Charakteristik von Körpern gilt der folgende Satz (vgl. [Mey2, Satz 6.1.2]):

1.1. Satz *Die Charakteristik eines Körpers ist entweder Null oder eine* Primzahl p, *im letzten Fall gilt* $p \cdot x = 0$ *für alle* $x \in K$.

1.2. Beispiel Betrachten wir nun einige Körper und deren Eigenschaften.

1. $\mathbb{Q}$ der Körper der *rationalen Zahlen*, $\mathbb{R}$ der Körper der *reellen Zahlen* und $\mathbb{C}$ der Körper der *komplexen Zahlen* sind Körper der Charakteristik 0.
2. Der Körper $\mathbb{Z}/p\mathbb{Z}$ (p Primzahl) der Restklassen modulo p ist ein *endlicher* Körper mit p Elementen. Die Charakterisik von $\mathbb{Z}/p\mathbb{Z}$ ist p.
3. $\mathbb{Q}$ und $\mathbb{Z}/p\mathbb{Z}$ enthalten keine echten Unterkörper; sie sind somit selbst *Primkörper*.
4. Der *algebraische Abschluß* von $\mathbb{R}$ ist $\mathbb{C}$.

Im folgenden werden wir mit $\mathbb{F}_p$ (p Primzahl) stets einen **endlichen Primkörper** bezeichnen. Der Körper $\mathbb{F}_p$ ist isomorph zu $\mathbb{Z}/p\mathbb{Z}$ und besitzt somit ebenfalls p Elemente.

Eine Eigenschaft von Körpern mit von Null verschiedener Charakteristik beschreibt der folgende Satz (vgl. [Mey2, Satz 6.1.6]).

1.3. Satz *Ist* K *ein Körper der Charakteristik* $p \neq 0$, *dann gilt für alle* $x, y \in K$ *und* $n \in \mathbb{N}$

$$(x+y)^{p^n} = x^{p^n} + y^{p^n}.$$

1.2 Elliptische Kurven über Körpern

Nachdem wir einige Grundlagen der Körpertheorie angegeben haben, wenden wir uns nun den *elliptischen Kurven* über Körpern zu. Dazu beschränken wir uns hier auf Körper der Charakteristik ungleich 2 und 3. Sei also im folgenden K ein Körper mit dieser Eigenschaft und $\overline{K}$ dessen algebraischer Abschluß.

1.4. Definition *Seien* $a, b \in K$, *so daß* $4a^3 + 27b^2 \neq 0$. *Eine* **elliptische Kurve** $E/K = (a, b)$ **über dem Körper** K *wird beschrieben durch die Gleichung*

$$E: \quad y^2 = x^3 + a \cdot x + b. \tag{1.1}$$

Die **Punktemenge** $E(\overline{K})$ **von** E *ist die Gesamtheit der Lösungen dieser Gleichung über* $\overline{K}$ *zusammen mit dem Punkt* $\mathcal{O} = (\infty, \infty)$*:*

$$E(\overline{K}) \quad = \quad \left\{ (x, y) \in \overline{K} \times \overline{K} \;\middle|\; y^2 = x^3 + a \cdot x + b \right\} \cup \{\mathcal{O}\}. \qquad (1.2)$$

Die Lösungen der Gleichung (1.1) über dem Körper K *bilden zusammen mit dem Punkt* $\mathcal{O}$ *die Menge*

$$E(K) \quad = \quad \left\{ (x, y) \in K \times K \mid y^2 = x^3 + a \cdot x + b \right\} \cup \{\mathcal{O}\} \qquad (1.3)$$

der K**-rationalen Punkte von E***.*

1.5. Bemerkung Die Gleichung (1.1) wird als **kurze Weierstraß-Normalform** bezeichnet. Für Körper der Charakteristik ungleich 2 und 3 entsteht diese aus der allgemeineren **Weierstraß-Gleichung**. Wir werden uns im folgenden stets mit elliptischen Kurven über endlichen Körpern **großer** Charakteristik oder über den komplexen Zahlen, also einem Körper der Charakteristik 0, beschäftigen. Daher können wir uns auf diese Darstellung beschränken.

1.6. Definition *Sei* $E = (a, b)$ *eine elliptische Kurve, dann heißt*

$$j(E) \quad = \quad 1728 \cdot \frac{4a^3}{4a^3 + 27b^2} \qquad (1.4)$$

die j**-Invariante** *von* E*.*

Auf der Punktmenge einer elliptischen Kurve kann eine Verknüpfung (Addition) definiert werden. Damit wird diese Menge dann sogar zu einer Gruppe. Wie diese Addition von Punkten der Kurve definiert wird, beschreibt der folgende Satz.

1.7. Satz *Sei* $E/K = (a, b)$ *eine elliptische Kurve und* $E(\overline{K})$ *die zugehörige Punktmenge von E. Dann bildet* $E(\overline{K})$ *eine (additive) abelsche Gruppe, wobei die Verknüpfung von Punkten dieser Menge durch folgendes Gesetz gegeben wird:*

- $\mathcal{O}$ *ist das neutrale Element in* $E(\overline{K})$*.*
- *Für einen von* $\mathcal{O}$ *verschiedenen Punkt* $P = (x, y) \in E(\overline{K})$ *wird das Inverse gegeben durch*
$$-P \quad = \quad (x, -y) \;\in\; E(\overline{K}).$$
- *Für zwei Punkte* $P = (x, y)$ *und* $Q = (x, -y)$ *ist* $P + Q = \mathcal{O}$*.*

- *Sind $P = (x_1, y_1)$ und $Q = (x_2, y_2)$ Punkte aus $E(\overline{K})$ mit $Q \neq -P$ so wird die Summe $P + Q = (x_3, y_3)$ definiert durch:*

$$\begin{aligned} x_3 &= -x_1 - x_2 + \lambda^2 \\ y_3 &= -y_1 + \lambda \cdot (x_1 - x_3) \end{aligned}$$

wobei

$$\lambda = \begin{cases} \frac{y_1 - y_2}{x_1 - x_2}, & \textit{falls } P \neq Q \\ \frac{3x_1^2 + a}{2y_1}, & \textit{falls } P = Q. \end{cases}$$

Beweis: Siehe dazu [Mü91, Satz 1.1]. ■

1.8. Bemerkung Da die Verknüpfung von Punkten ausschließlich mit Hilfe von Additionen, Multiplikationen und Invertierungen der Punktkoordinaten berechnet werden kann, können wir erkennen, daß die Koordinaten der Summe zweier Punkte stets in demselben Körper liegen wie die der Ausgangspunkte. Dies hat zur Folge, daß die oben definierte Punktaddition eingeschränkt auf die Menge der K-rationalen Punkte wiederum einen Punkt dieser Menge liefert. Damit ist $E(K)$ sogar eine *Untergruppe* von $E(\overline{K})$.

Da also die Punktmenge einer elliptischen Kurve eine Gruppenstruktur aufweist, existieren auch *Endomorphismen* dieser Punktgruppe.

1.9. Definition *Sei E/K eine über dem Körper K definierte elliptische Kurve und $E(\overline{K})$ die zugehörige Punktgruppe. Dann bildet die Gesamtheit der Endomorphismen dieser Gruppe einen Ring, den* **Endomorphismenring End($E(\overline{K})$)**. *Darin sind die Verknüpfungen definiert durch*

$$\begin{aligned} (\phi_1 + \phi_2)(P) &= \phi_1(P) + \phi_2(P) \\ (\phi_1 \cdot \phi_2)(P) &= \phi_1(\phi_2(P)). \end{aligned}$$

1.10. Beispiel Einfache Beispiele für Endomorphismen einer Punktgruppe sind die Abbildungen

$$n \cdot P = \underbrace{P + \ldots + P}_{n-mal} \quad \text{für } n \in \mathbb{N}$$

und

$$-n \cdot P = \underbrace{(-P) + \ldots + (-P)}_{n-mal} \quad \text{für } n \in \mathbb{N}.$$

Schließlich betrachten wir noch spezielle Abbildungen zwischen verschiedenen elliptischen Kurven (vgl. [Si85, Seite 70]).

1.11. Definition *Seien E_1 und E_2 elliptische Kurven über einem Körper K. Eine* **Isogenie** *zwischen E_1 und E_2 ist ein Gruppen-Homomorphismus zwischen den Punktgruppen der beiden Kurven*

$$\phi \; : \; E_1(\overline{K}) \longrightarrow E_2(\overline{K}).$$

E_1 und E_2 heißen **isogen**, *falls eine Isogenie ϕ zwischen ihnen existiert, für die $\phi(E_1) \neq \{\mathcal{O}\}$ gilt.*

Abschließend betrachten wir noch eine spezielle Klasse von elliptischen Kurven.

1.12. Definition *Sei $E = E/\mathbb{F}_p$ eine elliptische Kurve über dem endlichen Primkörper $\mathbb{F}_p$. Dann heißt E* **supersingulär**, *falls die Untergruppe der $\mathbb{F}_p$-rationalen Punkte von E genau $p+1$ Elemente enthält.*

1.3 Meromorphe Funktionen

In späteren Kapiteln werden wir bestimmte komplexwertige Funktionen verwenden, die sogenannten *meromorphen* Funktionen. Daher werden wir hier nun kurz einige Definitionen aus der Funktionentheorie angeben, die diesen Begriff erläutern (vgl. dazu [FiLieb80]).

1.13. Definition *Eine* **holomorphe Funktion** *auf einer offenen Teilmenge $U \subseteq \mathbb{C}$ ist eine auf U komplex differenzierbare Funktion f.*

1.14. Definition *Es sei z_0 ein Punkt der komplexen Ebene und f eine Funktion, die auf einer Umgebung von z_0 mit Ausnahme des Punktes z_0 selbst definiert und holomorph ist. Gilt weiterhin $\lim_{z \to z_0} |f(z)| = +\infty$, so heißt z_0* **Pol von** *f.*

Damit können wir nun eine weitere Klasse von Funktionen definieren.

1.15. Definition *Eine* **meromorphe Funktion** *auf einer offenen Menge $U \subseteq \mathbb{C}$ ist eine holomorphe Funktion $f : U - P_f \to \mathbb{C}$, wobei*

1. *P_f eine diskrete Teilmenge von U ist,*
2. *die Punkte von P_f Pole von f sind.*

Ist eine offene Teilmenge G von $\mathbb{C}$ sogar ein Gebiet (vgl. [FiLieb80, Def. 3.2, Seite 14]), so bildet die Gesamtheit aller auf G meromorphen Funktionen einen Körper ([FiLieb80, Satz 3.2, Seite 142]).

1.4 Modulgruppen

In diesem Abschnitt beschreiben wir spezielle Teilmengen der 2×2-Matrizen über $\mathbb{Z}$ und die von diesen induzierten Transformationen.

1.16. Definition *Die Teilmenge*

$$\Gamma \quad = \quad \left\{ \begin{pmatrix} a & b \\ c & d \end{pmatrix} \in \mathbb{Z}^{2,2} \,\middle|\, ad - bc = 1 \right\}$$

der Matrizen aus $\mathbb{Z}^{2,2}$ *mit Determinante* 1 *heißt* **(homogene) Modulgruppe**. *Weiter bezeichne* I *die Einheitsmatrix* $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$.

Alle Elemente $M \in \Gamma$ sind somit invertierbar, denn es gilt für $M = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \in \Gamma$:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \cdot \begin{pmatrix} d & -b \\ -c & a \end{pmatrix} \quad = \quad \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

Sei $\mathcal{H}$ die obere komplexe Halbebene, d.h. die Menge aller komplexen Zahlen mit positivem Imaginärteil

$$\mathcal{H} \quad = \quad \{\, \omega \in \mathbb{C} \mid \Im(\omega) > 0 \,\} \tag{1.5}$$

und

$$\mathcal{H}^* \quad = \quad \mathcal{H} \,\cup\, \mathbb{Q} \,\cup \{i\infty\}. \tag{1.6}$$

Dann *operiert* Γ linear gebrochen auf $\mathcal{H}^*$, d.h. jede Matrix $M = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \in \Gamma$ induziert eine **unimodulare Transformation**:

$$\begin{aligned} M \,:\, \mathcal{H}^* &\longrightarrow \mathcal{H}^* \\ \tau &\longmapsto M(\tau) = \frac{a\tau + b}{c\tau + d}. \end{aligned}$$

1.17. Bemerkung In den folgenden Kapiteln werden wir stets die Elemente aus Γ mit den von ihnen induzierten Transformationen identifizieren und nennen $M \in \Gamma$ eine Transformation. Desweiteren können wir den Zusammenhang zwischen Matrizen und Transformationen auch erweitern auf beliebige Matrizen aus $\mathbb{Z}^{2,2}$ mit einer von 0 verschiedenen Determinante.

Wir betrachten nun noch spezielle Untergruppen von Γ.

1.18. Definition *Sei n eine beliebige ganze Zahl. Dann heißt*

$$\Gamma_0(n) \quad = \quad \left\{ \left(\begin{array}{cc} a & b \\ c & d \end{array} \right) \in \Gamma \;\middle|\; c \equiv 0 mod\ n \right\} \tag{1.7}$$

eine **Kongruenz-Untergruppe** *von* Γ.

Abschließend betrachten wir noch das Verhalten beliebiger meromorpher Funktionen unter den Transformationen aus Γ. Wir interessieren uns dabei insbesondere für solche Funktionen, die invariant sind unter diesen Abbildungen.

1.19. Definition *Sei f eine auf $\mathcal{H}$ meromorphe Funktion. Dann heißt*

$$I_f \quad = \quad \{\ M \in \Gamma \ \mid\ f(M(\tau)) = f(\tau) \quad \textit{für alle} \ \ \tau \in \mathcal{H}\ \}$$

die **Invarianzgruppe von** f.

Kapitel 2

Der Algorithmus von Atkin

In diesem Kapitel werden wir beschreiben, wie modulare Polynome dazu verwendet werden können, die Punktanzahl elliptischer Kurven über endlichen Körpern zu bestimmen. Ein Algorithmus, der sich dieser Polynome bedient, wurde von Oliver Atkin entwickelt. Dieses Verfahren werden wir nun kurz skizzieren, um zu verdeutlichen, welche Funktion die modularen Polynome dabei erfüllen.

Eine genaue Untersuchung des Algorithmus kann man in [Mü94] nachlesen. Eine Beschreibung einer Implementierung dieses Algorithmus sowie praktische Resultate sind in [Ma94] zu finden.

Sei eine elliptische Kurve $E = (a, b) \in \mathbb{F}_p^2$ gegeben. Nach Definition ist die Gruppe der $\mathbb{F}_p$-rationalen Punkte $E(\mathbb{F}_p)$ eine Teilmenge von $\mathbb{F}_p \times \mathbb{F}_p$ und somit insbesondere endlich. Ziel ist es nun, die Anzahl $\#E(\mathbb{F}_p)$ der $\mathbb{F}_p$-rationalen Punkte von E zu berechnen. Eine erste Abschätzung für die gesuchte Gruppenordnung liefert uns der folgende Satz.

2.1. Satz (Hasse) *Sei E eine elliptische Kurve über einem endlichen Körper $\mathbb{F}_q$ der Charakteristik p. Dann gilt für die Anzahl $\#E(\mathbb{F}_q)$ der $\mathbb{F}_q$-rationalen Punkte von E*

$$|\,\#E(\mathbb{F}_q) - (q+1)\,| \quad \leq \quad 2\sqrt{q}. \tag{2.1}$$

Wir betrachten hier nur elliptische Kurven über endlichen Primkörpern. Daher sei im folgenden stets p eine Primzahl und $\mathbb{F}_p$ der Primkörper mit p Elementen. Dann können wir für eine über $\mathbb{F}_p$ definierte elliptische Kurve E die Anzahl $\#E := \#E(\mathbb{F}_p)$ der $\mathbb{F}_p$-rationalen Punkte darstellen in der Form :

$$\#E \quad = \quad p + 1 - c, \qquad \text{für ein } c \in \mathbb{Z} \;\; \text{mit } \; |c| \leq 2\sqrt{p}. \tag{2.2}$$

Zur Bestimmung der gesuchten Gruppenordnung genügt es somit, den Wert c in der *Hasse-Gleichung* (2.2) zu bestimmen. Da wir uns speziell für elliptische Kurven über Körpern *großer* Charakteristik interessieren (d.h. $p > 10^{375}$), ist damit auch c sehr groß. Daher wird im Algorithmus von Atkin zunächst versucht c modulo „kleiner“ Primzahlen l ($l < 900$) zu berechnen bzw. die Anzahl der möglichen Werte für $c \bmod l$ einzuschränken. Danach können die Teilergebnisse dann mit Hilfe des *Chinesischen Restsatzes* kombiniert werden. Wir werden den Algorithmus nun kurz beschreiben. Wir geben dabei aber keinerlei Beweise zu den verwendeten Aussagen an; diese sind in [Ma94], [Mü91] und [Mü94] zu finden.

Die Konstante c der Hasse-Gleichung (2.2) tritt auch in einem anderen Zusammenhang auf. Dazu betrachten wir einen speziellen Endomorphismus der gegebenen Kurve.

2.2. Definition *Die Abbildung*

$$\begin{array}{rrcl} \Phi: & E(\overline{\mathbb{F}_p}) & \longrightarrow & E(\overline{\mathbb{F}_p}) \\ & (x,y) & \longmapsto & (x^p, y^p) \end{array}$$

heißt der **Frobenius-Endomorphismus** *von E.*

Mit Hilfe von Satz 1.3 rechnet man leicht nach, daß diese Abbildung wirklich Punkte der Kurve ineinander überführt. Eine wichtige Eigenschaft des Frobenius-Endomorphismus liefert der folgende Satz.

2.3. Satz *Sei E eine über $\mathbb{F}_p$ definierte elliptische Kurve, und sei weiter die Anzahl $\#E$ der $\mathbb{F}_p$-rationalen Punkte von E gegeben durch die Hasse-Gleichung*

$$\#E = p + 1 - c.$$

Dann genügt der Frobenius-Endomorphismus von E der Gleichung

$$\Phi^2 - c \cdot \Phi + p = 0 \tag{2.3}$$

im Endomorphismenring der Kurve, d.h. für jeden Punkt $P \in E(\overline{\mathbb{F}_p})$ gilt:

$$\left(\Phi^2 - c \cdot \Phi + p\right)(P) = \mathcal{O}.$$

Damit können wir die Anzahl der $\mathbb{F}_p$-rationalen Punkte einer elliptischen Kurve bestimmen, wenn wir die Konstante c in der zugehörigen Gleichung des Frobenius-Endomorphismus der Kurve kennen. Wie wir zuvor bereits erwähnt haben, wollen wir zunächst den Wert für c modulo kleiner Primzahlen l bestimmen. Dazu betrachten wir eine spezielle Untergruppe von $E(\overline{\mathbb{F}_p})$.

2.4. Definition *Sei E eine über $\mathbb{F}_p$ definierte elliptische Kurve, und sei $n \in \mathbb{N}$ beliebig. Dann heißt die Menge*

$$E[n] \quad = \quad \left\{ P \mid P \in E(\overline{\mathbb{F}_p}) \text{ mit } n \cdot P = \mathcal{O} \right\} \tag{2.4}$$

die **n-Torsionsgruppe** *von E.*

2.5. Bemerkung Ein Punkt P der elliptischen Kurve ist also genau dann Element der n-Torsionsgruppe, wenn n ein *Vielfaches* der Punktordnung von P ist. Betrachten wir speziell für Primzahlen l die zugehörige l-Torsionsgruppe von E, so haben alle von $\mathcal{O}$ verschiedene Punkte aus $E[l]$ exakt die Punktordnung l.

Eine wichtige Aussage über die Struktur der Torsionsgruppen von E liefert der folgende Satz.

2.6. Satz *Sei die elliptische Kurve E über $\mathbb{F}_p$ definiert, und sei weiter $n \in \mathbb{N}_{>1}$ teilerfremd zu p. Dann gilt:*

$$E[n] \quad \cong \quad (\mathbb{Z}/n\mathbb{Z}) \times (\mathbb{Z}/n\mathbb{Z}). \tag{2.5}$$

Insbesondere ist somit

$$\#E[n] \quad = \quad n^2,$$

und $E[n]$ ist nicht die triviale Untergruppe $\{\mathcal{O}\}$.

Betrachten wir nun für eine Primzahl l ($l < p$) die Einschränkung des Frobenius-Endomorphismus auf die zugehörige l-Torsionsgruppe. Für einen beliebigen Punkt $Q \in E[l]$ gilt dann wegen der Homomorphieeigenschft von Φ:

$$l \cdot \Phi(Q) \quad = \quad \Phi(l \cdot Q) \quad = \quad \Phi(\mathcal{O}) \quad = \quad \mathcal{O}.$$

Damit ist auch $\Phi(Q)$ eine Element von $E[l]$, und die Einschränkung von Φ auf diese Untergruppe liefert einen Endomorphismus von $E[l]$:

$$\Phi_l : \quad E[l] \longrightarrow E[l]. \tag{2.6}$$

Da der Endomorphismus Φ_l mit dem Frobenius-Endomorphismus auf den Punkten aus $E[l]$ übereinstimmt, folgt aus Satz 2.3 für alle $P \in E[l]$

$$\left(\Phi_l^2 - c\Phi_l + p\right)(P) \quad = \quad \mathcal{O}.$$

Da weiterhin alle Punkte der l-Torsionsgruppe Ordnung l besitzen, gilt

$$(k \cdot l + r) \cdot P \quad = \quad \underbrace{k \cdot l \cdot P}_{=\mathcal{O}} + r \cdot P \quad = \quad r \cdot P$$

für alle $P \in E[l]$ und $k, r \in \mathbb{N}$. Unter Verwendung der letzten beiden Aussagen kann der folgende Satz gezeigt werden.

2.7. Satz *Sei E eine über $\mathbb{F}_p$ definierte elliptische Kurve, und l eine Primzahl ($l \neq p$). Dann genügt die Einschränkung Φ_l des Frobenius-Endomorphismus auf die l-Torsionsgruppe von E der* charakteristischen Gleichung

$$\Phi_l^2 - c_l\Phi + q = 0 \tag{2.7}$$

in $\mathrm{End}(E[l])$*. Dabei gilt insbesondere*

$$c_l \equiv c \pmod{l} \quad \textit{und} \quad q \equiv p \pmod{l}. \tag{2.8}$$

In (2.7) heißt c_l die *Spur* von Φ_l. Somit können wir die Konstante c der Hasse-Gleichung modulo l bestimmen, wenn wir die Spur von Φ_l kennen. Wir werden daher im folgenden beschreiben, wie wir diese Spur berechnen können.

Das *charakteristische Polynom* von Φ_l ist wegen (2.7)

$$\chi(X) = X^2 - c_lX + q \in \mathbb{F}_l[X]. \tag{2.9}$$

Die Nullstellen α und β dieses Polynoms vom Grad 2 liegen damit entweder in $\mathbb{F}_l$ oder aber in $\mathbb{F}_{l^2}$, und wir erhalten damit über $\mathbb{F}_{l^2}[X]$ die Zerlegung

$$\chi(X) = (X-\alpha)\cdot(X-\beta). \tag{2.10}$$

Durch Ausmultiplizieren der beiden Linearfaktoren erhalten wir:

$$c_l \equiv \alpha+\beta \pmod{l} \quad \text{und} \quad q \equiv \alpha\cdot\beta \pmod{l}. \tag{2.11}$$

Daraus können wir erkennen, daß die Spur c_l von Φ_l eng verbunden ist mit den Nullstellen des charakteristischen Polynoms. Die Idee des Algorithmus von Atkin besteht nun darin, durch Untersuchung der Nullstellen α und β von $\chi(X)$ nähere Informationen über c_l zu erhalten. Dabei wird zunächst festgestellt, in welchem Körper α und β liegen (in $\mathbb{F}_l$ oder in $\mathbb{F}_{l^2}$), ob sie voneinander verschieden sind oder ob sie übereinstimmen. Dabei liefert der Algorithmus auch die Ordnung des Elementes $\alpha\cdot\beta^{-1}$ in dem entsprechenden Körper. Mit Hilfe dieser Informationen können dann mögliche Werte für c_l berechnet werden. Für $x = \alpha\cdot\beta^{-1}$ erhalten wir aus (2.11)

$$q \equiv \alpha\cdot\beta \equiv (\alpha\beta^{-1})\cdot\beta^2 \equiv x\cdot\beta^2 \pmod{l}$$

sowie

$$c_l \equiv \alpha+\beta \equiv \beta\cdot x+\beta \equiv (x+1)\cdot\beta \pmod{l}.$$

Durch Kombination der beiden Gleichungen ergibt sich

$$c_l \equiv (x+1)\cdot\beta \equiv \pm\,(x+1)\cdot\sqrt{q\cdot x^{-1}} \pmod{l}. \tag{2.12}$$

Stimmen speziell α und β überein, so erhalten wir hieraus wegen $x = \alpha \cdot \beta^{-1} = 1$

$$c_l \equiv \pm 2 \cdot \sqrt{q}. \tag{2.13}$$

Allerdings wird das Element $\alpha \cdot \beta^{-1}$ im Algorithmus nicht korrekt bestimmt, sondern lediglich dessen Ordnung d in F_l (bzw. in $\mathbb{F}_{l^2}$). Für jedes Element x der Ordnung d im entsprechenden Körper werden dann die zwei möglichen Lösungen (2.12) berechnet. Ist dieses Verfahren dann für hinreichend viele Primzahlen l durchgeführt, so kann die Gesamtlösung in einer abschließenden „Babystep-/Giantstep-Phase“ mit Hilfe des Chinesischen Restsatzes aus einer Menge möglicher Lösungen bestimmt werden (vgl. dazu [Ma94]).

Betrachten wir nun, wie die Lage der Nullstellen α und β sowie die Ordnung von $\alpha \cdot \beta^{-1}$ bestimmt werden kann. Dazu beachten wir, daß α und β immer in demselben Körper liegen. Ist nämlich α ein Element aus $\mathbb{F}_l$, so ist

$$\beta = c_l - \alpha$$

ebenfalls ein Element dieses Körpers. Weiterhin können die beiden Nullstellen nur dann übereinstimmen, wenn sie bereits in $\mathbb{F}_l$ liegen. Denn für $\alpha = \beta$ gilt dann

$$c_l = 2 \cdot \alpha \quad \Leftrightarrow \quad \alpha = 2^{-1} \cdot c_l \quad \in \mathbb{F}_l.$$

Genau dann, wenn α und β in $\mathbb{F}_l$ liegen, sind sie *Eigenwerte* von Φ_l. Dann existiert also insbesondere ein Punkt $P \in E[l]$ mit der Eigenschaft:

$$\Phi_l(P) = \alpha \cdot P.$$

Betrachten wir einen beliebigen Punkt Q in der von P erzeugten Untergruppe $\langle P \rangle$ von $E[l]$, so können wir beobachten:

$$\Phi_l(Q) = \Phi_l(k \cdot P) = k \cdot \Phi_l(P) = k \cdot \alpha \cdot P \in \langle P \rangle.$$

Somit bildet Φ_l die zyklische Untergruppe $\langle P \rangle$ in sich selbst ab; $\langle P \rangle$ ist *invariant* unter Φ_l. Wir können damit also feststellen, ob α und β in $\mathbb{F}_l$ liegen, indem wir alle zyklischen Untergruppen von $E[l]$ danach untersuchen, ob sie invariant sind unter Φ_l oder nicht. Aussagen über Anzahl und Struktur der Untergruppen von $E[l]$ liefert der folgende Satz.

2.8. Satz *Sei für eine Primzahl l ($l \neq p$) $E[l]$ die l-Torsionsgruppe der elliptischen Kurve $E/\mathbb{F}_p$. Dann besitzt $E[l]$ neben den trivialen Untergruppen $\{\mathcal{O}\}$ und $E[l]$ genau $l+1$ verschiedene Untergruppen $C_1, \ldots, C_{l+1}$. Jede dieser Untergruppen ist* **zyklisch** *von der Ordnung l, und insbesondere gilt*

$$C_i \cap C_j = \{\mathcal{O}\} \qquad \text{für } i \neq j.$$

Genau diese Untergruppen C_i $(1 \leq i \leq l+1)$ müssen wir also untersuchen. Ist C eine dieser Untergruppen und *nicht* invariant unter Φ_l, so kann man zeigen, daß eine minimale Potenz Φ_l^d existiert, für die $\Phi_l^d(C) \subseteq C$ gilt. Aussagen über den Zusammenhang zwischen der Lage von α und β und der Invarianz der Untergruppen C_i $(1 \leq i \leq l+1)$ unter Φ_l bzw. unter minimalen Potenzen von Φ_l, erhalten wir aus dem nächsten Satz.

2.9. Satz *Seien $C_1, \ldots, C_{l+1}$ die l-elementigen Untergruppen von $E[l]$, und Φ_l die Einschränkung des Frobenius-Endomorphismus auf die l-Torsionsgruppe. Seien weiter α und β die Nullstellen von $\chi(X)$. Dann gelten die folgenden Aussagen:*

1. *Ist $\alpha = \beta \in \mathbb{F}_l$, so können zwei Fälle auftreten:*
   - *Alle Untergruppen C_i $(1 \leq i \leq l+1)$ sind invariant unter Φ_l, oder*
   - *genau eine dieser Gruppen ist invariant unter Φ_l und alle übrigen unter der minimalen Potenz Φ_l^l.*
2. *Sind $\alpha \neq \beta \in \mathbb{F}_l$, so sind genau zwei der Untergruppen invariant unter Φ_l, und alle anderen unter der minimalen Potenz Φ_l^d. Dabei ist d gleich der Ordnung von $\alpha \cdot \beta^{-1}$ in $\mathbb{F}_l$.*
3. *Sind $\alpha \neq \beta \in \mathbb{F}_{l^2}$, so sind alle Untergruppen C_i $(1 \leq i \leq l+1)$ erst unter Φ_l^d invariant, und dabei ist d die Ordnung von $\alpha \cdot \beta^{-1}$ in $\mathbb{F}_{l^2}$.*

Damit ist also das Verhalten der Untergruppen von $E[l]$ bis auf wenige Möglichkeiten eingeschränkt. Wir können auch erkennen, daß die C_i $(1 \leq i \leq l+1)$ in allen Fällen unterschiedliches Verhalten zeigen. Damit können wir auch auf die Lage von α und β zurückschließen, wenn wir wissen, unter welchen Potenzen von Φ_l die C_i invariant sind. Wir werden nun ein Hilfsmittel kennenlernen, das es uns ermöglicht festzustellen, welche minimale Potenz Φ_l^d von Φ_l eine gegebene echte Untergruppe C von $E[l]$ in sich selbst abbildet.

2.10. Satz *Sei E eine nicht supersinguläre elliptische Kurve über $\mathbb{F}_p$, und E nicht isogen zu einer Kurve mit $j-$Invariante 0 oder 1728. Sei weiter C eine endliche Untergruppe von E. Dann existieren eine elliptische Kurve E_C und eine Isogenie*

$$\varphi : \quad E \longrightarrow E_C$$

mit Kern $(\varphi) = C$. Dabei ist E_C bis auf Isomorphie eindeutig bestimmt. Die minimale Potenz von Φ_l, unter der die Untergruppe C invariant ist, ist Φ_l^d für

$$d \quad = \quad min \left\{ k \mid j(E_C) \in \mathbb{F}_{p^k} \right\}.$$

Mit Hilfe dieses Satzes können wir nun die gewünschten Aussagen über die Nullstellen von $\chi(X)$ gewinnen. Dazu müssen wir zu jeder echten Untergruppe C von $E[l]$ bestimmen, in welchem Erweiterungskörper von $\mathbb{F}_p$ die j-Invariante der zugehörigen Kurve E_C liegt. Daraus können wir ablesen, unter welchen Potenzen von Φ_l die Untergruppen invariant sind, und abschließend mit Hilfe von Satz 2.9 die Lage von α und β bestimmen.

Wir haben somit das Problem, die Anzahl der $\mathbb{F}_p$-rationalen Punkte von E zu bestimmen darauf reduziert, die Erweiterungskörper zu bestimmen, in denen die j-Invarianten der Kurven E_C aus Satz 2.10 liegen. Damit sind wir beim Thema dieser Arbeit angelangt. Wir werden in den folgenden Kapiteln ein Polynom beschreiben, das genau diese j-Invarianten als Nullstellen besitzt, das **l-te modulare Polynom**. Dieses ist ein Polynom in zwei Unbestimmten und hat sämtlich ganzzahlige Koeffizienten

$$\Psi_l(X,Y) \quad = \quad \sum_{\lambda=0}^{l+1}\sum_{\kappa=0}^{k} a_{\lambda,\kappa}\cdot Y^{\kappa}\cdot X^{\lambda}. \tag{2.14}$$

Setzt man für die Unbestimmte Y die j-Invariante der Kurve E ein, so gilt

$$\Psi_l(j(E_C), j(E)) \quad = \quad 0$$

für alle j-Invarianten $j(E_C)$ aus Satz 2.10. Insbesondere ist also

$$\Psi_l(X) \quad = \quad \Psi_l(X, j(E)) \quad = \quad \prod_{\substack{C\leq E[l] \\ C\neq\{\mathcal{O}\}}} (X - j(E_C))\,. \tag{2.15}$$

Dann spiegeln die Grade der irreduziblen Faktoren dieses Polynoms genau die Grade der minimalen Erweiterungkörper wider, in denen die j-Invarianten $j(E_C)$ liegen. Zur Untersuchung dieser Grade führen wir noch den Begriff des *Zerfällungstyps* eines Polynoms ein.

2.11. Definition *Sei $f(X) \in \mathbb{F}_p[X]$ ein beliebiges Polynom mit Koeffizienten aus $\mathbb{F}_p$. Sei weiter*

$$f(X) = f_1(X)\cdot f_2(X)\cdot\dots\cdot f_n(X)$$

die Zerlegung von $f(X)$ in irreduzible Faktoren mit

$$deg\,(f_i(X)) \quad = \quad d_i \quad \geq \quad 1 \qquad \text{für} \;\; 1 \leq i \leq n.$$

Unter dem **Zerfällungstyp von** *$f(X)$ verstehen wir die Liste $(d_1\, d_2\, \dots\, d_n)$ der Grade der Faktoren von $f(X)$. Dabei gelte $d_1 \leq d_2 \leq \dots \leq d_n$.*

2.12. Beispiel Analog können wir auch den Zerfällungstyp eines Polynoms über den ganzen Zahlen definieren. Betrachten wir ein beliebiges Polynom über $\mathbb{Z}$

$$f(X) \quad = \quad X^4 - 4X^3 + X - 2.$$

Dann erhalten wir in $\mathbb{Z}[X]$ eine Zerlegung in irreduzible Faktoren

$$f(X) \quad = \quad (X-1)\cdot(X-2)\cdot(X^2-X-1).$$

Der Zerfällungstyp von $f(X)$ ist somit $(1\,1\,2)$.

