

Extremale Gitter mit großen Automorphismen

MASTERARBEIT

von Simon Berger

Vorgelegt am Lehrstuhl D für Mathematik der RWTH-Aachen University

bei Prof. Dr. Gabriele Nebe (Erstgutachterin)
und Priv.-Doz. Dr. Markus Kirschmer (Zweitgutachter)

21. September 2018

Inhaltsverzeichnis

1	Einleitung	4
2	Grundbegriffe	6
2.1	Bilineare Vektorräume	6
2.2	Modulare Gitter	9
3	Ideal-Gitter	15
3.1	Definitionen	15
3.2	Strategie zur Klassifikation	19
3.3	Klassengruppe	22
3.4	Total-positive Erzeuger	23
3.5	Finaler Algorithmus und Ergebnisse	30
4	Sub-Ideal-Gitter	33
4.1	Einführung	33
4.2	Automorphismen von Primzahlordnung	35
4.3	Geschlechter	47
4.4	Kneser-Nachbarschaftsmethode	54
4.5	Konstruktion von Obergittern	59
4.6	Konstruktion von Gittern mit großem Automorphismus	62
4.7	Vollständigkeit der Ergebnisse	68

5	Zusammenfassung und Ausblick	76
6	Anhang	78
6.1	Ergebnisse der Ideal-Gitter-Klassifikation	78
6.2	Ergebnisse der Subideal-Gitter-Klassifikation	85
6.3	MAGMA-Implementierungen von Hilfsfunktionen	91
6.4	MAGMA-Implementierungen der Ideal-Gitter-Algorithmen	98
6.5	MAGMA-Implementierungen der Subideal-Gitter-Algorithmen	107
6.6	MAGMA-Implementierungen zur Aufzählung der charakteristischen Polynome	127
6.7	Eidesstattliche Versicherung	133
7	Literaturverzeichnis	134

1 Einleitung

Die Gittertheorie ist seit vielen Jahren ein wesentlicher Bestandteil der theoretischen Mathematik in Themengebieten wie der algebraischen Zahlentheorie und der Gruppentheorie. In der Anwendung sind oftmals Gitter von Interesse, welche besonders dichte Kugelpackungen liefern. Also Gitter, die im Vergleich zu ihrer Determinante ein möglichst großes Minimum besitzen. Die Klassifikation möglichst dichter Gitter stellt jedoch eine große Herausforderung dar.

H.-G. Quebbemann definiert in seiner Arbeit [Que95] den Begriff eines *modularen* Gitters und zeigt, dass die Thetareihen solcher modularen Gitter Modulformen einer bestimmten Gruppe sind. Diese Struktur erlaubt die Beschreibung sogenannter *extremaler* modularer Gitter, welche innerhalb der Klasse der modularen Gitter eine maximale Dichte haben. Die Klassifikation extremaler Gitter ist eines der großen Forschungsgebiete aus der Gittertheorie. In dieser Arbeit wird versucht, extremale Gitter mithilfe ihrer Automorphismengruppen zu untersuchen und zu konstruieren.

Zunächst definieren wir dazu unsere grundlegenden Begriffe und werfen einen genaueren Blick auf die Dichte extremaler Gitter. Anschließend beschreiben wir modulare Gitter L mit einer Struktur als gebrochene Ideale eines zyklotomischen Zahlkörpers $\mathbb{Q}(\zeta_m)$, sogenannte *Ideal-Gitter*. Diese Struktur ist gegeben, falls L einen Automorphismus σ mit $\mu_\sigma = \Phi_m$ besitzt, sodass $\varphi(m) = \text{Dim}(L)$ gilt. Mithilfe dieser zusätzlichen Struktur lässt sich ein Algorithmus entwickeln, welcher zu gegebener Stufe, Determinante

und Dimension eine vollständige Liste der Ideal-Gitter mit den festgelegten Parametern konstruiert. Dieser Algorithmus wurde von Michael Jürgens in [Jü15] beschrieben. Die einzelnen Schritte werden im Zuge dieser Arbeit genau untersucht und im Computer-Algebra-System **MAGMA** implementiert.

Im sich daran anschließenden Hauptteil der Arbeit erfolgt die Beschreibung extremaler modularer Gitter mit *großem* Automorphismus. Hier gehen wir lediglich von den Voraussetzungen $|\sigma| = m$ und $\Phi_m | \mu_\sigma$ mit $\varphi(m) > \frac{\dim(L)}{2}$ aus. In diesem Falle induziert σ ein Teilgitter $M := L \cap \text{Kern}(\Phi_m(\sigma)) \perp L \cap \text{Kern}(\frac{\mu_\sigma}{\Phi_m}(\sigma))$. Die erste Komponente hat dabei eine Struktur als Ideal-Gitter und kann deshalb mit dem Algorithmus des vorherigen Kapitels leicht konstruiert werden; für die zweite Komponente benötigen wir weitere Theorie. Jürgens und Nebe untersuchen in [Jü15] und [Neb13] Gitterautomorphismen σ mit Primzahlordnung p . Wie vorher induziert solch ein σ ein Teilgitter $M := L \cap \text{Kern}(\Phi_p(\sigma)) \perp L \cap \text{Kern}(\Phi_1(\sigma))$, in diesem Falle lassen sich allerdings wichtige Einschränkungen an die Determinanten der Komponenten sowie an den Index von M in L zeigen. Die Informationen über Automorphismen von Primzahlordnung können anschließend verwendet werden, um mittels der Primteiler von m auch Automorphismen allgemeinerer Ordnung zu untersuchen. Nebe wendet in [Neb13] diese Erkenntnisse an, um extremale unimodulare Gitter der Dimension 48 mit großem Automorphismus zu klassifizieren. In dieser Arbeit wird die dortige Vorgehensweise auf ℓ -modulare Gitter verallgemeinert und als Algorithmus zusammengefasst. Alle Voraussetzungen und Teilschritte werden dabei genau erläutert und in **MAGMA** implementiert. Auf diese Weise konnten insgesamt 98 bisher unbekannte extremale modulare Gitter konstruiert werden. Abschließend wird die Vollständigkeit der Ergebnisse genauer erläutert. Dazu zählen wir alle Möglichkeiten für die charakteristischen Polynome von Gittern auf, die nicht die Voraussetzungen des Algorithmus erfüllen und somit gegebenenfalls nicht gefunden werden konnten.

2 Grundbegriffe

§ 2.1 Bilineare Vektorräume

Wir wiederholen zunächst einige wichtige Begriffe aus der Gittertheorie, welche wir in der Arbeit häufig benötigen werden. Zunächst führen wir das Konzept eines bilinearen Vektorraumes ein. Die nun angeführten Definitionen sind [Kne02, Def. (2.1)] entnommen.

(2.1.1) Definition

- (i) Sei A ein Ring und E ein A -Modul. Für eine symmetrische Bilinearform $b : E \times E \rightarrow A$ heißt das Paar (E, b) ein *bilinearer A -Modul* (bzw. falls A ein Körper ist ein *bilinearer A -Vektorraum*).
- (ii) Eine *isometrische Abbildung* (oder kurz *Isometrie*) zwischen zwei bilinearen Moduln (E, b) und (E', b') ist ein Modulisomorphismus $f : E \rightarrow E'$ mit $b(x, y) = b'(f(x), f(y))$.
- (iii) Die Gruppe $O(E, b) := \{f : E \rightarrow E \mid f \text{ ist Isometrie}\}$ aller Isometrien eines bilinearen Vektorraums (E, b) in sich selbst heißt die *Isometriegruppe* von (E, b) .

Nun folgen Definitionen zum Gitterbegriff, zu finden in [Kne02, Def. (14.1), (14.2)].

(2.1.2) Definition

- (i) Sei K ein Körper, V ein endlich-dimensionaler K -Vektorraum mit Basis (b_1, \dots, b_n) . Ein R -Gitter in V ist ein R -Untermodul L von V , zu dem Elemente $a, b \in K^*$ existieren mit $a \sum_{i=1}^n Rb_i \subseteq L \subseteq b \sum_{i=1}^n Rb_i$.
- (ii) Sei b eine nicht-ausgeartete symmetrische Bilinearform auf V und L ein Gitter in V . Dann ist auch $L^\# := \{x \in V \mid b(x, y) \in R \text{ für alle } y \in L\}$ ein R -Gitter und heißt *das zu L duale Gitter* (bzgl. b).
- (iii) Für $m \in \mathbb{N}$ heißt das Gitter $L^{\#,m} := \frac{1}{m}L \cap L^\#$ *partielles Dualgitter* von L .
- (iv) Sei (V, b) ein bilinearer K -Vektorraum und L ein Gitter in V . Die Gruppe $\text{Aut}(L) := \{\sigma : V \rightarrow V \mid \sigma \text{ ist Isometrie und } \sigma(L) = L\}$ heißt die *Automorphismengruppe* von L .

(2.1.3) Bemerkung

Falls R ein Hauptidealbereich ist, vereinfacht sich die Definition erheblich, da Teilmoduln von endlich erzeugten freien Moduln über Hauptidealbereichen wieder frei sind. Ein R -Gitter ist per Definition zwischen zwei freien Moduln eingespannt, also sind die R -Gitter in diesem Fall genau die freien R -Moduln von Rang n .

Insbesondere interessieren uns \mathbb{Z} -Gitter in \mathbb{R}^n . Für diese folgen nun ein paar weitere aus [Kne02, Def. (1.7), (1.13), (14.7), (26.1)] abgeleitete Definitionen.

(2.1.4) Definition

Sei L ein \mathbb{Z} -Gitter mit Basis $B = (e_1, \dots, e_n)$ in (\mathbb{R}^n, b) , für eine symmetrische Bilinearform b .

- (i) Die Matrix $G := \text{Gram}(B) = (b(e_i, e_j))_{i,j=1}^n$ heißt *Gram-Matrix* von L , $\text{Det}(L) := \text{Det}(G)$ heißt die *Determinante* von L .
- (ii) Das Gitter L heißt *ganz*, falls $b(L, L) \subseteq \mathbb{Z}$ gilt.
- (iii) Das Gitter L heißt *gerade*, falls $b(x, x) \in 2\mathbb{Z}$ für alle $x \in L$ gilt.
- (iv) Die *Stufe* von L ist die kleinste Zahl $\ell \in \mathbb{N}$, sodass $\sqrt{\ell}L^\#$ ein gerades Gitter ist.
- (v) Das *Minimum* von L ist definiert als $\text{Min}(L) := \min\{b(x, x) \mid 0 \neq x \in L\}$.

(2.1.5) Bemerkung

- (i) Nach [Kne02, Satz (14.7)] gilt $\text{Det}(L) = |L^\# / L|$. Insbesondere ist die Determinante für \mathbb{Z} -Gitter unabhängig von der Wahl der Basis. Allgemeiner ist die Determinante von R -Gittern modulo $(R^*)^2$ eindeutig bestimmt [Kne02, (1.13)].
- (ii) Direkt aus der Definition des dualen Gitters folgt: L ist genau dann ganz, wenn $L \subseteq L^\#$.
- (iii) Ein gerades Gitter L ist notwendigerweise ganz; denn seien $x, y \in L$, dann ist

$$b(x, y) = \frac{b(x + y, x + y) - b(x, x) - b(y, y)}{2} \in \mathbb{Z}.$$

- (iv) Ist $B = (e_1, \dots, e_n)$ eine Basis von L , dann ist $B^* := (e_1^*, \dots, e_n^*)$ mit der Eigenschaft $b(e_i, e_j^*) = \delta_{ij}$ eine Basis von $L^\#$. Es gilt $\text{Gram}(B^*) = \text{Gram}(B)^{-1}$ [Kne02, (1.14)].

Da wir uns im Zuge dieser Arbeit in der Regel mit geraden Gittern quadratfreier Stufe beschäftigen, ist das folgende Lemma aus [Jü15, Lemma 1.1.1] von großer Bedeutung.

(2.1.6) Lemma

Sei L ein gerades Gitter der Stufe ℓ , wobei ℓ quadratfrei. Dann ist ℓ gleichzeitig die kleinste natürliche Zahl a , sodass $aL^\# \subseteq L$ gilt (also der Exponent der Diskriminantengruppe $L^\# / L$).

§ 2.2 Modulare Gitter

Wir kommen nun zum ursprünglich von Quebbemann eingeführten Konzept *modularer Gitter* [Que95]. Die hier verwendete Definition ist in [BFS05] zu finden.

(2.2.1) Definition

Sei L ein gerades Gitter und $\ell \in \mathbb{N}$.

- (i) L heißt *ℓ -modular*, falls $L \cong \sqrt{\ell}L^\#$.
- (ii) L heißt *stark ℓ -modular*, falls $L \cong \sqrt{m}L^{\#,m}$ für alle $m|l$, sodass $\text{ggT}(m, \frac{\ell}{m}) = 1$.

(2.2.2) Lemma

Ist L ein gerades Gitter der Dimension n .

- (i) Ist L ℓ -modular, dann ist $\text{Det}(L) = \ell^{\frac{n}{2}}$. Insbesondere muss daher n gerade sein.
- (ii) Ist L ℓ -modular und ℓ quadratfrei, dann hat L die Stufe ℓ .

(iii) Ist L stark ℓ -modular, von Stufe ℓ und ℓ quadratfrei, dann ist L auch ℓ -modular.