Mit dieser Vereinbarung und Satz 2.9 erhalten wir dann den folgenden Zusammenhang zwischen den Nullstellen α und β von $\chi(X)$ und dem Zerfällungstyp von $\Psi_l(X)$:

α,β	Zerf.typ von ψ_l
$\alpha,\beta\in\mathbb{F}_l$	
$\alpha=\beta$	$\begin{cases}(1\ l)\\(1\ \ldots\ 1)\end{cases}$
$\alpha\neq\beta$	$(1\,1\,d\ \ldots\ d)$
$\alpha,\beta\in\mathbb{F}_{l^2}\backslash\mathbb{F}_l$	
$\alpha\neq\beta$	$(d\ \ldots\ d)$

Dabei ist d jeweils die Ordnung von $\alpha\beta^{-1}$ in dem jeweiligen Körper. Eine genaue Beschreibung, wie der Zerfällungstyp eines Polynoms bestimmt wird, kann man in [Ma94] nachlesen. Zusammenfassend erhalten wir dann folgenden Algorithmus:

Berechnung der Spur des Frobenius-Endomorphismus Φ_l

Eingabe : elliptische Kurve E=(a,b) über $\mathbb{F}_p$
Primzahl l $(l>2)$ mit $l\neq p$
Ausgabe : Vektor möglicher Lösungen $c \bmod l$ für c aus (2.2)

<u>Berechnung der j-Invarianten</u>

(1) $j_E = 1728\cdot\frac{4a^3}{4a^3+27b^2}$

<u>Berechnung des modularen Polynoms</u>

(2) $\Psi_l(X)=\Psi_l(X,j_E)$

Berechnung der Spur des Frobenius-Endomorphismus Φ_l
(Fortsetzung)

<u>BERECHNUNG MÖGLICHER LÖSUNGEN</u>

typ = ZerfällTyp $(\Psi_l(X))$
if $(typ \in \{(1\ldots 1), (1\, l)\})$ **then**
  $c[0] = 2 \cdot \sqrt{q}$ // Berechn. in $\mathbb{F}_l$
  $c[1] = -2 \cdot \sqrt{q}$ // Berechn. in $\mathbb{F}_l$
else
  if $(typ == (1\,1\,d \ldots d))$ **then**
    Ber. alle $x_i \in \mathbb{F}_l$ mit Ordn. d
    for (jedes x_i) **do**
      $c[2i] = (x_i + 1)\sqrt{px_i^{-1}}$ // Arithm. in $\mathbb{F}_l$
      $c[2i+1] = -(x_i + 1)\sqrt{px_i^{-1}}$
    od
  else
    $//typ == (d \ldots d)$
    Ber. alle $x_i \in \mathbb{F}_{l^2}$ mit Ordn. d
    for (jedes x_i) **do**
      $c[2i] = (x_i + 1)\sqrt{px_i^{-1}}$ // Arithm. in $\mathbb{F}_{l^2}$
      $c[2i+1] = -(x_i + 1)\sqrt{px_i^{-1}}$
    od
  fi
fi
return (c)

Kapitel 3

Eine Arithmetik für Fourier-Reihen

In den folgenden Kapiteln werden wir Funktionen verwenden, die mit Hilfe sogenannter Fourier-Reihen dargestellt werden können. Daher werden wir uns in diesem Abschnitt zunächst mit Verknüpfungen formaler Fourier-Reihen (Addition, Multiplikation) beschäftigen. Danach betrachten wir Approximationen von Fourier-Reihen — d.h. endliche Darstellungen der unendlichen Summen — und deren Verhalten unter den zuvor vorgestellten Operationen. Abschließend werden wir dann beschreiben, wie solche Näherungen in der Praxis repräsentiert werden können.

3.1 Operationen auf Fourier-Reihen

Sei im folgenden R ein beliebiger Integritätsbereich — d.h. ein *nullteilerfreier, kommutativer Ring mit Einselement*. Wir betrachten nun die Gesamtheit der formalen Fourier-Reihen

$$R\,[[X]] \quad = \quad \left\{ \sum_{\nu=n}^{\infty} a_\nu \cdot X^\nu \mid a_\nu \in R, a_n \neq 0 \text{ und } n \in \mathbb{Z}, |n| < \infty \right\} \tag{3.1}$$

über R. Dabei erlauben wir also, daß in den *unendlichen* Summen höchstens *endlich viele* Potenzen von X mit *negativen* Exponenten auftreten. Für eine Fourier-Reihe $f(X) = \sum_{\nu=n}^{\infty} a_\nu \cdot X^\nu$ nennen wir n den **Startexponenten**, und bezeichnen mit

$$\operatorname{lc}\,(f(X)) \quad = \quad a_n$$

den **Leitkoeffizienten** dieser Reihe. Weiter vereinbaren wir $a_\nu = 0$ für alle $\nu < n$.

Auf der Menge $R[[X]]$ definieren wir nun eine Addition und eine Multiplikation. Seien dazu

$$f(X) \quad = \quad \sum_{\nu=n_f}^{\infty} a_\nu \cdot X^\nu \qquad \text{und} \qquad g(X) \quad = \quad \sum_{\nu=n_g}^{\infty} b_\nu \cdot X^\nu$$

gegeben. Dann sei die Summe (bzw. Differenz) dieser Reihen

$$(f+g)(X) \quad = \quad \sum_{\nu=n}^{\infty} (a_\nu + b_\nu) \cdot X^\nu \tag{3.2}$$

$$\text{und} \qquad (f-g)(X) \quad = \quad \sum_{\nu=n}^{\infty} (a_\nu - b_\nu) \cdot X^\nu . \tag{3.3}$$

Hierbei ist $(a_\nu \pm b_\nu) = 0$ für jedes $\nu < \min\,(n_f, n_g)$, und für den Startexponenten n der Summe (bzw. Differenz) gilt damit

$$n \quad \geq \quad \min\,(n_f, n_g). \tag{3.4}$$

In Summe und Differenz zweier Fourier-Reihen treten also ebenfalls höchstens endliche viele Potenzen von X mit negativen Exponenten auf. Damit sind $(f \pm g)(X)$ Elemente von $R[[X]]$.

3.1. Beispiel Betrachten wir die Summe zweier Fourier-Reihen über $R = \mathbb{F}_{101}$:

$$\begin{array}{rcccccccccccc}
f(X) = & & & 77 \cdot X^{-1} & + & 37 \cdot X^0 & + & 36 \cdot X^1 & + & 40 \cdot X^2 & + & 72 \cdot X^3 \\
g(X) = & 6 \cdot X^{-2} & + & 0 \cdot X^{-1} & + & 100 \cdot X^0 & + & 24 \cdot X^1 & + & 83 \cdot X^2 & + & \cdots \\
\\
(f+g)(X) = & 6 \cdot X^{-2} & + & 77 \cdot X^{-1} & + & 36 \cdot X^0 & + & 60 \cdot X^1 & + & 22 \cdot X^2 & + & \cdots
\end{array}$$

Das Produkt zweier Reihen ist definiert als

$$(f \cdot g)(X) \quad = \quad \sum_{\nu=n}^{\infty} c_\nu \cdot X^\nu \tag{3.5}$$

mit den Koeffizienten

$$c_\nu \quad = \quad \sum_{\substack{(i,j) \\ i+j=\nu}} a_i \cdot b_j \quad = \quad \sum_{i=n_f}^{\nu-n_g} a_i \cdot b_{\nu-i}. \tag{3.6}$$

Dabei ist $c_\nu = 0$ für $(\nu - n_g) < n_f$, und wir erhalten

$$c_{n_f+n_g} \quad = \quad a_{n_f} \cdot b_{n_g} \quad \neq 0$$

$$\text{und} \qquad c_\nu \quad = \quad 0 \quad \text{für} \quad \nu \; < \; (n_f + n_g).$$

Der Startexponent des Produktes ist damit

$$n \quad = \quad n_f + n_g. \tag{3.7}$$

Somit treten auch bei der Produktbildung höchstens endlich viele negative Potenzen von X auf, und $(f \cdot g)(X)$ liegt wiederum in $R[[X]]$.

3.2. Beispiel Das Produkt der beiden Reihen des vorigen Beispiels ist:

$$\begin{array}{rccccccccccc}
f(X) = & & & 77 \cdot X^{-1} & + & 37 \cdot X^0 & + & 36 \cdot X^1 & + & 40 \cdot X^2 & + & 72 \cdot X^3 \\
g(X) = & 6 \cdot X^{-2} & + & 0 \cdot X^{-1} & + & 100 \cdot X^0 & + & 24 \cdot X^1 & + & 83 \cdot X^2 & + & \cdots \\
& & & & & & & & & & & \\
(f \cdot g)(X) = 58 \cdot X^{-3} & + & 20 \cdot X^{-2} & + & 38 \cdot X^{-1} & + & 31 \cdot X^0 & + & 100 \cdot X^1 & + & \cdots &
\end{array}$$

Betrachten wir nun noch die Situation, daß speziell $f(X) = g(X)$ ist — d.h., daß insbesondere die Koeffizienten der Reihen übereinstimmen. Dann erhalten wir aus der Formel (3.6)

$$c_\nu \quad = \quad \sum_{i=n_f}^{\nu-n_f} a_i \cdot a_{\nu-i}.$$

Bestimmen wir diese Summe für ein konkretes Beispiel:

3.3. Beispiel Sei der Startexponent n_f von $f(X)$ gleich -1. Dann erhalten wir

$$\begin{aligned}
c_1 &= a_{-1} \cdot a_2 + a_0 \cdot a_1 + a_1 \cdot a_0 + a_2 \cdot a_{-1} \\
&= 2 \cdot (a_{-1} \cdot a_2 + a_0 \cdot a_1)
\end{aligned}$$

und

$$\begin{aligned}
c_2 &= a_{-1} \cdot a_3 + a_0 \cdot a_2 + a_1 \cdot a_1 + a_2 \cdot a_0 + a_3 \cdot a_{-1} \\
&= 2 \cdot (a_{-1} \cdot a_3 + a_0 \cdot a_2) + a_1^2.
\end{aligned}$$

Dabei können wir beobachten, daß die Produkte $a_i \cdot a_{\nu-i}$ in diesen Summen genau zweimal auftreten, falls i und $(\nu - i)$ voneinander verschieden sind.

Allgemein erhalten wir

$$c_\nu \quad = \quad 2 \cdot \sum_{i=n_f}^{(\nu-1)/2} a_i \cdot a_{\nu-i}, \qquad \text{falls} \quad \nu \ \text{ungerade} \tag{3.8}$$

und

$$c_\nu \quad = \quad 2 \cdot \sum_{i=n_f}^{\nu/2-1} a_i \cdot a_{\nu-i} + a_{\nu/2}^2, \qquad \text{falls} \quad \nu \ \text{gerade ist.} \tag{3.9}$$

3.4. Bemerkung Die speziellen Formeln zur Berechnung des Quadrates einer Fourier-Reihe können wir später bei der Implementierung der Arithmetik verwenden. Gegenüber einer gewöhnlichen Multiplikation werden zum Quadrieren einer Reihe nur noch halb so viele Koeffizienten-Multiplikationen benötigt.

Die Menge $R[[X]]$ ist somit unter den oben definierten Verknüpfungen abgeschlossen. Weiter kann der Ring R in $R[[X]]$ eingebettet werden durch:

$$\begin{array}{rcl} R & \longrightarrow & R[[X]] \\ a & \longmapsto & a \cdot X^0 \end{array} \tag{3.10}$$

Man kann leicht nachrechnen, daß $(R[[X]], +, \cdot)$ mit den oben definierten Operationen ebenfalls ein Integritätsbereich ist. Dabei sind die neutralen Elemente der Addition und Multiplikation gerade die Bilder der jeweiligen Elemente aus R unter der Abbildung (3.10) — also $0 \cdot X^0$ und $1 \cdot X^0$. Wir betrachten nun noch, wann eine Fourier-Reihe $f(X)$ in $R[[X]]$ eine *Einheit* ist, d.h. wann diese *multiplikativ* invertierbar ist. Es gilt der folgende Satz:

3.5. Satz *Sei $f(X) \in R[[X]]$ gegeben durch*

$$f(X) = \sum_{\nu=n_f}^{\infty} a_\nu \cdot X^\nu$$

Dann ist $f(X)$ eine Einheit in $R[[X]]$, falls der Leitkoeffizient a_{n_f} bereits in R eine Einheit ist.

Beweis: Zum Beweis sei zunächst $a_{n_f}^{-1}$ in R mit

$$a_{n_f} \cdot a_{n_f}^{-1} = 1$$

das multiplikative Inverse zu a_{n_f}. Dann können wir zu $f(X)$ die Reihe

$$f^{-1}(X) = \sum_{\nu=-n_f}^{\infty} d_\nu \cdot X^\nu$$

mit

$$d_\nu = \begin{cases} 0, & \text{für } \nu < -n_f \\ a_{n_f}^{-1}, & \text{für } \nu = -n_f \\ -a_{n_f}^{-1} \cdot \left(\sum_{\mu=-n_f}^{\nu-1} d_\mu \cdot a_{(n_f+\nu)-\mu} \right), & \text{für } \nu > -n_f \end{cases} \tag{3.11}$$

definieren. Mit Hilfe der Definition für die Multiplikation rechnen wir nach:

$$f(X)^{-1} \cdot f(X) = \left(\sum_{\nu=-n_f}^{\infty} d_\nu \cdot X^\nu \right) \cdot \left(\sum_{\nu=n_f}^{\infty} a_\nu \cdot X^\nu \right) = \sum_{\nu=0}^{\infty} c_\nu \cdot X^\nu.$$

Dabei gilt für die Koeffizienten

$$c_0 \;=\; c_{(-n_f+n_f)} \;=\; d_{-n_f} \cdot a_{n_f} \;=\; a_{n_f}^{-1} \cdot a_{n_f} \;=\; 1,$$

und für alle $\nu > 0$ erhalten wir nach Anwendung der Multiplikationsformeln (3.6)

$$c_\nu \;=\; \sum_{i=-n_f}^{\nu-n_f} d_i \cdot a_{\nu-i} \;=\; \underbrace{\left(\sum_{i=-n_f}^{(\nu-n_f)-1} d_i \cdot a_{\nu-i} \right)}_{-(d_{\nu-n_f} \cdot a_{n_f})} \;+\; d_{\nu-n_f} \cdot a_{n_f} \;=\; 0.$$

Also gilt

$$f(X)^{-1} \cdot f(X) \quad = \quad 1.$$

■

3.6. Bemerkung Betrachten wir speziell den Ring $\mathbb{Z}\,[[X]]$ der Fourier-Reihen mit ganzzahligen Koeffizienten, so ist darin eine Reihe invertierbar, falls der zugehörige Leitkoeffizient gleich 1 oder -1 ist. Sind andererseits $\mathbb{K}$ ein beliebiger Körper, und damit alle von 0 verschiedenen Elemente Einheiten in $\mathbb{K}$, dann ist in $\mathbb{K}\,[[X]]$ *jede* Fourier-Reihe $f(X) \neq 0$ invertierbar, und $\mathbb{K}\,[[X]]$ sogar selbst ein Körper.

3.7. Beispiel Berechnen wir das Inverse der Reihe $f(X)$ der vorangehenden Beispiele über dem Körper $\mathbb{F}_{101}$, so erhalten wir:

$$f(X) \;=\; 77 \cdot X^{-1} \;+\; 37 \cdot X^0 \;+\; 36 \cdot X^1 \;+\; 40 \cdot X^2 \;+\; 72 \cdot X^3 \;+\; \cdots$$

$$f^{-1}(X) \;=\; 21 \cdot X^1 \;+\; 45 \cdot X^2 \;+\; 63 \cdot X^3 \;+\; 86 \cdot X^4 \;+\; 20 \cdot X^5 \;+\; \cdots$$

3.2 Approximationen von Fourier-Reihen

Wenn wir Funktionen mit Hilfe von Fourier-Reihen — also *unendlichen* Summen — darstellen, so können wir in der Praxis nur *endlich* viele der zugehörigen Koeffizienten speichern. Damit verwenden wir lediglich *Approximationen* anstatt einer korrekten Darstellung. In diesem Abschnitt werden wir nun untersuchen, wie sich die „Güte“ solcher Approximationen durch Addition, Multiplikation und Inversion der Fourier-Reihen verändert.

3.8. Definition *Sei eine beliebige Fourier-Reihe $f(X) = \sum_{\nu=n_f}^{\infty} a_\nu X^\nu$ gegeben. Unter einer* **Approximation** *von $f(X)$ verstehen wir eine* endliche *Teilsumme von $f(X)$ und schreiben:*

$$f(X) \quad \approx \quad \sum_{\nu=n_f}^{N_f} a_\nu \cdot X^\nu. \tag{3.12}$$

(Alle Koeffizienten a_ν mit $\nu > N_f$ sind unbekannt.) Zu einer solchen Näherung definieren wir weiter die Funktionen

$$\begin{aligned} & \textit{first}\ (f(X)) && := && n_f && (3.13) \\ \textit{und} \quad & \textit{last}\ (f(X)) && := && N_f. && (3.14) \end{aligned}$$

Ausgehend von Approximationen von Fourier-Reihen erhalten wir auch nur näherungsweise Lösungen für die Summe und das Produkt der jeweiligen Reihen. Daher untersuchen wir nun, wie sich die Funktionen *first* und *last* auf Verknüpfungen von Reihen übertragen. Dazu seien im folgenden die Approximationen

$$f(X) \approx \sum_{\nu=n_f}^{N_f} a_\nu \cdot X^\nu \qquad \text{und} \qquad g(X) \approx \sum_{\nu=n_g}^{N_g} b_\nu \cdot X^\nu \tag{3.15}$$

der Reihen $f(X)$ und $g(X)$ aus $R[[X]]$ gegeben.

3.9. Satz *Seien die Approximationen (3.15) vorgegeben. Dann ergibt sich für die Summe (bzw. Differenz) der Reihen die Approximation*

$$(f \pm g)(X) \approx \sum_{\nu=n}^{N} (a_\nu \pm b_\nu) \cdot X^\nu.$$

Dabei gilt

$$\begin{aligned} & n && \geq && \textit{min}\ (n_f, n_g) && (3.16) \\ \textit{und} \quad & N && = && \textit{min}\ (N_f, N_g). && (3.17) \end{aligned}$$

Beweis: Die Aussage über den Startexponenten der Summe (bzw. Differenz) folgt leicht, da für alle Indizes ν mit $\nu < \min(n_f, n_g)$ die Koeffizienten *beider* Reihen gleich 0 sind. Somit sind auch die entsprechenden Summen (bzw. Differenzen) gleich 0. Wir beachten weiter, daß der Startexponent der Summe (bzw. Differenz) mit dem Minimum der Startexponenten zusammenfällt, falls diese voneinander verschieden sind. Sind beide Startexponenten gleich, so kann n evtl. durch „Auslöschung“ der führenden Koeffizienten größer werden.

Um die Aussagen über den höchsten Exponenten N der Approximation zu verifizieren, beachten wir, daß die Summen $a_\nu + b_\nu$ *genau dann* berechnet werden können, falls sowohl a_ν als auch b_ν bekannt sind. Damit kann der entsprechende Index ν (der Exponent der zugehörigen Potenz von X) nur kleiner oder gleich N_f und gleichzeitig kleiner oder gleich N_g sein. Für den minimalen dieser beiden Indizes kann der zugehörige Koeffizient berechnet werden. Somit ist N gleich dem Minimum von N_f und N_g ■

Für das Produkt zweier Fourier-Reihen liefert der folgende Satz analoge Aussagen.

3.10. Satz *Seien die Approximationen (3.15) gegeben. Dann gilt für das Produkt der beiden Reihen*

$$(f \cdot g)(X) \quad \approx \quad \sum_{\nu=n}^{N} c_\nu \cdot X^\nu$$

mit den in (3.6) definierten Koeffizienten c_ν. Dabei gilt

$$n \quad = \quad n_f \; + \; n_g \tag{3.18}$$

$$\textit{und} \qquad N \quad = \quad \min\,(n_f + N_g, n_g + N_f). \tag{3.19}$$

Beweis: Bereits bei der Definition des Produktes zweier Reihen haben wir gezeigt, daß sich dessen Startexponent gerade als Summe der Startexponenten der Ausgangsreihen ergibt. Zum Beweis von (3.19) betrachten wir noch einmal die Formeln (3.6) zur Berechnung der Koeffizienten c_ν:

$$c_\nu \quad = \quad \sum_{i=n_f}^{\nu-n_g} a_i \cdot b_{\nu-i}.$$

Darin treten die Koeffizienten

$$a_{n_f},\; a_{n_f+1},\; \ldots,\; a_{\nu-n_g}$$

der Approximation von $f(X)$ und die Koeffizienten

$$b_{\nu-n_f},\; b_{\nu-n_f-1},\; \ldots,\; b_{n_g}$$

der Approximation von $g(X)$ auf. Der Koeffizient c_ν des Produktes ist also genau dann korrekt berechenbar, falls alle auftretenden Koeffizienten von $f(X)$ und $g(X)$ bekannt sind. Dies bedeutet, daß sowohl

$$\nu \; - \; n_g \quad \leq \quad N_f \qquad \Leftrightarrow \qquad \nu \quad \leq \quad N_f \; + \; n_g$$

als auch

$$\nu \; - \; n_f \quad \leq \quad N_g \qquad \Leftrightarrow \qquad \nu \quad \leq \quad N_g \; + \; n_f$$

erfüllt sein muß. Damit ist c_ν genau dann berechenbar, wenn

$$\nu \quad \leq \quad \min\,(N_g + n_f, N_f + n_g).$$

■

Berechnen wir das Quadrat einer Approximation, so erhalten wir als Spezialfall dieses Satzes das folgende Korollar:

3.11. Korollar *Sei die Approximation von $f(X)$ wie in (3.15) gegeben. Dann gilt für die Approximation von $f^2(X)$:*

$$f^2(X) \quad \approx \quad \sum_{\nu=n}^{N} c_\nu \cdot X^\nu$$

mit den in (3.6) definierten Koeffizienten c_ν. Dabei ist

$$n \quad = \quad 2 \cdot n_f \tag{3.20}$$

$$\textit{und} \qquad N \quad = \quad N_f \; + \; n_f. \tag{3.21}$$

Ist $f(X)$ eine Einheit in $R[[X]]$, so können wir auch eine Approximationen des zugehörigen Inversen berechnen.

3.12. Satz *Sei die Approximation (3.15) von $f(X)$ gegeben, und sei weiter $f(X)$ eine Einheit in $R[[X]]$. Dann gilt für die Approximation von $f^{-1}(X)$:*

$$f^{-1}(X) \quad \approx \quad \sum_{\nu=n}^{N} d_\nu \cdot X^\nu$$

mit den in (3.11) definierten Koeffizienten d_ν, wobei

$$n \quad = \quad -n_f \tag{3.22}$$

$$\textit{und} \qquad N \quad = \quad N_f \; - \; 2 \cdot n_f. \tag{3.23}$$

Beweis: Aus der Aussage über den Startexponenten des Produktes zweier Fourier-Reihen können wir ersehen, daß der Startexponent 0 der Reihe $1 \cdot X^0 = f(X) \cdot f^{-1}(X)$ gleich der Summe $n_f + n$ sein muß — damit ist also $n = -n_f$. Zur Bestimmung des maximalen Exponenten N, ziehen wir die Formel zur Berechnung der Koeffizienten d_ν für den Fall $\nu > -n_f$ heran:

$$d_\nu \quad = \quad -a_{n_f}^{-1} \cdot \left(\sum_{\mu=-n_f}^{\nu-1} d_\mu \cdot a_{(n_f+\nu)-\mu} \right).$$

Dazu werden die Koeffizienten

$$a_{(n_f+\nu)+n_f}, \; a_{2\cdot n_f+\nu-1}, \; \ldots, \; a_{n_f-1}$$

von $f(X)$ benötigt, und wir können ablesen, daß d_ν genau dann berechnet werden kann, falls

$$2 \cdot n_f \; + \; \nu \quad \leq \quad N_f \qquad \Leftrightarrow \qquad \nu \quad \leq \quad N_f \; - \; 2 \cdot n_f. \qquad \blacksquare$$

Fassen wir die Aussagen der vorangehenden Sätze zusammen, so beschreibt das folgende Korollar das Verhalten der Startexponenten von Fourier-Reihen bei den jeweiligen Operationen:

3.13. Korollar *Seien $f(X)$ und $g(X)$ durch (3.15) approximiert. Dann gilt*

$$\begin{aligned} \mathit{first}\,((f \pm g)(X)) &= \min\,(\mathit{first}\,(f(X)), \mathit{first}\,(g(X))) \\ \mathit{first}\,((f \cdot g)(X)) &= \mathit{first}\,(f(X)) + \mathit{first}\,(g(X)) \\ \mathit{first}\,(f^{-1}(X)) &= -\,\mathit{first}\,(f(X)). \end{aligned}$$

Ebenso erhalten wir die folgenden Aussagen über die Präzisionen der Approximationen:

3.14. Korollar *Seien $f(X)$ und $g(X)$ durch (3.15) approximiert. Dann gilt*

$$\begin{aligned}
\mathit{last}\,((f \pm g)(X)) &= \mathit{min}\,(\mathit{last}\,(f(X)), \mathit{last}\,(g(X))) \\
\mathit{last}\,((f \cdot g)(X)) &= \mathit{min}\,(\mathit{first}\,(f(X)) + \mathit{last}\,(g(X)), \mathit{first}\,(g(X)) + \mathit{last}\,(f(X))) \\
\mathit{last}\,(f^{-1}(X)) &= \mathit{last}\,(f(X)) \;-\; 2 \cdot \mathit{first}\,(f(X)).
\end{aligned}$$

3.3 FFT für Fourier-Reihen

In diesem Abschnitt werden wir zeigen, daß die Multiplikation der approximierten Fourier-Reihen auf die Multiplikation von *Polynomen* zurückgeführt werden kann. Damit steht uns dann ein spezielles Verfahren zur Verfügung, diese Multiplikationen auszuführen — die *schnelle Fourier Transformation* (FFT).

3.3.1 Verwendung von Polynom-Multiplikation

Wir werden zunächst zeigen, daß für die Multiplikation approximierter Fourier-Reihen die übliche Polynom-Multiplikation verwendet werden kann. Dazu geben wir uns wieder zwei Approximationen vor:

$$f(X) \;\approx\; \sum_{\nu=n_f}^{N_f} a_\nu \cdot X^\nu \qquad \text{und} \qquad g(X) \;\approx\; \sum_{\nu=n_g}^{N_g} b_\nu \cdot X^\nu.$$

Diese können wir auch in der Form

$$f(X) \;\approx\; X^{n_f} \cdot \sum_{\nu=0}^{N_f-n_f} a_{n_f+\nu} \cdot X^\nu \qquad \text{und} \qquad g(X) \;\approx\; X^{n_g} \cdot \sum_{\nu=0}^{N_g-n_g} b_{n_g+\nu} \cdot X^\nu$$

schreiben und erhalten damit für das Produkt dieser Reihen

$$\begin{aligned}
(f \cdot g)(X) &\approx \left(X^{n_f} \cdot \sum_{\nu=0}^{N_f-n_f} a_{n_f+\nu} \cdot X^\nu \right) \cdot \left(X^{n_g} \cdot \sum_{\nu=0}^{N_g-n_g} b_{n_g+\nu} \cdot X^\nu \right) \\
&= X^{n_f+n_g} \cdot \left(\sum_{\nu=0}^{N_f-n_f} a_{n_f+\nu} \cdot X^\nu \right) \cdot \left(\sum_{\nu=0}^{N_g-n_g} b_{n_g+\nu} \cdot X^\nu \right).
\end{aligned}$$

Damit können wir uns bei der Untersuchung der Multiplikation darauf beschränken, Fourier-Reihen mit Startexponenten 0 zu betrachten — sind die Startexponenten von 0 verschieden, so müssen nach der Multiplikation lediglich die berechneten Koeffizienten um $(n_f + n_g)$ „verschoben“ werden. Seien also im folgenden $n_f = n_g = 0$ und damit

$$f(X) \quad \approx \quad \sum_{\nu=0}^{N_f} a_\nu \cdot X^\nu \qquad \text{und} \qquad g(X) \quad \approx \quad \sum_{\nu=0}^{N_g} b_\nu \cdot X^\nu.$$

Diese *endlichen* Summen können wir insbesondere als *Polynome* in X auffassen. Bilden wir dann mit der üblichen Polynom-Multiplikation das Produkt dieser Approximationen, so erhalten wir das Polynom:

$$\left(\sum_{\nu=0}^{N_f} a_\nu \cdot X^\nu\right) \cdot \left(\sum_{\nu=0}^{N_g} b_\nu \cdot X^\nu\right) \quad = \quad \sum_{\nu=0}^{N_f+N_g} c_\nu \cdot X^\nu$$

mit den Koeffizienten

$$c_\nu \quad = \quad \sum_{\substack{(i,j)\\ i+j=\nu}} a_i \cdot b_j \quad = \quad \sum_{i=\max(0,\nu-N_g)}^{\min(\nu,N_f)} a_i \cdot b_{\nu-i} \quad \text{für } 0 \le \nu \le N_f + N_g.$$

Vergleichen wir dies nun mit der Multiplikation der Fourier-Reihen und bestimmen, wann die Koeffizienten der Polynom-Multiplikation mit den Koeffizienten der Fourier-Reihen-Multiplikation übereinstimmen. Wir müssen dazu untersuchen, wann die folgende Gleichheit gilt (vgl. Formel (3.6) mit $n_f = n_g = 0$):

$$\sum_{i=0}^{\nu} a_i \cdot b_{\nu-i}. \quad \stackrel{?}{=} \quad \sum_{i=\max(0,\nu-N_g)}^{\min(\nu,N_f)} a_i \cdot b_{\nu-i}.$$

Dies gilt offenbar genau dann, wenn

$$\begin{array}{rcccccc} 0 & = & \max\,(0, \nu - N_g) & \Leftrightarrow & \nu & \le & N_g \\ \text{und} \quad \nu & = & \min\,(\nu, N_f) & \Leftrightarrow & \nu & \le & N_f \end{array}$$

erfüllt sind. Damit stimmen die beiden Formeln genau dann überein, wenn ν nicht größer ist als $\min\,(N_f, N_g)$. Da die Startexponenten beider Reihen gleich 0 vorausgesetzt waren, liefert uns Korollar 3.14, daß $\min\,(N_f, N_g)$ aber ohnehin der höchste Exponent ist, der in der Approximation von $(f \cdot g)(X)$ vorkommt.

Damit können wir das folgende Verfahren verwenden, um zwei Approximationen von Fourier-Reihen zu multiplizieren:

1. Betrachte die approximierten Reihen $f(X)$ und $g(X)$ als Polynome vom Grad N_f und N_g.
2. Berechne das Produkt dieser Polynome mit Hilfe einer Polynom-Multiplikation (z.B. mit FFT).
3. Kopiere die ersten $\min\,(N_f, N_g)$ Koeffizienten des Polynom-Produktes.

3.3.2 Die schnelle Fourier-Transformation für Polynome

Die Idee der schnellen Fourier-Transformation (FFT) für Polynome beruht darauf, daß ein Polynom $P(X)$ vom Grad n eindeutig bestimmt ist, falls man für $n+1$ paarweise verschiedene *Stützstellen* $x_0, x_2, \ldots x_n$ die zugehörigen Funktionswerte kennt, die $P(X)$ an diesen annimmt. Gilt beispielsweise

$$P(x_\nu) \quad = \quad y_\nu \qquad \text{für} \quad 1 \le \nu \le n,$$

so kann man die Koeffizienten von $P(X)$ mit Hilfe der *Lagrange'schen Interpolationsformeln* berechnen.

3.15. Satz *Seien* $(x_0, y_0), \ldots, (x_n, y_n)$ *gegeben. Dann wird das eindeutig bestimmte Polynom* $P(X)$ *mit*

$$P(x_\nu) \quad = \quad y_\nu$$

gegeben durch

$$P(X) \quad = \quad \sum_{\nu=0}^{n} y_\nu \cdot \prod_{\substack{k\neq\nu \\ k=0}}^{n} \frac{X - x_k}{x_\nu - x_k}$$

Beweis: Siehe dazu [Mey1, Satz 4.1.16]. ∎

3.16. Beispiel Seien vorgegeben

$$(x_0, y_0) \;=\; (2, 11), \quad (x_1, y_1) \;=\; (1, 6) \quad \text{und} \quad (x_2, y_2) \;=\; (0, 3).$$

Dann erhalten wir

$$\begin{aligned} P(X) &= 11 \cdot \tfrac{X-1}{2-1} \cdot \tfrac{X-0}{2-0} + 6 \cdot \tfrac{X-2}{1-2} \cdot \tfrac{X-0}{1-0} + 3 \cdot \tfrac{X-2}{0-2} \cdot \tfrac{X-1}{0-1} \\ &= \tfrac{11}{2}X^2 - \tfrac{11}{2}X + (-6)X^2 + 12X + \tfrac{3}{2}X^2 - \tfrac{9}{2}X + 3 \\ &= X^2 + 2X + 3. \end{aligned}$$

Somit kann man ein Polynom vom Grad n also auf zwei Arten darstellen:

1. Durch Abspeichern der Koeffizienten $a_0, a_1, \ldots, a_n$ oder
2. durch Speichern der Paare $(x_0, y_0), (x_2, y_2) \ldots, (x_n, y_n)$.

Wir nennen die Darstellung eines Polynoms mit Hilfe der Paare (x_i, y_i) **Point-Value** Repräsentation.