Beweis:

(i) Nach Bem. (2.1.5) ist $\text{Det}(L^\#) = \text{Det}(L)^{-1}$. Somit

$$\text{Det}(L) = \text{Det}(\sqrt{\ell}L^\#) = \ell^n \text{Det}(L^\#) = \frac{\ell^n}{\text{Det}(L)}.$$

Also folgt die Behauptung.

(ii) Sei a die Stufe von L , dann ist $\sqrt{a}L^\#$ gerade und hat insbesondere eine ganzzahlige Determinante. Nach (i) erhalten wir $\text{Det}(\sqrt{a}L^\#) = \left(\frac{a^2}{\ell}\right)^{\frac{n}{2}} \stackrel{!}{\in} \mathbb{Z}$. Da ℓ quadratfrei ist, sieht man also $\ell|a$. Andersherum ist $L \cong \sqrt{\ell}L^\#$, also selbstverständlich auch $\sqrt{\ell}L^\#$ gerade, somit $a|\ell$.

(iii) L hat die quadratfreie Stufe ℓ , also ist $\ell L^\# \subseteq L$ nach Lemma (2.1.6). Wir erhalten

$$L \cong \sqrt{\ell}L^{\#,\ell} = \sqrt{\ell} \left(\frac{1}{\ell} L \cap L^\# \right) = \sqrt{\ell}L^\#. \quad \square$$

Quebbemann zeigte in [Que95], dass die Theta-Reihen modularer Gitters Modulformen einer bestimmten Gruppe sind. Außerdem hat die Algebra der Modulformen eine besonders einfache Gestalt, wenn die Summe der Teiler von ℓ selbst ein Teiler von 24 ist. Konkret ist diese Eigenschaft für $\ell \in \{1, 2, 3, 5, 6, 7, 11, 14, 15, 23\}$ erfüllt. In der Literatur sind diese Stufen also besonders interessant. Es lässt sich zeigen (vgl. z.B. [Jü15, 1.2.2]), dass der Raum der Modulformen der erwähnten Gruppe in diesen Fällen ein eindeutiges Element θ der Form $1 + O(q^d)$ mit möglichst großem d und ganzzahligen Koeffizienten hat. Wir wollen den Begriff eines *extremalen Gitters* definieren als ein Gitter, welches ein möglichst großes Minimum besitzt, also ein Gitter mit Thetareihe θ . In unseren Spezialfällen gilt $d = 1 + \lfloor \frac{n}{k_1} \rfloor$, wobei k_1 Tabelle (2.1) zu entnehmen ist.

Wir können also definieren:

ℓ	1	2	3	5	6	7	11	14	15	23
k_1	24	16	12	8	8	6	4	4	4	2

Tabelle 2.1: k_1 Werte nach ℓ .

(2.2.3) Definition

Sei L ein ℓ -modulares Gitter der Dimension n und $\ell \in \{1, 2, 3, 5, 6, 7, 11, 14, 15, 23\}$.

Erfüllt L die Schranke

$$\text{Min}(L) \geq 2 \left(1 + \left\lfloor \frac{n}{k_1} \right\rfloor \right),$$

wobei k_1 gewählt ist wie in Tabelle (2.1), so nennen wir L ein *extremales Gitter*.

Die Dimensionen, welche jeweils echt von k_1 geteilt werden bezeichnet man häufig auch als *Sprungdimensionen*, da in diesen Fällen das Minimum im Vergleich zur nächst kleineren Dimension um 2 nach oben "springt".

Da die Determinante für ℓ -modulare Gitter in fester Dimension nach Lemma (2.2.2) eindeutig bestimmt ist, liefern modulare Gitter mit möglichst großem Minimum die dichtesten Kugelpackungen. In diesem Sinne ist die Klassifikation extremaler Gitter besonders interessant.

(2.2.4) Definition

Die Funktion

$$\gamma : \{L \mid L \text{ ist } n\text{-dimensionales } \mathbb{Z}\text{-Gitter}\} \rightarrow \mathbb{R}, L \mapsto \frac{\text{Min}(L)}{\text{Det}(L)^{\frac{1}{n}}} \quad (2.1)$$

heißt *Hermite-Funktion*. Der Wert

$$\gamma_n := \max\{\gamma(L) \mid L \text{ ist } n\text{-dimensionales } \mathbb{Z}\text{-Gitter}\}$$

heißt *Hermite-Konstante* zur Dimension n .

Ein höherer Wert bezüglich der Funktion γ bedeutet dabei ein dichteres Gitter im Hinblick auf die dazugehörige Kugelpackung. In der Literatur wird häufig alternativ mit der sogenannten *Zentrumsdichte* $\delta(L) = \frac{\text{Min}(L)^{\frac{n}{2}}}{2^n \sqrt{\text{Det}(L)}}$ gearbeitet (vgl. [CS93, (1.5)]). Cohn und Elkies haben in [CE03] obere Schranken für die Zentrumsdichte ermittelt. Mithilfe der Identität $\gamma(L) = 4\delta(L)^{\frac{2}{n}}$ lassen sich daraus obere Schranken für die Hermite-Konstante herleiten. Zusätzlich sind für die Dimensionen 1 bis 8 und 24 die Werte von γ_n explizit bekannt. Hierfür können wir also die Hermite-Funktionen der dichtesten bekannten Gitter als Schranken festhalten (vgl. [NS]). Die sich ergebenden oberen Schranken in Dimensionen 1 bis 36 sind in Tabelle (2.2) festgehalten.

n	$\gamma_n \leq$	n	$\gamma_n \leq$	n	$\gamma_n \leq$	n	$\gamma_n \leq$
1	1	10	2,2636	19	3.3975	28	4.4887
2	1.1547	11	2.3934	20	3.5201	29	4.6087
3	1.2599	12	2.3934	21	3.6423	30	4.7286
4	1.4142	13	2.6494	22	3.7641	31	4.8484
5	1.5157	14	2.7759	23	3.8855	32	4.9681
6	1.6654	15	2.9015	24	4.0000	33	5.0877
7	1.8115	16	3.0264	25	4.1275	34	5.2072
8	2.0000	17	3.1507	26	4.2481	35	5.3267
9	2.1327	18	3.2744	27	4.3685	36	5.4462

Tabelle 2.2: Obere Schranken für γ_n bei $1 \leq n \leq 36$.

Diese Schranken sind sehr nützlich, da sie in vielen Fällen die Existenz von bestimmten Gittern von vorneherein ausschließt. Beispielsweise hätte ein hypothetisches extremales

23-modulares Gitter L in Dimension 6 bereits ein Minimum ≥ 8 und Determinante 23^3 , also $\gamma(L) \geq \frac{8}{\sqrt{23}} \approx 1.6681 > 1.6654$ und kann somit nicht existieren. Genauer schließen die Schranken die folgenden extremalen Gitter aus:

(2.2.5) Lemma

Erfüllen $\ell \in \mathbb{N}$ und $1 \leq n \leq 36$ eine der Bedingungen

- $\ell = 1$ und $n \in \{2, 4, 6\}$,
- $\ell = 2$ und $n = 2$,
- $\ell = 11$ und $n \in \{20, 24, 28, 30, 32, 34, 36\}$,
- $\ell = 23$ und $n \in \{6, 8, 10, \dots, 34, 36\}$,

so existiert kein extremales ℓ -modulares Gitter in Dimension n .

Beweis:

Tabelle (2.2). □

Vergleicht man die hypothetischen Zentrumsdichten extremaler Gitter - für die die Frage nach der Existenz bisher nach [Jü15] noch offen ist - mit denen der dichtesten bisher bekannten Gitter (zu finden in [NS]), so fällt auf, dass die Entdeckung extremaler Gitter in den folgenden Stufen ℓ und Dimensionen $1 \leq n \leq 48$ jeweils neue dichteste Kugelpackungen liefern würden:

- $\ell = 3$ und $n \in \{36, 38\}$.
- $\ell = 5$ und $n \in \{32, 36, 40, 44, 48\}$.
- $\ell = 6$ und $n = 40$.

- $\ell = 7$ und $n \in \{32, 34, 38, 40, 36\}$.
- $\ell = 11$ und $n \in \{18, 22\}$.
- $\ell = 14$ und $n = 28$.
- $\ell = 15$ und $n = 28$.

Aufgrund der dargelegten Beispiele wird deutlich, dass die Erforschung extremaler modularer Gitter von großem Interesse für die Gittertheorie ist. Im nächsten Kapitel beschreiben wir nun eine Vorgehensweise, modulare Gitter zu klassifizieren, welche zusätzlich eine Struktur als gebrochenes Ideal eines Zahlkörpers aufweisen - sogenannte *Ideal-Gitter*.

3 Ideal-Gitter

§ 3.1 Definitionen

Wir geben nun die Definition eines Ideal-Gitter abgeleitet aus [BFS05] an.

(3.1.1) Definition

- (i) Ein (*algebraischer*) *Zahlkörper* ist eine endliche Erweiterung des Körpers \mathbb{Q} .
- (ii) Der *Ring der ganzen Zahlen* eines Zahlkörpers K ist der Ring

$$\mathbb{Z}_K := \{a \in K \mid \mu_{a,\mathbb{Q}}(X) \in \mathbb{Z}[X]\}.$$

- (iii) Die *Norm* eines Ideals \mathcal{I} von \mathbb{Z}_K ist definiert als

$$\mathcal{N}(\mathcal{I}) := |\mathbb{Z}_K / \mathcal{I}|.$$

- (iv) Ein Zahlkörper K heißt *total-reell*, wenn für alle Einbettungen $\iota : K \rightarrow \mathbb{C}$ gilt, dass $\iota(K) \subseteq \mathbb{R}$ ist. Analog heißt ein Element α eines Zahlkörpers K *total-reell*, wenn für alle Einbettungen $\iota : K \rightarrow \mathbb{C}$ gilt, dass $\iota(\alpha) \in \mathbb{R}$ ist.

- (v) Ein Zahlkörper K heißt *total-imaginär*, wenn für alle Einbettungen $\iota : K \rightarrow \mathbb{C}$ gilt, dass $\iota(K) \not\subseteq \mathbb{R}$ ist. Analog heißt ein Element α eines Zahlkörpers K total-imaginär, wenn für alle Einbettungen $\iota : K \rightarrow \mathbb{C}$ gilt, dass $\iota(\alpha) \notin \mathbb{R}$ ist.
- (vi) Ein Zahlkörper K heißt *CM-Körper*, falls K total-imaginär ist und ein total-reeller Teilkörper $K^+ \leq K$ existiert mit $[K : K^+] = 2$.
- (vii) Sei K ein CM-Körper und \mathbb{Z}_K der Ring der ganzen Zahlen in K . Ein *Ideal-Gitter* ist ein Gitter (\mathcal{I}, b) , sodass \mathcal{I} ein gebrochenes \mathbb{Z}_K -Ideal ist und $b : \mathcal{I} \times \mathcal{I} \rightarrow \mathbb{R}$ eine symmetrische positiv-definite Bilinearform mit $b(\lambda x, y) = b(x, \bar{\lambda}y)$ für $x, y \in \mathcal{I}$ und $\lambda \in \mathbb{Z}_K$. Die Abbildung $\lambda \mapsto \bar{\lambda}$ bezeichnet dabei die herkömmliche komplexe Konjugation.
- (viii) Ein Element $\alpha \in K^+$ heißt *total-positiv*, für alle Einbettungen $\iota : K^+ \hookrightarrow \mathbb{R}$ gilt, dass $\iota(\alpha) > 0$ ist. Wir schreiben dann auch $\alpha \gg 0$. Die Menge aller total-positiven Elemente in K^+ wird mit $K_{\gg 0}^+$ bezeichnet.

Bis auf weiteres sei im Folgenden stets K ein CM-Körper, \mathbb{Z}_K der Ring der ganzen Zahlen in K und K^+ der maximale total-reelle Teilkörper von K .

(3.1.2) Bemerkung

Die Eigenschaften der Bilinearform in der obigen Definition sind nach [BFS05] äquivalent dazu, dass ein total-positives Element $\alpha \in K^+$ existiert, sodass die Bilinearform b die Gestalt $b(x, y) = \text{Spur}_{K/\mathbb{Q}}(\alpha x \bar{y})$ annimmt. Wir können Ideal-Gitter daher auch durch die Notation (\mathcal{I}, α) beschreiben.

Ein Ideal-Gitter \mathcal{I} kann immer auch als \mathbb{Z} -Gitter betrachtet werden, indem man \mathbb{Z}_K -Erzeuger von \mathcal{I} und eine \mathbb{Z} -Basis von \mathbb{Z}_K zu \mathbb{Z} -Erzeugern von \mathcal{I} kombiniert. Im Folgenden bezeichnen wir daher \mathcal{I} als gerade, ganz, modular, etc., falls \mathcal{I} als \mathbb{Z} -Gitter diese Eigenschaften erfüllt und nennen $\mathcal{I}^\#$ das Dualgitter von \mathcal{I} als \mathbb{Z} -Gitter.

Wir beschäftigen uns in dieser Arbeit mit Ideal-Gittern über zyklotomischen Zahlkörpern, also Körpern der Form $\mathbb{Q}(\zeta_m)$ für primitive m -te Einheitswurzeln ζ_m . Solche Körper sind CM-Körper mit maximalem total-reellem Teilkörper $K^+ = \mathbb{Q}(\zeta_m + \overline{\zeta_m})$. Wir erhalten Körper dieser Form, indem wir Automorphismen von \mathbb{Z} -Gittern betrachten, die wie primitive Einheitswurzeln operieren. Diese Aussagen wollen wir nun präzisieren. Dazu eine kurze Definition:

(3.1.3) Definition

Sei K ein Körper und $m \in \mathbb{N}$.

1. Ein Element $\zeta \in K$ heißt primitive m -te Einheitswurzel, falls $|\langle \zeta \rangle| = m$ ist.
2. Gilt $\text{char}(K) \nmid m$ und sind ζ_1, \dots, ζ_n die primitiven m -ten Einheitswurzeln in einem Zerfällungskörper von $X^m - 1$, dann heißt das Polynom

$$\Phi_m(X) := \prod_{i=1}^n (X - \zeta_i)$$

das m -te *Kreisteilungspolynom*.

Einige wichtige bekannte Fakten zu Kreisteilungspolynomen (z.B. zu finden in [Mol11, Kap. 1]), sind die folgenden:

(3.1.4) Satz

- (i) Gilt $\text{char}(K) \nmid m$, so enthält der Zerfällungskörper von $X^m - 1$ genau $\varphi(m)$ primitive m -te Einheitswurzeln. Dabei ist $\varphi(m) := |(\mathbb{Z}/m\mathbb{Z})^*|$ die *Eulersche φ -Funktion*.
- (ii) Ist $\text{char}(K) = 0$, dann ist $\Phi_m \in \mathbb{Z}[X]$ und $X^m - 1 = \prod_{d|m} \Phi_d$.

- (iii) Speziell für $K = \mathbb{Q}$ gilt $[\mathbb{Q}(\zeta_m) : \mathbb{Q}] = \varphi(m)$ und $\Phi_m \in \mathbb{Q}[X]$ ist irreduzibel.
- (iv) Gilt $\text{char}(K) \nmid m$, so ist $K(\zeta_m)/K$ eine Galoiserweiterung.

Wir sehen also, dass ζ_m genau dann eine primitive m -te Einheitswurzel ist, wenn sie das Minimalpolynom Φ_m hat. Wir können \mathbb{Z} -Gitter somit auf die folgende Weise als Ideal-Gitter auffassen (vgl. [Neb13, Abschnitt (5.2)]):

(3.1.5) Lemma

Sei L ein \mathbb{Z} -Gitter in einem n -dimensionalen bilinearen Vektorraum (V, b) und $\sigma \in \text{Aut}(L)$ mit $\mu_\sigma = \Phi_m$ für ein $m \in \mathbb{N}$ mit $\varphi(m) = n$. Dann ist L isomorph zu einem Ideal-Gitter in $\mathbb{Q}(\zeta_m)$.

Beweis:

Durch die Operation von σ wird $\mathbb{Q}L$ mittels $\zeta_m \cdot x := \sigma(x)$ für $x \in \mathbb{Q}L$ zu einem eindimensionalen $\mathbb{Q}(\zeta_m)$ -Vektorraum und L zu einem ein $\mathbb{Z}[\zeta_m]$ -Modul. Wegen $\mathbb{Z}[\zeta_m] = \mathbb{Z}_{\mathbb{Q}[\zeta_m]}$ ist L also ein gebrochenes Ideal in $\mathbb{Q}(\zeta_m)$.

Da σ ein Automorphismus ist, ist die Bilinearform $b : L \times L \rightarrow \mathbb{Q}$ des Vektorraums ζ_m -invariant. Sei nun $\lambda \in \mathbb{Z}[\zeta_m]$ beliebig. Wir können $\lambda = \sum_{i=0}^{m-1} a_i \zeta_m^i$ für Koeffizienten $a_i \in \mathbb{Z}$ schreiben und sehen

$$b(\lambda x, y) = \sum_{i=0}^{m-1} a_i b(\zeta_m^i x, y) = \sum_{i=0}^{m-1} a_i b(x, \zeta_m^{-i} y) = \sum_{i=0}^{m-1} a_i b(x, \overline{\zeta_m^i} y) = b(x, \overline{\lambda} y),$$

womit die Eigenschaften eines Ideal-Gitters erfüllt sind. □

Mittels der Klassifikation der Ideal-Gitter über $\mathbb{Q}(\zeta_m)$ erhalten wir also zugleich alle \mathbb{Z} -Gitter mit Minimalpolynom Φ_m . Wie diese Klassifikation durchgeführt werden kann, erläutern wir in den nächsten Abschnitten.

§ 3.2 Strategie zur Klassifikation

Die in den nächsten Abschnitten beschriebenen Aussagen und Vorgehensweisen zur Klassifikation von Ideal-Gittern sind an [Jü15, Abschnitt (3.2)] und [Neb13, Abschnitt (5.2)] angelehnt.

(3.2.1) Definition

Das \mathbb{Z}_K -ideal

$$\Delta := \{x \in K \mid \text{Spur}_{K/\mathbb{Q}}(x\bar{y}) \in \mathbb{Z} \text{ für alle } y \in \mathbb{Z}_K\}$$

bezeichnet die *inverse Different* von \mathbb{Z}_K .

Wir können nun das Dual eines Idealgitters mithilfe der inversen Different ausdrücken.

(3.2.2) Lemma

Sei (\mathcal{I}, α) ein Ideal-Gitter. Dann ist $\mathcal{I}^\# = \bar{\mathcal{I}}^{-1} \Delta \alpha^{-1}$ das Dualgitter von \mathcal{I} als \mathbb{Z} -Gitter.

Beweis:

$$\begin{aligned} \mathcal{I}^\# &= \{x \in K \mid b(x, \mathcal{I}) \subseteq \mathbb{Z}\} \\ &= \{x \in K \mid \text{Spur}_{K/\mathbb{Q}}(\alpha x \bar{\mathcal{I}}) \subseteq \mathbb{Z}\} \\ &= \alpha^{-1} \{x \in K \mid \text{Spur}_{K/\mathbb{Q}}(x \bar{\mathcal{I}}) \subseteq \mathbb{Z}\} \\ &= \alpha^{-1} \bar{\mathcal{I}}^{-1} \{x \in K \mid \text{Spur}_{K/\mathbb{Q}}(x \overline{\mathbb{Z}_K}) \subseteq \mathbb{Z}\} \\ &= \bar{\mathcal{I}}^{-1} \Delta \alpha^{-1}. \end{aligned}$$

□

Mit Blick auf modulare Gitter kann man damit die nächste Folgerung ziehen:

(3.2.3) Korollar

Sei ℓ quadratfrei und (\mathcal{I}, α) ein gerades Ideal-Gitter der Stufe ℓ . Die Menge $\mathcal{B} := \alpha \mathcal{I} \bar{\mathcal{I}} \Delta^{-1}$ ist ein \mathbb{Z}_K -Ideal mit $\ell \mathbb{Z}_K \subseteq \mathcal{B}$ und Norm $\mathcal{N}(\mathcal{B}) = \det(\mathcal{I})$.

Beweis:

Da ℓ quadratfrei ist, gilt $\ell \mathcal{I}^\# \subseteq \mathcal{I}$ nach Lemma (2.1.6). Mit Lemma (3.2.2) bedeutet dies:

$$\begin{aligned} \ell \mathcal{I}^\# &\subseteq \mathcal{I} \subseteq \mathcal{I}^\# \\ \Leftrightarrow \ell \bar{\mathcal{I}}^{-1} \Delta \alpha^{-1} &\subseteq \mathcal{I} \subseteq \bar{\mathcal{I}}^{-1} \Delta \alpha^{-1} \\ \Leftrightarrow \ell \mathbb{Z}_K &\subseteq \alpha \mathcal{I} \bar{\mathcal{I}} \Delta^{-1} \subseteq \mathbb{Z}_K. \end{aligned}$$

Für die Norm gilt

$$\det(\mathcal{I}) = |\mathcal{I}^\# / \mathcal{I}| = |\mathbb{Z}_K / \left(\mathcal{I} \left(\mathcal{I}^\# \right)^{-1} \right)| = |\mathbb{Z}_K / \mathcal{B}| = \mathcal{N}(\mathcal{B}). \quad \square$$

Da es jeweils nur endlich viele \mathbb{Z}_K -Ideale mit bestimmter Norm gibt, existieren bei der Konstruktion von Idealgittern mit fester Determinante nur endlich viele Möglichkeiten für \mathcal{B} . Mithilfe der Primidealzerlegung lässt sich der rekursive Algorithmus (1) konstruieren, welcher alle Teiler eines Ideals \mathcal{J} mit bestimmter Norm n berechnen kann.

Konkret wird unsere Strategie daraus bestehen, alle (relevanten) Möglichkeiten für \mathcal{I} und \mathcal{B} durchzugehen und zu testen, für welche davon das Ideal $(\mathcal{I} \bar{\mathcal{I}})^{-1} \Delta \mathcal{B}$ ein Hauptideal mit total-positivem Erzeuger $\alpha \in K^+$ ist. In diesem Fall ist (\mathcal{I}, α) ein Ideal-Gitter. Um den Suchraum zu verkleinern, machen wir zunächst einige Einschränkungen.

Algorithmus 1 Berechnung aller Teiler mit fester Norm

```
1: Eingabe:  $\mathbb{Z}_K$ -Ideal  $\mathcal{I}$ , Norm  $n$ 
2: Ausgabe: Liste aller Teiler von  $\mathcal{I}$  mit Norm  $n$ 
3:
4: if  $n = 1$  then return  $[\mathbb{Z}_K]$ 
5: if  $n \nmid \mathcal{N}(\mathcal{I})$  then return  $[\ ]$ 
6: if  $\mathcal{N}(\mathcal{I}) = n$  then return  $[\mathcal{I}]$ 
7: Zerlege  $\mathcal{I}$  in Primideale  $\mathcal{I} = \mathfrak{p}_1^{s_1} \dots \mathfrak{p}_k^{s_k}$ 
8:  $n_{\mathfrak{p}} \leftarrow \mathcal{N}(\mathfrak{p}_1)$ 
9:  $Results \leftarrow [\ ]$ 
10: for  $j \in \{0, \dots, s_1\}$  do
11:   if  $n_{\mathfrak{p}}^j \mid n$  then
12:      $D \leftarrow$  Teiler von  $\mathfrak{p}_2^{s_2} \dots \mathfrak{p}_k^{s_k}$  mit Norm  $\frac{n}{n_{\mathfrak{p}}^j}$  (rekursiv)
13:     for  $\mathcal{J} \in D$  do
14:        $Results \leftarrow Results \cup [\mathfrak{p}_1^j \mathcal{J}]$ 
15: return  $Results$ 
```

§ 3.3 Klassengruppe

(3.3.1) Definition

Die *Klassengruppe*

$$\text{Cl}_K := \{J \mid J \text{ ist gebrochenes } \mathbb{Z}_K\text{-Ideal}\} / \{(c)_{\mathbb{Z}_K} \mid c \in K^*\}.$$

(3.3.2) Lemma

Seien \mathcal{I} ein gebrochenes \mathbb{Z}_K -Ideal und $\alpha \in K_{\gg 0}^+$. Für $\lambda \in K^*$ gilt $(\lambda\mathcal{I}, \alpha) \cong (\mathcal{I}, \lambda\bar{\lambda}\alpha)$.

Beweis:

Sei $b_\alpha : K \times K \rightarrow \mathbb{R}, (x, y) \mapsto \text{Spur}_{K/\mathbb{Q}}(\alpha x \bar{y})$ die zu α gehörige Bilinearform. Dann ist

$$b_\alpha(\lambda x, \lambda y) = \text{Spur}_{K/\mathbb{Q}}(\lambda \bar{\lambda} \alpha x \bar{y}) = b_{\lambda \bar{\lambda} \alpha}(x, y).$$

Folglich ist $\psi : (K, b_{\lambda \bar{\lambda} \alpha}) \rightarrow (K, b_\alpha), x \mapsto \lambda x$ eine Isometrie mit $\psi(\mathcal{I}) = (\lambda\mathcal{I})$. □

Mit dieser Aussage genügt es also, aus jeder Klasse jeweils nur einen Vertreter zu betrachten. Wählt man $\lambda \in \mathbb{Z}_K^*$, so zeigt das Lemma, dass $(\mathcal{I}, \alpha) \cong (\mathcal{I}, \lambda \bar{\lambda} \alpha)$. Für α reichen also Vertreter modulo $\{\lambda \bar{\lambda} \mid \lambda \in \mathbb{Z}_K^*\}$.

Wir wollen nun die zu untersuchenden Möglichkeiten für \mathcal{I} noch weiter einschränken: Ist K/\mathbb{Q} galoissch (wie es für zyklotomische Zahlkörper der Fall ist), so genügt ein Repräsentant modulo der Operation der Galosgruppe.

(3.3.3) Lemma

Sei K/\mathbb{Q} eine Galoiserweiterung, \mathcal{I} ein gebrochenes \mathbb{Z}_K -Ideal und $\alpha \in K_{\gg 0}^+$. Für $\sigma \in \text{Gal}(K/\mathbb{Q})$ ist $(\mathcal{I}, \alpha) \cong (\sigma(\mathcal{I}), \sigma(\alpha))$.