Der Vorteil der Point-Value Darstellung von Polynomen ist der, daß alle Operationen auf Polynomen linear in deren Graden ausgeführt werden können. Geben wir uns nämlich die Point-Value Darstellungen zweier Polynome $A(X)$ und $B(X)$ vor, dann gilt $(A+B)(x_i) = A(x_i) + B(x_i)$ für $0 \leq i \leq n$, und wir erhalten durch *punktweise* Addition die entsprechende Darstellung der Summe $A+B$. Analog gilt dies auch für die Multiplikation. Dabei müssen wir jedoch beachten, daß der Grad des Produktes zweier Polynome gleich der Summe der Grade der Ausgangspolynome ist:

$$\deg(A) = n \text{ und } \deg(B) = m \quad \Longrightarrow \quad deg(A \cdot B) = n + m.$$

Das Produkt dieser Polynome ist damit eindeutig bestimmt durch mindestens $n + m + 1$ Point-Value Paare (x_i, y_i). Damit müssen dann auch $A(X)$ und $B(X)$ an mindestens $n + m + 1$ verschiedenen Stützstellen ausgewertet werden, wenn die Polynom-Multiplikation in dieser Darstellung berechnet werden soll.

Damit bietet sich folgende Strategie an, um die Polynom-Multiplikation zu realisieren:

1. Konvertiere A und B von Koeffizientendarstellung zur Point-Value Darstellung durch Auswerten der Polynome an $n + m + 1$ paarweise verschiedenen Stützstellen.
2. Multipliziere die Point-Value Darstellungen der beiden Polynome punktweise.
3. Bestimme die Koeffizientendarstellung der Produktes $A{\cdot}B$ durch Interpolation.

Die schnelle Fourier-Transformation ist ein Verfahren, das es ermöglicht mit Hilfe von $O(n \log n)$ Operationen sowohl die Auswertung eines Polynoms vom Grad n zu berechnen als auch die Interpolation in dieser Laufzeit durchzuführen. Dabei werden die Stützstellen, an denen die Polynome ausgewertet werden müssen als n-te Einheitswurzeln gewählt. Dann kann man nachweisen, daß die Interpolation eines Polynoms mit Hilfe desselben Verfahrens berechnet werden kann wie die Auswertung (vgl. [CLR90, Seite 776ff]).

Hat man Funktionen zur Verfügung, die die Auswertung bzw. Interpolation nach dieser Methode durchführen, kann man leicht einen Algorithmus zur Berechnung einer Polynom-Multiplikation angeben. Seien diese Funktionen gegeben:

1. FFT$(a_0, a_1, \ldots, a_n)$ sei eine Funktion, die ein Polynom an $n+1$ verschiedenen Stützstellen auswertet und die jeweiligen Funktionswerte zurückliefert.
2. Die Funktion RevFFT$(A_0, A_1, \ldots, A_n)$ berechne die Koeffizienten eines Polynoms, das an vorgegebenen Stützstellen die Funktionswerte A_ν annimmt.

Polynom-Multiplikation mit FFT

EINGABE : Koeffizientenvektoren zweier Polynome vom Grad n:
$(a_0, a_1, \ldots, a_n)$ und $(b_0, b_1, \ldots, b_n)$
AUSGABE : Koeffizientenvektor des Produktes $(c_0, c_1, \ldots, c_{2n})$

BERECHNE POINT-VALUE-DARSTELLUNGEN

(1) $A = \mathrm{FFT}(a_0, a_1, \ldots, a_n)$ // $\#A = 2 \cdot n$
(2) $B = \mathrm{FFT}(b_0, b_1, \ldots, b_n)$ // $\#B = 2 \cdot n$

BERECHNE PUNKTWEISE PRODUKTE

(3) **for** $(i = 0, \ldots, 2n)$ **do**
(4) $C_i = A_i * B_i$
(5) **od**

BERECHNE KOEFFIZIENTEN-DARSTELLUNG DES PRODUKTES

(6) $c = \mathrm{RevFFT}(C_0, C_1, \ldots, C_{2n})$
(7) **return** (c)

3.17. Bemerkung Möchte man mit diesem Verfahren das Quadrat eines gegebenen Polynoms berechnen, so muß die Funktion FFT(...) dazu nur einmal verwendet werden. Dies führt dann zu einer Beschleunigung des Algorithmus im Vergleich zu einer gewöhnlichen Multiplikation.

Eine Implementierung der FFT für Polynome über beliebigen endlichen Primkörpern wurde von Victor Shoup entwickelt. Details zu den verwendeten Verfahren kann man in [Sho94] nachlesen.

3.4 Implementierung einer Arithmetik

Wir haben uns in den vorangehenden Abschnitten mit der Arithmetik auf Fourier-Reihen beschäftigt. Hier werden wir nun zwei verschiedene Methoden betrachten, Approximationen solcher Reihen in der Praxis zu repräsentieren. Für jede dieser Darstellungen geben wir dann an, wie die oben beschriebenen Verknüpfungen realisiert werden können.

Um eine Fourier-Reihe in der Praxis darzustellen, müssen wir den Startexponenten sowie die Koeffizienten der Reihe speichern. Wir unterscheiden Darstellungen danach, ob die zugehörigen Koeffizienten im allgemeinen von 0 verschieden sind (*dichte* Darstellung) oder ob ein „Großteil" bereits gleich 0 ist (*dünne* Darstellung). Für jede dieser Möglichkeiten werden wir im folgenden eine Realisierung beschreiben.

3.4.1 Dichte Darstellung von Fourier-Reihen

Wollen wir eine Approximation

$$f(X) \quad \approx \quad \sum_{\nu=n_f}^{N_f} a_\nu \cdot X^\nu$$

in *dichter* Darstellung repräsentieren, dann speichern wir *alle* Koeffizienten dieser Reihe. Wir verwenden dabei folgende Datenstruktur:

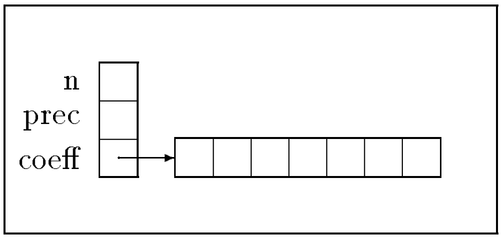

Abbildung 3.1: Dichte Darstellung von Fourier-Reihen

In der Komponente „n" dieser Struktur wird der Startexponent der Reihe gespeichert, der Vektor „coeff" enthält die Koeffizienten $a_{n_f}, a_{n_f}, \ldots, a_{N_f}$ und die Komponente „prec" enthält die aktuelle Größe des Vektors „coeff" ($prec = N_f - n_f + 1$). Durch die Position der Koeffizienten in dem Vektor ist somit eindeutig festgelegt, welchen Exponenten der Reihe er definiert. Es ist

$$c[i] \quad = \quad a_{i+n} \qquad \text{für} \quad 0 \leq i < prec.$$

Realisieren wir diese Darstellung, so verwenden wir die folgende Struktur:

```
// Datentyp zur Darstellung dicht besetzter Fourier-Reihen

struct
 {
     ADT   *coeff ;          // Koeffizientenfeld der Fourier-Reihe

     long   n     ;          // Startexponent
     long   prec  ;          // Groesse des Vektors coeff

 }   dense_powser ;
```

Wir werden im folgenden Algorithmen angeben, mit denen die Operationen Addition, Multiplikation und Inversion auf dieser Datenstruktur realisiert werden können.

Addition dicht besetzter Fourier-Reihen

Zur Berechnung der Summe zweier Fourier-Reihen, untersuchen wir zunächst, ob die eingegebenen Reihen gemeinsame Exponenten definieren. Trifft dies nicht zu, so können wir die Fourier-Reihe mit kleinerem Startexponenten einfach kopieren. Überschneiden sich die Reihen, dann addieren wir die entsprechenden Koeffizienten.

Addition dicht besetzter Fourier-Reihen

EINGABE : Fourier-Reihen a,b vom Typ `dense_powser`
AUSGABE : Summe $c = a + b$ vom Typ `dense_powser`

<u>BESTIMME STARTEXPONENT UND PRÄZISION DER SUMME</u>

(1) $last_a = a.n + a.prec - 1$
(2) $last_b = b.n + b.prec - 1$
(3) $first_c = \min\,(a.n, b.n)$
(4) $last_c = \min\,(last_a, last_b)$

Addition dicht besetzter Fourier-Reihen
(Fortsetzung)

Teste, ob die Reihen a und b gemeinsame Exponenten definieren

```
if (last_a < b.n) then                               // a liegt vor b
  c = a
  return (c)
else
  if (last_b < a.n) then                             // b liegt vor a
    c = b
    return (c)
  fi
fi
```

Führe Addition aus

```
if (a.n > b.n) then
  Vertausche a und b                                 // a beginnt vor b
fi
d = b.n − a.n                                        //Anz. der Koeff., die nur von a bestimmt werden
for (i = 0 ; i < d ; i++ ) do
  c.coeff[i] = a.coeff[i]                            // Koeff. von a übernehmen
od
for (i = d ; i ≤ last_c − a.n ; i++) do
  c.coeff[i] = a.coeff[i] + b.coeff[i − d]           // gemeinsame Koeff.
od
```

Teste, ob führende Koeffizienten gleich 0 sind

```
i = 0
while ( c.coeff[i] == 0 ) do
  i++
od
first_c = first_c + i
Verkürze c.coeff entsprechend
```

Addition dicht besetzter Fourier-Reihen
(Fortsetzung)

<u>SPEICHERE STARTEXPONENTEN UND PRÄZISION VON c</u>

(30) $c.n = first_c$
(31) $c.prec =$ Länge des Vektors c.coeff
(32) **return** (c)

Multiplikation dicht besetzter Fourier-Reihen

Die Multiplikation von Fourier-Reihen in dichter Darstellung können wir sofort mit Hilfe der Formel (3.6) implementieren. Wir berechnen dazu zunächst den Startexponenten des zu berechnenden Produktes und fassen dann die Koeffizientenvektoren der Eingabereihen als Fourier-Reihen mit Startexponent 0 auf.

Multiplikation dicht besetzter Fourier-Reihen

EINGABE : Fourier-Reihen a,b vom Typ `dense_powser`
AUSGABE : Produkt $c = a \cdot b$ vom Typ `dense_powser`

<u>BESTIMME STARTEXPONENT UND PRÄZISION DES PRODUKTES</u>

(1) $c.prec = \min\,(a.prec, b.prec)$
(2) $c.n = a.n + b.n$

Multiplikation dicht besetzter Fourier-Reihen
(Fortsetzung)

<u>Berechne Koeffizienten von c nach Formel (3.6)</u>

(3) **for** $(i = 0\ ;\ i < c.prec\ ;\ i++)$ **do** // Ber. i-ten Koeff.
(4) c.coeff[i] = 0
(5) **for** $(j = 0\ ;\ j \leq i\ ;\ j++)$ **do**
(6) tmp = a.coeff[i] $*$ b.coeff[$i-j$]
(7) c.coeff[i] = c.coeff[i] + tmp
(8) **od**
(9) **od**
(10) **return** (c)

Zur Multiplikation dicht besetzter Fourier-Reihen verwenden wir in der Praxis das im Abschnitt 3.3 vorgestellte Verfahren, die Fourier-Reihen-Multiplikation zurückzuführen auf die Multiplikation von Polynomen. Von Victor Shoup wurde uns dazu eine Programm-Bibliothek zur Verfügung gestellt, mit deren Hilfe wir die Polynom-Multiplikation durch FFT berechnen können. Damit erhalten wir dann den folgenden Algorithmus.

Multiplikation dicht besetzter Fourier-Reihen mit FFT

Eingabe : Fourier-Reihen a,b vom Typ `dense_powser`
Ausgabe : Produkt $c = a \cdot b$ vom Typ `dense_powser`

<u>Bestimme Startexponent und Präzision des Produktes</u>

(1) $c.prec = \min\ (a.prec, b.prec)$
(2) $c.n = a.n + b.n$

<u>Berechne Polynom-Multiplikation mit FFT</u>

(3) tmp = FFTMul(a.coeff, b.coeff)

Multiplikation dicht besetzter Fourier-Reihen mit FFT (Fortsetzung)

KOPIERE KORREKT BERECHNETE KOEFFIZIENTEN

(4) **for** $(i = 0 \; ; \; i < c.prec \; ; \; i++)$ **do**
(5) c.coeff[i] = tmp[i]
(6) **od**
(7) **return** (c)

Um die Auswirkungen der schnellen Fourier-Transformation in der Praxis zu verdeutlichen, geben wir an dieser Stelle einige Laufzeitvergleiche an. Dabei haben wir Fourier-Reihen über endlichen Körpern einer Charaktersitik $\approx 2^{26}$ betrachtet und je eine Multiplikation ausgeführt.

Präzision	Standard	FFT	FFT / Std.
1000	1.5 sec	0.3 sec	20.00 %
2000	6.5 sec	0.7 sec	10.78 %
4000	26.4 sec	1.5 sec	5.68 %
8000	1 min 45.0 sec	3.2 sec	0.03 %
16000	7 min 10.0 sec	6.8 sec	0.01 %

Schnelle Exponentiation

Mit Hilfe der Algorithmen zum Multiplizieren und Quadrieren von Fourier-Reihen können wir ein spezielles Verfahren angeben, um eine Potenz einer gegebenen Fourier-Reihe zu berechnen. Wir verwenden dazu den Algorithmus der Schnellen Exponentiation. Die Idee zu diesem Verfahren liefert die folgende Beobachtung:

3.18. Beispiel Sei $a = a(X)$ eine beliebige Fourier-Reihe. Dann gilt

$$\begin{aligned} a^{21} &= a^{16+4+1} = a^{16} \cdot a^4 \cdot a^1 \\ &= a^{2^4} \cdot a^{2^2} \cdot a^{2^0}. \end{aligned}$$

Betrachten wir die Binärdarstellung $(10101)_2$ von 21, so können wir erkennen, daß bei der Berechnung von a^{21} genau die Faktoren vorkommen, die einer 1 in dieser Darstellung entsprechen. Weiterhin können die einzelnen Faktoren durch sukzessives Quadrieren von a berechnet werden:

$$a^{16} \quad = \quad \left(\left(\left(a^2\right)^2\right)^2\right)^2.$$

Zur Berechnung einer beliebigen n-ten Potenz einer Fourier-Reihe a können wir diese also sukzessive quadrieren und dabei für jede 1 in der Binärdarstellung von n den zugehörigen Faktor aufmultiplizieren. Wir erhalten dann den folgenden Algorithmus.

Schnelle Exponentiation

EINGABE : Fourier-Reihe a vom Typ `dense_powser`
$n \in \mathbb{N}$
AUSGABE : $b = a^n$ vom Typ `dense_powser`

(1) $b = 1 \cdot X^0$ // konstante Reihe 1
(2) $tmp = a$
(3) **while** $(n \geq 1)$ **do**
(4) **if** (n ist ungerade) **then** // 1 in Binärdarst.
(5) $b = b \cdot tmp$
(6) **fi**
(7) $tmp = tmp^2$
(8) $n = n \,/\, 2$
(9) **od**

Inversion dicht besetzter Fourier-Reihen

Auch die Implementierung der Inversion von Fourier-Reihen in dichter Darstellung können wir einfach aus den Berechnungsvorschriften (3.11) übernehmen. Wiederum berechnen wir zunächst den Startexponenten der Inversen und fassen die Eingabe als Reihe mit Startexponenten 0 auf.

Invertierung einer dicht besetzten Fourier-Reihe

EINGABE : Fourier-Reihe a vom Typ `dense_powser`
AUSGABE : Inverses $b = a^{-1}$ vom Typ `dense_powser`

<u>BESTIMME STARTEXPONENT UND PRÄZISION DES INVERSEN</u>

(1) $b.prec = a.prec$
(2) $b.n = -a.n$ // vgl. (3.11)

<u>BERECHNE INVERSES DES LEITKOEFFIZIENTEN VON a</u>

(3) a_{inv} = a.coeff[0]$^{-1}$
(4) b.coeff[0] = a_{inv}

<u>BERECHNE KOEFFIZIENTEN VON b NACH FORMEL (3.11)</u>

(5) **for** $(i = 1 \; ; \; i < b.prec \; ; \; i++)$ **do**
(6) b.coeff[i] = 0
(7) **for** $(j = 0 \; ; \; j \leq i \; ; \; j++)$ **do**
(8) tmp = a.coeff[j] $*$ b.coeff[$i - j$]
(9) b.coeff[i] = b.coeff[i] + tmp
(10) **od**
(11) b.coeff[i] = b.coeff[i] $*(-a_{inv})$
(12) **od**
(13) **return** (b)

3.4.2 Dünne Darstellung von Fourier-Reihen

Wenn wir bereits a priori wissen, daß die Koeffizienten einer Approximation zu einem „großen" Teil gleich 0 sind, so wählen wir eine *dünne* Darstellung. Dabei werden nur diejenigen Koeffizienten gespeichert, die von 0 verschieden sind. Für eine Approximation

$$f(X) \quad \approx \quad \sum_{\nu=n_f}^{N_f} a_\nu \cdot X^\nu$$

speichern wir den Startexponenten n_f sowie den maximal auftretenden Exponenten N_f und für jeden Koeffizient a_ν das Paar (a_ν, ν). Somit ergibt sich die folgende Struktur:

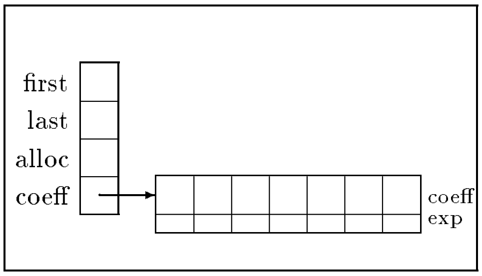

Abbildung 3.2: Dünne Darstellung von Fourier-Reihen

Dabei werden in den Komponenten „first" und „last" die Exponenten n_f und N_f gespeichert, und „coeff" ist ein Vektor, dessen Einträge jeweils aus einem Koeffizienten und dem zugehörigen Exponenten bestehen. In dieser Struktur enthält „alloc" die aktuelle Größe des Vektors „coeff".

Die Darstellung dünner Fourier-Reihen, kann in der Implementierung durch den Datentyp

```
// Datentyp zur Darstellung duenn besetzter Fourier-Reihen

struct
 {
     coefficient *coeff ;    // Koeffizientenfeld der Fourier-Reihe

     long   first  ;         // Startexponent
     long   last   ;         // Hoechster bekanner Koeffizient
     long   alloc  ;         // Groesse des Vektors coeff

 }   sparse_powser ;
```

```
struct
 {
     ADT    coeff  ;                    //  Koeffizient ungleich 0
     long   exp    ;                    //  zugehoeriger Exponent

 }  coefficient ;
```

realisiert werden. Wir geben nun wieder an, wie die Algorithmen für Addition, Multiplikation und Inversion auf dieser Datenstrukur arbeiten.

Addition dünn besetzter Fourier-Reihen

Zur Berechnung der Summe zweier Fourier-Reihen in dünner Darstellung berechnen wir zunächst den Startexponenten „first" und den maximalen Exponenten „last" der resultierenden Reihe. Damit können wir dann feststellen, wie weit wir die Koeffizienten-Vektoren der eingegebenen Reihen betrachten müssen. Wir untersuchen dann in jedem Schritt je einen Eintrag in jeder Reihe. Durch Vergleich der gespeicherten Exponenten können wir dann entscheiden, ob die zugehörigen Koeffizienten addiert werden müssen oder nicht. Stimmen die Exponenten der Einträge nicht überein, so übernehmen wir den Koeffizienten, der den kleineren Exponenten definiert.

Addition dünn besetzter Fourier-Reihen

EINGABE : Fourier-Reihen a,b vom Typ `sparse_powser`
AUSGABE : Summe $c = a + b$ vom Typ `sparse_powser`

<u>BESTIMME STARTEXPONENT UND HÖCHSTEN EXPONENT DER SUMME</u>

(1) $c.first = \min\ (a.first, b.first)$
(2) $c.last = \min\ (a.last, b.last)$

<u>BESTIMME INDIZIES DER HÖCHSTEN ZU BETRACHTENDEN KOEFFIZIENEN</u>

(3) Best. max_a mit a.coeff[i].exp $>$ c.last,
für $i > max_a$
(4) Best. max_b mit b.coeff[i].exp $>$ c.last,
für $i > max_b$

Addition dünn besetzter Fourier-Reihen
(Fortsetzung)

<u>Führe Addition aus</u>

```
ac = a.coeff                                   // Hilfsvariable
bc = b.coeff
i = j = k = 0                                  // k = Anz. Koeff. in c
while ( i ≤ max_a ∧ j ≤ max_b) do
  if (ac[i].exp == bc[j].exp) then             // gleiche Exponenten
    c.coeff[k].coeff = ac[i].coeff + bc[j].coeff
    c.coeff[k].exp = ac[i].exp
    if (c.coeff[k].coeff ≠ 0) then             // Koeff. 0 nicht speichern
      k++
    fi
    i++
    j++
  fi
  if (ac[i].exp < bc[j].exp) then              // Koeff. von a
    c.coef[k] = ac[i].coeff
    c.coef[k].exp = ac[i].exp
    k++
    i++
  fi
  if (ac[i].exp > bc[j].exp) then              // Koeff von b
    c.coef[k] = bc[j].coeff
    c.coef[k].exp = bc[j].exp
    k++
    j++
  fi
od
```

Addition dünn besetzter Fourier-Reihen
(Fortsetzung)

GGF. RESTLICHE REIHE ÜBERNEHMEN

```
while (i ≤ max_a) do                        // Reihe a kopieren
   c.coef[k] = ac[i].coeff
   c.coef[k].exp = ac[i].exp
   k++
   i++
od
while (j ≤ max_b) do                        // Reihe b kopieren
   c.coef[k] = bc[j].coeff
   c.coef[k].exp = bc[j].exp
   k++
   j++
od
```

SPEICHERPLATZ ANPASSEN

```
Reserviere Platz für k Koeff. in c
c.alloc = k
return (c)
```

Multiplikation dünn besetzter Fourier-Reihen

Zur Multiplikation von Fourier-Reihen untersuchen wir wiederum zuerst die Exponenten „first“ und „last“ des Produktes. Danach betrachten wir alle Paare von Koeffizienten je einer Reihe. Die Summe der zugehörigen Exponenten ist dann gleich dem Exponenten des Produkts, zu dem die Koeffizienten beitragen. Ist dies ein Exponent, dessen zugehöriger Koeffizient in der Approximation des Produktes korrekt bestimmt werden kann, dann berechnen wir das entsprechende Produkt der Koeffizienten.

Multiplikation dünn besetzter Fourier-Reihen

EINGABE : Fourier-Reihen a,b vom Typ `sparse_powser`
AUSGABE : Produkt $c = a \cdot b$ vom Typ `sparse_powser`

<u>BESTIMME STARTEXPONENT UND HÖCHSTEN EXPONENT DES PRODUKTS</u>

(1) $c.first = a.first + b.first$
(2) $c.last = \min\ (a.first + b.last, b.first + a.last)$
(3) Reserviere Platz für $c.last - c.first + 1$ Koeffizienten in c
(4) Initialisiere alle Koeff. mit 0

<u>FÜHRE MULTIPLIKATION AUS</u>

```
 (5)  ac = a.coeff                               // Hilfsvariable
 (6)  bc = b.coeff
 (7)  for (i = 0 ; i < a.alloc ; i++) do
 (8)     for (j = 0 ; j < b.alloc ; j++) do      // Betr. alle (i,j)
 (9)        exp = ac[i].exp + bc[j].exp
(10)        if (exp < c.last) then               // gültiger Exponent
(11)           tmp = ac[i].coeff * bc[j].coeff
(12)           c.coeff[exp - c.first] += tmp
(13)        fi
(14)     od
(15)  od
```

<u>SPEICHERPLATZ ANPASSEN</u>

(16) Lösche alle Koeffizienten 0 in c
(17) Verkleinere ggf. den reservierten Speicherplatz
(18) **return** (c)

3.19. Bemerkung Zur Berechnung von Potenzen einer gegebenen Reihe in dünner Darstellung können wir den Algorithmus der Schnellen Exponentiation, wie auf Seite 43 beschrieben, wörtlich übernehmen.

Inversion dünn besetzter Fourier-Reihen

Wollen wir das Inverse einer Fourier-Reihe berechnen, so verwenden wir einfach die Formeln (3.11). Die darin benötigten Koeffizienten der Eingabereihe bestimmen wir jeweils durch eine Suche in deren Koeffizientenvektor.

Invertierung einer dünn besetzten Fourier-Reihe

EINGABE : Fourier-Reihe a vom Typ `sparse_powser`
AUSGABE : Inverses $b = a^{-1}$ vom Typ `sparse_powser`

<u>BESTIMME STARTEXPONENT UND HÖCHSTEN EXPONENT DES INVERSEN</u>

(1) $b.first = -a.first$
(2) $b.last = a.last - 2a.first$
(3) Reserviere Platz für $b.last - b.first +$ 1 Koeff. in b
(4) Initialisiere alle Koeff. mit 0

<u>BERECHNE INVERSES DES LEITKOEFFIZIENTEN VON A</u>

(5) a_{inv} = a.coeff[0].coeff^{-1}
(6) b.coeff[0] = a_{inv}
(7) k=1 // Anz. Koeff. in b

<u>BERECHNE KOEFFIZIENTEN MIT FORMEL (3.11)</u>

(8) ac = a.coeff // Hilfsvariable
(9) bc = b.coeff

Invertierung einer dünn besetzten Fourier-Reihe
(Fortsetzung)

```
(10) for (i = b.first ; i ≤ b.last ; i++) do            // Koeff. zu Exp. i
(11)   sum = 0
(12)   for (u = i−1 ; u ≥ b.first ; u−−) do
(13)     l = i − b − first − u                          // l + u = (i-b.first)
(14)     Suche i_0 mit ac[i_0].exp = l
(15)     Suche j_0 mit bc[j_0].exp = u
(16)     if (i_0 und j_0 existieren ) then
(17)       tmp = ac[i_0] * bc[j_0]
(18)       sum = sum + tmp
(19)     fi
(20)   od
(21)   if (sum ≠ 0) then
(22)     bc[k].coeff = −a_inv * sum
(23)     bc[k].exp = i
(24)     k++
(25)   fi
(26) od
```

<u>Speicherplatz anpassen</u>

```
(27) Verkleinere Koeffizientenfeld von b
     auf Länge k
(28) return (b)
```

Kapitel 4

Modulare Polynome

Bei der Beschreibung des Algorithmus von Atkin haben wir festgestellt, daß das Problem, die Punktanzahl einer elliptischen Kurve über einem endlichen Körper zu berechnen, zurückgeführt werden kann auf die Bestimmung minimaler Erweiterungskörper von $\mathbb{F}_p$, in denen die j-Invarianten geeigneter Kurven liegen. Zur Untersuchung dieser j-Invarianten können spezielle Polynome verwendet werden — die *modularen Polynome*.

In diesem Kapitel werden wir diese Polynome genauer beschreiben. Dazu werden wir zunächst die Situation für elliptische Kurven über den komplexen Zahlen betrachten und dann angeben, wie die dort konstruierten Polynome auf endliche Körper übertragen werden können.

4.1 Elliptische Kurven über $\mathbb{C}$

Betrachten wir elliptische Kurven über dem Körper $\mathbb{C}$ der komplexen Zahlen, so kann man einen engen Zusammenhang zu *Gittern* beobachten.

4.1. Definition *Seien $\omega_1, \omega_2 \in \mathbb{C}$ zwei von 0 verschiedene komplexe Zahlen mit der Eigenschaft, daß $\Im\left(\frac{\omega_1}{\omega_2}\right) > 0$. Dann ist die diskrete Teilmenge*

$$L \quad = \quad \mathbb{Z}\,\omega_1 \,+\, \mathbb{Z}\,\omega_2$$

von $\mathbb{C}$ ein zweidimensionales **Gitter** *in $\mathbb{C}$.*

Ein Gitter bildet damit insbesondere auch eine additive Untergruppe von $\mathbb{C}$. Damit ist dann auch die zugehörige Faktorgruppe $\mathbb{C}/L$ bildbar. Den Zusammenhang zwischen elliptischen Kurven über $\mathbb{C}$ und Gittern beschreibt der folgende Satz.

4.2. Satz *Sei L ein zweidimensionales Gitter in $\mathbb{C}$. Dann existieren dazu komplexe Zahlen a_L und b_L sowie ein Isomorphismus zwischen $\mathbb{C}/L$ und der Punktgruppe $E(\mathbb{C})$ der elliptischen Kurve $E = (a_L, b_L)$ mit*

$$E(\mathbb{C}) \quad \cong \quad \mathbb{C}/L.$$

Beweis: Vergleiche dazu [Mü94]. ■

Damit lassen sich also auch Aussagen über Gitter auf die zugehörigen elliptischen Kurven übertragen. Damit wir uns im folgenden darauf beschränken können, Gitter einfacher Gestalt zu betrachten, untersuchen wir nun, wie sich Isomorphie elliptischer Kurven auf die jeweiligen Gitter überträgt. Dazu definieren wir zunächst den Begriff äquivalenter Gitter.

4.3. Definition *Zwei Gitter L_1 und L_2 in $\mathbb{C}$ heißen* **äquivalent**, *falls eine Zahl $\alpha \in \mathbb{C}$, $\alpha \neq 0$, existiert mit*

$$L_1 \quad = \quad \alpha \cdot L_2.$$

4.4. Lemma *Seien E_1 und E_2 elliptische Kurven über den komplexen Zahlen und L_1 sowie L_2 die zugehörigen Gitter, so daß $E_i(\mathbb{C}) \cong \mathbb{C}/L_i$ $(i = 1, 2)$. Dann sind E_1 und E_2 genau dann* isomorph, *wenn die Gitter L_1 und L_2* äquivalent *sind.*

Beweis: Siehe [Si85, Korollar 4.1.1]. ■

Aufgrund dieser Aussagen genügt es also, wenn wir im folgenden stets Gitter der Form

$$L \quad = \quad \mathbb{Z} + \mathbb{Z}\tau$$

betrachten ($\tau \in \mathbb{C}$ mit $\Im(\tau) > 0$).

4.5. Definition *Zu einem Gitter L in $\mathbb{C}$ sei die* **j-Invariante** *$j(L)$ definiert als*

$$j(L) \quad = \quad 1728 \cdot \frac{g_2^3(L)}{g_2^3(L) - g_3^2(L)}.$$

Dabei sind $g_2(L)$ und $g_3(L)$ von L abhängige Konstanten (vgl. [La87]).

Eine Eigenschaft der j-Invariante von Gittern beschreibt der folgende Satz:

4.6. Satz *Zwei Gitter L und M in $\mathbb{C}$ sind genau dann äquivalent, wenn ihre j-Invarianten übereinstimmen, d.h.*

$$j(L) \quad = \quad j(M).$$

Die j-Invariante liefert aber auch eine Funktion auf den komplexen Zahlen. Geben wir zu einem Gitter stets ein dazu äquivalentes Gitter der Form $L = \mathbb{Z} + \mathbb{Z}\tau$ an, so können wir $j(L)$ auch auffassen als Funktion in τ: $j(L) = j(\tau)$. Diese Funktion werden wir bei der Herleitung eines geeigneten Polynoms benötigen.

4.2 Der Körper der Modulfunktionen

In diesem Abschnitt werden wir eine wichtige Eigenschaft der zuvor vorgestellten Funktion $j(\tau)$ beschreiben. Dazu betrachten wir einen speziellen Körper meromorpher Funktionen.

Sei dazu wie in Definition 1.16

$$\Gamma \quad = \quad \left\{ \begin{pmatrix} a & b \\ c & d \end{pmatrix} \in \mathbb{Z}^{2,2} \;\middle|\; ad - bc = 1 \right\}$$

die homogene Modulgruppe. Sei weiter $\mathcal{H}$ die obere komplexe Halbebene

$$\mathcal{H} \quad = \quad \{\, \omega \in \mathbb{C} \mid \Im(\omega) > 0 \,\}$$

und

$$\mathcal{H}^* \quad = \quad \mathcal{H} \cup \mathbb{Q} \cup \{i\infty\}.$$

Dann definieren wir eine gewisse Klasse von Funktionen (vgl. [Sc74, Seite 26]):

4.7. Definition *Eine Funktion*

$$f: \;\; \mathcal{H}^* \longrightarrow \mathbb{C} \cup \{\infty\}$$

heißt eine **Modulfunktion**, *falls sie den folgenden Bedingungen genügt:*

1. *f ist meromorph auf $\mathcal{H}$.*
2. *Für alle $M \in \Gamma$ und für alle $\tau \in \mathcal{H}^*$ gilt $f(M(\tau)) = f(\tau)$.*
3. *Es gibt $a \in \mathbb{R}_{>0}$ und $k_0 \in \mathbb{Z}$, so daß $f(\tau)$ für alle τ mit $\Im(\tau) > a$ eine Entwicklung der Form*

$$f(\tau) \quad = \quad \sum_{k=k_0}^{\infty} b_k \cdot \exp(2\pi i \tau)^k$$

mit $b_{k_0} \neq 0$ besitzt.

Die Funktion $j(\tau)$ ist eine solche Modulfunktion, und sie besitzt in dieser Menge noch eine wichtige Eigenschaft.

4.8. Satz *Die Menge aller Modulfunktionen bildet einen Körper K_Γ. Insbesondere gilt*

$$K_\Gamma = \mathbb{C}\,(j(\tau))\,.$$

Beweis: Siehe dazu [Sc74, Satz 8, Seite 36]. ∎

4.3 Modulare Polynome

Seien nun $E/\mathbb{C}$ eine elliptische Kurve über $\mathbb{C}$ und $L = \mathbb{Z}+\mathbb{Z}\,\tau$ das zugehörige Gitter mit der Eigenschaft $E(\mathbb{C}) \cong \mathbb{C}/L$. Sei weiter $j(\tau)$ die j-Invariante dieses Gitters. Mit Hilfe der Transformationen

$$\alpha_n = \begin{pmatrix} 1 & n \\ 0 & l \end{pmatrix} \qquad \text{für} \qquad 0 \le n < l \tag{4.1}$$

$$\alpha_l = \begin{pmatrix} l & 0 \\ 0 & 1 \end{pmatrix} \tag{4.2}$$

können wir nun die gewünschten Polynome definieren.