Beweis:

Da die Spur invariant unter der Galoisgruppe ist, erhält man die folgende Gleichungskette.

$$\begin{aligned} b_{\sigma(\alpha)}(\sigma(x), \sigma(y)) &= \text{Spur}_{K/\mathbb{Q}}(\sigma(\alpha)\sigma(x)\overline{\sigma(y)}) \\ &= \text{Spur}_{K/\mathbb{Q}}(\sigma(\alpha x \bar{y})) = \text{Spur}_{K/\mathbb{Q}}(\alpha x \bar{y}) = b_{\alpha}(x, y) \end{aligned}$$

Also induziert σ eine Isometrie $\sigma : (K, b_{\alpha}) \rightarrow (K, b_{\sigma(\alpha)})$. □

(3.3.4) Bemerkung

Mit **MAGMA** kann die Klassengruppe berechnet werden, in gewissen Fällen ist der zugehörige Algorithmus jedoch sehr ressourcenintensiv. Unter Annahme der unbewiesenen *verallgemeinerten Riemann-Hypothese* erlauben gewisse Schranken eine leichtere Berechnung. Für die Implementierung gehen wir daher von der Korrektheit dieser Hypothese aus. Sollte diese falsch sein, so sind die später angeführten Listen der gefundenen modularen Gitter möglicherweise unvollständig.

§ 3.4 Total-positive Erzeuger

Wir benötigen nun einen Test, welcher für ein gegebenes gebrochenes \mathbb{Z}_K -Ideal \mathcal{I} überprüft, dieses von einem total-positiven Element $\alpha \in K_{\gg 0}^+$ erzeugt wird. Dazu untersuchen wir zuerst, ob \mathcal{I} überhaupt von einem Element aus K^+ erzeugt ist und anschließend, wann ein Ideal $\alpha'\mathbb{Z}_K$ für $\alpha' \in K^+$ einen total-positiven Erzeuger hat.

(3.4.1) Satz

Sei \mathcal{I} ein gebrochenes \mathbb{Z}_K -Ideal. Es existiert genau dann ein $\alpha' \in K^+$ mit $\mathcal{I} = \alpha' \mathbb{Z}_K$, wenn $\mathcal{I} \cap K^+ = \alpha' \mathbb{Z}_{K^+}$ und für jeden Primteiler \mathfrak{p} von \mathcal{I} gilt

- Ist \mathfrak{p} verzweigt in K/K^+ , so ist $\nu_{\mathfrak{p}}(\mathcal{I}) \in 2\mathbb{Z}$.
- Ist \mathfrak{p} unverzweigt in K/K^+ , so ist $\nu_{\mathfrak{p}}(\mathcal{I}) = \nu_{\bar{\mathfrak{p}}}(\mathcal{I})$.

Beweis:

Wir zeigen zunächst, dass die Bedingungen an die Primteiler äquivalent dazu sind, dass $\mathcal{I} = (\mathcal{I} \cap K^+) \mathbb{Z}_K$.

Sei dazu zuerst $\mathcal{I} = (\mathcal{I} \cap K^+) \mathbb{Z}_K$ erfüllt. Seien

$$\mathcal{I}' := \mathcal{I} \cap K^+ = \prod_{\mathfrak{a}} \mathfrak{a}^{\nu_{\mathfrak{a}}(\mathcal{I} \cap K^+)}, \quad \mathcal{I} = \prod_{\mathfrak{p}} \mathfrak{p}^{\nu_{\mathfrak{p}}(\mathcal{I})}$$

die Primidealzerlegungen. Dann folgt

$$\prod_{\mathfrak{a}} \mathfrak{a}^{\nu_{\mathfrak{a}}(\mathcal{I}')} \mathbb{Z}_K = \prod_{\mathfrak{p}} \mathfrak{p}^{\nu_{\mathfrak{p}}(\mathcal{I})}.$$

Aufgrund der Eindeutigkeit der Primidealzerlegung bedeutet dies

$$\mathfrak{a}^{\nu_{\mathfrak{a}}(\mathcal{I}')} \mathbb{Z}_K = \prod_{\mathfrak{p}|\mathfrak{a}} \mathfrak{p}^{\nu_{\mathfrak{p}}(\mathcal{I})}$$

für jedes Primideal \mathfrak{a} von \mathbb{Z}_{K^+} . Es ist $[K : K^+] = 2$, also kann $\mathfrak{a} \mathbb{Z}_K$ nur eine der Formen \mathfrak{p} , \mathfrak{p}^2 , oder $\mathfrak{p}\bar{\mathfrak{p}}$ für ein Primideal \mathfrak{p} in \mathbb{Z}_K annehmen.

- Falls $\mathfrak{a} \mathbb{Z}_K = \mathfrak{p}^2$ (also falls \mathfrak{p} verzweigt ist), so folgt $\nu_{\mathfrak{p}}(\mathcal{I}) = 2\nu_{\mathfrak{a}}(\mathcal{I}') \in 2\mathbb{Z}$.
- In den anderen beiden Fällen (also falls \mathfrak{p} unverzweigt ist) gilt $\nu_{\mathfrak{p}}(\mathcal{I}) = \nu_{\mathfrak{a}}(\mathcal{I}') = \nu_{\bar{\mathfrak{p}}}(\mathcal{I})$.

Seien nun andersherum die Primideal-Bedingungen erfüllt. Definiert man

$$\mathcal{I}' := \prod_{\mathfrak{a}} \mathfrak{a}^{\nu_{\mathfrak{a}}(\mathcal{I}')} , \quad \nu_{\mathfrak{a}}(\mathcal{I}') = \begin{cases} \nu_{\mathfrak{p}}(\mathcal{I}) & \mathfrak{a}\mathbb{Z}_K \in \{\mathfrak{p}, \mathfrak{p}\bar{\mathfrak{p}}\} \\ \frac{1}{2}\nu_{\mathfrak{p}}(\mathcal{I}) & \mathfrak{a}\mathbb{Z}_K = \mathfrak{p}^2 \end{cases}$$

so gilt

$$\mathcal{I} = \prod_{\mathfrak{p}} \mathfrak{p}^{\nu_{\mathfrak{p}}(\mathcal{I})} = \prod_{\mathfrak{a}} (\mathfrak{a}\mathbb{Z}_K)^{\nu_{\mathfrak{a}}(\mathcal{I}')} = \mathcal{I}'\mathbb{Z}_K.$$

Also folgt $(\mathcal{I} \cap K^+)\mathbb{Z}_K = (\mathcal{I}'\mathbb{Z}_K \cap K^+)\mathbb{Z}_K = \mathcal{I}'\mathbb{Z}_K = \mathcal{I}$ und es gilt die behauptete Äquivalenz.

Die Behauptung des Satzes wurde somit darauf reduziert, dass genau dann $\mathcal{I} = \alpha'\mathbb{Z}_K$, wenn $\mathcal{I} \cap K^+ = \alpha'\mathbb{Z}_{K^+}$ und $\mathcal{I} = (\mathcal{I} \cap K^+)\mathbb{Z}_K$ für $\alpha' \in K^+$. Dies folgt allerdings leicht mithilfe von $(\alpha'\mathbb{Z}_K) \cap K^+ = \alpha'\mathbb{Z}_{K^+}$. \square

Mithilfe dieses Satzes können wir nun Algorithmus (2) formulieren, welcher zu einem gegebenen Ideal \mathcal{I} testet, ob dieses einen Erzeuger in K^+ hat und - falls ja - einen solchen zurückgibt. Ein Primideal \mathfrak{a} wie im Beweis unseres Satzes erhalten wir, indem wir eine Primzahl p finden, sodass $\mathfrak{p} \mid (p\mathbb{Z}_K)$, dann muss \mathfrak{a} eines der Primideale aus der Faktorisierung von $p\mathbb{Z}_{K^+}$ teilen.

(3.4.2) Lemma

Sei $\alpha' \in K^+$. Ein total-positives Element $\alpha \in K_{\gg 0}^+$ ist genau dann ein Erzeuger des Ideals $\alpha'\mathbb{Z}_K$, wenn eine Einheit $\epsilon \in \mathbb{Z}_{K^+}^*$ existiert mit $\alpha = \alpha'\epsilon$ und $\text{sign}(\iota(\epsilon)) = \text{sign}(\iota(\alpha'))$ für alle Einbettungen $\iota : K^+ \hookrightarrow \mathbb{R}$.

Beweis:

Ein weiterer Erzeuger hat immer die Gestalt $\alpha = \alpha'\epsilon$ für eine Einheit $\epsilon \in (\mathbb{Z}_{K^+})^*$. Damit α total-positiv wird muss für alle Einbettungen $\iota : K^+ \hookrightarrow \mathbb{R}$ gelten:

$$1 \stackrel{!}{=} \text{sign}(\iota(\alpha)) = \text{sign}(\iota(\alpha')\iota(\epsilon)) = \text{sign}(\iota(\alpha')) \text{sign}(\iota(\epsilon))$$

Algorithmus 2 Berechnung eines total-reellen Erzeugers

```
1: Eingabe:  $\mathbb{Z}_K$ -Ideal  $\mathcal{I}$ 
2: Ausgabe: Element  $\alpha' \in K^+$  mit  $\alpha'\mathbb{Z}_K = \mathcal{I}$ , oder false, falls ein solches Element
   nicht existiert
3:
4:  $\mathcal{I}' \leftarrow 1\mathbb{Z}_{K^+}$ 
5:  $\text{Split} \leftarrow [ ]$ 
6: Zerlege  $\mathcal{I} = \mathfrak{p}_1^{s_1} \dots \mathfrak{p}_k^{s_k}$  in Primideale.
7: for  $i \in \{1 \dots k\}$  do
8:   if  $i \in \text{Split}$  then continue
9:    $p \leftarrow$  Minimale natürliche Zahl  $p \in \mathbb{N}$  mit  $\mathfrak{p}_i \mid (p\mathbb{Z}_K)$ 
10:  Zerlege  $p\mathbb{Z}_{K^+}$  in Primideale:  $p\mathbb{Z}_{K^+} = \mathfrak{q}_1^{t_1} \dots \mathfrak{q}_l^{t_l}$ 
11:   $\mathfrak{a} \leftarrow \mathfrak{q}_j$  mit  $\mathfrak{p}_i \mid (\mathfrak{q}_j\mathbb{Z}_K)$ 
12:  if  $\mathfrak{a}\mathbb{Z}_K = \mathfrak{p}_i^2$  then
13:    if  $2 \nmid s_i$  then return false
14:     $\mathcal{I}' \leftarrow \mathcal{I}' \mathfrak{a}^{\frac{s_i}{2}}$ 
15:  else if  $\mathfrak{a}\mathbb{Z}_K = \mathfrak{p}_i$  then
16:     $\mathcal{I}' \leftarrow \mathcal{I}' \mathfrak{a}^{s_i}$ 
17:  else if  $\mathfrak{a}\mathbb{Z}_K = \mathfrak{p}_i \overline{\mathfrak{p}_i}$  then
18:    if  $\nu_{\mathfrak{p}_i}(\mathcal{I}) \neq \nu_{\overline{\mathfrak{p}_i}}(\mathcal{I})$  then return false
19:     $\mathcal{I}' \leftarrow \mathcal{I}' \mathfrak{a}^{s_i}$ 
20:     $j \leftarrow j'$  mit  $\mathfrak{p}_{j'} = \overline{\mathfrak{p}_i}$ 
21:     $\text{Split} \leftarrow \text{Split} \cup [j]$ 
22: if  $\mathcal{I}'$  kein Hauptideal then
23:   return false
24: else
25:   return Erzeuger von  $\mathcal{I}'$ 
```

Also müssen die Vorzeichen jeweils identisch sein. \square

Elemente aus $(\mathbb{Z}_{K^+}^*)^2$ haben immerzu ein positives Signum bezüglich aller Einbettungen. Außerdem liefern total-positive Elemente, die in der gleichen Klasse modulo Quadraten liegen, nach Lemma (3.3.2) isomorphe Idealgitter. Es genügt also, sich bei der Suche nach einer Einheit wie im vorherigen Lemma auf Vertreter modulo Quadraten zu beschränken. Nach dem Dirichletschen Einheitsensatz [Neu92, Theorem (7.4)] hat die Einheitengruppe die Struktur

$$\mathbb{Z}_{K^+}^* = \{\pm 1\} \times \mathbb{Z}^{t-1}$$

mit $t := [K^+ : \mathbb{Q}]$. Die Erzeuger $(\epsilon_1, \dots, \epsilon_t)$ der Gruppe heißen *Grundeinheiten*. Jede Einheit ϵ lässt sich also darstellen in der Form $\epsilon = \epsilon_1^{\nu_1} \dots \epsilon_t^{\nu_t}$. Das folgende Korollar liefert uns nun die Lösung auf unsere Frage nach den total-positiven Erzeugern.

Dann lässt sich folgendes Korollar ziehen:

(3.4.3) Korollar

Sei $\alpha' \in K^+$, seien die Einbettungen von K^+ in \mathbb{R} gegeben durch ι_1, \dots, ι_t und seien $\epsilon_1, \dots, \epsilon_t$ die Grundeinheiten von $\mathbb{Z}_{K^+}^*$. Definiere die Matrix

$$M \in \mathbb{F}_2^{t \times t}, \quad M_{ij} = \begin{cases} 1 & , \text{sign}(\iota_j(\epsilon_i)) = -1 \\ 0 & , \text{sign}(\iota_j(\epsilon_i)) = 1 \end{cases}$$

und den Vektor

$$V \in \mathbb{F}_2^{1 \times t}, \quad V_i = \begin{cases} 1 & , \text{sign}(\iota_i(\alpha')) = -1 \\ 0 & , \text{sign}(\iota_i(\alpha')) = 1. \end{cases}$$

Dann sind die total-positiven Erzeuger des Ideals $\alpha' \mathbb{Z}_K$ genau die Elemente der Menge $\{\alpha' \epsilon_1^{x_1} \dots \epsilon_t^{x_t} \epsilon^2 \mid x \in \mathbb{F}_2^{1 \times t}, xM = V, \epsilon \in (\mathbb{Z}_{K^+}^*)\}$.

Beweis:

Nach Lemma (3.4.2) und da Quadrate immerzu ein positives Signum haben, sind die total-positiven Erzeuger gegeben durch die Elemente $u = \alpha' \epsilon_1^{x_1} \dots \epsilon_t^{x_t} \epsilon^2$, wobei $x \in \mathbb{F}_2^{1 \times t}$, $\epsilon \in \mathbb{Z}_{K+}^*$ und $\epsilon_1^{x_1} \dots \epsilon_t^{x_t}$ bezüglich aller Einbettungen dasselbe Signum wie α' hat. Das Signum bezüglich einem ι_i ist genau dann gleich, wenn

$$|\{j \mid \text{sign}(\iota_j(\epsilon_i)) = -1 \text{ und } x_i = 1\}| \equiv \begin{cases} 1 \pmod{2} & , \text{sign}(\iota_j(\alpha')) = -1 \\ 0 \pmod{2} & , \text{sign}(\iota_j(\alpha')) = 1 \end{cases}.$$

Diese Kongruenz ist aber genau dann erfüllt, wenn x Lösung des linearen Gleichungssystems $xM = V$ ist. \square

(3.4.4) Bemerkung

Um später in der Implementierung Zeit zu sparen, kann man bemerken, dass sich verschiedene total-positive Erzeuger des gleichen Ideals jeweils lediglich um eine total-positive Einheit unterscheiden. Um die Effizienz zu erhöhen, kann man also zu Beginn des Algorithmus die Menge aller total-positiven Einheiten (diese korrespondieren zum Kern von M) berechnen. Auf diese Weise muss später pro Ideal jeweils nur eine spezielle Lösung des Gleichungssystems gefunden werden und die Menge aller total-positiven Erzeuger lässt sich durch Multiplikation mit den vorher berechneten total-positiven Einheiten erstellen.

Eine weitere Anmerkung zur Implementierung: **MAGMA** kann mit der Funktion `pFundamentalUnits` eine Untergruppe von \mathbb{Z}_{K+}^* mit ungeradem Index berechnen. Wie das folgende Lemma zeigt, reicht dies für unser Vorhaben bereits aus, da wir nur ein Vertretersystem der Einheiten modulo Quadraten benötigen.

(3.4.5) Lemma

Sei G eine abelsche Gruppe und $U \leq G$ mit $[G : U]$ ungerade. Dann ist

$$G/G^2 \cong U/U^2.$$

Beweis:

Betrachte den Epimorphismus $\pi : G \rightarrow G/G^2$. Es ist bereits $\pi|_U$ surjektiv, denn sei $gG^2 \in G/G^2$, dann ist $g^{[G:U]} \in U$, da $(gU)^{[G:U]} = U$ und - weil der Index ungerade ist - auch $\pi(g^{[G:U]}) = gG^2$. Zudem ist $\text{Kern}(\pi|_U) = U \cap G^2 = U^2$, denn für $g^2 \in U \cap G^2$ muss $|gU| \leq 2$ gelten. Die Ordnung kann aber wegen des ungeraden Index nicht 2 sein, also folgt bereits $g \in U$ und somit $g^2 \in U^2$. Mit dem Homomorphiesatz ergibt sich nun die Behauptung. \square

Anhand der gewonnenen Erkenntnisse erstellen wir nun einen Algorithmus (3), der zu einem Ideal $\mathcal{I} = \alpha' \mathbb{Z}_K$ für $\alpha' \in K^+$ ein Vertretersystem aller total-positiven Erzeuger $\alpha \in K_{\gg 0}^+$ modulo $\lambda \bar{\lambda}$ für $\lambda \in \mathbb{Z}_K^*$ liefert. Die Ergebnisse der Zeilen 6 – 14 können in der Implementierung nach einmaliger Durchführung abgespeichert werden, sodass die Resultate anschließend für jedes zu prüfende α' wiederverwertet werden können.

Algorithmus 3 Berechnung total-positiver Erzeuger

```
1: Eingabe: Erzeuger  $\alpha' \in K^+$  von  $\mathcal{I}$ 
2: Ausgabe: Liste von Vertretern der Menge aller total-positiven Erzeuger  $\alpha \in K_{\gg 0}^+$ 
   von  $\mathcal{I}$  modulo  $\{\lambda\bar{\lambda} \mid \lambda \in \mathbb{Z}_K^*\}$  zurückgibt
3:
4:  $\iota_1, \dots, \iota_t \leftarrow$  Einbettungen  $K^+ \hookrightarrow \mathbb{R}$ 
5:  $\epsilon_1, \dots, \epsilon_t \leftarrow$  Erzeuger einer Untergruppe von  $\mathbb{Z}_{K^+}^*$  mit ungeradem Index
6:  $M \leftarrow 0 \in \mathbb{F}_2^{t \times t}$ 
7: for  $(i, j) \in \underline{t} \times \underline{t}$  do
8:   if  $\iota_j(\epsilon_i) < 0$  then
9:      $M_{ij} \leftarrow 1$ 
10:  $U' \leftarrow [\epsilon_1^{a_1} \dots \epsilon_t^{a_t} \mid a \in \text{Kern}(M)]$ 
11:  $U \leftarrow []$ 
12: for  $u' \in U'$  do
13:   if  $u' \neq u\lambda\bar{\lambda}$  für alle  $u \in U, \lambda \in \mathbb{Z}_K^*$  then
14:      $U \leftarrow U \cup [u']$ 
15:  $V \leftarrow 0 \in \mathbb{F}_2^{1 \times t}$ 
16: for  $i \in \{1, \dots, t\}$  do
17:   if  $\iota_i(\alpha') < 0$  then
18:      $V_i \leftarrow 1$ 
19:  $x \leftarrow$  Lösung von  $xM = V$ 
20: return  $\alpha' \epsilon_1^{x_1} \dots \epsilon_t^{x_t} U$ 
```

§ 3.5 Finaler Algorithmus und Ergebnisse

Alle bisherigen Bestandteile können nun zu einem Algorithmus zusammengesetzt werden, der zu einem quadratfreien $\ell \in \mathbb{N}$, einer vorgegebenen Determinante d und einem

CM-Körper K mit total-reellem Teilkörper K^+ alle Ideal-Gitter berechnet.

Algorithmus 4 Berechnung von Ideal-Gittern

```

1: Eingabe: Quadratfreies  $\ell \in \mathbb{N}$ ,  $d \in \mathbb{N}$ , CM-Körper  $K$ , maximaler total-reeller
   Teilkörper  $K^+$  von  $K$ 
2: Ausgabe: Per Isomorphie reduzierte Liste aller geraden Ideal-Gitter  $(\mathcal{I}, \alpha)$  über  $K$ 
   mit Determinante  $d$ , deren Stufe  $\ell$  teilt
3:
4:  $\mathfrak{A} \leftarrow$  Vertretersystem von  $Cl_K / \text{Gal}(K/\mathbb{Q})$ 
5:  $\mathfrak{B} \leftarrow [\mathcal{B} \mid \mathcal{B} \text{ ist } \mathbb{Z}_K\text{-Ideal mit } \ell\mathbb{Z}_K \subseteq \mathcal{B} \subseteq \mathbb{Z}_K \text{ und } \mathcal{N}(\mathcal{B}) = d]$  (nach Algorithmus
   (1))
6:  $Results \leftarrow []$ 
7: for  $(\mathcal{I}, \mathcal{B}) \in (\mathfrak{A}, \mathfrak{B})$  do
8:    $\mathcal{J} \leftarrow (\mathcal{I}\bar{\mathcal{I}})^{-1} \Delta \mathcal{B}$ 
9:   if  $\exists \alpha' \in K^+$  mit  $\mathcal{J} = \alpha' \mathbb{Z}_K$  (nach Algorithmus (2)) then
10:      $X \leftarrow [\alpha \in K_{\gg 0}^+ \mid \mathcal{J} = \alpha \mathbb{Z}_K]$  (nach Algorithmus (3))
11:     for  $\alpha \in X$  do
12:       if  $(\mathcal{I}, \alpha)$  ist gerades Gitter then
13:         if  $(\mathcal{I}, \alpha) \not\cong (\tilde{\mathcal{I}}, \tilde{\alpha})$  für alle  $(\tilde{\mathcal{I}}, \tilde{\alpha}) \in Results$  then
14:            $Results \leftarrow Results \cup [(\mathcal{I}, \alpha)]$ 

```

Mit diesem Algorithmus kann man nun alle ℓ -modularen Gitter in Dimension n klassifizieren, welche einen Automorphismus σ besitzen mit $\mu_\sigma = \Phi_m$ und $\varphi(m) = n$. Dazu wendet man Algorithmus (4) wie in Lemma (2.2.2) und Lemma (3.1.5) besprochen mit $d = l^{\frac{n}{2}}$ und $K = \mathbb{Q}(\zeta_m)$ an. Eine weitere kleine Erleichterung bringt in diesem Spezialfall die Tatsache, dass $\mathbb{Q}(\zeta_m) \cong \mathbb{Q}(\zeta_{2m})$, falls $m \equiv 1 \pmod{2}$. Insbesondere sind die Ideal-Gitter über $\mathbb{Q}(\zeta_m)$ und $\mathbb{Q}(\zeta_{2m})$ dieselben. Man kann also für eine vollständige Aufzählung alle m mit $m \equiv 2 \pmod{4}$ weglassen. Eine Implementierung in **MAGMA** liefert nun alle ℓ -modularen Ideal-Gitter mit Dimension $n \leq 36$ (eine Steigerung der Dimension

$\begin{array}{c c} & \ell \\ \hline n & \end{array}$	1	2	3	5	6	7	11	14	15	23
4	—	1(1)	1(1)	—	—	—	1(1)	1(1)	—	1(1)
6	—	—	1(1)	—	—	1(1)	—	—	—	—
8	1(1)	1(1)	1(1)	1(1)	1(1)	1(1)	2(1)	2(1)	1(1)	3(—)
10	—	—	—	—	—	—	1(1)	—	—	—
12	—	1(1)	2(1)	1(1)	1(1)	1(—)	1(—)	1(1)	—	1(—)
16	1(1)	2(1)	3(2)	1(—)	2(1)	4(3)	5(—)	5(—)	3(1)	5(—)
18	—	—	1(—)	—	—	—	—	—	—	—
20	—	1(1)	—	—	1(1)	1(—)	2(—)	—	—	—
22	—	—	—	—	—	—	—	—	—	2(—)
24	4(1)	2(1)	7(1)	5(1)	5(2)	8(—)	7(—)	8(—)	5(—)	14(—)
32	7(5)	13(4)	13(7)	10(—)	12(—)	19(—)	42(—)	21(—)	23(—)	—
36	—	6(3)	8(—)	8(—)	—	—	2(—)	36(—)	4(—)	—

Tabelle 3.1: Anzahl der ℓ -modularen Ideal-Gitter in Dimension $n \leq 36$ sowie der Anzahl der extremalen Gitter darunter

beansprucht exponentiell höheren Zeitaufwand) und $\ell \in \{1, 2, 3, 5, 6, 7, 11, 14, 15, 23\}$. In Tabelle (3.5) sind die Gesamtzahlen der Ideal-Gitter zu finden; außerdem befindet sich in Anhang (6.1) eine ausführlichere Zusammenfassung der Klassifikationsergebnisse mit zusätzlicher Angabe der zugrundeliegenden zyklotomischen Zahlkörpern und der Anzahl der Gitter, aufgeschlüsselt nach Minimum. Beachte dabei: Der Zahlkörper ist im Allgemeinen nicht eindeutig, ein Gitter kann möglicherweise Ideal-Gitter-Struktur über mehreren Kreisteilungskörpern gleichzeitig aufweisen; in der Tabelle im Anhang ist jeweils nur einer davon genannt.

4 Sub-Ideal-Gitter

§ 4.1 Einführung

Im letzten Kapitel haben wir gesehen, wie Gitter in Dimension n mit einem Automorphismus σ klassifiziert werden können, falls $\mu_\sigma = \Phi_m$ und $\varphi(m) = n$ erfüllt sind. In diesem Kapitel wollen wir versuchen, Aussagen über Gitter L zu treffen, welche nicht selbst Ideal-Gitter-Struktur aufweisen, aber zumindest ein Ideal-Gitter enthalten.

Ist L ein Gitter und $\sigma \in \text{Aut}(L)$ von endlicher Ordnung mit Minimalpolynom $\mu_\sigma = \Phi_{m_1} \cdots \Phi_{m_k}$, so können wir den zugrundeliegenden Vektorraum V in σ -invariante Teilräume aufspalten:

$$V = \text{Kern}(\Phi_{m_1}(\sigma)) \oplus \cdots \oplus \text{Kern}(\Phi_{m_k}(\sigma)).$$

Wie das folgende Lemma zeigt, ist diese Zerlegung sogar orthogonal:

(4.1.1) Lemma

Sei (V, b) ein bilinearer Vektorraum und $\sigma \in O(V, b)$ mit $\mu_\sigma = \Phi_{m_1} \Phi_{m_2} \cdots \Phi_{m_k}$, dann ist

$$V = \text{Kern}(\Phi_{m_1}(\sigma)) \perp \text{Kern}(\Phi_{m_2}(\sigma)) \perp \cdots \perp \text{Kern}(\Phi_{m_k}(\sigma))$$

eine Zerlegung in σ -invariante, orthogonale Teilräume.