4.9. Definition *Sei $l > 2$ eine Primzahl und α_n $(0 \le n \le l)$ die in (4.1) und (4.2) definierten Transformationen. Dann ist das* l**-te modulare Polynom über** $\mathbb{C}$ *gegeben durch*

$$\Psi_l(X) = \prod_{n=0}^{l} (X - j(\alpha_n(\tau)))\,. \tag{4.3}$$

Eine wichtige Eigenschaft modularer Polynome ist, daß deren Koeffizienten mit Hilfe von ganzzahligen Polynomen in $j(\tau)$ dargestellt werden können, d.h

$$\Psi_l(X) \in \mathbb{Z}[j(\tau)][X].$$

In [Sc74, Seite 143] wird gezeigt, daß der Grad dieser Koeffizientenpolynome in $j(\tau)$ gleich $l+1$ ist. Damit ist also Ψ_l von der Form

$$\Psi_l(X) = \sum_{\lambda=0}^{l+1}\sum_{\kappa=0}^{l+1} a_{\lambda,\kappa}\cdot X^\lambda \cdot j(\tau)^\kappa = \Psi_l(X, j(\tau)).$$

Ersetzen wir darin $j(\tau)$ durch eine weitere Unbestimmte Y, so erhalten wir das Polynom

$$\Psi_l(X,Y) = \sum_{\lambda=0}^{l+1}\sum_{\kappa=0}^{l+1} a_{\lambda,\kappa}\cdot X^\lambda \cdot Y^\kappa \in \mathbb{Z}[X,Y]. \tag{4.4}$$

Die beiden Darstellungen (4.3) und (4.4) des l-ten modularen Polynoms werden wir im folgenden identifizieren. Die Bedeutung dieser Polynome für den Algorithmus von Atkin beschreibt der nächste Satz.

4.10. Satz *Seien E_1 und E_2 elliptische Kurven über $\mathbb{C}$ und $l > 2$ eine Primzahl. Sei weiter $\Psi_l(X,Y)$ das l-te modulare Polynom über $\mathbb{C}$. Genau dann existiert eine Isogenie zwischen E_1 und E_2 mit einem Kern der Ordnung l, wenn $j(E_2)$ Nullstelle von $\Psi_l(X, j(E_1))$ ist.*

Beweis: Siehe [La87, Seite 59]. ■

Erinnern wir uns an das Problem, das wir zur Bestimmung der Punktanzahl einer elliptischen Kurve E über einem endlichen Körper mit Hilfe der modularen Polynome lösen wollten. Ziel war es dabei, ein Polynom zu konstruieren, welches genau die j-Invarianten der elliptischen Kurven E_C zu allen l-elementigen Untergruppen der l-Torsionsgruppe von E als Nullstellen besitzen sollte (vgl. Satz 2.10). Dabei haben wir festgestellt, daß diese Kurven E_C stets zu E isogen sind vermöge einer Isogenie φ mit Kern $(\varphi) = C$. Es treten also genau solche Isogenien auf, deren Kern die Ordnung l besitzt. Gelingt es nun, Satz 4.10 auf die Situation endlicher Körper zu übertragen, dann können wir das zugehörige Polynome im Algorithmus von Atkin verwenden. Dies leistet der folgende Satz.

4.11. Satz *Sei $\mathbb{F}_p$ ein endlicher Körper der Charakteristik $p > 3$, E eine über $\mathbb{F}_p$ definierte elliptische Kurve und l eine ungerade Primzahl ($l \neq p$). Sei weiter $\Psi_l(X,Y)$ das l-te modulare Polynom über $\mathbb{C}$ und $\overline{\Psi}_l(X,Y)$ aus diesem durch Reduktion der Koeffizienten modulo p entstanden. Dann sind die Nullstellen von $\overline{\Psi}_l(X, j(E)) \in \mathbb{F}_p[X]$ genau die j-Invarianten $j(E_C)$ (vgl. Satz 2.10) für die Gesamtheit der l-elementigen Untergruppen C von $E(\overline{\mathbb{F}_p})$.*

Beweis: Siehe dazu [Mü94]. ■

Erfüllt die elliptische Kurve noch einige weitere Bedingungen, so ist das Polynom $\overline{\Psi}_l(X,Y)$ in Satz 4.11 doppelwurzelfrei.

4.12. Lemma *Sei E eine nicht supersinguläre Kurve über $\mathbb{F}_p$ mit j-Invariante ungleich 0 oder 1728, und sei $l > 2$ eine zu p teilerfremde Primzahl. Dann besitzt das Polynom $\overline{\Psi}_l(X, j(E)) \in \mathbb{F}_p[X]$ in Satz 4.11 nur einfache Nullstellen.*

Beweis: Vergleiche [Mü94]. ■

Damit haben wir also ein Polynom mit den gewünschten Eigenschaften erhalten. Kennen wir die Koeffizienten $a_{\lambda,\kappa} \in \mathbb{Z}$ des modularen Polynoms, so müssen wir diese lediglich modulo der Charakteristik p des jeweiligen Körpers reduzieren. Berechnen wir dann die Koeffizientenpolynome durch Einsetzen der j-Invarianten der zu untersuchenden Kurve, so liefert uns das daraus resultierende Polynom die benötigten Informationen.

Wir werden später sehen, daß die Koeffizienten $a_{\lambda,\kappa}$ des Polynoms über $\mathbb{C}$ in der Praxis um so aufwendiger zu berechnen sind, je größer der Grad von $j(\tau)$ in den Koeffizientenpolynomen ist. Daher werden wir im folgenden Abschnitt untersuchen, wie man anstatt der modularen Polynome andere Polynome verwenden kann, die immer noch dieselbe Information über die Lage der speziellen j-Invarianten liefern, deren Berechnung in der Praxis jedoch schneller durchführbar ist.

4.4 Alternativen zu modularen Polynomen

Wir wollen nun in diesem Abschnitt beschreiben, wie man anstelle modularer Polynome andere Polynome berechnen kann, die für den Algorithmus von Atkin dieselbe Information liefern, deren *Zerfällungstyp* also mit dem der jeweiligen modularen Polynome übereinstimmt. Solche Polynome nennen wir zum modularen Polynom **äquivalent**.

Dazu werden wir zunächst ein Polynom konstruieren, das ebenfalls über $\mathbb{Z}[j(\tau)]$ definiert ist und dessen Galoisgruppe mit der des entsprechenden modularen Polynoms übereinstimmt. Das zugehörige Polynom über dem endlichen Körper $\mathbb{F}_p$ liefert dann wieder die koeffizientenweise Reduktion modulo p.

4.4.1 Die Galoisgruppe modularer Polynome über $\mathbb{C}$

Betrachten wir zunächst die Galoisgruppen für modulare Polynome über den komplexen Zahlen. Dazu definieren wir noch spezielle Gruppen.

4.13. Definition *Sei K ein beliebiger Körper. Dann bezeichne $GL_2(K)$ die* **generelle lineare Gruppe** *aller invertierbaren Matrizen aus $K^{2,2}$ sowie $SL_2(K)$ die* **spezielle lineare Gruppe** *aller Matrizen aus $GL_2(K)$ mit Determinante 1. Dann seien*

$$PGL_2(K) = GL_2(K)/Z\left(GL_2(K)\right)$$

und

$$PSL_2(K) = SL_2(K)/Z\left(SL_2(K)\right).$$

Dabei bezeichne $Z(\cdot)$ das **Zentrum** *der jeweiligen Gruppe.*

Mit diesen Vereinbarungen können wir den folgenden Satz formulieren.

4.14. Satz *Sei $l > 2$ eine Primzahl. Dann ist die Galoisgruppe des l-ten modularen Polynoms über dem Körper $\mathbb{Q}(j(\tau))$ gleich $PGL_2(\mathbb{F}_l)$. Ist der Grundkörper jedoch*

$$\mathbb{Q}\left(j(\tau), \sqrt{(-1)^{(l-1)/2} \cdot l}\right),$$

so ist die Galoisgruppe $PSL_2(\mathbb{F}_l)$.

Beweis: Siehe [We08, Seite 287-289]. ∎

4.4.2 Übergang zu endlichen Körpern

Hier untersuchen wir nun, wie sich Galoisgruppen bzw. Zerfällungstypen von Polynomen über $\mathbb{Z}[j(\tau)]$ unter einer Reduktionsabbildung verhalten (siehe dazu [Mü94]).

4.15. Satz *Seien l und p voneinander verschiedene Primzahlen ungleich 2, R ein Oberring von $\mathbb{Z}[j(\tau)]$ und Δ der Quotientenkörper von R. Sei weiterhin $J \in \mathbb{F}_{p^d}^*$ und ein Reduktionshomomorphismus*

$$\begin{array}{rcll} \overline{} : R & \longrightarrow & \mathbb{F}_p[J] & \\ x & \longmapsto & x \bmod p & \text{für } x \in \mathbb{Z} \\ j(\tau) & \longmapsto & J & \end{array}$$

gegeben. Seien ein Polynom $f(X) \in R[X]$ sowie dessen Bild $\overline{f}(X) \in \mathbb{F}_p[J, X]$ ohne mehrfache Nullstellen.

Besitzt ein Polynom $g(X) \in R[X]$ über Δ dieselbe Galoisgruppe wie $f(X)$, so hat auch $\overline{g}(X)$ keine mehrfache Nullstellen, und die Galoisgruppen von $\overline{f}(X)$ und $\overline{g}(X)$ über $\mathbb{F}_p(J)$ sind gleich.

Beweis: Siehe [Mü94]. ∎

4.16. Korollar *Unter den Voraussetzungen von Satz 4.15 besitzen $\overline{f}(X)$ und $\overline{g}(X)$ über $\mathbb{F}_p(J)$ denselben Zerfällungstyp.*

Diese beiden Aussagen liefern uns eine Methode, Polynome zu finden, die wir anstelle der modularen Polynome verwenden können. Wir können jedes Polynom verwenden, das über $\mathbb{Z}[j(\tau)]$ definiert ist und dessen Galoisgruppe mit der des modularen Polynoms übereinstimmt.

4.4.3 Konstruktion äquivalenter Polynome

Wir haben bereits gesehen, daß die Funktion $j(\tau)$ den Körper K_Γ der Modulfunktionen erzeugt, d.h. $K_\Gamma = \mathbb{C}(j(\tau))$. Insbesondere umfaßt also K_Γ den Ring $\mathbb{Z}[j(\tau)]$, und damit ist für $l \in \mathbb{P}_{>2}$ das l-te modulare Polynom erst recht auch über K_Γ definiert.

$$\Psi_l(X) \quad \in \quad K_\Gamma[X].$$

In diesem Polynomring ist $\Psi_l(X)$ irreduzibel, und wir erhalten einen Erweiterungskörper von K_Γ durch

$$K_\Gamma[X]\,/(\Psi_l(X)) \quad \supseteq \quad K_\Gamma.$$

Nach Definition ist $j(l\tau)$ Nullstelle des l-ten modularen Polynoms. Somit ist $\Psi_l(X)$ das Minimalpolynom von $j(l\tau)$ über dem Körper K_Γ und damit

$$K_\Gamma[X]\,/(\Psi_l(X)) \quad \cong \quad K_\Gamma(j(l\tau)).$$

Wollen wir nun ein Polynom konstruieren, das dieselbe Galoisgruppe besitzt wie $\Psi_l(X)$, so wählen wir ein Polynom $G(X) \in K_\Gamma[X]$, das insbesondere auch

$$K_\Gamma[X]\,/(G(X)) \quad \cong \quad K_\Gamma(j(l\tau))$$

erfüllt. Dazu untersuchen wir nun den Körper $K_\Gamma(j(l\tau))$ etwas genauer. Wir werden sehen, daß die Funktionen aus $K_\Gamma(j(l\tau))$ eine gemeinsame Eigenschaft haben. Dazu betrachten wir zunächst folgende Definition:

4.17. Definition *Sei Γ_1 eine Untergruppe von Γ mit $-I \in \Gamma_1$, und habe diese endlichen Index in Γ. Eine Funktion $f(\tau)$ auf $\mathcal{H}^*$ heißt* **modulare Funktion für** *Γ_1, wenn gilt:*

1. *f ist meromorph auf $\mathcal{H}$.*
2. *Für alle $S \in \Gamma_1$ und alle $\tau \in \mathcal{H}^*$ gilt $f(S(\tau)) = f(\tau)$.*
3. *An jeder rationalen Spitze $-\frac{d}{c}$ mit $ggT\,(d,c) = 1$ besitzt $f(\tau)$ eine für komplexe Zahlen mit hinreichend großem Imaginärteil gültige Entwicklung der Form*

$$f(\tau) \quad = \quad \sum_{k=k_0}^{\infty} b_k \cdot \exp\left(\frac{2\pi i}{\kappa} \cdot A(\tau)\right)^k \qquad \mathit{mit}\ A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \in \Gamma,$$

und $\kappa \in \mathbb{N}$.

Den Körper aller dieser Funktionen bezeichne K_{Γ_1}.

Damit können wir dann den folgenden Satz formulieren:

4.18. Satz *Ist $f(\tau)$ eine Funktion mit Invarianzgruppe I_f mit $-I \in I_f$ und besitzt diese in Γ endlichen Index, so gilt*

$$K_{I_f} \quad = \quad K_\Gamma(f(\tau)).$$

Beweis: Vergleiche [Sc74, Satz 5, Seite 129]. ■

Wir können also insbesondere $K_\Gamma(j(l\tau))$ als Körper von modularen Funktionen für eine spezielle Untergruppe von Γ auffassen, wenn wir die Invarianzgruppe von $j(l\tau)$ kennen. Mit Hilfe des folgenden Lemmas können wir diese bestimmen.

4.19. Lemma *Sei $f(\tau)$ eine Modulfunktion. Dann ist die Invarianzgruppe der Funktion $f(l\tau)$ gleich $\Gamma_0(l)$.*

Beweis: Siehe [Mü94]. ■

Da nach Satz 4.8 die Funktion $j(\tau)$ eine Modulfunktion ist, erhalten wir unmittelbar das nächste Korollar.

4.20. Korollar *Die Invarianzgruppe von $j(l\tau)$ ist $\Gamma_0(l)$.*

Wir kennen nun also die Invarianzgruppe von $j(l\tau)$ und können aus Satz 4.18 sofort ablesen:

$$K_\Gamma(j(l\tau)) \quad = \quad K_{\Gamma_0(l)}.$$

Fordern wir, daß das zu konstruierende Polynom denselben Erweiterungskörper über K_Γ erzeugt wie $\Psi_l(X)$, so können wir dies erreichen, indem wir ein irreduzibles Polynom konstruieren, welches als Nullstelle eine Funktion mit Invarianzgruppe $\Gamma_0(l)$ besitzt. Ein Polynom mit diesen Eigenschaften beschreiben wir im folgenden Satz.

4.21. Satz *Sei $f(\tau)$ eine Funktion mit Invarianzgruppe $\Gamma_0(l)$ und seien die folgenden Transformationen gegeben:*

$$S_\nu \quad = \quad \begin{pmatrix} 0 & -1 \\ 1 & \nu \end{pmatrix} \quad \textit{für} \quad 0 \le \nu < l \quad \textit{und} \quad S_l \quad = \quad \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

Dann ist das Polynom

$$G_l^f(X) \quad = \quad \prod_{\nu=0}^{l} \left(X - f\left(S_\nu(\tau)\right)\right) \quad \in \quad K_\Gamma[X] \tag{4.5}$$

irreduzibel.

Beweis: Siehe dazu [Mü94]. ■

Zu jeder Funktion $f(\tau)$ mit Invarianzgruppe $\Gamma_0(l)$ können wir also ein Polynom $G_l^f(X) \in K_\Gamma[X]$ erzeugen, das unseren bisherigen Bedingungen genügt. Weiterhin stimmt sogar die Galoigruppe von $G_l^f(X)$ über $\mathbb{Q}(j(\tau))$ bzw. über $\mathbb{Q}\left(j(\tau), \sqrt{(-1)^{(l-1)/2} \cdot l}\right)$ mit der Galoigruppe des l-ten modularen Polynoms überein (vgl. Satz 4.14). Um dies zu zeigen genügt es, wenn wir nachweisen, daß die Zerfällungskörper von $\Psi_l(X)$ und $G_l^f(X)$ über K_Γ übereinstimmen. Betrachten wir die Untergruppe

$$\Gamma^*(l) \quad = \quad \left\{ S \in \Gamma \mid S \equiv \begin{pmatrix} a & 0 \\ 0 & a \end{pmatrix} \bmod l \right\}$$

von Γ für ein $n \in \mathbb{N}$, so gilt nach Satz 18 aus [Sc74], daß der Zerfällungskörper von $\Psi_l(X)$ über K_Γ gleich dem Körper $K_{\Gamma^*(l)}$ der modularen Funktionen für $\Gamma^*(l)$ ist. Die Äquivalenz der Polynome $\Psi_l(X)$ und $G_l^f(X)$ folgt dann direkt aus dem folgenden Satz.

4.22. Satz *Der Zerfällungskörper des Polynoms $G_l^f(X) \in K_\Gamma[X]$ aus (4.5) ist $K_{\Gamma^*(l)}$.*

Beweis: Vergleiche [Mü94]. ■

Wenn wir in den folgenden Kapiteln die Berechnung der Polynome $G_l^f(X)$ beschreiben, werden wir zeigen, daß eine Funktion $f(\tau)$ so gewählt werden kann, daß das zugehörige Polynom sogar in $\mathbb{Z}[j(\tau)][X]$ liegt. Für auf diese Weise gewählte Funktionen ist dann das nächste Korollar anwendbar.

4.23. Korollar *Ist das Polynom $G_l^f(X)$ aus (4.5) sogar ein Element aus $\mathbb{Z}[j(\tau)][X]$, so besitzt dessen Bild unter der Reduktionsabbildung aus Satz 4.15 denselben Zerfällungstyp wie das reduzierte l-te modulare Polynom.*

Beweis: Man wende Korollar 4.16 auf die Polynome $\Psi_l(X)$ und $G_l^f(X)$ an. ■

Kapitel 5

Berechnung äquivalenter Polynome (Typ 1)

Im vorherigen Abschnitt haben wir beschrieben, daß man, ausgehend von einer Funktion mit Invarianzgruppe $\Gamma_0(l)$, ein Polynom konstruieren kann, das denselben Zerfällungstyp aufweist wie das gesuchte *modulare* Polynom. Hier werden wir nun zwei Typen von Funktionen vorstellen, deren zugehörige Polynome wir verwenden können und Algorithmen angeben, wie die Koeffizienten der jeweiligen Polynome berechnet werden können.

5.1 Eine unter $\Gamma_0(l)$ invariante Funktion

Wir werden eine Funktion wählen, die mit Hilfe der *Dedekind'schen η-Funktion* berechnet werden kann.

5.1. Definition *Sei $q_\tau = \exp(2\pi i \tau)$. Dann wird die* Dedekind'sche η-Funktion *definiert durch das unendliche Produkt*

$$\eta(\tau) = q_\tau^{1/24} \cdot \prod_{n=1}^{\infty} (1 - q_\tau^n) \tag{5.1}$$

Diese Funktion läßt sich auch darstellen durch eine Fourier-Reihenentwicklung in q_τ. Der folgende Satz liefert eine einfache Methode, wie wir diese berechnen können.

5.2. Satz *Die Dedekind'sche η-Funktion läßt sich entwickeln in eine Fourier-Reihe in q_τ mit ganzzahligen Koeffizienten. Es gilt:*

$$\eta(\tau) = q_\tau^{1/24} \cdot \left(1 + \sum_{n=1}^{\infty} (-1)^n \cdot \left(q_\tau^{n(3n-1)/2} + q_\tau^{n(3n+1)/2}\right)\right). \tag{5.2}$$

Beweis: Siehe dazu [We08]. ∎

Ausgehend von der Funktion $\eta(\tau)$ können wir nun eine Funktion mit Invarianzgruppe $\Gamma_0(l)$ definieren. Sei dazu $s \in \mathbb{N}$ so gewählt, daß $\frac{s\cdot(l-1)}{12}$ eine ganze Zahl ist. Wir wählen s minimal mit dieser Eigenschaft und erhalten:

$$s = \frac{12}{\text{ggT } (l-1, 12)}. \tag{5.3}$$

Im folgenden sei dann stets

$$v = \frac{s\cdot(l-1)}{12} \in \mathbb{N}. \tag{5.4}$$

Mit Hilfe dieser Festlegungen konstruieren wir nun die gewünschte Funktion.

5.3. Definition *Sei $\eta(\tau)$ die oben definierte Dedekind'sche η-Funktion. Dann definieren wir*

$$f(\tau) = \left(\frac{\eta(\tau)}{\eta(l\tau)}\right)^{2s}. \tag{5.5}$$

Daß die so definierte Funktion die geforderte Invarianzgruppe besitzt, beschreibt der folgende Satz.

5.4. Satz *Die Funktion $f(\tau)$ in (5.5) besitzt die Invarianzgruppe $\Gamma_0(l)$.*

Beweis: Vergleiche dazu [Mü94] ∎

Im späteren Algorithmus werden wir die Fourier-Reihenentwicklung der Funktion $f(\tau)$ verwenden. Aussagen über deren Gestalt liefert das folgende Lemma. Dazu sei im folgenden stets $q_\tau = \exp(2\pi i\tau)$.

5.5. Lemma *Die Funktion $f(\tau) = \left(\frac{\eta(\tau)}{\eta(l\tau)}\right)^{2s}$ besitzt eine Fourier-Reihenentwicklung in q_τ mit ganzzahligen Koeffizienten. In dieser Entwicklung ist der minimale auftretende Exponent gleich -v. Es gilt somit*

$$f(\tau) = q_\tau^{-v}\cdot\sum_{n=0}^{\infty} a_n\cdot q_\tau^n \tag{5.6}$$

wobei $a_n \in \mathbb{Z}$ für alle $n \geq 0$. Insbesondere gilt $a_0 = 1$.

Beweis: Aus der q_τ-Entwicklung (5.2) der η-Funktion erhalten wir

$$\eta(\tau)^{2s} = q_\tau^{2s/24} \cdot \sum_{n=0}^{\infty} b_n \cdot q_\tau^n$$

mit $b_n \in \mathbb{Z}$ für $n \geq 0$, und $b_0 = 1$. Somit gilt auch

$$\eta(l\tau)^{2s} = q_\tau^{l\cdot(2s/24)} \cdot \sum_{n=0}^{\infty} b_n \cdot q_\tau^{l\cdot n}.$$

Da $b_0 = 1$, ist $\eta(l\tau)^{2s}$ als formale Fourier-Reihe über $\mathbb{Z}$ invertierbar. Wir erhalten also

$$\eta(l\tau)^{-2s} = q_\tau^{-l\cdot(2s/24)} \cdot \sum_{n=0}^{\infty} c_n \cdot q_\tau^n$$

mit $c_n \in \mathbb{Z}$ für $n \geq 0$, und $c_0 = 1/b_0 = 1$. Mit $\eta(\tau)^{2s}$ und $\eta(l\tau)^{-2s}$ enthält auch deren Produkt $\eta(\tau)^{2s} \cdot \eta(l\tau)^{-2s} = f(\tau)$ ganzzahlige Koeffizienten in seiner q_τ-Entwicklung. Wir erhalten also:

$$\begin{aligned}
f(\tau) &= \eta(\tau)^{2s} \cdot \eta(l\tau)^{-2s} \\
&= \left(q_\tau^{2s/24} \cdot \sum_{n=0}^{\infty} b_n \cdot q_\tau^n \right) \cdot \left(q_\tau^{-l\cdot(2s/24)} \cdot \sum_{n=0}^{\infty} c_n \cdot q_\tau^n \right) \\
&= q_\tau^{-\frac{2s\cdot(l-1)}{24}} \cdot \left(\sum_{n=0}^{\infty} b_n \cdot q_\tau^n \right) \cdot \left(\sum_{n=0}^{\infty} c_n \cdot q_\tau^n \right) \\
&= q_\tau^{-v} \cdot \sum_{n=0}^{\infty} a_n \cdot q_\tau^n
\end{aligned}$$

und dabei gilt $a_n \in \mathbb{Z}$ für $n \geq 0$ und $a_0 = b_0 \cdot c_0 = 1$. ■

Abschließend untersuchen wir noch, wie sich die Funktion $f(\tau)$ unter der Transformation $\tau \to -\frac{1}{l\tau}$ verhält. Es gilt das folgende Lemma.

5.6. Lemma *Sei* $f(\tau) = \left(\frac{\eta(\tau)}{\eta(l\tau)}\right)^{2s}$. *Dann gilt*

$$f\left(-\frac{1}{l\tau}\right) = \frac{l^s}{f(\tau)}.$$

Beweis: Siehe [Mü94]. ■

Da der „Leitkoeffizient“ a_0 in der Entwicklung von $f(\tau)$ gleich eins ist, ist $f(\tau)$ als formale Fourier-Reihe über $\mathbb{Z}$ invertierbar. Somit erhalten wir auch für die Funktion

$$g(\tau) \quad := \quad \frac{l^s}{f(\tau)} \tag{5.7}$$

eine Fourier-Reihenentwicklung

$$g(\tau) \quad = \quad q_\tau^{+v} \cdot \sum_{n=0}^{\infty} \tilde{a}_n \cdot q_\tau^n, \tag{5.8}$$

deren Koeffizienten ganzzahlig sind.

Da mit $f(\tau)$ auch $g(\tau)$ Invarianzgruppe $\Gamma_0(l)$ besitzt, können wir auch zu $g(\tau)$ das Polynom $G_l^g(X) \in \mathbb{Z}[j(\tau), X]$ wie in (4.5) berechnen. Auch dieses liefert dann eine zum *modularen Polynom* äquivalente Gleichung.

Unter Verwendung der beiden Funktionen $f(\tau)$ und $g(\tau)$ wollen wir nun im nächsten Abschnitt beschreiben, wie wir die Koeffizienten des Polynoms

$$G_l^g(X) \quad = \quad \sum_{\lambda=0}^{l+1} \sum_{\kappa=0}^{k} a_{\lambda,\kappa} \cdot j(\tau)^\kappa \cdot X^\lambda \tag{5.9}$$

bestimmen können. Dazu werden wir auch die Funktion $j(\tau)$, dargestellt durch ihre Fourier-Reihenentwicklung, benötigen. Diese können wir mit Hilfe der *2. Weber'schen Funktion*

$$f_2(\tau) \quad = \quad \sqrt{2} \cdot \frac{\eta(2\tau)}{\eta(\tau)} \tag{5.10}$$

aus der Entwicklung der η-Funktion erhalten.

5.7. Satz *Für die Funktion $j(\tau)$ gilt mit der 2. Weber'schen Funktion (5.10):*

$$j(\tau) = \frac{(f_2^{24}(\tau) + 16)^3}{f_2^{24}(\tau)} \tag{5.11}$$

Beweis: Vergleiche dazu [We08, Seite 179]. ■

Man erhält mit dieser Gleichung die Fourier-Reihenentwicklung mit ganzzahligen Koeffizienten:

$$j(\tau) = q_\tau^{-1} + 744 + 196884 \cdot q_\tau + \ldots \tag{5.12}$$

Insbesondere besitzt diese Darstellung den „Leitkoeffizienten“ 1, und der kleinste auftretende Exponent ist -1. Betrachten wir auch für $j(\tau)$ das Verhalten unter der Transformation $\tau \to -\frac{1}{l\tau}$, so erhalten wir:

5.8. Lemma *Für die j-Invariante $j(\tau)$ gilt*

$$j\left(-\frac{1}{l\tau}\right) = j(l\tau). \tag{5.13}$$

Beweis: Wir wissen, daß $j(\tau)$ eine *Modulfunktion* ist. Damit ist diese Funktion insbesondere invariant unter allen unimodularen Transformationen aus Γ. Betrachten wir also speziell die Transformation $\tau \to -\frac{1}{\tau}$, so gilt

$$j\left(-\frac{1}{\tau}\right) = j(\tau)$$

für alle komplexen Zahlen mit positivem Imaginärteil. Da mit τ auch $l \cdot \tau$ positiven Imaginärteil besitzt, folgt daraus sofort:

$$j\left(-\frac{1}{l\tau}\right) = j(l\tau).$$

■

5.2 Berechnung des Polynoms für $\Gamma_0(l)$

Nach Konstruktion ist die Funktion $g(\tau)$ aus (5.7) Nullstelle des gesuchten Polynoms $G_l^g(X)$. Damit erhalten wir aus (4.5) die Gleichung

$$\sum_{\lambda=0}^{l+1} \sum_{\kappa=0}^{k} a_{\lambda,\kappa} \cdot j(\tau)^{\kappa} \cdot g(\tau)^{\lambda} = 0. \tag{5.14}$$

Die maximale Potenz k von $j(\tau)$ in dieser Gleichung, werden wir in einem der folgenden Abschnitte bestimmen.

Betrachten wir das Verhalten dieser Gleichung unter der Transformation $\tau \to -\frac{1}{l\tau}$, so erhalten wir wegen

$$g\left(-\frac{1}{l\tau}\right) = \frac{l^s}{f\left(-\frac{1}{l\tau}\right)} = f(\tau) \qquad \text{und} \qquad j\left(-\frac{1}{l\tau}\right) = j(l\tau)$$

die Aussage

$$\sum_{\lambda=0}^{l+1} \sum_{\kappa=0}^{k} a_{\lambda,\kappa} \cdot j(l\tau)^{\kappa} \cdot f(\tau)^{\lambda} = 0. \tag{5.15}$$

Wir werden nun zeigen, wie man mit Hilfe der Gleichung (5.15) die Koeffizienten $a_{\lambda,\kappa}$ bestimmen kann. Dazu untersuchen wir das Verhalten von $f(\tau)$ und $j(l\tau)$ unter den folgenden Transformationen:

5.9. Definition *Es seien*

$$V_n = \begin{pmatrix} l & n \\ 0 & l \end{pmatrix} \quad \textit{für} \quad 0 \leq n < l \tag{5.16}$$

$$\textit{und} \quad V_l = \begin{pmatrix} 0 & -1 \\ l^2 & 0 \end{pmatrix}. \tag{5.17}$$

Für die oben definierten Funktionen $f(\tau)$ und $j(\tau)$ gilt das folgende Lemma:

5.10. Lemma *Seien V_n für $0 \leq n \leq l$ wie in Definition 5.9. Dann gilt :*

$$\begin{aligned} j(l \cdot V_n(\tau)) &= j(l\tau) \quad \textit{für} \quad 0 \leq n \leq l, \\ f(V_n(\tau)) &= f\left(\frac{l\tau + n}{l}\right) = f\left(\tau + \frac{n}{l}\right) \quad \textit{für} \quad 0 \leq n < l, \\ \textit{und} \quad f(V_l(\tau)) &= \frac{l^s}{f(l\tau)}. \end{aligned}$$

Beweis: Die Aussagen über die Funktion $f(\tau)$ folgen sofort nach Anwendung der jeweiligen Transformationen V_n für $0 \leq n < l$; die Funktion $f(V_l(\tau))$ kann mit Hilfe von Lemma 5.6 berechnet werden.