Beweis:

Seien $\pi_i : V \rightarrow \text{Kern}(\Phi_{m_i}(\sigma))$ für $i = 1, \dots, k$ die Projektionen auf die Komponenten. Da σ eine Isometrie ist, gilt $\sigma^{ad} = \sigma^{-1}$, also ist σ insbesondere normal und damit auch die π_i , welche sich als Polynome in σ mit rationalen Koeffizienten ausdrücken lassen. Normale Projektionen sind allerdings selbstadjungiert, also gilt für alle $x, y \in V$ sowie $i \neq j$:

$$b(\pi_i(x), \pi_j(y)) = b(x, \pi_i^{ad}(\pi_j(y))) = b(x, \pi_i(\pi_j(y))) = b(x, 0) = 0$$

und daher $\pi_i(V) \perp \pi_j(V)$. Die angegebene Zerlegung von V ist somit orthogonal. \square

Besitzt μ_σ einen Teiler Φ_m , wobei $\frac{n}{2} < \varphi(m) \leq n$, so muss $\text{Kern}(\Phi_m(\sigma))$ die Dimension $\varphi(m)$ haben, wird also zu einem eindimensionalen $\mathbb{Q}(\zeta_m)$ -Vektorraum. Die orthogonale Zerlegung des Vektorraums induziert also ein volles Teilgitter

$$M := (L \cap \text{Kern}(\Phi_m(\sigma))) \perp \left(L \cap \text{Kern} \left(\frac{\mu_\sigma}{\Phi_m}(\sigma) \right) \right) \leq L$$

und $L \cap \text{Kern}(\Phi_m(\sigma))$ hat eine Struktur als Ideal-Gitter über dem zyklotomischen Zahlkörper $\mathbb{Q}(\zeta_m)$. Vergleiche dazu [Neb13, Abs. (5.3)]. Dies halten wir in der folgenden Definition fest.

(4.1.2) Definition

Sei L ein \mathbb{Z} -Gitter der Dimension n .

- (i) Ein *großer Automorphismus* von L ist ein $\sigma \in \text{Aut}(L)$ von Ordnung $m \in \mathbb{N}$, sodass $\Phi_m | \mu_\sigma$ und $\frac{n}{2} < \varphi(m) \leq n$.
- (ii) Ist $\sigma \in \text{Aut}(L)$ ein großer Automorphismus, so bezeichnet man das Ideal-Gitter $L \cap \text{Kern}(\Phi_m(\sigma))$ über $\mathbb{Q}(\zeta_m)$ als *Sub-Ideal-Gitter* von L .

Unsere Strategie zur Klassifikation der Gitter L besteht darin, Eigenschaften des induzierten Teilgitters M zu zeigen sowie die Operation von σ genauer zu untersuchen, um eine Liste möglicher Kandidaten für M und σ zu finden. Anschließend konstruieren wir L als σ -invariantes Obergitter von M .

Für Gitter mit großen Automorphismen können wir die Ideal-Gitter-Komponente mithilfe der Algorithmen aus dem letzten Kapitels effizient konstruieren. Probleme bereitet uns allerdings der andere Teil $\text{Kern}\left(\frac{\mu_\sigma}{\Phi_m}(\sigma)\right)$ des Vektorraums, über welchen wir a priori nicht viel aussagen können. Abhilfe schaffen uns unter gewissen Umständen die Automorphismen von Primzahlordnung.

§ 4.2 Automorphismen von Primzahlordnung

Der folgende Abschnitt ist an [Jü15, Kap. 4] und [Neb13, Kap. 4] angelehnt.

Sei L in diesem Abschnitt ein \mathbb{Z} -Gitter in einem n -dimensionalen bilinearen \mathbb{Q} -Vektorraum (V, b) und $\sigma \in \text{Aut}(L)$ von Primzahlordnung p . Dann ist $\mu_\sigma \in \{\Phi_p, \Phi_1\Phi_p\}$. Wie vorher erhält man eine σ -invariante Zerlegung

$$V = \text{Kern}(\Phi_1(\sigma)) \perp \text{Kern}(\Phi_p(\sigma)) =: V_1 \oplus V_p.$$

Es ist $\Phi_1(X) = X - 1$, also $V_p = \text{Bild}(\sigma - 1)$ und $V_1 = \text{Kern}(\sigma - 1)$. Seien n_p die Dimension von V_p und n_1 die Dimension von V_1 über \mathbb{Q} . Da V_p eine Struktur als $\mathbb{Q}(\zeta_p)$ -Vektorraum hat und $\dim_{\mathbb{Q}}(\mathbb{Q}(\zeta_p)) = p - 1$, muss n_p von $p - 1$ geteilt werden.

Die durch die orthogonale Zerlegung von V induzierten Gitter $L_1 := L \cap V_1$ und $L_p := L \cap V_p$ nennen wir das von σ induzierte *Fix-Gitter* und das von σ induzierte *Bild-Gitter*. Außerdem sei $M := L_1 \perp L_p \leq L$. Im Folgenden wollen wir die Struktur von M näher untersuchen.

(4.2.1) Lemma

Seien σ , L_1 und L_p wie oben definiert.

- (i) Es existiert ein Polynom $v \in \mathbb{Z}[X]$ mit $1 = \frac{1}{p}\Phi_p + \frac{1}{p}v \cdot \Phi_1$.
- (ii) Es gilt $pL \subseteq L_1 \perp L_p \subseteq L$.

Beweis:

- (i) Nach (3.1.4) ist $\Phi_p(X) = \frac{X^p-1}{X-1} = X^{p-1} + X^{p-2} + \dots + 1$, also ist $\Phi_p(1) = p$ und somit 1 eine Nullstelle von $p - \Phi_p \in \mathbb{Z}[X]$. Da $\Phi_1(X) = X - 1$ folgt daher

$$p - \Phi_p = v \cdot \Phi_1 \text{ für ein } v \in \mathbb{Q}[X]. \quad (4.1)$$

Mit dem Lemma von Gauß muss $v \in \mathbb{Z}[X]$ gelten. Umstellen der Gleichung (4.1) liefert die Behauptung.

- (ii) Zu zeigen ist $px \in L_1 \perp L_p$ für alle $x \in L$. Wegen $(\Phi_1\Phi_p)(\sigma) = 0$ und der σ -Invarianz von L ist

$$\begin{aligned} px &= \Phi_p(\sigma)(x) + (v \cdot \Phi_1)(\sigma)(x) \in L \cap \text{Kern}(\Phi_1(\sigma)) + L \cap \text{Kern}(\Phi_p(\sigma)) \\ &= (L \cap V_1) \perp (L \cap V_p) = L_1 \perp L_p. \quad \square \end{aligned}$$

Mit diesem Lemma muss M ein Gitter der Dimension n sein, also gilt $\text{Dim}(L_p) = n_p$ und $\text{Dim}(L_1) = n_1$. Ist L gerade und von quadratfreier Stufe ℓ , so gilt $\ell L^\# \subseteq L$. Lemma (4.2.1)(ii) ist äquivalent zu $pM^\# \subseteq L^\#$. Zusammen erhält man folglich

$$\ell pM^\# \subseteq \ell L^\# \subseteq L.$$

Schneidet man mit V_p , so folgt

$$\ell p(M^\# \cap V_p) \subseteq (L \cap V_p) \Leftrightarrow \ell pL_p^\# \subseteq L_p$$

und analog $\ell p L_1^\# \subseteq L_1$.

Im Spezialfall $\text{ggT}(\ell, p) = 1$ bedeutet dies, dass die Stufe der Gitter L_1 und L_p das Produkt ℓp teilt.

Als nächstes wollen wir die Determinanten von L_1 und L_p untersuchen. Dazu werden die partiellen Dualgitter betrachtet.

(4.2.2) Lemma

Sei L ein gerades Gitter der quadratfreien Stufe ℓ und $\sigma \in \text{Aut}(L)$ von Primzahlordnung p mit $\text{ggT}(p, \ell) = 1$.

$$(i) \quad p L_1^{\#,p} \subseteq L_1.$$

$$(ii) \quad (1 - \sigma) L_p^{\#,p} \subseteq L_p.$$

Beweis:

Teil (i) folgt bereits aus der Definition des partiellen Duals, denn es gilt

$$p L_1^{\#,p} = p \left(\frac{1}{p} L_1 \cap L_1^\# \right) = L_1 \cap p L_1^\# \subseteq L_1.$$

Kommen wir nun zu Teil (ii). Definiere dazu die Projektionen $\pi_1 := \frac{1}{p} \Phi_p(\sigma)$ und $\pi_p := 1 - \pi_1$ auf V_1 bzw. V_p (vgl. Lemma (4.2.1)). Es zeigt sich:

$$\begin{aligned} (1 - \sigma) \pi_p &= (1 - \sigma)(1 - \pi_1) \\ &= 1 - \sigma - \pi_1 + \sigma \pi_1 \\ &= 1 - \sigma - \pi_1 + \frac{1}{p} (\sigma^p + \sigma^{p-1} + \dots + \sigma) \\ &= 1 - \sigma - \pi_1 + \frac{1}{p} (1 + \sigma^{p-1} + \dots + \sigma) \\ &= 1 - \sigma - \pi_1 + \pi_1 \\ &= 1 - \sigma. \end{aligned}$$

Sei nun (b_1, \dots, b_n) eine Basis von L mit zugehöriger Dualbasis $(b_1^\#, \dots, b_n^\#)$, sodass (b_1, \dots, b_{n_p}) Basis von L_p ist. Dann gilt

$$\pi_p(L^\#) = \pi_p(\langle b_1^\#, \dots, b_n^\# \rangle) = \langle b_1^\#, \dots, b_{n_p}^\# \rangle = L_p^\#.$$

Setzt man diese beiden Fakten zusammen, so erhält man

$$(1 - \sigma)L_p^\# = (1 - \sigma)\pi_p(L^\#) = (1 - \sigma)L^\# \stackrel{\text{Stufe } \ell}{\subseteq} (1 - \sigma)\frac{1}{\ell}L \stackrel{L \text{ } \sigma\text{-invariant}}{\subseteq} \frac{1}{\ell}L.$$

Außerdem ist $(1 - \sigma)L_p^\# \subseteq V_p$, also zusammen

$$(1 - \sigma)L_p^\# \subseteq \frac{1}{\ell}L \cap V_p = \frac{1}{\ell}L_p.$$

Für das partielle Dual ergibt sich hiermit

$$(1 - \sigma)L_p^{\#,p} = (1 - \sigma) \left(\frac{1}{p}L_p \cap L_p^\# \right) \subseteq \frac{1}{p}L_p \cap \frac{1}{\ell}L_p = \frac{1}{\text{ggT}(p, \ell)}L_p = L_p. \quad \square$$

Wir benötigen noch ein weiteres Hilfslemma.

(4.2.3) Lemma

Sei Λ ein gerades Gitter, dessen Stufe $p\ell$ teilt, wobei p prim und ℓ quadratfrei mit $\text{ggT}(p, \ell) = 1$ sind. Dann ist $\Lambda^{\#,p} / \Lambda \cong \Lambda^\# / \Lambda^{\#,\ell}$.

Beweis:

Sei $\psi : \Lambda^{\#,p} \rightarrow \Lambda^\# / \Lambda^{\#,\ell}, x \mapsto x + \Lambda^{\#,\ell}$.

Surjektivität: Sei $x \in \Lambda^\#$. Wegen $p\ell\Lambda^\# \subseteq \Lambda$ ist $p\Lambda^\# \subseteq \Lambda^{\#,\ell}$ und $\ell\Lambda^\# \subseteq \Lambda^{\#,p}$.

Nach Euklid existieren Zahlen $s, t \in \mathbb{Z}$ mit $sp + t\ell = 1$. Dann ist $x = spx + t\ell x \subseteq \Lambda^{\#,\ell} + \Lambda^{\#,p}$ und somit $\psi(t\ell x) = x + \Lambda^{\#,\ell}$.

Kern: Der Kern der Abbildung ist $\Lambda^{\#,p} \cap \Lambda^{\#,\ell}$. Es ist einerseits

$$\Lambda^{\#,p} \cap \Lambda^{\#,\ell} \subseteq \frac{1}{p}\Lambda \cap \frac{1}{\ell}\Lambda = \frac{1}{\text{ggT}(p, \ell)}\Lambda = \Lambda$$

und andersherum per Definition $\Lambda \subseteq \Lambda^{\#,p}$ und $\Lambda \subseteq \Lambda^{\#,\ell}$. Insgesamt ist $\text{Kern}(\psi) = \Lambda$.

Die Behauptung folgt nun mit dem Homomorphiesatz. \square

Nun wird ein wichtiger Satz zur Bestimmung der Determinanten dargelegt:

(4.2.4) Satz

Sei L wie vorher von quadratfreier Stufe ℓ und $\sigma \in \text{Aut}(L)$ mit $|\sigma| = p$, $\text{ggT}(p, \ell) = 1$. Seien außerdem L_1 und L_p die von σ induzierten Fix- und Bildgitter mit Dimensionen n_1 und n_p . Dann gilt:

$$L_1^{\#,p} / L_1 \cong \mathbb{F}_p^s \cong L_p^{\#,p} / L_p$$

für ein $s \in \{0, \dots, \min(n_1, \frac{n_p}{p-1})\}$.

Beweis:

Wir zeigen zunächst $L_1^{\#,p} / L_1 \cong L_p^{\#,p} / L_p$. Dies ist nach Lemma (4.2.3) äquivalent zu $L_1^{\#} / L_1^{\#,\ell} \cong L_p^{\#} / L_p^{\#,\ell}$.