Da $j(\tau)$ eine Modulfunktion ist, ist diese insbesondere invariant unter der Transformation $\tau \to \tau + n$. Wir erhalten daher für $0 \leq n < l$

$$j(\tau) = j(\tau + n) \quad \Longrightarrow \quad j(l\tau) = j(l\tau + n).$$

Damit folgt

$$j(l \cdot V_n(\tau)) = j\left(l \cdot \frac{l\tau + n}{l}\right) = j(l\tau + n) = j(l\tau).$$

Mit Hilfe von Lemma 5.8 berechnen wir schließlich

$$j(l \cdot V_l(\tau)) = j\left(-l \cdot \frac{1}{l^2\tau}\right) = j\left(-\frac{1}{l\tau}\right) = j(l\tau). \qquad \blacksquare$$

Insbesondere ist also $j(l\tau)$ *invariant* unter all diesen Transformationen, und die $f(V_n(\tau))$ sind *paarweise verschieden* für $0 \leq n \leq l$. Wenden wir nun die Transformationen V_n $(0 \leq n \leq l)$ auf die Gleichung (5.15) an, so erhalten wir

$$\sum_{\lambda=0}^{l+1} \sum_{\kappa=0}^{k} a_{\lambda,\kappa} \cdot j(l\tau)^\kappa \cdot f(V_n(\tau))^\lambda = 0 \quad \text{für} \quad n = 0, \ldots, l. \tag{5.18}$$

Somit bilden die $f(V_n(\tau))$ $(0 \leq n \leq l)$ die Gesamtheit der Nullstellen des Polynoms

$$H(X) \quad := \quad \sum_{\lambda=0}^{l+1} \underbrace{\left(\sum_{\kappa=0}^{k} a_{\lambda,\kappa} \cdot j(l\tau)^{\kappa} \right)}_{=: k_\lambda(\tau)} \cdot X^{\lambda}. \tag{5.19}$$

Unser Ziel ist es nun, mit Hilfe der Nullstellen $f(V_n(\tau))$ $(0 \leq n \leq l)$ die Koeffizienten $k_\lambda(\tau)$ des Polynoms $H(X)$ zu bestimmen und aus diesen die $a_{\lambda,\kappa}$ zu berechnen. Kennen wir nämlich die q_τ-Entwicklungen der Funktionen $k_\lambda(\tau)$ für $0 \leq \lambda \leq l+1$

$$k_\lambda(\tau) = \sum_{i=m}^{\infty} b_{\lambda,i} \cdot q_\tau^i, \tag{5.20}$$

so müssen wir lediglich die Koeffizienten der q_τ-Potenzen beider Seiten der Gleichung

$$\sum_{\kappa=0}^{k} a_{\lambda,\kappa} \cdot j(l\tau)^{\kappa} \quad = \quad \sum_{i=m}^{\infty} b_{\lambda,i} \cdot q_\tau^i \tag{5.21}$$

vergleichen. Aus der Darstellung (5.12) von $j(\tau)$ folgt, daß der Startexponent von $j(l\tau)$ gleich $-l$ ist. Damit gilt für alle Potenzen $j(l\tau)^\kappa$ $(\kappa \geq 0)$

$$\text{first } \left(j(l\tau)^{\kappa}\right) \quad = \quad -l \cdot \kappa.$$

Der Startexponent der linken Seite der Gleichung (5.21) ist somit das Minimum dieser Werte für $0 \leq \kappa \leq k$, also $-lk$. Weiterhin wird der „Leitkoeffizient" dieser Summe ausschließlich durch den Summanden $a_{\lambda,k} \cdot j(l\tau)^k$ bestimmt, und für diesen gilt:

$$\text{lc } \left(a_{\lambda,k} \cdot j(l\tau)^{k}\right) \;=\; a_{\lambda,k} \cdot \text{lc } \left(j(l\tau)^{k}\right) \;=\; a_{\lambda,k} \cdot 1 \;=\; a_{\lambda,k}.$$

Damit muß also $a_{\lambda,k}$ mit dem entsprechenden Koeffizienten der rechten Seite von (5.21) übereinstimmen. Wir müssen dabei unterscheiden, ob die Startexponenten der beiden Seiten der Gleichung übereinstimmen oder nicht. Der minimale Exponent der rechten Seite kann jedoch höchstens *größer* sein als auf der linken Seite — dies wird durch die Wahl von k gewährleistet. Stimmen beide Exponenten überein, so können wir einfach den entsprechenden Koeffizienten übernehmen, anderenfalls muß $a_{\lambda,\kappa} = 0$ sein:

$$a_{\lambda,k} \quad = \quad \begin{cases} b_{\lambda,-lk} & \text{, falls} \quad m = -lk \\ 0 & \text{, falls} \quad m > -lk \end{cases} \tag{5.22}$$

Damit ist $a_{\lambda,k}$ bestimmt und durch Subtraktion von $a_{\lambda,k} \cdot j(l\tau)^k$ auf beiden Seiten der Gleichung (5.21) folgt:

$$\sum_{\kappa=0}^{k-1} a_{\lambda,\kappa} \cdot j(l\tau)^\kappa \quad = \quad \sum_{i=m'}^{\infty} b'_{\lambda,i} \cdot q_\tau^i. \tag{5.23}$$

In dieser Gleichung wird der Startexponent, mit denselben Begründungen, durch den Term $a_{\lambda,k-1} \cdot j(l\tau)^{k-1}$ definiert, und der zugehörige Leitkoeffizient $a_{\lambda,k-1}$ kann bestimmt werden. Führen wir dieses Verfahren fort, so können wir nacheinander alle Koeffizienten $a_{\lambda,k}, a_{\lambda,k-1}, \ldots a_{\lambda,0}$ berechnen. Wir erhalten somit folgenden Algorithmus für jedes λ mit $0 \leq \lambda \leq l+1$:

5.11. Algorithmus [Koeffizientenvergleich für festes λ]

(1)	$h(\tau) = k_\lambda(\tau)$	// Fourier-Reihendarstellung
(2)	**for** $(\kappa = k, \ldots, 0)$ **do**	
(3)	**if** $(-l\kappa <$ first $(h(\tau)))$ **then**	// vgl. (5.22)
(4)	$a_{\lambda,\kappa} = 0$	
(5)	**else**	
(6)	$a_{\lambda,\kappa} = \mathrm{lc}\ (h(\tau))$	
(7)	$h(\tau) = h(\tau) - a_{\lambda,\kappa} \cdot j(l\tau)^\kappa$	
(8)	**fi**	
(9)	**od**	

Wenden wir uns nun der Berechnung der $k_\lambda(\tau)$ zu. Es ist bekannt, daß die Koeffizienten eines (normierten) Polynoms berechnet werden können mit Hilfe einfacher Polynome in den jeweiligen Nullstellen – den *elementarsymmetrischen Funktionen.* Diese Funktionen werden wir im folgenden betrachten, ebenso deren Bezug zu *Potenzsummen* (vgl. dazu auch [Wa50], Seite 92).

5.12. Definition *Für einen beliebigen kommutativen Ring R mit Einselement betrachten wir den Polynomring $R[X_1, \ldots, X_n]$ in n Unbestimmten. Die Polynome*

$$\sigma_\rho(X_1, \ldots, X_n) \quad := \quad \sum_{i_1 < i_2 < \cdots < i_\rho} X_1^{i_1} \cdot X_2^{i_2} \cdots X_\rho^{i_\rho}, \qquad \text{für} \qquad \rho \leq n \tag{5.24}$$

heißen die **elementarsymmetrischen Funktionen**, *die Polynome*

$$s_\rho(X_1, \ldots, X_n) \quad := \quad \sum_{\nu=1}^{n} X_\nu^\rho \tag{5.25}$$

die **Potenzsummen** *der n Unbestimmten.*

Die Polynome σ_ρ und s_ρ sind in dem Sinne *symmetrisch*, als daß sie invariant sind unter allen Permutationen der Unbestimmten $X_1, \ldots, X_n$. Den Zusammenhang zwischen den Polynomen σ_ρ und s_ρ beschreiben die *Newton'schen Formeln* (vgl. [Wa50, S.92]: Im Polynomring in n Unbestimmten gilt für $\rho \leq n$

$$s_\rho - s_{\rho-1}\sigma_1 + s_{\rho-2}\sigma_2 - \cdots + (-1)^{\rho-1}s_1\sigma_{\rho-1} + (-1)^\rho \cdot \rho \cdot \sigma_\rho \quad = \quad 0. \tag{5.26}$$

Mit der Festlegung $\sigma_0 := 1$ erhalten wir somit

$$\rho \cdot \sigma_\rho \quad = \quad \sum_{i=1}^{\rho} (-1)^{i-1} \cdot s_i \cdot \sigma_{\rho-i}. \tag{5.27}$$

Die für uns wichtige Bedeutung der *elementarsymmetrischen Funktionen* liegt in der folgenden Aussage (siehe auch [Wa50]).

5.13. Proposition *Sei R ein kommutativer Ring mit Einselement; $\alpha_1, \ldots, \alpha_n \in R$. Seien weiter $\sigma_\nu := \sigma_\nu(\alpha_1, \ldots, \alpha_n)$ für $\nu = 1, \ldots, n$ die elementarsymmetrischen Funktionen in den $\alpha_1, \ldots, \alpha_n$. Dann gilt:*

$$\prod_{\nu=1}^{n} (X - \alpha_\nu) \quad = \quad X^n - \sigma_1 X^{n-1} + \sigma_2 X^{n-2} - \cdots + (-1)^n \sigma_n. \tag{5.28}$$

Damit sind also die Koeffizienten eines (normierten) Polynoms genau die *elementarsymmetrischen* Funktionen in dessen Nullstellen.

Diese Aussage liefert uns somit ein Verfahren, die Koeffizienten $k_\lambda(\tau)$ des Polynoms $H(X)$ aus (5.19) zu berechnen. Wegen

$$H(X) \quad = \quad \prod_{n=0}^{l} (X - f(V_n(\tau))) \tag{5.29}$$

können wir aus Gleichung (5.19) und Proposition 5.13 ablesen, daß

$$\begin{aligned} k_\lambda(\tau) &= (-1)^\lambda \cdot \sigma_{l+1-\lambda}(f(V_0(\tau)), \ldots, f(V_l(\tau))) \\ &= (-1)^\lambda \cdot \sigma_{l+1-\lambda}(\tau) \end{aligned} \tag{5.30}$$

gilt. Somit müssen wir also die Fourier-Reihenentwicklung der *elementarsymmetrischen Funktionen* in den vorgegebenen Nullstellen berechnen und in der Gleichung

$$\sum_{\kappa=0}^{k} a_{\lambda,\kappa} \cdot j(l\tau)^{\kappa} \quad = \quad (-1)^{\lambda} \cdot \sigma_{l+1-\lambda}(\tau) \tag{5.31}$$

einen Koeffizientenvergleich durchführen.

Zur Berechnung der *elementarsymmetrischen Funktkionen* in den $f(V_n(\tau))$ verwenden wir die *Newton'schen Formeln* (5.27). Dazu müssen wir zunächst die q_τ-Entwicklungen der Potenzsummen der Nullstellen $f(V_n(\tau))$ $(0 \leq n \leq l)$ berechnen. Wir erhalten mit Lemma 5.10 für $\rho \geq 0$:

$$s_\rho(\tau) \quad = \quad \underbrace{\sum_{n=0}^{l-1} f\left(\tau + \frac{n}{l}\right)^{\rho}}_{=:s_{\rho,1}} + \underbrace{\left(\frac{l^s}{f(l\tau)}\right)^{\rho}}_{=:s_{\rho,2}}. \tag{5.32}$$

Dabei ergibt sich die Fourier-Reihenentwicklung von $s_{\rho,2}$ einfach durch Potenzieren der q_τ-Entwicklung von $\left(\frac{l^s}{f(l\tau)}\right)$. Die Entwicklungen der $s_{\rho,1}$ lassen sich mit Hilfe der folgenden Berechnungen aus der Darstellung von $f(\tau)^\rho$ bestimmen (vgl. [Mü94]). Sei dazu

$$f(\tau)^{\rho} \quad = \quad q_\tau^u \cdot \left[\sum_{\mu=0}^{\infty} a_\mu \cdot q_\tau^\mu\right] \qquad \text{mit} \qquad q_\tau = \exp(2\pi i \tau).$$

Dann erhalten wir daraus für ein $n \in \{0, \ldots, l-1\}$

$$\begin{aligned} f\left(\tau + \frac{n}{l}\right)^{\rho} &= \exp\left(2\pi i \left(\tau + \frac{n}{l}\right) u\right) \cdot \left[\sum_{\mu=0}^{\infty} a_\mu \cdot \exp\left(2\pi i (\tau + \frac{n}{l})\mu\right)\right] \\ &= q_\tau^u \cdot \exp\left(\frac{2\pi i n u}{l}\right) \cdot \left[\sum_{\mu=0}^{\infty} a_\mu \cdot q_\tau^\mu \cdot \exp\left(\frac{2\pi i n \mu}{l}\right)\right] \\ &= q_\tau^u \cdot \left[\sum_{\mu=0}^{\infty} a_\mu \cdot q_\tau^\mu \cdot \exp\left(\frac{2\pi i n(\mu + u)}{l}\right)\right]. \end{aligned}$$

Bilden wir dann die Summe dieser Terme für $n = 0, \ldots, l-1$, so ergibt sich

$$s_{\rho,1}(\tau) \quad = \quad \sum_{n=0}^{l-1} f\left(\tau + \frac{n}{l}\right)^{\rho}$$

$$
\begin{aligned}
&= \sum_{n=0}^{l-1} q_\tau^u \cdot \left[\sum_{\mu=0}^{\infty} a_\mu \cdot q_\tau^\mu \cdot \exp\left(\frac{2\pi i n(\mu+u)}{l}\right)\right] \\
&= q_\tau^u \cdot \sum_{\mu=0}^{\infty} a_\mu \cdot q_\tau^\mu \cdot \underbrace{\sum_{n=0}^{l-1} \exp\left(\frac{2\pi i n(\mu+u)}{l}\right)}_{=:l_\mu}.
\end{aligned}
$$

Untersuchen wir nun noch die Werte der Ausdrücke l_μ und betrachten dazu $\frac{\mu+u}{l}$. Ist dies eine **ganze** Zahl, so ist $\exp\left(2\pi i n\frac{\mu+u}{l}\right) = 1$ für jedes n, und wir erhalten :

$$
l_\mu \quad = \quad \sum_{n=0}^{l-1} 1 \quad = \quad l, \qquad \text{falls} \qquad l \mid (\mu+u).
$$

Ist $\mu + u$ *nicht* teilbar durch l, so beachten wir, daß

$$
\exp\left(\frac{2\pi i n(\mu+u)}{l}\right)^l \quad = \quad \exp\left(2\pi i n(\mu+u)\right) \quad = \quad 1
$$

für $0 \leq n < l$. Die Werte $\exp\left(\frac{2\pi i n(\mu+u)}{l}\right)$ sind paarweise verschieden und bilden somit die Gesamtheit der komplexen Nullstellen des Polyoms $X^l - 1$. Wie wir bereits in Proposition 5.13 festgestellt haben, ist dann die Summe dieser Nullstellen gerade gleich dem Koeffizienten von X^{l-1} dieses Polynoms, also gleich 0. Damit gilt:

$$
l_\mu \quad = \quad \sum_{n=0}^{l-1} \exp\left(\frac{2\pi i n(\mu+u)}{l}\right) \quad = \quad 0, \qquad \text{falls} \qquad l \not| (\mu+u).
$$

Zusammenfassend sehen wir also, daß wir die Koeffizienten der $s_{\rho,1}$ „ausschneiden“ können aus der Darstellung von $f(\tau)^\rho$; wir erhalten das folgende Lemma:

5.14. Lemma *Sei die Funktion $f(\tau)$ wie in (5.3) definiert, und sei die q_τ-Entwicklung von $f(\tau)^\rho$ gegeben durch*

$$
f(\tau)^\rho = q_\tau^u \cdot \left[\sum_{\mu=0}^{\infty} a_\mu \cdot q_\tau^\mu\right].
$$

Dann gilt für $s_{\rho,1}(\tau)$ aus (5.32):

$$
s_{\rho,1}(\tau) = q_\tau^u \cdot \left[\sum_{\mu=0}^{\infty} b_\mu \cdot q_\tau^\mu\right] \tag{5.33}
$$

wobei für die Koeffizienten b_μ gilt:

$$b_\mu = \begin{cases} l \cdot a_\mu & , \quad \textit{falls} \quad l \mid (\mu + u) \\ 0 & , \quad \textit{falls} \quad l \nmid (\mu + u) \end{cases} \tag{5.34}$$

Damit können wir nun die rechte Seite der Gleichung (5.31) berechnen. Um den Vergleich der Koeffizienten durchführen zu können, müssen wir lediglich noch den maximalen Grad k bestimmen, in dessen Potenz die Funktion $j(l\tau)$ in (5.31) auftreten kann; dies werden wir im folgenden Abschnitt beschreiben. Ist dieser Wert bekannt, so können wir einen Algorithmus zur Berechnung der gesuchten $a_{\lambda,\kappa}$ angeben.

Algorithmus zur Berechnung des Polynoms für $\Gamma_0(l)$

EINGABE : Fourier-Reihenentwicklung der Funktionen $f(\tau)$ und $j(l\tau)$
AUSGABE : Feld der Koeffizenten $a_{\lambda,\kappa}$ für $0 \leq \lambda \leq l+1$ und $0 \leq \kappa \leq k$

LEITKOEFFIZIENT 1 SETZEN

for $(1 \leq \kappa \leq k)$ **do**
  $a_{l+1,\kappa} = 0$
od
$a_{l+1,0} = 1$

INITIALISIERUNG

$\sigma_0(\tau) = 1$
$h_1(\tau) = h_2(\tau) = 1$
for $(\lambda = 1 \ldots l+1)$ **do** //Berechnung der $s_\lambda(\tau)$
  $h_1(\tau) = h_1(\tau) \cdot f(\tau)$ // $h_1(\tau) = f(\tau)^\lambda$
  $h_2(\tau) = h_2(\tau) \cdot \left(\frac{l^s}{f(l\tau)}\right)$ // $h_2(\tau) = \left(\frac{l^s}{f(l\tau)}\right)^\lambda$
  Berechne $s_{\lambda,1}(\tau)$ aus $h_1(\tau)$ mit (5.34)
  $s_\lambda(\tau) = s_{\lambda,1}(\tau) + h_2(\tau)$
od

Algorithmus zur Berechnung des Polynoms für $\Gamma_0(l)$
(Fortsetzung)

KOEFFIZIENTENVERGLEICH

for $(\lambda = 0, \ldots, l)$ **do**
    $h_1(\tau) = (-1)^\lambda \cdot \sigma_{l+1-\lambda}(\tau)$ // verwende (5.27)
    for $(\kappa = k, \ldots, 0)$ **do**
        if $(-l\kappa <$ first $(h_1(\tau)))$ **then**
            $a_{\lambda,\kappa} = 0$
        else
            $a_{\lambda,\kappa} = \mathrm{lc}\ (h_1(\tau))$
            $h_1(\tau) = h_1(\tau) - a_{\lambda,\kappa} \cdot j(l\tau)^\kappa$
        fi
    od
od

Dieser Algorithmus liefert die Koeffizienten eines äquivalenten Polynoms zunächst als ganze Zahlen. Für die Anwendung, den Algorithmus von Atkin, benötigen wir das Polynom jedoch über dem Körper $\mathbb{F}_p$. Bei der Beschreibung der zum modularen Polynome äquivalenten Polynome haben wir festgestellt, daß wir dies durch Reduktion der Koeffizienten modulo p erreichen können. Somit ergeben sich für die Praxis zwei verschiedene Möglichkeiten, um ein Polynom über $\mathbb{F}_p$ zu berechnen.

1. Wir berechnen für eine beliebige Charakteristik p die Koeffizienten des gesuchten Polynoms über dem Körper $\mathbb{F}_p$, d.h. daß alle im Algorithmus auftretenden Operationen in $\mathbb{F}_p$ durchgeführt werden.

2. Wir berechnen die Koeffizienten des Polynoms über $\mathbb{Z}$, und speichern sie in einer Datei. Dann müssen diese zur Verwendung in einem speziellen Körper $\mathbb{F}_p$ lediglich eingelesen und modulo p reduziert werden.

Für die Implementierung des Algorithmus haben wir uns dazu entschieden, die zweite Alternative zu realisieren. Damit muß jedes Polynom genau einmal berechnet werden und kann dann für jeden beliebigen endlichen Primkörper verwendet werden.

5.3 Ein Beispiel

Zur Veranschaulichung des Algorithmus zeigen wir hier anhand eines konkreten Beispiels die einzelnen Rechenschritte auf. Wir berechnen die Koeffizienten des Polynoms für $l = 5$ modulo 101. Für die Größen s und v aus (5.3) und (5.4) erhalten wir

$$s = 3 \qquad \text{und} \qquad v = 1.$$

Dazu berechnen wir zunächst die Dedekind'sche η-Funktion und erhalten:

$$\begin{array}{rcl}
\eta(\tau) & \approx & q_\tau^{\frac{1}{24}} \cdot (\ 1 \cdot q_\tau^0 + 100 \cdot q_\tau^1 + 100 \cdot q_\tau^2 + 0 \cdot q_\tau^3 + 0 \cdot q_\tau^4 + 1 \cdot q_\tau^5 + 0 \cdot q_\tau^6\ ) \\
\eta(5\tau) & \approx & q_\tau^{\frac{5}{24}} \cdot (\ 1 \cdot q_\tau^0 + 0 \cdot q_\tau^1 + 0 \cdot q_\tau^2 + 0 \cdot q_\tau^3 + 0 \cdot q_\tau^4 + 100 \cdot q_\tau^5 + 0 \cdot q_\tau^6\ ) \\
\eta(5\tau)^{-1} & \approx & q_\tau^{-\frac{5}{24}} \cdot (\ 1 \cdot q_\tau^0 + 1 \cdot q_\tau^1 + 0 \cdot q_\tau^2 + 0 \cdot q_\tau^3 + 0 \cdot q_\tau^4 + 1 \cdot q_\tau^5 + 0 \cdot q_\tau^6\ )
\end{array}$$

Dann müssen wir die Funktion

$$f(\tau) = \left(\frac{\eta(\tau)}{\eta(5\tau)}\right)^6$$

und deren Potenzen berechnen. Es gilt dann:

$$\begin{array}{rl}
f(\tau) \approx & 1 \cdot q_\tau^{-1} + 95 \cdot q_\tau^0 + 9 \cdot q_\tau^1 + 10 \cdot q_\tau^2 + 71 \cdot q_\tau^3 + 6 \cdot q_\tau^4 + 76 \cdot q_\tau^5 \\
f(\tau)^2 \approx & 1 \cdot q_\tau^{-2} + 89 \cdot q_\tau^{-1} + 54 \cdot q_\tau^0 + 13 \cdot q_\tau^1 + 2 \cdot q_\tau^2 + 47 \cdot q_\tau^3 + 44 \cdot q_\tau^4 \\
\\
f(\tau)^5 \approx & 1 \cdot q_\tau^{-5} + 71 \cdot q_\tau^{-4} + 1 \cdot q_\tau^{-3} + 42 \cdot q_\tau^{-2} + 5 \cdot q_\tau^{-1} + 40 \cdot q_\tau^0 + 51 \cdot q_\tau^1 \\
f(\tau)^6 \approx & 1 \cdot q_\tau^{-6} + 65 \cdot q_\tau^{-5} + 89 \cdot q_\tau^{-4} + 79 \cdot q_\tau^{-3} + 38 \cdot q_\tau^{-2} + 92 \cdot q_\tau^1 + 41 \cdot q_\tau^0
\end{array}$$

Berechnen wir daraus die Potenzsummen $s_\rho(\tau)$ mit Hilfe der Formel (5.32) und Lemma 5.14, so ergeben sich diese zu

$$\begin{array}{rl}
s_1(\tau) \approx & 71 \cdot q_\tau^0 + 0 \cdot q_\tau^1 + \ldots + 0 \cdot q_\tau^5 \\
s_2(\tau) \approx & 68 \cdot q_\tau^0 + 0 \cdot q_\tau^1 + \ldots + 0 \cdot q_\tau^4 \\
\\
s_5(\tau) \approx & 5 \cdot q_\tau^{-5} + 0 \cdot q_\tau^{-4} + \ldots \quad 99 \cdot q_\tau^0 + 0 \cdot q_\tau^1 \\
s_6(\tau) \approx & 22 \cdot q_\tau^{-5} + 0 \cdot q_\tau^{-4} + \ldots \quad 3 \cdot q_\tau^0.
\end{array}$$

Mit Hilfe dieser Vorberechnungen kann nun der Koeffizientenvergleich durchgeführt werden. Wir benötigen dazu noch die Approximation der Funktion $j(5\tau)$ und erhalten :

$$\begin{aligned} j(5\tau) &\approx 1 \cdot q_\tau^{-5} + 0 \cdot q_\tau^{-4} + 0 \cdot q_\tau^{-3} + 0 \cdot q_\tau^{-2} + 0 \cdot q_\tau^{-1} + 37 \cdot q_\tau^{0} \\ &\approx 1 \cdot q_\tau^{-5} + 37 \cdot q_\tau^{0}. \end{aligned}$$

Zu Beginn des Vergleichs können wir die bereits bekannten Koeffizienten initialisieren

$$a_{6,0} = 1 \qquad \text{und} \qquad a_{6,1} = 0,$$

und beobachten dann die eigentliche Berechnung. Die approximierten Fourier-Reihen auf den rechten Seiten der folgenden Gleichungen stellen die jeweiligen elementarsymmetrischen Funktionen dar.

$$a_{5,0} \cdot 1 + a_{5,1} \cdot j(5\tau) = -71 \cdot q_\tau^{0} = 30 \cdot q_\tau^{0}$$

$$\boxed{\Longrightarrow \qquad a_{5,0} = 30 \qquad \textbf{und} \qquad a_{5,1} = 0}$$

$$a_{4,0} \cdot 1 + a_{4,1} \cdot j(5\tau) = 12 \cdot q_\tau^{0}$$

$$\boxed{\Longrightarrow \qquad a_{4,0} = 12 \qquad \textbf{und} \qquad a_{4,1} = 0}$$

Weiterhin erhalten wir

$$\begin{aligned} a_{3,0} &= 88 & a_{3,1} &= 0 \\ a_{2,0} &= 60 & a_{2,1} &= 0. \end{aligned}$$

Und schließlich

$$\begin{aligned} a_{1,0} \cdot 1 + a_{1,1} \cdot j(5\tau) &= -(1 \cdot q_\tau^{-5} + 95 \cdot q_\tau^{0}) \\ &= 100 \cdot q_\tau^{-5} + 6 \cdot q_\tau^{0} \end{aligned}$$

$$\boxed{\Longrightarrow \qquad a_{1,1} \quad = \quad 100}$$

Subtrahieren wir $100 \cdot j(5\tau)$ von beiden Seiten der Gleichung, so können wir auch $a_{1,0}$ bestimmen.

$$\begin{aligned} a_{1,0} \cdot 1 &= (100 \cdot q_\tau^{-5} + 6 \cdot q_\tau^0) - (100 \cdot q_\tau^{-5} + 64 \cdot q_\tau^0) \\ &= 43 \cdot q_\tau^0. \end{aligned}$$

$$\boxed{\Longrightarrow \qquad a_{0,0} \quad = \quad 43}$$

$$a_{0,0} \cdot 1 \quad + \quad a_{0,1} \cdot j(5\tau) \quad = \quad 24 \cdot q_\tau^0$$

$$\boxed{\Longrightarrow \qquad a_{0,0} \quad = \quad 24 \qquad \textbf{und} \qquad a_{0,1} \quad = \quad 0}$$

Damit erhalten wir modulo 101 die Koeffizienten:

$$\begin{array}{rcrrcr} a_{0,0} &=& 24 & a_{0,1} &=& 0 \\ a_{1,0} &=& 43 & a_{1,1} &=& 100 \\ a_{2,0} &=& 60 & a_{2,1} &=& 0 \\ a_{3,0} &=& 88 & a_{3,1} &=& 0 \\ a_{4,0} &=& 12 & a_{4,1} &=& 0 \\ a_{5,0} &=& 30 & a_{5,1} &=& 0 \\ a_{6,0} &=& 1 & a_{6,1} &=& 0 \end{array}$$

Führen wir die Berechnungen modulo mehrerer Primzahlen durch und kombinieren die Ergebnisse, so erhalten wir über $\mathbb{Z}$ die Koeffizienten:

$$\begin{array}{rcrrcr} a_{0,0} &=& 125 & a_{0,1} &=& 0 \\ a_{1,0} &=& 750 & a_{1,1} &=& -1 \\ a_{2,0} &=& 1575 & a_{2,1} &=& 0 \\ a_{3,0} &=& 1300 & a_{3,1} &=& 0 \\ a_{4,0} &=& 315 & a_{4,1} &=& 0 \\ a_{5,0} &=& 30 & a_{5,1} &=& 0 \\ a_{6,0} &=& 1 & a_{6,1} &=& 0 \end{array}$$

5.4 Beschreibung der Implementierung

Im vorigen Abschnitt haben wir beschrieben, wie wir eine Funktion mit Invarianzgruppe $\Gamma_0(l)$ berechnen können und wie es damit möglich ist, ein zum l-ten *modularen* Polynom äquivalentes Polynom zu berechnen. Dabei haben wir sämtliche auftretenden Funktionen repräsentiert durch (unendliche) Fourier-Reihenentwicklungen in $q_\tau = \exp(2\pi i\tau)$. In einer Implementierung des angegebenen Verfahrens werden wir daher nur Approximationen der verwendeten Funktionen zur Verfügung haben. Hier werden wir nun zunächst untersuchen, mit welcher Präzision die auftretenden Größen berechnet werden müssen, um einen korrekten Ablauf des Algorithmus zu gewährleisten. Danach geben wir an, wie wir durch geschicktere Berechnung der elementarsymmetrischen Funktionen eine Beschleunigung des Verfahrens erzielen können.

5.4.1 Präzision der Fourier-Reihen

Um den Koeffizientenvergleich (5.21) korrekt durchführen zu können, müssen wir gewährleisten, daß die Koeffizienten $k_\lambda(\tau)$ des Polynoms $H(X)$ „hinreichend“ gut approximiert werden, d.h. daß genügend viele Koeffizienten in deren Fourier-Reihenentwicklung bekannt sind. Dies bedeutet somit, daß die elementarsymmetrischen Funktionen in den Nullstellen $f(V_n(\tau))$ bis zu einem gewissen Exponenten von q_τ bekannt sein müssen.

Da die elementarsymmetrischen Funktionen mit Hilfe der Funktion $f(\tau)$ berechnet werden, untersuchen wir nun, inwieweit die Präzision der $\sigma_\lambda(\tau)$ von der Approximation der Funktion $f(\tau)$ abhängig ist. Dazu werden wir in den folgenden Sätzen die Startexponenten sowie die maximalen bekannten Exponenten dieser Funktionen betrachten:

5.15. Satz *Seien $f(\tau)$ die in (5.5) definierte Funktion und $\sigma_\lambda(\tau)$ (resp. $s_\lambda(\tau)$) für $1 \leq \lambda \leq l+1$ die elementarsymmetrischen Funktionen (resp. Potenzsummen) in den $f(V_0(\tau)), \ldots, f(V_l(\tau))$ gegeben durch die jeweiligen* **approximierten** *Fourier-Reihenentwicklungen in q_τ.*

Dann gilt für alle λ mit $1 \leq \lambda \leq l+1$:

$$\mathit{first}\ (s_\lambda(\tau)) \geq -\lambda v \tag{5.35}$$
$$\mathit{first}\ (\sigma_\lambda(\tau)) \geq -\lambda v \tag{5.36}$$

Beweis: Sei λ beliebig mit $1 \leq \lambda \leq l+1$. Aus (5.32) können wir ablesen:

$$\text{first}\ (s_\lambda(\tau)) = \text{first}\ (s_{\lambda,1}(\tau) + s_{\lambda,2}(\tau))$$

$$
\begin{aligned}
&= \min\left(\text{first }(s_{\lambda,1}(\tau)), \text{first }\left(\left(\frac{l^s}{f(l\tau)}\right)^{\lambda}\right)\right) \\
&= \min\left(\text{first }(s_{\lambda,1}(\tau)), \lambda \cdot lv\right) \\
&= \text{first }(s_{\lambda,1}(\tau)) \\
&\geq \text{first }(f(\tau)^{\lambda}) \quad = \quad -\lambda v.
\end{aligned}
$$

Damit ist die erste Aussage bewiesen. Den Beweis für (5.36) führen wir mit Induktion über λ. Dabei folgt die Behauptung für $\lambda = 1$ wegen $\sigma_1 = s_1$ sofort. Sei nun also $1 < \lambda < l+1$. Dann erhalten wir mit Hilfe der *Newton'schen Formeln* (5.27):

$$
\begin{aligned}
\text{first }(\sigma_\lambda(\tau)) &= \text{first }\left(\sum_{i=1}^{\lambda}(-1)^{i-1} \cdot s_i \cdot \sigma_{\lambda-i}\right) \\
&= \min\ \{\text{first }(s_i \cdot \sigma_{\lambda-i}) \mid i = 1, \ldots, \lambda\}.
\end{aligned}
$$

Dabei gilt für jedes i mit $1 \leq i \leq \lambda$:

$$
\begin{aligned}
\text{first }(s_i \cdot \sigma_{\lambda-i}) &= \text{first }(s_i) + \text{first }(\sigma_{\lambda-i}) \\
&\underset{I.V.}{\geq} -iv \ + \ (-(\lambda-i)\cdot v) \ = \ -\lambda v.
\end{aligned}
$$

Somit überträgt sich diese Abschätzung auch auf das Minimum dieser Startexponenten, und die Ungleichung ist bewiesen. ∎

Nachdem die Startexponenten der $\sigma_\lambda(\tau)$ bestimmt sind, betrachten wir nun die Präzisionen dieser Reihen in Abhängigkeit von $f(\tau)$ und erhalten:

5.16. Satz *Sei die Entwicklung der Funktion $f(\tau)$ bis zu dem maximalen Exponenten $M_f \geq -v$ berechnet:*

$$
f(\tau) \approx \sum_{n=-v}^{M_f} a_n \cdot q_\tau^n
$$

Dann gilt für alle λ mit $1 \leq \lambda \leq l+1$:

$$
\begin{aligned}
\textit{last }(s_\lambda(\tau)) &= M_f - (\lambda-1)\cdot v && (5.37) \\
\textit{last }(\sigma_\lambda(\tau)) &= \textit{last }(s_\lambda(\tau)) && (5.38)
\end{aligned}
$$

Beweis: Zum Beweis von (5.37) betrachten wir:

$$\begin{aligned}
\text{last }(s_\lambda(\tau)) &= \text{last }(s_{\lambda,1}(\tau)+s_{\lambda,2}(\tau)) \\
&= \min\ (\text{last }(s_{\lambda,1}(\tau)), \text{last }(s_{\lambda,2}(\tau))) \\
&= \min\ \left(\text{last }(f(\tau)^\lambda), \text{last }\left(\left(\frac{l^s}{f(l\tau)}\right)^\lambda\right)\right) \\
&= \min\ (M_f-(\lambda-1)\cdot v, l\cdot M_f+2lv) \\
&= M_f-(\lambda-1)\cdot v.
\end{aligned}$$

Die Aussage (5.38) zeigen wir, analog zum Beweis des vorigen Satzes, durch Induktion über λ. Wiederum ist der Induktionsanfang für $\lambda=1$ trivial. Sei also $1<\lambda<l+1$. Dann ist

$$\begin{aligned}
\text{last }(\sigma_\lambda(\tau)) &= \text{last }\left(\sum_{i=1}^{\lambda}(-1)^{i-1}\cdot s_i\cdot\sigma_{\lambda-i}\right) \\
&= \min\ \{\text{last }(s_i\cdot\sigma_{\lambda-i})) \mid i=1,\ldots,\lambda\}.
\end{aligned}$$

Für jedes i mit $1\le i\le\lambda-1$ gilt:

$$\begin{aligned}
\text{last }(s_i\cdot\sigma_{\lambda-i}) &= \min\ (\text{first }(s_i)+\text{last }(\sigma_{\lambda-i}), \text{first }(\sigma_{\lambda-i})+\text{last }(s_i)) \\
&\underset{I.V.}{=} \min\ (\text{first }(s_i)+\text{last }(s_{\lambda-i}), \text{first }(\sigma_{\lambda-i})+\text{last }(s_i)) \\
&\ge \min\ (-iv+M_f-(\lambda-i-1)v, -(\lambda-i)v+M_f-(i-1)v) \\
&= M_f-(\lambda-1)\cdot v,
\end{aligned}$$

und für $i=\lambda$ ist

$$\text{last }(s_\lambda\cdot\sigma_0)=\text{last }(s_\lambda)=M_f-(\lambda-1)\cdot v.$$

Fassen wir dies zusammen, so erhalten wir :

$$\text{last }(\sigma_\lambda(\tau)) \quad = \quad \min\ \{\ \text{last }(s_i\cdot\sigma_{\lambda-i}) \mid 1\le i\le\lambda\ \} \quad = \quad \text{last }(s_\lambda) \qquad \blacksquare$$

Mit Hilfe von Satz (5.15) sind wir nun in der Lage, eine Struktur in den Koeffizienten $a_{\lambda,\kappa}$ zu erkennen. Wir können damit feststellen, welche dieser Werte immer gleich 0 sein müssen. Es gilt das folgende Lemma:

5.17. Lemma *Sei s wie in (5.3) und v wie in (5.4) definiert. Sei weiter $t=\frac{12}{s}=\frac{l-1}{v}$.*

Dann gilt mit $j\in\mathbb{Z}_{\ge 0}$ für alle λ mit $2+j\cdot t\le\lambda\le l+1$

$$a_{\lambda,\kappa} \quad = \quad 0, \qquad \textit{für alle} \quad \kappa\ge v-j.$$

Beweis: Sei $j \in \mathbb{Z}_{\geq 0}$ beliebig gewählt, so daß $\lambda_0 := 2 + j \cdot t \leq l + 1$ noch erfüllt ist. Zum Beweis des Lemmas zeigen wir, daß der maximale Exponent k von $j(l\tau)$ in der Gleichung

$$\sum_{\kappa=0}^{k} a_{\lambda_0,\kappa} \cdot j(l\tau)^{\kappa} = (-1)^{\lambda_0} \cdot \sigma_{l+1-\lambda_0}(\tau)$$

kleiner als $(v-j)-1$ gewählt werden kann. Wir beachten dabei, daß der Koeffizient $a_{\lambda_0,k}$ gleich null sein muß, falls der Startexponent der linken Seite dieser Gleichung kleiner ist als der Startexponent der rechten Seite. Wenden wir Satz 5.15 an, so erhalten wir für jedes λ mit $1 \leq \lambda \leq l+1$

$$\text{first}\left(((-1)^{\lambda} \cdot \sigma_{l+1-\lambda}(\tau)\right) \geq -(l+1-\lambda)v.$$

Der minimale Exponent der linken Seite ist

$$\text{first}\left(\sum_{\kappa=0}^{k} a_{\lambda,\kappa} \cdot j(l\tau)^{\kappa}\right) = \min\ \{ -\kappa \cdot l \mid 0 \leq \kappa \leq k \} = -kl.$$

Also gilt $a_{\lambda,k} = 0$, falls bereits

$$\begin{array}{cccc} & -kl & < & -(l+1-\lambda) \cdot v \\ \Leftrightarrow & k & > & \left(\dfrac{l+1-\lambda}{l}\right) \cdot v. \end{array}$$

Für λ_0 erhalten wir somit

$$\begin{aligned} \left(\frac{l+1-\lambda_0}{l}\right) v &= \left(\frac{l+1-2-jt}{l}\right) v \\ &= \left(\frac{l-1}{l}\right) v - \left(\frac{jt}{l}\right) v \\ &= \left(\frac{l-1}{l}\right) v - j \left(\frac{\frac{l-1}{v}}{l}\right) v \\ &= \left(\frac{l-1}{l}\right) v - j \left(\frac{l-1}{l}\right) \\ &= \underbrace{\left(\frac{l-1}{l}\right)}_{<1} \cdot (v-j)\,. \end{aligned}$$

Damit kann für alle $\lambda \geq \lambda_0$ $k \in \mathbb{Z}$ also größer oder gleich $(v-j)-1$ gewählt werden. ∎

Speziell folgt aus diesem Lemma das folgende Korollar.