Sei $y \in L_1^{\#}$ beliebig. Die Abbildung $L_1 \rightarrow \mathbb{Z}, x \mapsto b(x, y)$ ist eine Linearform. Da L_1 der Schnitt von L mit dem Untervektorraum V_1 ist, ist L_1 ein primitiver Teilmodul von L und diese Linearform lässt sich zu einer Linearform auf ganz L fortsetzen. Unter Ausnutzung der Isomorphie $\text{Hom}_{\mathbb{Z}}(L, \mathbb{Z}) \cong L^{\#}$ existiert ein Element $\hat{y} \in L^{\#}$, welches diese Linearform darstellt; insbesondere gilt also $b(x, y) = b(x, \hat{y})$ für alle $x \in L_1$. Zunächst zeigt sich für das Element $\hat{y} - y$:

$$b(x, \hat{y} - y) = 0 \quad \text{für alle } x \in L_1$$

und somit $\hat{y} - y \in V_1^{\perp} = V_p$. Außerdem ist

$$b(x, \hat{y} - y) = b(x, \hat{y}) - b(x, y) = b(x, \hat{y}) \in \mathbb{Z} \quad \text{für alle } x \in L_p.$$

Insgesamt gilt damit $\hat{y} - y \in L_p^\#$. Wir können somit die folgende Abbildung definieren:

$$\psi : L_1^\# \rightarrow L_p^\# / L_p^{\#, \ell}, y \mapsto (\hat{y} - y) + L_p^{\#, \ell}.$$

Wir zeigen nun, dass ψ ein wohldefinierter Epimorphismus mit Kern $L_1^{\#, \ell}$ ist und folgern dann die Behauptung erneut mit dem Homomorphiesatz.

Wohldefiniert: Es definiere $\tilde{y} \in L^\#$ eine weitere Fortsetzung. Da L von Stufe ℓ ist, gilt $\hat{y} - \tilde{y} \in L^\# \subseteq \frac{1}{\ell}L$. Wir schlussfolgern für $y \in L_1^\#$:

$$(\hat{y} - y) - (\tilde{y} - y) = \hat{y} - \tilde{y} \in \frac{1}{\ell}L \cap L_p^\# = \frac{1}{\ell}L_p \cap L_p^\# = L_p^{\#, \ell}.$$

Das Bild unter ψ hängt daher nicht von der gewählten Fortsetzung ab.

Linearität: Seien $y_1, y_2 \in L_1^\#$ mit Elementen $\hat{y}_1, \hat{y}_2 \in L^\#$, welche die zugehörigen fortgesetzten Linearformen darstellen. Für $s, t \in \mathbb{Z}$ definiert dann $s\hat{y}_1 + t\hat{y}_2$ eine Fortsetzung der Linearform $x \mapsto b(x, sy_1 + ty_2)$.

Surjektivität: Sei $y' \in L_p^\#$. Es korrespondiere $\hat{y} \in L^\#$ zu einer Fortsetzung von $x \mapsto b(x, y) \in \text{Hom}_{\mathbb{Z}}(L_p, \mathbb{Z})$ auf L . Wie zuvor liegt dann das Element $y := \hat{y} - y'$ in $L_1^\#$. Durch \hat{y} wird zudem eine Fortsetzung der Linearform $L_1 \rightarrow \mathbb{Z}, x \mapsto b(x, y)$ dargestellt, denn für alle $x \in L_1$ ist

$$b(x, y) = b(x, \hat{y} - y') = b(x, \hat{y}) - b(x, y') = b(x, \hat{y}).$$

Somit ist $\psi(y) = (\hat{y} - y) + L_1^{\#, \ell} = y' + L_1^{\#, \ell}$.

Kern: Es ist $\text{Kern}(\psi) \subseteq L_1^{\#, \ell}$, denn sei $y \in \text{Kern}(\psi)$, so gilt $\hat{y} - y \in \frac{1}{\ell}L_p^{\#, \ell} \subseteq \frac{1}{\ell}L_p \subseteq \frac{1}{\ell}L$. Da zudem $\hat{y} \in L^\# \subseteq \frac{1}{\ell}L$ ist, folgt $y = \hat{y} - (\hat{y} - y) \in \frac{1}{\ell}L$. Insgesamt gilt daher $y \in \frac{1}{\ell}L \cap L_1^\# = \frac{1}{\ell}L_1 \cap L_1^\# = L_1^{\#, \ell}$.

Andersherum ist $L_1^{\#, \ell} \subseteq \text{Kern}(\psi)$, denn sei $y \in L_1^{\#, \ell}$, so ist $y \in \frac{1}{\ell}L_1 \subseteq \frac{1}{\ell}L$. Somit gilt $\hat{y} - y \in \frac{1}{\ell}L \cap L_p^\# = \frac{1}{\ell}L_p \cap L_p^\# = L_p^{\#, \ell}$ und damit $y \in \text{Kern}(\psi)$.

Die erste Behauptung folgt nun aus dem Homomorphiesatz.

Verwendet man nun Lemma (4.2.2), so zeigt sich, dass $L_1^{\#,p}/L_1$ ein Quotient der Gruppe $L_1^{\#,p}/pL_1^{\#,p} \cong \mathbb{F}_p^{n_1}$ ist und somit die Gestalt \mathbb{F}_p^s für ein $s \in \{0, \dots, n_1\}$ besitzt.

Analog zeigt dasselbe Lemma, dass $L_p^{\#,p}/L_p$ ein Faktor der Gruppe $L_p^{\#,p}/(1-\sigma)L_p^{\#,p} \cong (\mathbb{Z}[\zeta_p]/(1-\zeta_p)\mathbb{Z}[\zeta_p])^{\frac{n_p}{p-1}}$ ist. Hier ist

$$\begin{aligned} p + (1 - \zeta_p)\mathbb{Z}[\zeta_p] &= \underbrace{1 + \dots + 1}_{p \text{ mal}} + (1 - \zeta_p)\mathbb{Z}[\zeta_p] \\ &= 1^{p-1} + \dots + 1^0 + (1 - \zeta_p)\mathbb{Z}[\zeta_p] \\ &= \zeta_p^{p-1} + \dots + \zeta_p^0 + (1 - \zeta_p)\mathbb{Z}[\zeta_p] \\ &= 0 + (1 - \zeta_p)\mathbb{Z}[\zeta_p], \end{aligned}$$

diese enthält also genau p Elemente und wir erhalten finalerweise, dass $L_p^{\#,p}/L_p$ ein Quotient von $(\mathbb{F}_p)^{\frac{n_p}{p-1}}$ ist. Damit folgt $s \leq \frac{n_p}{p-1}$. \square

Wir können entsprechend die Faktorgruppen $L_1^{\#,p}/L_1$ und $L_p^{\#,p}/L_p$ als \mathbb{F}_p -Vektorräume der Dimension s auffassen.

Nach Lemma (4.2.3) ist

$$\text{Det}(L_p) = [L_p^{\#} : L_p] = [L_p^{\#} : L_p^{\#, \ell}] \cdot [L_p^{\#, \ell} : L_p] = [L_p^{\#, p} : L_p] \cdot [L_p^{\#, \ell} : L_p].$$

Außerdem teilt p den Index $[L_p^{\#, \ell} : L_p]$ nicht, da der Exponent der Faktorgruppe $L_p^{\#, \ell}/L_p$ wegen $\ell L_p^{\#, \ell} \subseteq L_p$ ein Teiler von ℓ sein muss und $\text{ggT}(\ell, p) = 1$. Ist also s wie im vorigen Satz, so ist s bereits die p -Bewertung der Determinante von L_p . Die Überlegungen funktionieren selbstverständlich analog für L_1 . Der Satz sagt uns also, dass $\text{Det}(L_1) = p^s c$ und $\text{Det}(L_p) = p^s d$ für gewisse, zu p teilerfremde, $c, d \in \mathbb{N}$. Nach Lemma (4.2.1) ist der Index $[L : M]$ allerdings eine p -Potenz, während die Determinante von L teilerfremd zu p ist. Daher muss $c \cdot d = \text{Det}(L)$ gelten und sich der in Abbildung (4.1) dargestellte Inklusionsverband ergeben.

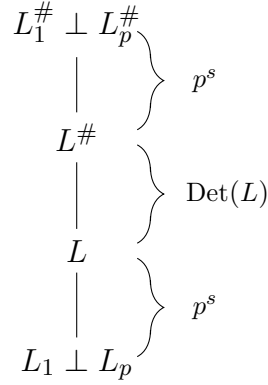


Abbildung 4.1: Inklusionsverband

Diese Überlegungen erlauben uns folgende Definition:

(4.2.5) Definition

Sei L ein gerades Gitter der quadratfreien Stufe ℓ . Sei weiterhin $\sigma \in \text{Aut}(L)$ von Ordnung p und L_1 und L_p mit Dimensionen n_1 und n_p wie zuvor die von σ induzierten Teilgitter. Die Primfaktorzerlegung von ℓ sei gegeben durch $\ell = q_1 \dots q_m$. Ist $\text{Det}(L_1) = p^s q_1^{k_{1,1}} \dots q_m^{k_{1,m}}$ und $\text{Det}(L_p) = p^s q_1^{k_{p,1}} \dots q_m^{k_{p,m}}$, so nennen wir das Tupel

$$p - (n_1, n_p) - s - q_1 - (k_{1,1}, k_{p,1}) - q_2 - (k_{1,2}, k_{p,2}) - \dots - q_m - (k_{1,m}, k_{p,m})$$

den *Typen* von σ .

Ist $m = 1$, also ℓ prim, so können wir die Schreibweise verkürzen zu

$$p - (n_1, n_p) - s - (k_1, k_p).$$

Wir können nun einige Einschränkungen an den Typen eines solchen Automorphismus machen. Im Spezialfall $p = 2$ hat die Gruppe $M^{\#,2}/M$ Exponent 2, in diesem Fall gilt eine veränderte Version von [Neb13, Lemma (4.9)].

(4.2.6) Lemma

Sei M ein gerades Gitter in einem bilinearen Vektorraum (V, b) und $M^{\#,2}/M$ habe Exponent 2. Dann enthält M ein Teilgitter isometrisch zu $\sqrt{2}U$, wobei U ein Gitter mit $U = U^{\#,2}$ ist und der Index $[M : \sqrt{2}U]$ eine Zweierpotenz.

Beweis:

Wir betrachten die 2-adische Jordanzerlegung (vgl. [CS93, (7.1)])

$$\mathbb{Z}_2 \otimes M \cong f_1 \perp \sqrt{2}f_2$$

mit einem geraden Gitter f_1 und einem ganzen Gitter f_2 , sodass $\text{Det}(f_1)$ und $\text{Det}(f_2)$ teilerfremd zu 2 sind. Da f_1 ein regulärer \mathbb{Z}_2 -Modul ist, erlaubt uns [Kne02, Satz (4.1)] eine Zerlegung

$$f_1 = E_1 \perp E_2 \perp \cdots \perp E_m$$

mit regulären Teilmoduln E_i von Dimension ≤ 2 . Da jedes E_i ein gerades Gitter sein muss, folgt $\text{Dim}(E_i) = 2$ für alle $i = 1, \dots, m$. Insbesondere hat f_1 die gerade Dimension $2m$.

Ist $m = 0$, so sind wir fertig mit $U := f_2$. Sei also nun $m > 0$.

Wir zeigen nun, f_1 enthält ein Element v mit $b(v, v) \in 2\mathbb{Z}_2^*$. Dazu schreiben wir $E_1 = \langle x, y \rangle$ und definieren s, t durch $b(x, x) \in 2^s\mathbb{Z}_2^*$ und $b(y, y) \in 2^t\mathbb{Z}_2^*$. Ist $s = 1$ oder $t = 1$, so haben wir mit $v := x$ bzw. $v := y$ ein solches Element gefunden. Seien also nun $s, t \geq 2$. Dann folgt

$$b(x - y, x - y) = b(x, x) + b(y, y) - 2b(x, y).$$

Nun muss $b(x, y)$ in \mathbb{Z}_2^* liegen, da sonst die Gram-Matrix von E_1 nur gerade Einträge und somit auch eine gerade Determinante hätte. Somit erhalten wir

$$b(x - y, x - y) \in 2^s\mathbb{Z}_2^* + 2^t\mathbb{Z}_2^* + 2\mathbb{Z}_2^* = 2(\underbrace{2^{s-1}\mathbb{Z}_2^* + 2^{t-1}\mathbb{Z}_2^*}_{\subseteq 2\mathbb{Z}_2} + \mathbb{Z}_2^*) \subseteq 2\mathbb{Z}_2^*.$$

Damit ist $v := x - y$ ein Element wie gesucht.

Nach den obigen Überlegungen können ohne Einschränkung annehmen, dass $E_1 = \langle x, v \rangle$, sonst vertausche x und y . Setze nun $w := b(v, v)x - b(v, x)v$, dann ist $b(v, w) = b(v, v)b(v, x) - b(v, x)b(v, v) = 0$ und mit $b(x, v) \in \mathbb{Z}_2^*$ außerdem

$$b(w, w) = b(v, v)^2 b(x, x) - b(v, v)b(x, v)^2 \in 2^{2+s}\mathbb{Z}_2^* + 2\mathbb{Z}_2^* = 2\mathbb{Z}_2^*.$$

Nun folgt: $\langle v \rangle \perp \langle w \rangle \perp E_2 \perp \cdots \perp E_m$ ist ein Teilgitter von f_1 vom Index 2 und $\langle v \rangle \perp \langle w \rangle = \sqrt{2}g$ für ein reguläres, ganzes Gitter g . Insgesamt ist somit

$$U' := \langle v \rangle \perp \langle w \rangle \perp E_2 \perp \cdots \perp E_m \perp \sqrt{2}f_2$$

ein Teilgitter von M vom Index 2 und mit Jordanzerlegung $(E_2 \perp \cdots \perp E_m) \perp \sqrt{2}(g \perp f_2)$.