5.18. Korollar *Die Funktion $j(l\tau)$ tritt im Koeffizientenvergleich in höchstens v-ter Potenz auf. D.h. in Gleichung (5.21) gilt $k = v$.*

Wir erhalten also für den Koeffizientenvergleich

$$\sum_{\kappa=0}^{v} a_{\lambda,\kappa} \cdot j(l\tau)^{\kappa} \quad = \quad (-1)^{\lambda} \cdot \sigma_{l+1-\lambda}(\tau)$$

für $0 \leq \lambda \leq l+1$. Bei der Herleitung des Algorithmus 5.11 haben wir festgestellt, daß wir für jedes λ die zugehörigen Koeffizienten $a_{\lambda,\kappa}$ in der Reihenfolge $a_{\lambda,v}, a_{\lambda,v-1}, \ldots, a_{\lambda,0}$ berechnen müssen. Somit benötigen wir nacheinander die Koeffizienten zu den Exponenten $-vl, -(v-1)l, \ldots, -l$ und 0 in den jeweiligen Fourier-Reihen. Daher müssen sowohl alle Potenzen $j(l\tau)^{\kappa}$ $(0 \leq \kappa \leq v)$ als auch die elementarsymmetrischen Funktionen σ_{λ} $(1 \leq \lambda \leq l)$ bis zum Exponenten 0 approximiert werden. Dies können wir dadurch gewährleisten, daß wir von der Entwicklung von $j(l\tau)$ alle Koeffizienten bis zum Exponenten $M_{jl} := (v-1) \cdot l$, und die Funktionen $f(\tau)$ bis zum Exponenten $M_f = l \cdot v$ berechen. Dann gilt nämlich wegen Korollar 3.14 und 5.16:

$$\text{last } (j(l\tau)^{\kappa}) = M_{jl} - (\kappa - 1) \cdot l = (v-1) \cdot l - (\kappa - 1) \cdot l = \underbrace{(v - \kappa)}_{\geq 0} \cdot l \geq 0$$

$$\text{last } (\sigma_{\lambda}(\tau)) = M_f - (\lambda - 1) \cdot v = l \cdot v - (\lambda - 1) \cdot v = \underbrace{(l + 1 - \lambda)}_{\geq 0} \cdot v \geq 0.$$

Approximieren wir also die q_τ-Entwicklungen von $f(\tau)$ und $j(l\tau)$ bis zu den Summanden $q_\tau^{M_f}$ bzw. $q_\tau^{M_{jl}}$, so kann der Koeffizientenvergleich korrekt durchgeführt werden.

Da $f(\tau)$ und $j(l\tau)$ mit Hilfe der *Dedekind'schen η-Funktion* berechnet werden, untersuchen wir abschließend, wie genau $\eta(\tau)$ berechnet werden muß, um die gewünschten Werte für M_f und M_{jl} zu erhalten. Dabei beachten wir, daß wegen $q_{l\tau} = q_\tau^l$

$$M_{jl} \quad = \quad \text{last } (j(l\tau)) \quad = \quad l \cdot \underbrace{\text{last } (j(\tau))}_{=:M_j}$$

gilt. Wir kennen also $j(l\tau)$ bis zum Exponenten M_{jl}, wenn wir $j(\tau)$ bis zum Exponenten $M_j = \left\lfloor \frac{M_{jl}}{l} \right\rfloor$ berechnet haben.

5.19. Satz *Seien die Funktionen $f(\tau)$ und $j(\tau)$ wie oben definiert. Sei weiter $\eta(\tau)$ gegeben durch die Approximation*

$$\eta(\tau) \quad \approx \quad q_\tau^{1/24} \cdot \sum_{\mu=0}^{M_\eta} e_\mu \cdot q_\tau^{\mu}.$$

Dann erhalten wir :

$$\begin{aligned} last\ (f(\tau)) &= M_\eta - v & (5.39)\\ last\ (j(\tau)) &= M_\eta - 1 & (5.40)\end{aligned}$$

Beweis: Aus der Entwicklung von $\eta(\tau)$ erhalten wir

$$\eta(l\tau) \approx q_\tau^{l/24} \cdot \sum_{\mu=0}^{l\cdot M_\eta} f_\mu \cdot q_\tau^\mu,$$

und damit

$$\eta(l\tau)^{-1} \approx q_\tau^{-(l/24)} \cdot \sum_{\mu=0}^{l\cdot M_\eta} \tilde{f}_\mu \cdot q_\tau^\mu.$$

Bilden wir das Produkt dieser beiden Reihen, so gilt

$$\eta(\tau)\cdot\eta(l\tau)^{-1} \approx q_\tau^{(1-l/24)} \cdot \sum_{\mu=0}^{M_\eta} d_\mu \cdot q_\tau^\mu,$$

und wir erhalten für $f(\tau)$

$$\begin{aligned} f(\tau) &= \left(\eta(\tau)\cdot\eta(l\tau)^{-1}\right)^{2s}\\ &\approx q_\tau^{-v} \cdot \sum_{\mu=0}^{M_\eta} a_\mu \cdot q_\tau^\mu\\ &= \sum_{\mu=-v}^{M_\eta - v} a_{\mu+v} \cdot q_\tau^\mu.\end{aligned}$$

Der höchste auftretende Exponent ist somit last $(f(\tau)) = M_\eta - v$. Damit ist die erste Aussage des Satzes bewiesen. Mit denselben Berechnungen folgt für die *2. Weber'sche Funktion* (5.10):

$$\begin{aligned} f_2(\tau) &= \sqrt{2}\cdot\eta(2\tau)\cdot\eta(\tau)^{-1} \approx q_\tau^{1/24} \cdot \sum_{\mu=0}^{M_\eta} w_\mu \cdot q_\tau^\mu,\\ f_2^{24}(\tau) &\approx q_\tau^{1} \cdot \sum_{\mu=0}^{M_\eta} u_\mu \cdot q_\tau^\mu\\ \text{und}\qquad f_2^{-24}(\tau) &\approx q_\tau^{-1} \cdot \sum_{\mu=0}^{M_\eta} \tilde{u}_\mu \cdot q_\tau^\mu.\end{aligned}$$

Fassen wir dies mit (5.11) zusammen, so erhalten wir

$$\begin{aligned} j(\tau) &= (f_2^{24}(\tau)+16)^3 \cdot f_2^{-24}(\tau) \\ &\approx \left(16 + q_\tau^1 \cdot \sum_{\mu=0}^{M_\eta} u_\mu \cdot q_\tau^\mu\right)^3 \cdot \left(q_\tau^{-1} \cdot \sum_{\mu=0}^{M_\eta} \tilde{u_\mu} \cdot q_\tau^\mu\right) \\ &= q_\tau^{-1} \cdot \sum_{\mu=0}^{M_\eta} b_\mu \cdot q_\tau^\mu \end{aligned}$$

Auch hier können wir ablesen, daß last $(j(\tau)) = M_\eta - 1$. ■

Mit Hilfe des letzten Satzes haben wir nun alle Informationen zur Verfügung, um die benötigten Fourier-Reihen hinreichend genau zu approximieren. Wir konnten zuvor bereits aussagen, daß die q_τ-Entwicklung für $f(\tau)$ bis zum Exponenten $l \cdot v$ berechnet werden muß. Wenden wir die soeben bewiesenen Aussagen an, so wissen wir, daß dazu die *Dedekind'sche η-Funktion* bis $M_\eta = lv + v$ entwickelt werden muß; zur Berechnung von $j(l\tau)$ kann $M_\eta = (v-1)+1$ gewählt werden.

5.4.2 Berechnung der Polynome über $\mathbb{Z}$

Berechnen wir die Polynome über den ganzen Zahlen, so tritt das Problem auf, daß die auftretenden Koeffizienten sehr groß werden. Daher bestimmen wir die Koeffizienten zunächst modulo „vieler“ kleiner Primzahlen. Der Chinesische Restsatz ermöglicht es uns danach, die berechneten Teilergebnisse zu kombinieren, so daß wir die Koeffizienten des gesuchten Polynoms dann modulo dem Produkt all dieser Primzahlen kennen. Ist dieses Produkt hinreichend groß, so kennen wir die berechneten Werte sogar korrekt. Wollen wir ein Polynom für l berechnen, so müssen wir bei der Auswahl der Primzahlen q_i beachten, daß zur Berechnung der Newton'schen Formeln (5.27) stets durch alle natürlichen Zahlen kleiner als l dividiert werden muß (vgl. Seite 71). Damit wir modulo q_i nicht durch 0 dividieren, müssen wir diese größer als l wählen.

Man kann thoeretisch nachrechnen, wie viele „kleine“ Primzahlen man in diesem Verfahren höchstens verwenden muß, um durch Anwendung des Chinesischen Restsatzes mit Sicherheit die über $\mathbb{Z}$ richtigen Koeffizienten zu erhalten. Diese Schranken sind jedoch sehr groß. Daher verwenden wir in der Praxis das folgende (heuristische) Verfahren:

- Wir berechnen zunächst die Koeffizienten modulo einer „kleinen“ Primzahl q_1.
- Danach führen wir den Algorithmus modulo anderer Primzahlen $(q_2, q_3, \ldots)$ durch und kombinieren nach jeder Berechnung die neuen Ergebnisse mit den zuvor bereits bekannten Werten.

- Wir vergleichen dann, ob sich die neu entstandenen Koeffizienten gegenüber den zuletzt berechneten verändert haben oder nicht.
- Verändern sich die Koeffizienten in zwei aufeinanderfolgenden Iterationen nicht, so nehmen wir an, daß diese bereits korrekt vorliegen.

Um die Korrektheit der so berechneten Polynome zu testen, verwenden wir diese im Algorithmus von Atkin. Dabei überprüfen wir dessen Ergebnisse für eine Reihe von Testeingaben.

5.4.3 Beschleunigung des Algorithmus

Bisher haben wir beschrieben, wie man ein zum modularen Polynom äquivalentes Polynom berechnen kann. Die kostenintensivsten Berechnungen stellen dabei die Multiplikationen von Fourier-Reihen dar. In diesem Abschnitt werden wir beschreiben, wie die Anzahl der nötigen Fourier-Reihen-Multiplikationen reduziert werden kann.

Eine erste Idee für eine Beschleunigung des Algorithmus liefern uns die Betrachtungen über die Präzisionen der verwendeten Fourier-Reihen. Wir haben festgestellt, daß es genügt, die q_τ-Entwicklungen der Funktionen $\sigma_\lambda(\tau)$ bis zum Term q_τ^0 zu berechnen. Aus den Beweisen des vorigen Abschnitts können wir ersehen, daß dazu die q_τ-Entwicklung der Funktion $s_1(\tau)$ bis zum Exponenten $l \cdot v$ bekannt sein muß, und daß die höchsten bekannten Exponenten aller anderen Potenzsummen kleiner sind als $l \cdot v$. Für $s_{\lambda,2}(\tau)$ in (5.32) erhalten wir aber bereits:

$$\text{first }(s_{\lambda,2}(\tau)) \;=\; \text{first }\left(\left(\frac{l^s}{f(l\tau)}\right)^\lambda\right) \;=\; -\lambda\cdot\text{first }(f(l\tau)) \;=\; \lambda\cdot l\cdot v.$$

Für $\lambda > 1$ ist also der Startexponent von $s_{\lambda,2}(\tau)$ echt größer als $l \cdot v$, und die Koeffizienten von $s_\lambda(\tau)$, die zu Exponenten kleiner oder gleich lv gehören, werden nur durch $s_{\lambda,1}(\tau)$ bestimmt. Damit müssen $s_{\lambda,2}(\tau)$ für $\lambda > 1$ *nicht* berechnet werden.

Verwendung von Hilfstabellen

Eine weitere kostenintensive Berechnung innerhalb des Algorithmus ist die Berechnung der elementarsymmetrischen Funkitonen $\sigma_\lambda(\tau)$ mit Hilfe der Newton'schen Formeln. Betrachten wir die Gleichung (5.27), so erkennen wir, daß λ viele Multiplikationen benötigt werden, um $\sigma_\lambda(\tau)$ zu bestimmen (nämlich $s_i \cdot \sigma_{\lambda-i}$ für $1 \le i \le \lambda$). Insgesamt müssen also für $\lambda = 1, \ldots, l+1$ also $\frac{1}{2}(l^2 + l)$ viele solcher Produkte berechnet werden. Wir werden nun zeigen, wie diese Anzahl reduziert werden kann.

Betrachten wir dazu nochmals die *Newton'schen Formeln* und die Gleichung für den Koeffizientenvergleich. Sei dazu $1 \leq \lambda \leq l$. Dann gilt:

$$\begin{aligned}
\sum_{\kappa=0}^{v} a_{\lambda,\kappa} \cdot j(l\tau)^{\kappa} &= (-1)^{\lambda} \cdot \sigma_{l+1-\lambda}(\tau) \\
\sigma_{l+1-\lambda}(\tau) &= \frac{1}{l+1-\lambda} \cdot \sum_{i=1}^{l+1-\lambda} (-1)^{i-1} \cdot s_i(\tau) \cdot \sigma_{l+1-\lambda-i}(\tau)
\end{aligned}$$

Durch Kombination dieser beiden Gleichungen erhalten wir:

$$\begin{aligned}
\sigma_{l+1-\lambda}(\tau) &= \frac{1}{l+1-\lambda} \cdot \sum_{i=1}^{l+1-\lambda} (-1)^{i-1} \cdot s_i(\tau) \cdot \sigma_{l+1-(\lambda+i)}(\tau) \\
&= \frac{1}{l+1-\lambda} \cdot \sum_{i=1}^{l+1-\lambda} (-1)^{i-1} \cdot s_i(\tau) \cdot \left((-1)^{\lambda+i} \sum_{\kappa=0}^{v} a_{\lambda+i,\kappa} \cdot j(l\tau)^{\kappa} \right) \\
&= \frac{1}{l+1-\lambda} \cdot \sum_{i=1}^{l+1-\lambda} \sum_{\kappa=0}^{v} (-1)^{\lambda-1} \cdot a_{\lambda+i,\kappa} \cdot [s_i(\tau) \cdot j(l\tau)^{\kappa}] . \qquad (5.41)
\end{aligned}$$

Wir können also die Koeffizienten $a_{\lambda,\kappa}$ für $(0 \leq \kappa \leq v)$ auf diese Weise bestimmen, falls die Werte für $a_{\hat{\lambda},\kappa}$ bereits für alle $\hat{\lambda} > \lambda$ zuvor bereits berechnet wurden. Die Berechnung kann also auf diese Weise durchgeführt werden, wenn wir die $a_{\lambda,\kappa}$ für $\lambda = l+1, \ldots, 0$ — also in absteigender Reihenfolge — bestimmen. Wir beobachten dabei, daß die Produkte $s_i(\tau) \cdot j(l\tau)^{\kappa}$ für verschiedene Werte von λ wiederholt auftreten. Speichern wir diese in einer Hilfstabelle, so muß jedes dieser Produkte höchstens einmal berechnet werden; ist der zugehörige Koeffizient $a_{\lambda+i,\kappa}$ in Gleichung (5.41) gleich *null*, so kann die Berechnung des Fourier-Reihenproduktes sogar entfallen.

Insgesamt benötigen wir für diese Berechnung wegen $1 \leq i \leq l$ und $0 \leq \kappa \leq v$ also höchstens $l \cdot v$ viele Multiplikationen von Fourier-Reihen. Für $v << l$ erhalten wir somit eine Beschleunigung gegenüber der ursprünglichen Berechnung.

Quadrieren statt Multiplizieren

Im vorgestellten Algorithmus berechnen wir die Potenzsummen $s_{\lambda}(\tau)$ der Nullstellen des gesuchten Polynoms, indem wir die Koeffizienten der zugehörigen q_{τ}-Entwicklung aus der Darstellung der Funktion $f(\tau)^{\lambda}$ „ausschneiden“. Zur Berechnung der Potenzen $f(\tau)^{\lambda}$ für $\lambda = 2, \ldots, l+1$ müssen wir in jedem Schritt eine Fourier-Reihen-Multiplikation durchführen. Wir haben aber bereits im Abschnitt über die Fourier-Reihenarithmetik festgestellt, daß das Quadrieren einer Fourier-Reihe schneller durchgeführt werden kann als eine Multiplikation zweier verschiedener Reihen. Daher verwenden wir folgendes Verfahren:

$$f(\tau)^\lambda = \begin{cases} \left(f(\tau)^{\lambda/2}\right)^2, & \text{falls } \lambda \text{ gerade} \\ \frac{1}{2} \cdot \left[\left(f(\tau)^{\frac{\lambda-1}{2}} + f(\tau)^{\frac{\lambda+1}{2}}\right)^2 - f(\tau)^{\lambda-1} - f(\tau)^{\lambda+1}\right], & \text{falls } \lambda \text{ ungerade} \end{cases}$$

Speichern wir während der Berechnung die Zwischenergebnisse in einer Tabelle, so erhalten wir:

- Ist λ gerade, so kann $f(\tau)^\lambda$ mit Hilfe *einer* Quadrierung berechnet werden.
- Für ungerade λ benötigt man *eine* Quadrierung und *drei* Additionen von Fourier-Reihen.

Da wir $f(\tau)^\lambda$ für $\lambda = 1, \ldots, l+1$ berechnen müssen, können wir also l Multiplikationen ersetzen durch l Quadrierungen und $\lceil \frac{l}{2} \rceil$ Additionen von Fourier-Reihen.

5.4.4 Praktische Resultate

Betrachten wir nun noch einen Laufzeitvergleich. Wir haben dazu für einige Primzahlen l die zugehörigen Polynome modulo einer Primzahl der Größenordnung 2^{26} berechnet. Die Tests wurden auf einer SUN Sparc Station ELC durchgeführt.

Wir vergleichen zunächst nur die Laufzeiten zur Berechnung der Potenzen der Funktion $f(\tau)$ (vgl. Schritte (7)–(12) im Algorithmus Seite 74). Dabei berechnen wir diese Potenzen einmal durch sukzessive Multiplikation und danach mit Hilfe des oben angegebenen Quadrierungs-Tricks. Zur Berechnung der Produkte und Quadrate verwenden wir die schnelle Fourier-Transformation. Die Laufzeiten sind hier in Sekunden angegeben.

l	Reihenlänge	sukzessiv	Quadr. Trick	Quadr. / sukz.
53	689	16,7	11,5	68,8 %
101	2525	146,17	99,5	68,1 %
151	3775	221,0	151,6	68,6 %
211	7385	680,3	462,7	68,0 %
283	13301	1887,7	1287,9	68,2 %

Betrachten wir diese Ergebnisse noch im Vergleich zur Gesamtlaufzeit des Algorithmus. Wir geben dabei in der letzten Spalte der folgenden Tabelle noch an, für wieviele verschiedene Primzahlen der Algorithmus ausgeführt werden mußte, um die über $\mathbb{Z}$ korrekten Koeffizienten zu erhalten (vgl. Seite 85).

l	Reihenlänge	Quadr. Trick	Koeff.vgl	Anzahl Pz.
53	689	11,5	1,5	16
101	2525	99,5	15,3	32
151	3775	151,6	33,3	35
211	7385	462,7	115,0	52
283	13301	1287,9	356,8	72

In diesen Tabellen können wir auch beobachten, daß die Präzisionen der verwendeten Reihen sehr rasch wächst. Untersuchen wir diese Reihenlängen auch noch für größere Primzahlen l. Dabei geben wir auch den Wert der Konstanten v aus (5.4) an.

l	v	Reihenlänge	l	v	Reihenlänge
307	51	15657	547	91	49777
317	79	25043	599	299	179101
359	179	64261	641	160	102560
443	221	97903	701	175	122675
503	251	126253	719	359	258121

Wir können daran erkennen, daß für einige, ungünstige Primzahlen l so große Approximationen von Fourier-Reihen verwendet werden müssen, daß die beschriebenen Polynome in der Praxis nicht mehr berechenbar sind. Daher werden wir uns im folgenden Kapitel damit beschäftigen Polynome zu konstruieren, die im allgemeinen kürzere Approximationen der verwendeten Reihen erfordern.

Kapitel 6

Berechnung äquivalenter Polynome (Typ 2)

Im vorigen Kapitel haben wir eine Funktion $f(\tau)$ mit Invarianzgruppe $\Gamma_0(l)$ verwendet, deren Fourier-Reihenentwicklung in q_τ im ungünstigsten Fall den Startexponenten $-\frac{l-1}{2}$ aufweisen konnte. Da die Präzision der verwendeten Fourier-Reihen, und damit die Laufzeit des Algorithmus, wesentlich von diesem Startexponenten abhängig sind, werden wir nun Funktionen verwenden, bei denen möglichst kleine Startexponenten auftreten.

6.1 Eine unter $\Gamma_0^*(l)$ invariante Funktion

Wir erweitern $\Gamma_0(l)$ um eine Menge von (nicht unimodularen) Transformation und definieren

$$\Gamma_0^*(l) \quad = \quad \Gamma_0(l) \bigcup \left\{ \begin{pmatrix} 0 & -1 \\ l & 0 \end{pmatrix} \right\} \cdot \Gamma_0(l). \tag{6.1}$$

Im folgenden werden wir ein Funktion definieren, die invariant ist unter den Transformationen aus $\Gamma_0^*(l)$; die somit die Invarianzgruppe $\Gamma_0(l)$ besitzt. Diese Funktion liefert uns dann wiederum eine Polynom, das wir anstelle des modularen Polynoms verwenden können.

Mit der *Dedekind'sche η-Funktion* sei

$$g(\tau) \quad = \quad \eta(\tau) \cdot \eta(l\tau)$$

definiert. Mit Hilfe der q_τ-Entwicklung (5.2) der η-Funktion können wir erkennen, daß auch $g(\tau)$ in eine Fourier-Reihe mit ganzzahligen Koeffizienten entwickelt werden kann. Wir erhalten

$$g(\tau) \quad = \quad q_\tau^{(l+1)/24} \cdot \sum_{n=0}^{\infty} b_n \cdot q_\tau^n.$$

Dabei ist insbesondere $b_0 = 1$, da die Leitkoeffizienten in den zu $\eta(\tau)$ und $\eta(l\tau)$ gehörenden Fourier-Reihen ebenfalls gleich eins sind. Somit ist $g(\tau)$ als formale Fourier-Reihe über $\mathbb{Z}$ invertierbar, und es ist

$$g(\tau)^{-1} \quad = \quad q_\tau^{-(l+1)/24} \cdot \sum_{n=0}^{\infty} c_n \cdot q_\tau^n,$$

mit *ganzzahligen* Koeffizienten c_n $(n \geq 0)$. In diesen Fourier-Reihenentwicklungen treten allerdings sämtlich *rationale* Exponenten auf — sie sind alle von der Form $n \pm \frac{l+1}{24}$. Unser Ziel ist es nun, mit Hilfe von $g(\tau)$ eine Funktion zu konstruieren, deren q_τ-Entwicklung ganzzahlige Koeffizieten und Exponenten besitzt. Dazu werden wir $g(\tau)$ durch bestimmte *Operatoren* modifizieren.

6.1. Definition *Für eine Funktion $f(\tau)$ ist der* **n-te Hecke-Operator** T_n *definiert als*

$$(T_n f)(\tau) \quad = \quad \sum_{d|n} \frac{1}{d} \cdot \sum_{b=0}^{d-1} \cdot f\left(\frac{n\tau + bd}{d^2}\right). \tag{6.2}$$

Da wir Funktionen stets mit Hilfe von Fourier-Reihenentwicklungen darstellen, betrachten wir nun noch, wie die Hecke-Operatoren T_n auf diese wirken.

6.2. Satz *Sei eine Funktion $f(\tau)$ gegeben durch eine Fourier-Reihenentwicklung in q_τ*

$$f(\tau) \quad = \quad \sum_{m=0}^{\infty} c(m) \cdot q_\tau^m,$$

und n eine natürliche Zahl, $n \in \mathbb{N}$. Dann hat $(T_n f)(\tau)$ die q_τ-Entwicklung

$$(T_n f)(\tau) \quad = \quad \sum_{m=0}^{\infty} \gamma_n(m) \cdot q_\tau^m,$$

wobei die Koeffizienten $\gamma_n(m)$ gegeben sind durch

$$\gamma_n(m) \quad = \quad \sum_{d|(n,m)} c\left(\frac{mn}{d^2}\right).$$

Beweis: Siehe dazu [Ap89, Satz 6.6, Seite 121]. ■

Unter Verwendung der Hecke-Operatoren konstruieren wir uns nun die gewünschte Funktion. Dazu seien im folgenden stets r und s so gewählt, daß

$$\frac{l+1}{24} = \frac{r}{s} \quad \text{mit} \quad \text{ggT } (r,s) = 1.$$

6.3. Definition *Sei $n \in \mathbb{P}$ so gewählt, daß sowohl n ein Quadrat modulo l als auch l ein Quadrat modulo n ist, und daß s ein Teiler von $n-1$ ist. Sei weiter $g(\tau)$ die oben definierte Funktion $g(\tau) = \eta(\tau)\eta(l\tau)$. Dann definieren wir*

$$A(\tau) = \frac{(T_n g)(\tau)}{g(\tau)}. \tag{6.3}$$

6.4. Bemerkung Wir beachten dabei, daß die Fourier-Reihe der Funktion $g(\tau)$ den Startexponenten $-\frac{r}{s}$ besitzt, und daß damit alle Exponenten in dieser Darstellung rationale Zahlen sind. Somit ist Satz 6.2 nicht direkt anwendbar. Die Exponenten in der q_τ-Entwicklung von $g(\tau)$ haben aber alle den gemeinsamen Nenner s. Die Funktion $g(s\tau)$ besitzt dann also eine Fourier-Entwicklung, bei der nur ganzzahlige Exponenten auftreten, und wir können $T_n(g(s\tau))$ mit Hilfe von Satz 6.2 berechnen. Ist dabei der Index n des Hecke-Operators gemäß Definition 6.3 gewählt, so erhalten wir damit eine Fourier-Reihe in $q_{s\tau}$, also eine Funktion in $s \cdot \tau$

$$T_n(g(s\tau)) = T(s \cdot \tau) \quad \Longrightarrow \quad (T_n g)(\tau) = T(\tau).$$

Daraus können wir die Funktion $(T_n g)(\tau)$ berechnen. Diese hat dann den Startexponenten $\frac{r'}{s}$, wobei (wiederum durch die Wahl des Operators T_n gewährleistet) $r' - r$ ein Vielfaches von s ist. Damit erhalten wir für $A(\tau)$ die Fourier-Entwicklung

$$A(\tau) = q_\tau^{(r'-r)/s} \cdot \sum_{\nu=0}^{\infty} a_\nu q_\tau^\nu$$

mit ganzzahligen Exponenten, und Koeffizienten aus $\mathbb{Z}$.

Die in Definition 6.3 definierte Funktion $A(\tau)$ ist also nicht nur abhängig von l, sondern auch von der Wahl des Hecke-Operators T_n. Somit haben wir eine ganze Menge von Funktionen zur Verfügung, die wir zur Berechnung des gesuchten Polynoms verwenden können.

Durch die Wahl verschiedener Hecke-Operatoren, kann man die Startexponenten der Funktion $A(\tau)$ beeinflussen. Wir werden später sehen, daß die Größe des Startexponenten von $A(\tau)$ Einfluß hat auf die Präzision der verwendeten Approximationen der Fourier-Reihen — je größer der Startexponent von $A(\tau)$ desto kürzer werden wir diese berechnen müssen. Damit werden wir in der Praxis den Hecke-Operator T_n so wählen, daß der Startexponent von $A(\tau)$ möglichst groß wird.

6.5. Beispiel Betrachten wir die Situation für $l = 281$, dann erhalten wir

$$\frac{l+1}{24} = \frac{282}{24} = \frac{47}{4}.$$

Damit ist also $r = 47$ und $s = 4$. Mögliche Hecke-Operatoren sind T_{17}, T_{53} und T_{29}. Berechnen wir dazu jeweils die Funktion $A(\tau)$, so ergeben sich die folgenden Startexponenten.

Hecke-Operator	Startexponent
T_{17}	-9
T_{53}	-8
T_{29}	-7

Für $l = 281$ wählen wir daher den Operator T_{29}.

Wir müssen nun noch gewährleisten, daß die Funktion $A(\tau)$ aus (6.3) Invarianzgruppe $\Gamma_0(l)$ hat, und somit für die Konstruktion eines zum modularen Polynom äquivalenten Polynoms geeignet ist. Dies bestätigt der folgende Satz.

6.6. Satz *Die in (6.3) definierte Funktion $A(\tau)$ ist invariant unter allen Transformationen aus $\Gamma_0(l)$.*

Beweis: Vergleiche [Mü94]. ■

Damit existiert zu $A(\tau)$ das Polynom $G_l^A(X)$ aus (4.5), das wir anstelle des modularen Polynoms verwenden können. Betrachten wir, wie im vorangehenden Abschnitt, noch das Verhalten der Funktion $A(\tau)$ unter der (nicht unimodularen) Transformation $\tau \to -\frac{1}{l\tau}$. Eine Aussage darüber liefert der nächste Satz.

6.7. Satz *Die Funktion $A(\tau)$ aus (6.3) ist* **invariant** *unter der Transformation $\tau \to -\frac{1}{l\tau}$, d.h.*

$$A\left(-\frac{1}{l\tau}\right) = A(\tau).$$

Beweis: Vergleiche [Mü94]. ■

Diese Aussagen werden wir im folgenden Abschnitt dazu verwenden, die Koeffizienten des Polynom $G_l^A(X)$ zu berechnen.

6.2 Berechnung des Polynoms für $\Gamma_0^*(l)$

Sei im folgenden für einen geeigneten Hecke-Operator die Funktion $A(\tau)$ wie in (6.3) definiert. Da diese Funktion Invarianzgruppe $\Gamma_0(l)$ besitzt, existiert ein zum l-ten modularen Polynom äquivalentes Polynom $G_l^A(X)$ wie in (4.5)

$$G_l^g(X) \quad = \quad \sum_{\lambda=0}^{l+1}\sum_{\kappa=0}^{k} a_{\lambda,\kappa}\cdot j(\tau)^\kappa \cdot X^\lambda.$$

Dabei werden wir die maximale Potenz k von $j(\tau)$ in dieser Gleichung später bestimmen. Insbesondere ist wiederum $A(\tau)$ Nullstelle dieses Polynoms, und wir erhalten daraus

$$\sum_{\lambda=0}^{l+1}\sum_{\kappa=0}^{k} a_{\lambda,\kappa}\cdot j(\tau)^\kappa \cdot A(\tau)^\lambda \quad = \quad 0. \tag{6.4}$$

Wir haben zuvor bereits festgestellt, daß $A(\tau)$ *invariant* ist unter der Transformation $\tau \to -\frac{1}{l\tau}$. Es gilt also

$$A\left(-\frac{1}{l\tau}\right) \quad = \quad A(\tau).$$

Wenden wir diese Transformation auf obige Gleichung an, so erhalten wir

$$\sum_{\lambda=0}^{l+1}\sum_{\kappa=0}^{k} a_{\lambda,\kappa}\cdot j(l\tau)^\kappa \cdot A(\tau)^\lambda \quad = \quad 0. \tag{6.5}$$

Wie bereits im vorigen Abschnitt untersuchen wir nun noch das Verhalten der Funktion $A(\tau)$ unter den Transformationen V_n $(0 \le n \le l)$ aus (5.9). Wir erhalten dann das folgende Lemma.

6.8. Lemma *Sei die Funktion $A(\tau)$ wir in (6.3) definiert. Dann gilt für die Transformationen V_n $(0 \le n \le l)$ aus (5.9)*

$$\begin{aligned} A(V_n(\tau)) &= A\left(\tau + \frac{n}{l}\right) \qquad \textit{für} \quad 0 \le n < l, \\ \textit{und} \qquad A(V_l(\tau)) &= A(l\tau). \end{aligned}$$

Beweis: Für $0 \le n < l$ rechnet man einfach nach:

$$A(V_n(\tau)) = A\left(\frac{l\tau + n}{l}\right) = A\left(\tau + \frac{n}{l}\right).$$

Zum Beweis der zweiten Aussage beachten wir , daß wegen der Eigenschaft

$$A\left(-\frac{1}{l\tau}\right) \quad = \quad A(\tau)$$

speziell auch die folgende Aussage gilt:

$$A(V_l(\tau)) \;=\; A\left(-\frac{1}{l^2\tau}\right) \;=\; A\left(-\frac{1}{l\cdot(l\tau)}\right) \;=\; A(l\tau). \qquad \blacksquare$$

Bereits in Lemma 5.10 haben wir gezeigt, daß $j(l\tau)$ unter den Transformationen V_n $(0 \leq n \leq l)$ invariant ist. Wenden wir nun diese Transformationen auf (6.5) an, so erhalten wir also für alle n mit $(0 \leq n \leq l)$:

$$\sum_{\lambda=0}^{l+1}\sum_{\kappa=0}^{k} a_{\lambda,\kappa}\cdot j(l\tau)^{\kappa}\cdot A(V_n(\tau))^{\lambda} \quad = \quad 0. \tag{6.6}$$

Also bilden auch hier die $A(V_n(\tau))$ für $(0 \leq n \leq l)$ die Gesamtheit der Nullstellen des Polynoms

$$H(X) \quad := \quad \sum_{\lambda=0}^{l+1}\underbrace{\left(\sum_{\kappa=0}^{k} a_{\lambda,\kappa}\cdot j(l\tau)^{\kappa}\right)}_{=:\,k_\lambda(\tau)}\cdot X^{\lambda}. \tag{6.7}$$

Die zugehörigen Koeffizienten sind damit wieder die elementarsymmetrischen Funktionen in den $A(V_n(\tau))$. Da wir die Funktion $A(\tau)$ stets so wählen, daß die zugehörige Fourier-Entwicklung nur ganzzahlige Koeffizienten enthält, besitzen auch die elementarsymmetrischen Funktionen eine q_τ-Entwicklung mit Koeffizienten aus $\mathbb{Z}$. Also sind insbesondere auch die Koeffizienten $a_{\lambda,\kappa}$ ganze Zahlen.

Der Algorithmus zur Bestimmung der Koeffizienten, wie wir ihn für die Funktion $f(\tau)$ aus (5.3) vorgestellt haben, ist also theoretisch auch hier anwendbar. Es wird sich jedoch zeigen, daß dieses Verfahren in der Praxis nicht geeignet ist. Dazu werden wir zunächst bestimmen, in welcher maximalen Potenz $j(l\tau)$ in Gleichung (6.5) vorkommen kann und danach untersuchen, mit welcher Präzision die verwendeten Fourier-Reihen approximiert werden müssen.

6.9. Lemma *Sei die Funktion $A(\tau)$ wie in (6.3) definiert und durch ihre Fourier-Reihenentwicklung gegeben:*

$$A(\tau) \quad = \quad \sum_{n=-v}^{\infty} a_n\cdot q_\tau^n.$$

Dann kann die höchste Potenz k der Funktion $j(l\tau)$ in Gleichung (6.5) gleich $2\cdot v$ gewählt werden.

Beweis: Wir wissen bereits, daß die Koeffizienten $k_\lambda(\tau)$ den elementarsymmetrischen Funktionen in den Nullstellen des Polynoms $H(X)$ entsprechen. Diese besitzen eine Fourier-Entwicklung mit ganzzahligen Exponenten, obwohl in den Darstellungen der Nullstellen rationale Exponenten auftreten:

$$A\left(\tau + \frac{n}{l}\right) = \sum_{i=-v}^{\infty} a_i q_{\tau+n/l}^i = \sum_{i=-v}^{\infty} a_i q_\tau^i \cdot q_{n/l}^i.$$

Darin ist die minimale Potenz, in der q_τ vorkommt gleich q_τ^{-v}. Weiter ist der Startexponent von $A(V_l(\tau))$

$$\text{first}\ (A(V_l(\tau))) = \text{first}\ (A(l\tau)) = -lv.$$

Für die i-te elementarsymmetrische Funktion $\sigma_i(\tau) = \sigma_i(A(V_0(\tau)), \ldots A(V_l(\tau)))$ können wir damit nachrechnen

$$\text{first}\ (\sigma_i(\tau)) \geq -(i-1) \cdot v - lv,$$

und dies ist minimal für $i = (l+1)$. ■

Wollen wir die Funktionen $\sigma_i(\tau)$ wieder mit Hilfe der Newton'schen Formeln berechnen, so benötigen wir dazu auch die Potenzsummen in den Nullstellen. Betrachten wir die Startexponenten dieser Funktionen, und seien dazu $s_\rho(\tau) = s_\rho(A(V_0(\tau)), \ldots A(V_l(\tau)))$. Dann schreiben wir wieder

$$s_\rho(\tau) = \underbrace{\sum_{n=0}^{l-1} A\left(\tau + \frac{n}{l}\right)^\rho}_{=:s_{\rho,1}} + \underbrace{A(l\tau)^\rho}_{=:s_{\rho,2}} \tag{6.8}$$

und erhalten für die Startexponenten dieser Funktionen

$$\begin{aligned} \text{first}\ (s_{\rho,1}(\tau)) &\geq -\rho \cdot v \\ \text{first}\ (s_{\rho,2}(\tau)) &= -\rho \cdot lv \end{aligned}$$

und damit

$$\text{first}\ (s_\rho(\tau)) = -\rho \cdot lv. \tag{6.9}$$

Wir haben früher bereits gezeigt, daß es für den korrekten Ablauf des Algorithmus erforderlich ist, daß alle verwendeten Approximationen von Fourier-Reihen mindestens bis zu dem Term q_τ^0 bekannt sind. Dies hat nun zur Folge, daß wir die hier auftretenden Potenzsummen durch mindestens $l^2 v$ viele Koeffizienten darstellen müssen. Dies ist in der Praxis nicht mehr anwendbar.

Um diesem Problem auszuweichen, werden wir zwei andere Methoden vorstellen, die es uns ermöglichen, den Koeffizientenvergleich unter Verwendung kürzerer Reihen durchzuführen.

6.2.1 Der simultane Koeffizientenvergleich

Wir werden nun zeigen, daß die Koeffizienten $a_{\lambda,\kappa}$ $(0 \leq \lambda \leq l+1, 0 \leq \kappa \leq 2v)$ des Polynoms bereits eindeutig durch die Gleichungen

$$\sum_{\lambda=0}^{l+1} \sum_{\kappa=0}^{2v} a_{\lambda,\kappa} \cdot j(\tau)^\kappa \cdot A(\tau)^\lambda = 0 \tag{6.10}$$

und

$$\sum_{\lambda=0}^{l+1} \sum_{\kappa=0}^{2v} a_{\lambda,\kappa} \cdot j(l\tau)^\kappa \cdot A(\tau)^\lambda = 0 \tag{6.11}$$

bestimmt sind. Damit ist es möglich, durch einen „simultanen“ Koeffizientenvergleich dieser beiden Aussagen die Koeffizienten $a_{\lambda,\kappa}$ zu bestimmen ohne die elementarsymmetrischen Funktionen in den Nullstellen berechnen zu müssen. Da wir bereits wissen, daß das gesuchte Polynom normiert ist, gilt auch hier

$$a_{l+1,\kappa} = \begin{cases} 1, & \text{falls } \kappa = 0, \\ 0, & \text{falls } 1 \leq \kappa \leq 2v. \end{cases}$$

Damit erhalten wir aus obigen Gleichungen

$$\sum_{\lambda=0}^{l} \sum_{\kappa=0}^{2v} a_{\lambda,\kappa} \cdot j(\tau)^\kappa \cdot A(\tau)^\lambda = -A(\tau)^{l+1} \tag{6.12}$$

und

$$\sum_{\lambda=0}^{l} \sum_{\kappa=0}^{2v} a_{\lambda,\kappa} \cdot j(l\tau)^\kappa \cdot A(\tau)^\lambda = -A(\tau)^{l+1}. \tag{6.13}$$

Um einen Vergleich der Koeffizienten in diesen Gleichungen durchzuführen, müssen also die Produkte $j(\tau)^\kappa \cdot A(\tau)^\lambda$ bzw. $j(l\tau)^\kappa \cdot A(\tau)^\lambda$ berechnet werden. Da es in der Praxis günstiger ist, mit dünn besetzten Fourier-Reihen zu rechnen, werden wir hauptsächlich Gleichung (6.13) verwenden. Wir ermitteln dann jeweils den Summanden, der den Startexponenten der linken Seite definiert und bestimmen dann den Wert des zugehörigen Koeffizienten. Zur Berechnung der $a_{\lambda,\kappa}$ werden wir also die Leitkoeffizienten der Summanden

$$a_{\lambda,\kappa} \cdot j(l\tau)^\kappa \cdot A(\tau)^\lambda \tag{6.14}$$

mit den entsprechenden Koeffizienten der jeweiligen rechten Seite vergleichen. Ist der Leitkoeffizient der Fourier-Entwicklung von $A(\tau)$ gleich c, so ist

$$\mathrm{lc}\left(A(\tau)^\lambda\right) = c^\lambda.$$

Da weiterhin der Leitkoeffizient von $j(l\tau)$ gleich 1 ist, gilt auch:

$$\mathrm{lc}\ \left(a_{\lambda,\kappa} \cdot j(l\tau)^{\kappa} \cdot A(\tau)^{\lambda}\right) \quad = \quad a_{\lambda,\kappa} \cdot c^{\lambda}.$$

Wir müssen also zur Berechnung der $a_{\lambda,\kappa}$ den führenden Koeffizienten der jeweiligen rechten Seite durch c^{λ} dividieren. Da wir bereits wissen, daß alle Koeffizienten $a_{\lambda,\kappa}$ ganze Zahlen sind, ist diese Division in $\mathbb{Z}$ möglich. Ist dann ein Koeffizient $a_{\lambda,\kappa}$ bestimmt, subtrahieren wir den entsprechenden Term auf beiden Seiten der Gleichung.

Betrachten wir nun die Exponenten, deren zugehörige Koeffizienten wir vergleichen müssen. Die Startexponenten dieser Fourier-Reihen sind

$$\mathrm{first}\ (a_{\lambda,\kappa} \cdot j(l\tau)^{\kappa} \cdot A(\tau)^{\lambda}) \quad = \quad -(\lambda v \ + \ \kappa l).$$

Wir vergleichen dann in den jeweiligen q_τ-Entwicklungen die Koeffizienten zu den Exponenten $-(\lambda v + \kappa l)$. Treten diese Startexponenten genau einmal auf, so genügt es, für den Koeffizientenvergleich nur Gleichung (6.13) zu verwenden. Tritt allerdings ein Startexponent e mehrmals unter den Summanden (6.14) auf, so kann der zugehörige Koeffizient nicht eindeutig bestimmt werden. Gilt zum Beispiel

$$-\left(\lambda_1 v + \kappa_1 l\right) \ = \ e \ = \ -(\lambda_2 v + \kappa_2 l), \tag{6.15}$$

so können a_{λ_1,κ_1} und a_{λ_2,κ_2} nicht allein mit Hilfe von Gleichung (6.13) berechnet werden.

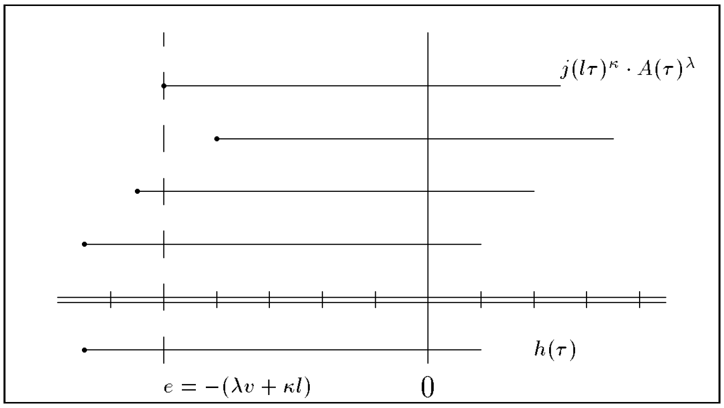

Abbildung 6.1: Koeffizientenvergleich

Ist dann nämlich e der Startexponent der aktuellen rechten Seite $h(\tau)$, dann gilt

$$\begin{aligned} \mathrm{lc}\ (h(\tau)) \ &= \ \mathrm{lc}\ (a_{\lambda_1,\kappa_1} \cdot j(l\tau)^{\kappa_1} \cdot A(\tau)^{\lambda_2}) + \mathrm{lc}\ (a_{\lambda_2,\kappa_2} \cdot j(l\tau)^{\kappa_2} \cdot A(\tau)^{\lambda_2}) \\ &= \ a_{\lambda_1,\kappa_1} \cdot c^{\lambda_1} \ + \ a_{\lambda_2,\kappa_2} \cdot c^{\lambda_2}. \end{aligned}$$

In diesen Situationen werden wir zunächst einen der Koeffizienten a_{λ_1,κ_1} mit Hilfe von Gleichung (6.12) bestimmen. Dann kann der Wert für a_{λ_2,κ_2} wieder durch Vergleich in (6.13) berechnet werden.

Wir werden zeigen, daß in (6.13) nur eine kleine Anzahl von Indexpaaren (λ, κ) mehrfach auftretende Exponenten definiert. Dazu betrachten wir das folgende Lemma.

6.10. Lemma *Seien für* $0 \leq \lambda_1, \lambda_2 \leq l$ *und* $0 \leq \kappa_1, \kappa_2 \leq 2v$

$$a_{\lambda_1,\kappa_1} \cdot j(l\tau)^{\kappa_1} \cdot A(\tau)^{\lambda_1} \quad \textit{und} \quad a_{\lambda_2,\kappa_2} \cdot j(l\tau)^{\kappa_2} \cdot A(\tau)^{\lambda_2}$$

zwei Summanden aus der Gleichung (6.13), die denselben Startexponenten e besitzen. Dann gilt

1. $\lambda_1 = l,$
2. $\lambda_2 = 0,$
3. $\kappa_1 \leq v$ *und*
4. $\kappa_2 = \kappa_1 + v.$

Beweis: Betrachten wir die Startexponenten, die von den jeweiligen Summanden definiert werden, dann erhalten wir

$$e_1 = -(\lambda_1 v + \kappa_1 l)$$

und

$$e_2 = -(\lambda_2 v + \kappa_2 l).$$

Stimmen diese Startexponenten überein, so gilt

$$\lambda_1 v + \kappa_1 l = \lambda_2 v + \kappa_2 l \quad \Longleftrightarrow \quad (\lambda_1 - \lambda_2) \cdot v = (\kappa_1 - \kappa_2) \cdot l.$$

Betrachten wir diese Gleichung modulo l, so erhalten wir

$$(\lambda_1 - \lambda_2) \cdot v \equiv 0 \pmod{l},$$

Da v stets kleiner als l und von 0 verschieden ist, ist es auch modulo l invertierbar. Daher muß bereits

$$(\lambda_1 - \lambda_2) \equiv 0 \pmod{l}$$

gelten. Die Indizes λ_1 und λ_2 können sich demnach also nur um ein Vielfaches von l unterscheiden. Da wir diese nur zwischen 0 und l betrachten, muß also

$$\lambda_1 = l \quad \text{und} \quad \lambda_2 = 0$$

gelten. Daraus können wir auch leicht die Aussagen über κ_1 und κ_2 herleiten:

$$lv + \kappa_1 l = \kappa_2 l \quad \Longleftrightarrow \quad v + \kappa_1 = \kappa_2.$$

Dabei muß stets

$$\kappa_2 \leq 2v \quad \Longleftrightarrow \quad \kappa_1 \leq v$$

erfüllt sein. ■

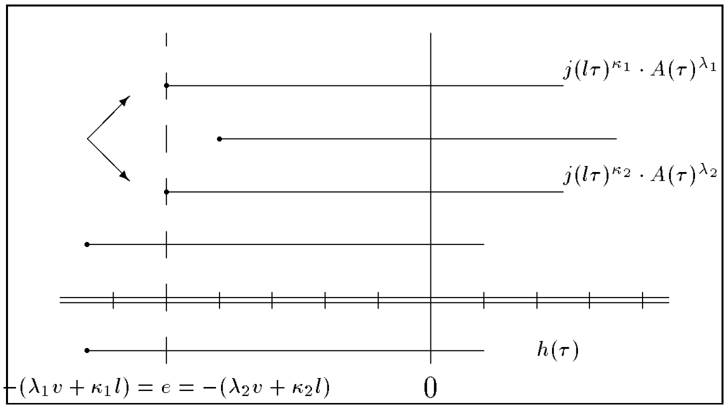

Abbildung 6.2: Mehrfach auftretende Startexponenten

Mit Hilfe dieses Lemmas können wir nun alle Indexpaare (λ, κ) angeben, für die die zugehörigen Koeffizienten $a_{\lambda,\kappa}$ nicht ausschließlich mit Gleichung (6.13) berechenbar sind.

Exponent	$-2lv$	...	$-(lv+v)$	$-lv$
Indexpaar	(l,v)	...	$(l,1)$	$(l,0)$
	$(0,2v)$	...	$(0,v+1)$	$(0,v)$

Erreichen wir während der Berechnung einen Exponenten e wie in (6.15), so bestimmen wir zunächst den Koeffizienten a_{l,κ_1} mit Gleichung (6.12) und gehen dann zurück zu (6.13), um a_{0,κ_2} berechnen.

Zeigen wir nun, daß dieses Verfahren durchführbar ist. Sei dazu $a_{l,\kappa}$ der Koeffizient, den wir mit Hilfe von Gleichung (6.12) bestimmen wollen. Dieser Koeffizient definiert in der ersten Gleichung den Exponenten

$$\text{first } (a_{l,\kappa} \cdot j(\tau)^\kappa \cdot A(\tau)^l) \quad = \quad -(lv \; + \; \kappa).$$

Wir müssen also gewährleisten, daß alle Koeffizienten, die in (6.12) Exponenten kleiner als $-(lv + \kappa)$ definieren, bereits bekannt sind. Dazu betrachten wir die Startexponenten zu den jeweiligen Indexpaaren in dieser Gleichung.

Exp.	$-(lv+2v)$	$-(lv+v+1)$	$-(lv+v)$	$-(lv+v-1)$	$-(lv+1)$	$-lv$
Index	$(l,2v)$	$(l,v+1)$	(l,v)	$(l,v-1)$	$(l,1)$	$(l,0)$
			$(l-1,2v)$	$(l-1,2v-1)$	$(l-1,v+1)$	$(l-1,v)$
						$(l-2,2v)$

Der erste Koeffizient, den wir in (6.12) bestimmen, ist $a_{l,v}$. Also müssen dann die Koeffizienten $a_{l,2v}, \ldots, a_{l,v+1}$ bekannt sein; man kann sogar nachrechnen, daß diese gleich 0 sein müssen.

6.11. Lemma *Die Koeffizienten $a_{l,v+1}, \ldots, a_{l,2v}$ des Polynoms sind stets 0.*

Beweis: Betrachten wir dazu Gleichung (6.12) zu Beginn der Berechnung. Dann gilt für die rechte Seite dieser Gleichung

$$\text{first } (-A(\tau)^{l+1}) \quad = \quad -(l+1) \cdot v \quad = \quad -(lv+v).$$

Die Koeffizienten $a_{l,2v}, \ldots, a_{l,v+1}$ definieren in dieser Gleichung die Exponenten $-(lv + 2v), \ldots, -(lv + v + 1)$, und diese sind sämtlich kleiner als $-(lv + v)$. Damit müssen die Koeffizienten gleich 0 sein. ■

Wir können durch ähnliche Beobachtungen noch weitere Koeffizienten sofort bestimmen.

6.12. Lemma *Sei $c = lc\,(A(\tau))$ der Leitkoeffizient der Fourier-Entwicklung der Funktion $A(\tau)$. Dann gilt*

$$a_{l,v} \quad = \quad -c \qquad \textit{und} \qquad a_{0,2v} \quad = \quad c^{l+1}. \tag{6.16}$$

Beweis: Zum Beweis beachten wir, daß die Koeffizienten $a_{l,v}$ und $a_{0,2v}$ in Gleichung (6.13) den Exponenten $-2lv$ definieren. Dieser ist jedoch kleiner als der Startexponent der Reihe $-A(\tau)^{l+1}$, und damit gilt

$$\begin{array}{rrcl} & a_{l,v} \cdot c^l + a_{0,2v} & = & 0 \\ \Longleftrightarrow & a_{0,2v} & = & -a_{l,v} \cdot c^l. \end{array} \tag{6.17}$$

Betrachten wir in Gleichung (6.12) die Koeffizienten zu dem Exponenten $-(l+1)v$, so erhalten wir

$$a_{l,v} \cdot c^l + a_{l-1,2v} = lc(-A(\tau)^{l+1}) = -c^{l+1}. \tag{6.18}$$

Der Koeffizient $a_{l-1,2v}$ definiert jedoch in (6.13) eindeutig einen Exponenten, der kleiner ist als $-(l+1)v$. Somit muß dieser Koeffizient gleich 0 sein. Setzen wir dies in (6.18) ein, dann gilt

$$a_{l,v} \cdot c^l = -c^{l+1} \qquad \Longleftrightarrow \qquad a_{l,v} = -c.$$

Schließlich erhalten wir damit aus (6.17)

$$a_{0,2v} = -(-c) \cdot c^l = c^{l+1}. \qquad \blacksquare$$

Aber wir können erkennen, daß auch in (6.12) einige Startexponenten mehrfach vorkommen; die Koeffizienten $a_{l,\kappa}$ und $a_{l-1,\kappa+v}$ definieren in (6.12) dieselben Startexponenten. Um die Koeffizienten $a_{l,\kappa}$ $(0 \leq \kappa \leq v)$ dennoch korrekt bestimmen zu können, müssen wir zeigen, daß die $a_{l-1,\kappa+v}$ dann bereits bekannt sind. Dies bestätigt das folgenden Lemma.

6.13. Lemma *Wenn der Koeffizient $a_{l,\kappa}$ mit Hilfe von Gleichung (6.12) bestimmt wird, dann wurde zuvor bereits $a_{l-1,\kappa+v}$ in Gleichung (6.13) berechnet.*

Beweis: Zum Beweis dieser Aussage müssen wir lediglich zeigen, daß das Indexpaar $(l-1, \kappa+v)$ in Gleichung (6.13) einen kleineren Exponenten definiert als (l, κ), d.h

$$\begin{array}{rrcl} & -((l-1)v + (\kappa+v)l) & < & -(lv + \kappa l) \\ \Longleftrightarrow & (l-1)v + (\kappa+v)l & > & lv + \kappa l \\ \Longleftrightarrow & lv - v + \kappa l + lv & > & lv + \kappa l \\ \Longleftrightarrow & l & > & 1. \end{array}$$

Dies ist stets erfüllt. $\blacksquare$

Mit denselben Überlegungen folgt dann auch, daß der Koeffizient $a_{l-2,2v}$ bereits bekannt ist, wenn $a_{l,0}$ berechnet werden soll. Damit haben wir also nachgewiesen, daß die Koeffizienten des Polynoms nur unter Verwendung der Gleichungen (6.12) und (6.13) berechnet werden können. Wir erhalten zusammenfassend den folgenden Algorithmus.

Berechnung des Polynoms für $\Gamma_0^*(l)$ (simultaner Vergleich)

Eingabe : Fourier-Reihen für $A = A(\tau)$, $j = j(\tau)$ und $j_l = j(l\tau)$
Ausgabe : Feld der Koeffizienten $a_{\lambda,\kappa}$ für $0 \le \lambda \le l+1$ und $0 \le \kappa \le 2v$

Festlegen der bekannten Koeffizienten

for $(\kappa = 1, \ldots 2v)$ **do**
  $a_{l+1,\kappa} = 0$
od
$c = \mathrm{lc}\ (A)$
$a_{l+1,0} = 1$ // Poly.-Leitkoeff. = 1
$a_{l,v} = -c$ // s. Lemma 6.12
$a_{0,2v} = c^{l+1}$

Initialisieren der rechten Seiten

$h_1(\tau) = -A^{l+1} + c \cdot j^v A^l - c^{l+1} \cdot j^{2v}$ // R.S. für (6.12)
$h_2(\tau) = -A^{l+1} + c \cdot j_l^v A^l - c^{l+1} \cdot j_l^{2v}$ // R.S. für (6.13)

Führe Koeffizientenvergleich durch

while $(h_2(\tau) \neq 0)$ **do**
  $e = \mathrm{first}\ (h_2(\tau))$
  Bestimme (λ, κ) mit $\lambda v + \kappa l = e$
  if $(\lambda = l \wedge \kappa \le v)$ **then** // Übergang zu h_1
    if $(e = lv)$ **then** // 3 Indexpaare in (6.12)
      $h_1(\tau) -= a_{l-2,2v} \cdot j^{2v} \cdot A^{l-2}$
      $h_1(\tau) -= a_{l-1,v} \cdot j^v \cdot A^{l-1}$
      $a_{l,0} = \mathrm{lc}\ (h_1(\tau))/c^l$
    else
      $h_1(\tau) -= a_{l-1,\kappa+v} \cdot j^{\kappa+v} \cdot A^{l-1}$
      $a_{l,\kappa} = \mathrm{lc}\ (h_1(\tau))/c^l$
    fi

Berechnung des Polynoms für $\Gamma_0^*(l)$
(Fortsetzung)

// Best. $a_{0,\kappa+v}$ in $h_2(\tau)$

    $h_2(\tau) \mathrel{-}= a_{\lambda,\kappa} \cdot j_l^\kappa \cdot A^\lambda$
    $a_{0,\kappa+v} = \mathrm{lc}\ (h_2(\tau))$
  else
    $a_{\lambda,\kappa} = \mathrm{lc}\ (h_2(\tau))/c^\lambda$
    $h_2(\tau) \mathrel{-}= a_{\lambda,\kappa} \cdot j_l^\kappa \cdot A^\lambda$
  fi
od

6.2.2 Verwendung elementarsymmetrischer Funktionen

Zu Beginn dieses Abschnittes haben wir feststellen können, daß die Potenzsummen in den Funktionen $A(V_n(\tau))$ $(0 \leq n \leq l)$ Fourier-Entwicklungen besitzen, die sehr kleine Startexponenten aufweisen. Damit hat sich die Berechnung der elementarsymmetrischen Funktionen mit Hilfe der Newton'schen Formeln als nicht anwendbar erwiesen. Wir haben daher einen Algorihtmus vorgestellt, der auf die Berechnung dieser Potenzsummen verzichtet.

Hier wollen wir nun beschreiben, wie wir die elementarsymmetrischen Funktionen in den Nullstellen berechnen können, ohne jedoch mit „sehr großen" Approximationen von Fourier-Reihen rechnen zu müssen.

Unser Ziel war es, mit Hilfe von $l+1$ vorgegebenen Nullstellen das Polynom

$$H(X) \quad = \quad \prod_{n=0}^{l} (X - A(V_n(\tau)))$$

zu berechnen. Wir haben gezeigt, daß die Koeffizienten dieses Polynoms genau die elementarsymmetrischen Funktionen in diesen Nullstellen sind. Fassen wir in obigem Produkt nur l dieser Nullstellen zusammen, so erhalten wir

$$\begin{aligned} H(X) &= \prod_{n=0}^{l} (X - A(V_n(\tau))) \\ &= \prod_{n=0}^{l-1} (X - A(V_n(\tau))) \cdot (X - A(V_l(\tau))) \\ &= H_1(X) \cdot (X - A(V_l(\tau))) \,. \end{aligned}$$

Die Koeffizienten von $H_1(X)$ sind dann die elementarsymmetrischen Funktionen in den ersten l Nullstellen $A(\tau + \frac{n}{l})$ $(0 \leq n < l)$. Seien also

$$\sigma_i(\tau) = \sigma_i \left(A(V_0(\tau), \ldots, A(V_{l-1}(\tau)) \right).$$

Dann erhalten wir

$$H_1(X) = X^l - \sigma_1(\tau) X^{l-1} + \sigma_2(\tau) X^{l-2} - \ldots + (-1)^l \sigma_l(\tau).$$

Multiplizieren wir dieses Polynom mit dem Linearfaktor $X - A(V_l(\tau)) = X - A(l\tau)$, so gilt

$$\begin{array}{rrrrrl} H(X) = & & X^{l+1} & & & \\ & - & \sigma_1(\tau)\cdot X^l & + \quad \sigma_2(\tau)\cdot X^{l-1} & -\ldots- & \sigma_l(\tau)\cdot X \\ & - & A(l\tau)\cdot X^l & + \ A(l\tau)\cdot\sigma_1(\tau)\cdot X^{l-1} & -\ldots- & A(l\tau)\cdot\sigma_{l-1}(\tau)\cdot X \\ & + & A(l\tau)\cdot\sigma_l(\tau), & & & \end{array}$$

und damit

$$\begin{aligned} H(X) = \quad & X^{l+1} \\ - \ & \left(\sigma_1(\tau) + A(l\tau)\right) \ X^l \\ + \ & \left(\sigma_2(\tau) + A(l\tau)\cdot\sigma_1(\tau)\right) X^{l-1} \\ & \vdots \\ - \ & \left(\sigma_l(\tau) + A(l\tau)\cdot\sigma_{l-1}(\tau)\right) \ X \\ + \ & A(l\tau)\cdot\sigma_l(\tau). \end{aligned}$$

Wir können somit die Koeffizienten von X^λ in $H(X)$ durch

$$(-1)^\lambda \left(\sigma_{l-\lambda+1}(\tau) + A(l\tau)\cdot\sigma_{l-\lambda}(\tau)\right) \quad \text{für} \quad 1 \leq \lambda \leq l \tag{6.19}$$

darstellen. Wir werden im folgenden Abschnitt nachweisen, daß es dann ausreicht, alle auftretenden Fourier-Reihen durch ungefähr $2lv$ viele Koeffizienten zu approximieren, falls v der Startexponent der q_τ-Entwicklung von $A(\tau)$ ist. Damit können wir zur Berechnung der $a_{\lambda,\kappa}$ einen ähnlichen Algorithmus verwenden, wie wir ihn bereits zum Koeffizientenvergleich des Polynoms für $\Gamma_0(l)$ verwendet haben (vgl. Abschnitt 5.2). Ziehen wir diese Methode zur Berechnung der gesuchten Koeffizienten heran, so erhalten wir

$$\sum_{\kappa=0}^{2v} a_{\lambda,\kappa}\cdot j(l\tau)^\kappa = (-1)^\lambda \left(\sigma_{l-\lambda+1}(\tau) + A(l\tau)\cdot\sigma_{l-\lambda}(\tau)\right) \quad \text{für} \quad 1 \leq \lambda \leq l$$

und

$$\sum_{\kappa=0}^{2v} a_{0,\kappa}\cdot j(l\tau)^\kappa = A(l\tau)\cdot\sigma_l(\tau).$$

Dabei berechnen wir die Funktionen $\sigma_\lambda(\tau)$ auch hier wieder mit Hilfe der Newton'schen Formeln aus den Potenzsummen

$$s_\rho(\tau) = \sum_{n=0}^{l-1} A\left(\tau + \frac{n}{l}\right)^\rho . \tag{6.20}$$

Diese können wir analog zur Berechnung des Polynoms für $\Gamma_0(l)$ aus der Fourier-Entwicklung der Potenzen von $A(\tau)$ berechnen (vgl. Seite 73). Geben wir uns diese q_τ-Entwicklungen als

$$A(\tau)^\rho = q_\tau^u \cdot \left[\sum_{\mu=0}^{\infty} a_\mu \cdot q_\tau^\mu\right]$$

vor, so erhalten wir

$$s_\rho(\tau) = q_\tau^u \cdot \left[\sum_{\mu=0}^{\infty} b_\mu \cdot q_\tau^\mu\right] . \tag{6.21}$$

Dabei gilt für die Koeffizienten b_μ:

$$b_\mu = \begin{cases} l \cdot a_\mu & , \quad \text{falls} \quad l \mid (\mu + u) \\ 0 & , \quad \text{falls} \quad l \nmid (\mu + u). \end{cases} \tag{6.22}$$

Daraus erhalten wir dann den folgenden Algorithmus.

Berechnung des Polynoms für $\Gamma_0^*(l)$

Eingabe : Approximationen der Fourier-Reihen für
$A \approx A(\tau)$, $A_l \approx A(l\tau)$ und $j_l \approx j(l\tau)$

Ausgabe : Feld der Koeffizienten $a_{\lambda,\kappa}$ für $0 \le \lambda \le l{+}1$ und $0 \le \kappa \le 2v$

<u>Leitkoeffizient 1 setzen</u>

```
for (κ = 1,...,2v) do
    a_{l+1,κ} = 0
od
a_{l+1,0} = 1
```

Berechnung des Polynoms für $\Gamma_0^*(l)$
(Fortsetzung)

<u>Initialisierung</u>

$\sigma_0(\tau) = 1$
$h(\tau) = 1$
for $(\lambda = 1, \ldots, l+1)$ **do** // Berechnung der $s_\lambda(\tau)$
  $h(\tau) = h(\tau) \cdot A$ //$h(\tau) = A(\tau)^\lambda$
  Berechne $s_\lambda(\tau)$ aus $h(\tau)$ mit (6.22)
od

<u>Berechne elem.symm. Funktionen in den ersten l Nullstellen</u>

for $(\lambda = 1, \ldots, l)$ **do**
  $\sigma_\lambda(\tau) = \frac{1}{\lambda} \cdot \sum_{i=1}^{\lambda} (-1)^{i-1} \cdot s_i(\tau) \cdot \sigma_{\lambda-i}(\tau)$ // vergleiche (5.27)
od

<u>Koeffizientenvergleich</u>

$h(\tau) = A_l \cdot \sigma_l(\tau)$
for $(\kappa = 2v, \ldots, 0)$ **do** // Berechn. von $a_{0,\kappa}$
  if $(-l\kappa <$ first $(h(\tau)))$ **then**
    $a_{0,\kappa} = 0$
  else
    $a_{0,\kappa} = \mathrm{lc}\ (h(\tau))$
    $h(\tau) = h(\tau) - a_{0,\kappa} \cdot j_l^\kappa$
  fi
od
for $(\lambda = 1, \ldots, l)$ **do**
  $h(\tau) = (-1)^\lambda \cdot (\sigma_{l-\lambda+1}(\tau) + A_l \cdot \sigma_{l-\lambda}(\tau))$
  for $(\kappa = 2v, \ldots, 0)$ **do**
    if $(-l\kappa <$ first $(h(\tau)))$ **then**
      $a_{\lambda,\kappa} = 0$
    else
      $a_{\lambda,\kappa} = \mathrm{lc}\ (h(\tau))$
      $h(\tau) = h(\tau) - a_{\lambda,\kappa} \cdot j(l\tau)^\kappa$
    fi
  od
od

6.3 Beschreibung der Implementierung

Wir haben nun zwei verschiedene Algorithmen beschrieben, Polynome für $\Gamma_0^*(l)$ zu berechnen. In der Praxis hat sich gezeigt, daß die Verwendung der elementarsymmetrischen Funktionen dem simultanen Koeffizientenvergleich vorzuziehen ist — am Ende dieses Abschnittes werden wir einige Laufzeitvergleiche angeben. Daher beschränken wir uns hier auf die Betrachtung des zuletzt vorgestellten Algorithmus. Wir werden, analog zum Abschnitt 5.4.1, zunächst berechnen, mit welcher Präzision die verwendeten approximierten Fourier-Reihen zur Verfügung gestellt werden müssen. Danach beschreiben wir einen Algorithmus, mit dem die Hecke-Operatoren T_n aus Definition 6.1 zu gegebenen Funktionen berechnet werden können. Abschließend geben wir dann einige praktische Laufzeitresultate an.

6.3.1 Präzision der Fourier-Reihen

Bei der Beschreibung des Algorithmus haben wir hergeleitet, daß wir einen Koeffizientenvergleich in den Gleichungen

$$\sum_{\kappa=0}^{2v} a_{\lambda,\kappa} \cdot j(l\tau)^\kappa \quad = \quad (-1)^\lambda \left(\sigma_{l-\lambda+1}(\tau) + A(l\tau)\cdot\sigma_{l-\lambda}(\tau)\right) \qquad \text{für} \quad 1 \le \lambda \le l$$

und

$$\sum_{\kappa=0}^{2v} a_{0,\kappa} \cdot j(l\tau)^\kappa \quad = \quad A(l\tau)\cdot\sigma_l(\tau)$$

durchführen müssen. Wir haben früher bereits gezeigt, daß alle dabei verwendeten Approximationen von Fourier-Reihen mindestens bis zu dem Term q_τ^0 bekannt sein müssen. Um daraus Aussagen über die Approximation der Funktion $A(\tau)$ zu erhalten, betrachten wir zunächst die Startexponenten aller hier auftretenden Summanden.

Sei dazu im folgenden stets die Approximation

$$A(\tau) \quad \approx \quad \sum_{m=-v}^{M_A} a_m \cdot q_\tau^m \tag{6.23}$$

der Funktion $A(\tau)$ gegeben. Insbesondere sei damit also

$$\text{first}\,(A(\tau)) \quad = \quad -v.$$

Dann können wir analog zu Satz 5.15 auf Seite 79 (bzw. Satz 5.16 auf Seite 80) zeigen, daß die Startexponenten der elementarsymmetrischen Funktionen

$$\begin{aligned} \text{first}\,(\sigma_i(\tau)) \quad &\ge \quad -i\cdot v \\ \text{last}\,(\sigma_i(\tau)) \quad &= \quad M_A-(i-1)\cdot v \qquad \text{für} \quad 0\le i\le l \end{aligned}$$

erfüllen. Diese Aussagen werden wir zum Beweis des folgenden Lemmas benötigen.

6.14. Lemma *Sei die Fourier-Entwicklung der Funktion $A(\tau)$ durch die Approximation (6.23) gegeben. Seien weiter $\sigma_\lambda(\tau)$ die elementarsymmterischen Funktionen in den Funktionen $A\left(\tau + \frac{n}{l}\right)$ für $0 \leq n < l$. Dann gilt für die approximierten Fourier-Entwicklungen im Koeffizientenvergleich*

$$\begin{array}{lll} \textit{first}\ (\sigma_{l-\lambda+1}(\tau) + A(l\tau)\cdot\sigma_{l-\lambda}(\tau)) & \geq & -2lv + \lambda v \quad \textit{für} \quad 1 \leq \lambda \leq l, \\ \textit{und} \qquad \textit{first}\ (A(l\tau)\cdot\sigma_l(\tau)) & \geq & -2lv. \end{array}$$

Beweis: Berechnen wir zum Beweis dieser Aussagen zunächst die Startexponenten der beteiligten Fourier-Reihen. Wir erhalten dann für $0 \leq \lambda \leq l$

$$\begin{aligned} \text{first}\ (A(l\tau)\cdot\sigma_{l-\lambda}(\tau)) &= \text{first}\ (A(l\tau)) + \text{first}\ (\sigma_{l-\lambda}(\tau)) \\ &\geq -lv - (l-\lambda)\cdot v \\ &= -2lv + \lambda v. \end{aligned}$$

Speziell gilt damit also auch

$$\text{first}\ (A(l\tau)\cdot\sigma_l(\tau)) \quad \geq \quad -2lv,$$

womit die zweite Aussage des Lemmas gezeigt ist. Zum Beweis der ersten Abschätzung, müssen wir jetzt nur noch überprüfen, ob für $1 \leq \lambda \leq l$ auch

$$\text{first}\ (\sigma_{l-\lambda+1}(\tau)) \quad \geq \quad -2lv + \lambda v$$

erfüllt ist. Dazu betrachten wir:

$$\begin{array}{rrcl} & -(l-\lambda+1)\cdot v & \geq & -2lv + \lambda v \\ \Longleftrightarrow & -lv + \lambda v - v & \geq & -2lv + \lambda v \\ \Longleftrightarrow & -v & \geq & -lv \\ \Longleftrightarrow & 1 & \leq & l. \end{array}$$

Dabei ist die letzte Aussage immer erfüllt. ∎

Betrachten wir nun analoge Aussagen über die höchsten bekannten Exponenten der Approximationen.

6.15. Lemma *Seien die Voraussetzungen und Bezeichnungen wie in Lemma 6.14. Dann gilt:*

$$\begin{array}{lll} \textit{last}\ (\sigma_{l-\lambda+1}(\tau) + A(l\tau)\cdot\sigma_{l-\lambda}(\tau)) & \geq & M_A - 2lv + (\lambda+1)v \quad \textit{für} \quad 1 \leq \lambda \leq l, \\ \textit{und} \qquad \textit{last}\ (A(l\tau)\cdot\sigma_l(\tau)) & \geq & M_A - 2lv + v. \end{array}$$

Beweis: Verwenden wir die Aussage

$$\text{last }(\sigma_{l-\lambda}(\tau)) \quad = \quad M_A - (l-\lambda-1)\cdot v,$$

für $0 \leq \lambda \leq l$ zusammen mit Lemma 6.14, so erhalten wir

$$\begin{aligned}\text{last }(A(l\tau)\cdot\sigma_{l-\lambda}(\tau)) \quad &\geq \quad \min\,(M_A - 2lv + (\lambda+1)\cdot v, lM_A - 2lv + \lambda v) \\ &= \quad M_A - 2lv + (\lambda+1)\cdot v.\end{aligned}$$

Vergleichen wir dies mit

$$\text{last }(\sigma_{l-\lambda+1}(\tau)) \quad = \quad M_A - (l-\lambda)\cdot v,$$

so können wir nachrechnen:

$$\begin{aligned} & M_A - 2lv + (\lambda+1)\cdot v \;\leq\; M_A - (l-\lambda)\cdot v \\ \Longleftrightarrow \quad & -lv + v \;\leq\; 0 \\ \Longleftrightarrow \quad & l \;\geq\; 1. \end{aligned}$$

Fassen wir diese Aussagen zusammen, so erhalten wir:

$$\text{last }(\sigma_{l-\lambda+1}(\tau) + A(l\tau)\cdot\sigma_{l-\lambda}(\tau)) \quad \geq \quad M_A \;-\; 2lv \;+\; (\lambda+1)v \quad \text{für} \quad 1 \leq \lambda \leq l,$$

und das Lemma ist bewiesen. ■

Mit Hilfe der soeben gezeigten Aussagen über die Approximationen der verwendeten Reihen, können wir erkennen, daß der kleinste maximale Exponent, der je unter den Ausdrücken auftritt, gegeben wird durch

$$M_A \;-\; 2lv \;+\; v.$$

Wollen wir gewährleisten, daß alle auftretenden Approximationen von Fourier-Reihen mindestens bis zum Term q_τ^0 bekannt sind, so können wir dies erreichen, indem wir die Approximantion (6.23) der Funktion $A(\tau)$ bis zum Exponenten M_A berechnen, der der folgenden Bedingung genügt:

$$M_A \;>\; 2lv \;+\; v. \tag{6.24}$$

6.3.2 Berechnung des Hecke-Operators

Wir haben in Abschnitt 6.1 beschrieben, wie wir mit Hilfe von *Hecke-Operatoren* Funktionen mit Invarianzgruppe $\Gamma_0(l)$ berechnen können. Für die Funktion $g(\tau) = \eta(\tau)\cdot\eta(l\tau)$ und eine Menge von Hecke-Operatoren T_n haben wir definiert:

$$A(\tau) \quad = \quad \frac{T_n(g(\tau))}{g(\tau)}.$$

Wir haben uns dabei beschränkt auf Hecke-Operatoren T_n mit $n \in \mathbb{P}$. In der Praxis haben wir jedoch festgestellt, daß auch solche Operatoren verwendet werden können, deren Index *keine* Primzahl ist. Allerdings müssen wir dann noch weitere Forderungen an diese Indizes stellen. Seien dazu im folgenden wieder r und s so gewählt, daß

$$\frac{l+1}{24} = \frac{r}{s} \quad \text{mit} \quad \text{ggT}\ (r,s) = 1.$$

Dann beschränken wir uns auf diejenigen Operatoren T_n, deren Index n die folgenden Eigenschaften erfüllt.

$$d_1 \mid n \quad \text{und} \quad d_2 \mid n \quad \Longrightarrow \quad d_1^2 \equiv d_2^2 \pmod{s}. \tag{6.25}$$

Weiterhin muß gelten:

$$\text{ggT}\ (n,s) = 1. \tag{6.26}$$

6.16. Bemerkung Sind T_{n_1} und T_{n_2} zwei Hecke-Operatoren, die die geforderten Eigenschaften besitzen, so können wir diese auch linear kombinieren, d.h. wir erhalten die Funktion

$$A(\tau) = \frac{c_1 \cdot (T_{n_1} g)(\tau) + c_2 \cdot (T_{n_2} g)(\tau)}{g(\tau)}.$$

Sei die Funktion $g(\tau) = \eta(\tau)\eta(l\tau)$ durch ihre approximierte Fourier-Reihentwicklung gegeben:

$$\begin{aligned} g(\tau) &= \eta(\tau) \cdot \eta(l\tau) \\ &= \left(q_\tau^{1/24} \cdot \sum_{i=0}^{M_g} b_i \cdot q_\tau^i \right) \cdot \left(q_\tau^{l/24} \cdot \sum_{i=0}^{M_g} b_i \cdot q_\tau^{li} \right) \\ &= q_\tau^{(l+1)/24} \cdot \sum_{i=0}^{M_g} \tilde{b}_i \cdot q_\tau^i. \end{aligned} \tag{6.27}$$

Dann hat $g(s\tau)$ eine Fourier-Reihenentwicklung mit ganzzahligen Exponenten:

$$g(s\tau) = q_\tau^r \cdot \sum_{i=0}^{M_g} \tilde{b}_i \cdot q_\tau^{si} = \sum_{i=0}^{s \cdot M_g} c(i) \cdot q_\tau^i. \tag{6.28}$$

Damit können wir nun mit Hilfe von [Ap89, Theorem 6.6] die Funktion $T_n(g(s\tau))$ berechnen. Es gilt der folgende Satz:

6.17. Satz *Sei die Funktion $g(s\tau)$ durch (6.28) gegeben. Dann gilt für die (approximierte) Entwicklung von $T_n(g(s\tau))$*

$$T_n(g(s\tau)) \quad = \quad \sum_{m=0}^{M_T} \gamma_n(m) \cdot q_\tau^m \tag{6.29}$$

mit den in Satz 6.2 definierten Koeffizienten $\gamma_n(m)$.

Dabei gilt für die Präzision der Approximation (6.29)

$$M_T \quad = \quad \left\lfloor \frac{s \cdot M_g}{n} \right\rfloor .$$

Beweis: Die Gestalt der Approximation (6.29) folgt sofort aus Satz 6.2. Wir müssen also nur noch die Aussage über die Präzision M_T begründen.

Der Koeffizient $\gamma_n(m)$ kann berechnet werden, falls in (6.28) für jeden gemeinsamen Teiler d von n und m der zugehörige Wert für $c(\frac{mn}{d^2})$ bekannt ist. Der größte dabei auftretende Index ist $m \cdot n$. Daraus können wir sehen, daß $\gamma_n(m)$ genau dann korrekt bestimmt werden kann, wenn

$$m \cdot n \quad \leq \quad s \cdot M_g \qquad \Longleftrightarrow \qquad m \quad \leq \quad \left\lfloor \frac{s \cdot M_g}{n} \right\rfloor = M_T$$

erfüllt ist. Die Präzision von $T_n(g(s\tau))$ ist also gleich M_T. ∎

Mit Hilfe dieser Aussagen können wir nun sofort einen Algorithmus zur Berechnung von $T_n(g(s\tau))$ angeben.

Berechnung des n-ten Hecke-Operators für $g(s\tau)$

EINGABE : Approximation für $g(s\tau) : \sum_{i=0}^{s\cdot M_g} c(i) \cdot q_\tau^i$
AUSGABE : Approximation für $T_n(g(s\tau))$ mit Koeff. $\gamma_n(m)$

(1) k = Anzahl der Teiler von n
(2) **for** (i=1,...,k) **do**
(3) d_i = i-ter Teiler von n
(4) **od**
(5) **for** ($m = 0, \ldots, M_T$) **do**
(6) $\gamma_n(m) = 0$
(7) **for** ($i = 0, \ldots, k$) **do**
(8) **if** ($m \bmod d_i == 0$) **then** // gemeins. Teiler von n, m
(9) $j = \frac{m}{d} \cdot \frac{n}{d}$
(10) $\gamma_n(m) = \gamma_n(m) + c(j)$
(11) **fi**
(12) **od**
(13) **od**

In der Praxis verwenden wir ein Verfahren, welches die „Struktur" von $g(s\tau)$ berücksichtigt. Betrachten wir dazu noch einmal die q_τ-Entwicklungen von $g(\tau)$ und $g(s\tau)$

$$\begin{aligned} g(\tau) &= q_\tau^{r/s} \cdot \sum_{i=0}^{M_g} \tilde{b}_i \cdot q_\tau^i \\ \text{und} \qquad g(s\tau) &= q_\tau^r \cdot \sum_{i=0}^{s\cdot M_g} \tilde{b}_i \cdot q_\tau^{si} = \sum_{i=0}^{s\cdot M_g} c(i) \cdot q_\tau^i, \end{aligned}$$

so können wir erkennen, daß die Exponenten in der Darstellung von $g(s\tau)$, mit von 0 verschiedenen Koeffizienten, alle von der Form $k \cdot s + r$ sind — diese sind also insbesondere *kongruent r modulo s*. Zur der Berechnung von $T_n(g(s\tau))$ müssen wir die Koeffizienten

$$\gamma_n(m) = \sum_{d|\,\mathrm{ggT}(n,m)} c\left(\frac{mn}{d^2}\right)$$

berechnen. Darin sind die Summanden $c(\frac{m}{d} \cdot \frac{n}{d}) = 0$, falls bereits $\frac{m}{d} \cdot \frac{n}{d}$ kleiner ist als der Startexponent r von $g(s\tau)$. Nur dann sind diese Koeffizienten von Null verschieden, falls die zugehörigen Exponenten

$$\frac{m \cdot n}{d^2} \equiv r \pmod{s} \tag{6.30}$$

erfüllen. Da n und s teilerfremd sind, ist n invertierbar modulo s und wir erhalten mit $m = \mu \cdot d$

$$\mu \equiv r \cdot d \cdot n^{-1} \pmod{s}. \tag{6.31}$$

Wir können daraus ablesen: Für einen Teiler d von n trägt der Koeffizient $c(\frac{m}{d} \cdot \frac{n}{d})$ für $m = \mu \cdot d$ genau dann zu $\gamma_n(m)$ bei, falls μ die Kongruenz (6.31) erfüllt und $\mu \cdot \frac{n}{d} \geq r$ ist. Ist $\mu_d \geq 0$ minimal gewählt mit diesen Eigenschaften, und ist weiter $m_d = \mu_d \cdot d$, so werden die $c\left(\frac{m}{d} \cdot \frac{n}{d}\right)$ zur Berechnung von $\gamma_n(m)$ genau für $m = (\mu_d + ks) \cdot d = m_d + ksd$ $(k \geq 0)$ benötigt.

Daraus leiten wir das folgende Verfahren ab, um $T_n(g(s\tau))$ bis zum Exponenten M_T zu berechnen:

1. Initialisiere alle Koeffizienten $\gamma_n(m)$ mit 0 für $m \geq 0$
2. Für jeden Teiler d von n berechne:
   - $\mu_d \geq 0$ minimal mit (6.31) und $\left(\mu_d \cdot \frac{n}{d}\right) \geq r$
   - für alle $k \geq 0$ mit $m_d(k) := m_d + ksd \leq M_T$ addiere die jeweiligen Koeffizienten von $g(s\tau)$ zu $\gamma_n(m_d(k))$

Wir erhalten den folgenden Algorithmus:

Berechnung des n-ten Hecke-Operators für $g(s\tau)$

EINGABE : Approximation für $g(s\tau) : \sum_{i=0}^{s \cdot M_g} c(i) \cdot q_\tau^i$
AUSGABE : Approximation für $T_n(g(s\tau))$ mit Koeff. $\gamma_n(m)$

<u>INITIALISIERUNG</u>

(1) $n' = n^{-1} \bmod s$
(2) k = Anzahl der Teiler von n
(3) **for** (i=1,...,k) **do**
(4) $\quad d_i = i$-ter Teiler von n
(5) **od**

Berechnung des n-ten Hecke-Operators für $g(s\tau)$
(Fortsetzung)

<u>Berechnung der $\mu_i := \mu_{d_i}$ und $m_i = m_{d_i}$</u>

for $(i = 0, \ldots, k)$ **do**
  $\mu_i = (r \cdot d \cdot n')\bmod s$ // vgl. (6.31)
  while $(\mu_i \cdot \frac{n}{d_i} < r)$ **do**   // suche min. μ_i
    $\mu_i = \mu_i + s$
  od
  $m_i = \mu_i \cdot d_i$
od

<u>Berechnung der Koeffizienten $\gamma_n(m)$</u>

for $(i = 0, \ldots, k)$ **do**
  while $(m_i \leq M_T)$ **do**
    $\gamma_n(m_i) = \gamma_n(m_i) + c\left(\mu_i \cdot \frac{n}{d_i}\right)$
    $\mu_i = \mu_i + s$ // nächster Exp. in g
    $m_i = m_i + sd$
  od
od

6.3.3 Praktische Resultate

Betrachten wir auch hier wieder praktische Laufzeiten. Wir haben dazu für einige Primzahlen l die zugehörigen Polynome modulo einer Primzahl der Größenordnung 2^{26} berechnet. Die Tests wurden auf einer SUN SPARC STATION ELC durchgeführt.

Wir betrachten zunächst den Algorithmus, der die Koeffizienten des gesuchten Polynoms mit Hilfe der elementarsymmetrischen Funktionen berechnet. Dabei trennen wir die einzelnen Phasen dieses Verfahrens.

l	Potenzen von $A(\tau)$	elem.Fkt.	K.vergleich
307	11 min 00 sec	4 min 00 sec	2 sec
349	12 min 28 sec	5 min 52 sec	3 sec
401	14 min 18 sec	8 min 57 sec	3 sec
449	34 min 19 sec	13 min 54 sec	5 sec
503	38 min 28 sec	17 min 03 sec	5 sec
557	1 h 42 min 4 sec	40 min 00 sec	12 sec
599	45 min 50 sec	34 min 45 sec	9 sec
641	1 h 57 min 29 sec	1 h 3 min 14 sec	16 sec
719	2 h 11 min 52 sec	1 h 22 min 50 sec	18 sec

Vergleichen wir nun noch die Gesamtlaufzeiten dieses Algorithmus mit den Laufzeiten des simultanen Koeffizientenvergleiches.

l	simultaner Koeff.Vergleich	Verwendung der elem.symmetr.Fkt.
307	53 min	16 min
349	1 h 15 min	19 min
401	1 h 44 min	25 min
449	3 h 24 min	51 min
503	3 h 31 min	58 min
557	12 h 15 min	2 h 32 min
599	7 h 50 min	1 h 26 min
641	15 h 44 min	3 h 14 min
719	17 h 00 min	3 h 49 min

Wir haben die Betrachtung der Polynome für $\Gamma_0^*(l)$ (Typ 2) damit begründet, daß die Approximationen der Fourier-Reihen bei den zuerst beschriebenen Polynomen in der Praxis zu groß werden. Daher wollen wir abschließend untersuchen, wie sich im Vergleich dazu die Reihenlängen bei den hier vorgestellten Algorithmen verhalten.

l	Reihenlänge (Typ 1)	Reihenlänge (Typ 2)
307	15657	6140
317	25043	6974
359	64261	4308
443	97903	10632
503	126253	10060
547	49777	19692
599	179101	14376
641	102560	20512
701	122675	29442
719	258121	21570

Kapitel 7

Die Programme

In diesem Kapitel beschreiben wir zunächst eine verteilte Anwendung zur Berechnung der beschriebenen Polynome. Danach geben wir eine Auflistung der wichtigsten Programme und Funktionen an.

7.1 Verteilte Anwendung mit LiPS

Auf Seite 85 haben wir bereits beschrieben, wie wir die Koeffizienten der gesuchten Polynome in der Praxis berechnen. Wir müssen die angegebenen Algorithmen dazu modulo mehrerer Primzahlen durchführen und die Ergebnisse mit Hilfe des Chinesischen Restsatzes kombinieren. Die Struktur dieses Verfahrens eignet sich sehr gut zur Parallelisierung. Daher haben wir im Rahmen der Implementierung der Algorithmen auch eine verteilte Version des Programmes entwickelt.

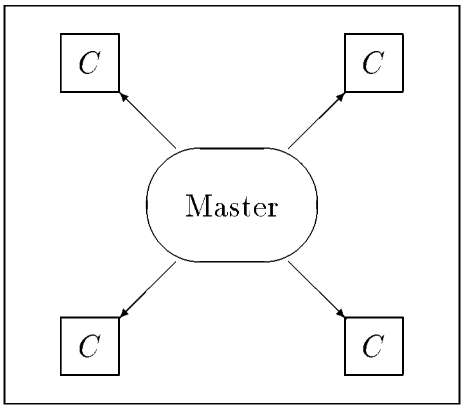

Abbildung 7.1: Verteilte Anwendung mit LiPS

Für die verteilte Anwendung haben wir die Bibliothek LiPS verwendet, die es erlaubt, Programme auf ein Netz von Workstations zu verteilen (siehe auch [Li93]). In einem Netz von Rechnern befindet sich ein ausgezeichneter *Master*. Dieser verteilt dann *Jobs* an alle *Klienten* im Verbund und erhält von diesen ggf. Ergebnisse von Berechnungen zurück.

In unserer Anwendung besteht ein Job darin, die Koeffizienten des gesuchten Polynoms modulo einer Primzahl q zu berechnen. Hat einer der Klienten den gesamten Algorithmus abgearbeitet, sendet er die berechneten Koeffizienten zu dem Master zurück und erwartet einen neuen Job.

Erhält der Master von einem der Klienten die Ergebnisse von dessen Berechnungen, so wird eine Routine gestartet, die mit Hilfe des Chinesischen Restsatzes die bereits bekannten Resultate mit den neuen kombiniert. Danach wird getestet, ob sich die Koeffizienten verändert haben oder nicht. Bleiben diese in zwei aufeinanderfolgenden Iterationen unverändert, so speichert das Programm die Koeffizienten in einer Datei und terminiert. Wir werden nun die Struktur der Programme für Master und Klienten kurz zusammenfassen.

MEQ-Master

EINGABE : Primzahl 1

AUSGABE : Die ganzzahligen Koeffizienten eines zum l-ten modularen Polynom äquivalenten Polynoms

Starte alle zur Verfügung stehenden Klienten
Sende l zu allen Klienten
Erzeuge eine Liste $q_1, q_2, \ldots, q_n$ von Primzahlen
Sende zu jedem Klient je eine dieser Primzahlen
while (not ready) **do**
    Warte auf Ergebnisse eines Klienten C
    Kombiniere diese mit den bekannten Koeff.
    Prüfe, ob die Koeff. verändert wurden
    if (Koeff. zweimal unverändert) **then**
        ready = TRUE
        Stoppe alle Klienten
    else
        Sende neue Primzahl q zu Klient C
    fi
od
Speichere die Koeffizienten in einer Datei

In diesem Programm wird zusätzlich eine Liste geführt, um feststellen zu können, welche der Primzahlen q_i bereits an die Klienten verteilt wurden. Sind alle q_i schon einmal versendet worden, so werden diejenigen aus der Liste, für die noch keine Resultate zurückgeschickt wurden, wiederum verteilt.

MEQ-Worker
(1) Empfange Primzahl l (2) Führe Initialisierungen durch (3) **while** (TRUE) **do** (4) Empfange Primzahl q vom Master (5) Berechne die Koeffizienten des Polynoms modulo q (6) Sende Ergebnisse zurück zum Master (7) **od**

Dieses Programm führt jeder Klient solange aus, bis die gesamte Anwendung durch MEQ-Master beendet wird.

Die Ausgabedateien, die von MEQ-Master erzeugt werden, tragen die Namen „out...“ und enthalten die berechneten Koeffizienten in der folgenden Struktur.

$$\begin{array}{cccc} a_{0,0} & a_{0,1} & \dots & a_{0,v} \\ a_{1,0} & a_{1,1} & \dots & a_{1,v} \\ \vdots & \vdots & \dots & \vdots \\ a_{l+1,0} & a_{l+1,1} & \dots & a_{l+1,v} \end{array}$$

Diese Dateien werden zunächst im ASCII-Format erzeugt. Zur Konvertierung dieser Files stehen zwei Hilfsprogramme zur Verfügung:

- Das Programm **atobin** wandelt eine von MEQ-Master erzeugte Datei von Koeffizienten in ein binäres Format um. Dabei werden die Datenstrukturen der Langzahl-Arithmetik **libI** verwendet. Die Ausgabedateien von „atobin“ erhalten die Namenserweiterung „.bin“.

- Das Hilfsprogramm **bintoa** konvertiert die von „atobin“ erzeugten Dateien wieder zurück in ASCII-Format. Die Ausgabedateien von „bintoa“ erhalten die Namenserweiterung „.asc“.

Mit den Programmen MEQ-Master und MEQ-Worker haben wir bisher für jede Primzahl $l < 800$ ein zum l-ten modularen Polynom äquivalentes Polynom berechnet. Diese stehen damit für die Verwendung im Algorithmus von Atkin zur Verfügung.

7.2 Programme und Funktionen

Die folgenden Programme stehen zur Berechnung der Polynome zur Verfügung:

- **MEQ-Master:** Master-Programm für die verteilte Anwendung
- **MEQ-Worker:** Programm für die Klienten in der verteilten Anwendung
- **atobin:** Konvertierung der Ausgabedateien von MEQ-Master in ein binäres Format
- **bintoa:** Konvertierung der Ausgabedateien von atobin in ASCII-Format
- **meq:** Einzelrechner-Version zur Berechnung der Polynome über $\mathbb{Z}$

Für die wichtigsten Funktionen der Implementierung geben wir an, in welchen Dateien sie zu finden sind und beschreiben in Stichworten deren Aufgabe.

Funktionen zur Berechnung der Polynome

Datei	Funktion	Aufgabe
lmod_equ.c	lmod_equ()	Steuerungsfunktion zur Berechnung der Polynome
examine_l.c	examine_l()	liefert Typ (1 oder 2) des Polynoms für eine Primzahl l sowie den Index des Hecke-Operators
lcompute_eta.c	lcompute_eta_de()	Berechnung der η-Funktion in *dichter* Darstellung
	lcompute_eta_sp()	Berechnung der η-Funktion in *dünner* Darstellung
lheckeop.c		verschiedene Funktionen zur Berechnung von Hecke-Operatoren

Datei	Funktion	Aufgabe
lcompute_f.c	lcompute_f()	Berechnung der Funktion $f(\tau)$ aus (5.5)
	lcompute_j()	Berechnung der j-Invarianten $j(\tau)$
lcompute_A.c	lcompute_A()	Berechnung der Funktion $A(\tau)$ aus (6.3)
lcompute_ark.c	lcompute_ark()	Koeff.vergl. für Polynom zu $\Gamma_0(l)$ (Typ 1)
lcompute_ars.c	lcompute_ars()	Koeff.vergl. für Polynom zu $\Gamma_0^*(l)$ (Typ 2)

Funktionen der Fourier-Reihen-Arithmetik

Datei	Funktion	Aufgabe
ldense_psr.c		Funktionen für Datentyp `dense_powser`
	lgenpsr_de()	Konstruktor-Funktion
	lfreepsr_de()	Destruktor-Funktion
	lcopypsr_de()	Zuweisung
	laddpsr_de()	Addition
	lsubpsr_de()	Subtraktion
	lplainmultpsr_de()	Multiplikation (Standard-Algorithmus)
	lfftmult_psr_de()	Multiplikation (Verwendung von FFT)
	linvpsr_de()	Inversion
	lpotpsr_de()	Schnelle Exponentiation
lsparse_psr.c		Funktionen für Datentyp `sparse_powser`
	lgenpsr_sp()	Konstruktor-Funktion
	lfreepsr_sp()	Destruktor-Funktion
	lcopypsr_sp()	Zuweisung
	laddpsr_sp()	Addition
	lsubpsr_sp()	Subtraktion
	lmultpsr_de()	Multiplikation (Standard-Algorithmus)
	linvpsr_sp()	Inversion
	lpotpsr_sp()	Schnelle Exponentiation

Literaturverzeichnis

[Ap89] T.M. Apostol: *Modular Function and Dirichlet Series in Number Theory*, 2. Auflage Springer-Verlag 1989

[At88] A.O.L. Atkin: *The number of points on an elliptic curve modulo a prime*, Manuskript, veröffentlicht über das Number Theory Net 1988

[At92] A.O.L. Atkin: *The number of points on an elliptic curve modulo a prime (II)*, Manuskript, veröffentlicht über das Number Theory Net 1992

[BuMü91] Johannes Buchmann, Volker Müller: *Computing the Number of Points of Elliptic Curves over Finite Fields*, Proceedings of ISSAC 91, ACM Press, 1991, Seite 179-82

[CLR90] T.H. Cormen, C.E. Leiserson, R.L. Rivest: *Introduction to Algorithms*, The MIT-Press 1990

[FiLieb80] W. Fischer, I. Lieb *Funktionentheorie*, Vieweg-Verlag 1980

[Ko86] Neal Koblitz: *Elliptic curve cryptosystems*, Math. Comp. 48, 1987, Seite 203-209

[La87] Serge Lang: *Elliptic Functions*, Springer-Verlag 1987

[Li93] Ralf Roth, Thomas Setz: *LiPS, a system for distributed processing on workstations*, Universität des Saarlandes, 1993

[LMMS94] F. Lehmann, M. Maurer, V. Müller, V. Shoup: *Counting the Number of Points on Elliptic Curves over Finite Fields of Characteristic Greater than Three*, Proceedings of ANTS I, 1994

[Ma94] Markus Maurer: *Eine Implementierung des Algorithmus von Atkin* , Diplomarbeit 1994; Universität des Saarlandes

[Mey1] K. Meyberg: *Algebra Teil 1*, 2. Auflage Hanser-Verlag 1980

[Mey2] K. Meyberg: *Algebra Teil 2*, 1. Auflage Hanser-Verlag 1976

[Mi86] Victor Miller: *Uses of elliptic curves in cryptography*, Advances in Cryptology: Proceedings of Crypto '85, Lecture Notes in Computer Science 218, 1986, Springer-Verlag, Seite 417-426

[Mü91] Volker Müller: *Die Berechnung der Punktanzahl elliptischer Kurven über endlichen Primkörpern*, Universität des Saarlandes, Diplomarbeit 1991

[Mü94] Volker Müller: *Die Berechnung der Punktanzahl elliptischer Kurven über endlichen Körpern der Charakteristik größer 3* , Universität des Saarlandes, Dissertation, voraussichtliche Veröffentlichung: 1994

[Sc74] Bruno Schoeneberg: *Elliptic Modular Functions*, Springer-Verlag 1974

[Sho94] Victor Shoup: *Practical computation with large polynomials over finite fields* in Vorbereitung (1994)

[Si85] Joseph Silverman: *The Arithmetic Of Elliptic Curves*, Springer-Verlag 1985

[Wa50] B.L. van der Waerden: *Moderne Algebra*, 3.Auflage Springer-Verlag 1950

[We08] Heinrich Weber: *Lehrbuch der Algebra, Band 3*, Chelsea Publishing Company New York 1908