Eine Induktion liefert nun die Behauptung. \square

Ist M wie im obigen Lemma mit Determinante $2^s a$ für ein ungerades $a \in \mathbb{N}$, dann hat das Gitter U Determinante a und Minimum $\text{Min}(U) \geq \frac{\text{Min}(M)}{2}$. Nun können wir einige Einschränkungen an die Typ-Parameter zeigen.

(4.2.7) Satz

Sei L ein gerades, n -dimensionales Gitter der quadratfreien Stufe ℓ mit Determinante $\text{Det}(L) = \ell^k$. Sei zudem $\sigma \in \text{Aut}(L)$ von Typ $p - (n_1, n_p) - s - q_1 - (k_{1,1}, k_{p,1}) - \cdots - q_m - (k_{1,m}, k_{p,m})$, wobei $\text{ggT}(p, \ell) = 1$. Dann gelten folgende Einschränkungen (für alle $i \in \underline{m}$):

- (i) $n_1 + n_p = n$.
- (ii) $s \in \{0, \dots, \min(n_1, \frac{n_p}{p-1})\}$.
- (iii) $s \equiv_2 \frac{n_p}{p-1}$ und für $p = 2$ zusätzlich $s \equiv_2 0$.

(iv) $k_{1,i} \in \{0, \dots, \min(n_1, k)\}$.

(v) $k_{1,i} \equiv_2 k$.

(vi) $k_{p,i} \in \{0, \dots, \min(n_p, k)\}$.

(vii) $k_{p,i} \equiv_2 0$.

(viii) $(2f(q_i)) \mid k_{p,i}$, wobei $f(q_i)$ den Trägheitsgrad von $q_i \mathbb{Z}_{\mathbb{Q}(\zeta_p + \zeta_p^{-1})}$ bezeichne.

(ix) $k_{1,i} + k_{p,i} = k$.

Beweis:

Eigenschaft (i) ist klar, (ii) ist Satz (4.2.4), Nummer (iv), (vi) und (ix) ergeben sich daraus, dass die Stufen von L_p und L_1 Teiler von p^ℓ sind und der Tatsache, dass $\frac{\text{Det}(L_1)\text{Det}(L_p)}{p^{2s}} = \text{Det}(L)$.

Nach [Jü15, Satz (3.1.4)(d) und Lemma (3.1.1)] existiert ein $\mathbb{Z}_{\mathbb{Q}(\zeta_p + \zeta_p^{-1})}$ -Ideal \mathfrak{a} mit

$$\text{Det}(L_p) = p^s q_1^{k_{p,1}} \dots q_m^{k_{p,m}} = p^{(p-2)\frac{n_p}{p-1}} \cdot \mathcal{N}(\mathfrak{a})^2.$$

Mit $\text{ggT}(p, \ell) = 1$ zeigt die Gleichung sofort Eigenschaft (vii). Außerdem kann man ablesen, dass $s \equiv_2 (p-2)\frac{n_p}{p-1}$ sein muss. Für (iii) bleibt also lediglich zu zeigen, dass im Falle $p = 2$ ebenfalls $s \equiv_2 \frac{n_p}{p-1}$ gilt. Hierfür verwenden wir Lemma (4.2.6) mit $M := L_p$. Für das U aus dem Lemma gilt

$$2^{n_p} = |(\sqrt{2}U)^{\#,2} / \sqrt{2}U| = |L_2^{\#,2} / L_2| [L_2 : U]^2 = 2^{s+2\nu_2([L_2:U])}.$$

Zuletzt ergibt sich (v) aus (vii) und (ix). Teil (viii) ist [Jü15, Korollar (4.1.9)]. □

Wir können nun mithilfe der Einschränkungen aus Satz (4.2.7) und den Hermite-Schranken aus (2.2) den Algorithmus (5) entwerfen, welcher die möglichen Automorphismen gerader Gitter mit quadratfreier Stufe zurückgibt.

Algorithmus 5 Aufzählung von Automorphismen-Typen

1: **Eingabe:** Quadratfreies $\ell \in \mathbb{N}$, $k \in \mathbb{N}$, $n \in \mathbb{N}$, $t \in \mathbb{N}$.

2: **Ausgabe:** Liste aller Typen von Automorphismen mit Primzahlordnung p von geraden Gittern der Stufe ℓ , Determinante ℓ^k , Dimension n , und Minimum $\geq t$, wobei $\text{ggT}(p, \ell) = 1$.

3:

4: $\text{Res} \leftarrow []$

5: $b \leftarrow$ Liste von Schranken für die Hermite-Konstante γ_i für $1 \leq i \leq n$

6: $q_1, \dots, q_m \leftarrow$ Primfaktoren von ℓ

7: **for** $p \in \mathbb{P}_{\leq n+1} - \{q_1, \dots, q_m\}$ **do**

8: $f_i :=$ Trägheitsgrad von $q_i \mathbb{Z}_{\mathbb{Q}(\zeta_p + \zeta_p^{-1})}$ für $i = 1, \dots, m$

9: **for** $n_p \in \{i(p-1) \mid 1 \leq i \leq \lfloor \frac{n}{p-1} \rfloor\}$ **do**

10: $n_1 \leftarrow n - n_p$

11: **for** $(k_{p,1}, k_{p,2}, \dots, k_{p,m}) \in \prod_{i=1}^m \{(2f_i)j \mid j \in \{0, \dots, \lfloor \frac{\min(n_p, k)}{2f_i} \rfloor\}\}$ **do**

12: $k_{1,i} \leftarrow k - k_{p,i}, \quad i \in \underline{m}$

13: **if** $\exists i \in \underline{m} : (k_{1,i} > \min(n_1, k)) \vee (k_{1,i} \not\equiv_2 k) \vee (k_{p,i} \not\equiv_2 0)$ **then**

14: **continue**

15: **for** $s \in \{0, \dots, \min(n_1, \frac{n_p}{p-1})\}$ **do**

16: **if** $s \not\equiv_2 (p-2) \frac{n_p}{p-1}$ **then continue**

17: $\gamma_1 \leftarrow \frac{t}{(p^s q_1^{k_{1,1}} \dots q_m^{k_{1,m}})^{1/n_1}}$

18: $\gamma_p \leftarrow \frac{t}{(p^s q_1^{k_{p,1}} \dots q_m^{k_{p,m}})^{1/n_p}}$

19: **if** $\gamma_1 > b_{n_1}$ oder $\gamma_p > b_{n_p}$ **then continue**

20: **if** $p = 2$ **then**

21: $\gamma'_1 \leftarrow \frac{t/2}{(q_1^{k_{1,1}} \dots q_m^{k_{1,m}})^{1/n_1}}$

22: $\gamma'_p \leftarrow \frac{t/2}{(q_1^{k_{p,1}} \dots q_m^{k_{p,m}})^{1/n_p}}$

23: **if** $\gamma'_1 > b_{n_1}$ oder $\gamma'_p > b_{n_p}$ **then continue**

24: $\text{Res} \leftarrow \text{Res} \cup [p - (n_1, n_p) - s - q_1 - (k_{1,1}, k_{p,1}) - \dots - (k_{1,m}, k_{p,m})]$

25: **return** Res

Wir haben in diesem Kapitel die möglichen Typen von Automorphismen mit Primzahlordnung studiert und Aussagen über die Determinanten der induzierten Teilgitter getroffen. Als nächstes definieren wir den Begriff des *Geschlechts* von Gittern und beschreiben eine Möglichkeit zur Aufzählung eines Geschlechts

§ 4.3 Geschlechter

(4.3.1) Definition

Wir sagen, zwei ganze Gitter L und L' liegen im selben *Geschlecht*, wenn

$$\mathbb{R} \otimes L \cong \mathbb{R} \otimes L' \quad \text{und} \quad \mathbb{Z}_p \otimes L \cong \mathbb{Z}_p \otimes L' \quad \text{für alle } p \in \mathbb{P}.$$

Dabei bezeichnet \mathbb{Z}_p den Ring der p -adischen ganzen Zahlen.

Conway und Sloane geben in [CS93, Kap. 15, Abs. 7] eine trennende Invariante der Geschlechter von Gittern an, das sogenannte *Geschlechtssymbol*. Dessen Gestalt und Eigenschaften werden im Folgenden erläutert.

Das gesamte Symbol setzt sich aus mehreren lokalen Symbolen zusammen; eines für jede Primzahl und eines für (-1) .

Sei L ein ganzes Gitter in einem bilinearen \mathbb{Q} -Vektorraum (V, b) . Das (-1) -adische Symbol ist von der Gestalt

$$+^{r-s}$$

und beschreibt die Signatur der Bilinearform b .

Für die p -adischen Symbole, wobei $p \in \mathbb{P}$, betrachte die Jordanzerlegung

$$\mathbb{Z}_p \otimes L \cong f_1 \perp \sqrt{p}f_p \perp \cdots \perp \sqrt{p}^k f_{p^k}.$$

Klar ist: Haben zwei Gitter die gleiche Signatur und über allen Primzahlen die gleiche Jordanzerlegung, so liegen sie im gleichen Geschlecht. Die Jordanzerlegung ist im allgemeinen nicht eindeutig, es ist jedoch bekannt, inwiefern sich zwei unterschiedliche Jordanzerlegungen desselben Gitters unterscheiden.

Für $p > 2$ ist die Zerlegung bis auf die Determinanten der Teilgitter f_1, \dots, f_{p^k} eindeutig, welche sich um ein Quadrat unterscheiden können. Genauer: Definiere die Invarianten

$$n_q := \dim(f_q), \quad \epsilon_q := \left(\frac{\text{Det}(f_q)}{p} \right) := \begin{cases} +1 & , \text{Det}(f_q) \in (\mathbb{Z}_p^*)^2 \\ -1 & , \text{Det}(f_q) \notin (\mathbb{Z}_p^*)^2 \end{cases}$$

für $q \in \{1, p, \dots, p^k\}$, dann sind zwei Gitter genau dann isometrisch über \mathbb{Z}_p , wenn sie dieselben Invarianten n_q und ϵ_q haben. Wir definieren nun das p -adische Symbol für $p > 2$ als das formale Produkt

$$1^{\epsilon_1 n_1} p^{\epsilon_p n_p} \dots (p^k)^{\epsilon_{p^k} n_{p^k}}.$$

Beispielsweise bedeutet das Symbol $1^{-2}3^{+5}$, dass jede Jordanzerlegung des Gitters über \mathbb{Z}_3 die Form $\mathbb{Z}_3 \otimes L \cong f_1 \perp \sqrt{3}f_3$ besitzt mit $\dim(f_1) = 2$, $\dim(f_3) = 5$. Außerdem ist $\text{Det}(f_1)$ kein Quadrat, aber $\text{Det}(f_3)$ ein Quadrat mod 3.

Der Fall $p = 2$ ist aufwändiger. Da b ursprünglich eine Bilinearform über \mathbb{Q} ist, können wir sie nach [Kne02, Satz (1.20)] diagonalisieren. Sei G_q jeweils die diagonalisierte Matrix der Bilinearform auf dem zu f_q gehörigen Teilraum. Wir definieren

$$\begin{aligned} n_q &:= \dim(f_i) \\ S_q &:= \begin{cases} \text{I} & , f_q \text{ ist kein gerades Gitter} \\ \text{II} & , f_q \text{ ist gerades Gitter} \end{cases} \\ \epsilon_q &:= \left(\frac{\text{Det}(f_q)}{2} \right) := \begin{cases} +1 & , \text{Det}(f_q) \equiv_8 \pm 1 \\ -1 & , \text{Det}(f_q) \equiv_8 \pm 3 \end{cases} \end{aligned}$$

$$t_q := \begin{cases} \text{Spur}(G_q) \bmod 8 & , S_q = \text{I} \\ 0 & , S_q = \text{II} \end{cases}$$

für $q \in \{1, 2, 4, \dots, 2^k\}$. Wir erhalten nun das vorläufige Symbol

$$1_{t_1/\text{II}}^{\epsilon_1 n_1} 2_{t_2/\text{II}}^{\epsilon_2 n_2} \dots (2^k)_{t_{2^k}/\text{II}}^{\epsilon_{2^k} n_{2^k}},$$

Wobei der Index für Komponenten mit $S_q = \text{I}$ den Wert t_q darstellt und andernfalls II ist. Dieses Symbol ist noch immer nicht eindeutig, wir benötigen zusätzliche Normierungsbedingungen. Wir fassen nun maximale Teilintervalle mit der Eigenschaft, dass alle Faktoren in den Intervallen den Typ $S_q = \text{I}$ haben (mit eckigen Klammern) zu sogenannten *Abteilen* zusammen. Außerdem trennen wir (mit Doppelpunkten) das Symbol in maximale Teilintervalle, genannt *Züge*, sodass in jedem Zug mindestens einer von je zwei aufeinanderfolgenden Faktoren den Typ $S_q = \text{I}$ hat. Beispielsweise wird so das Symbol

$$1_{\text{II}}^{+2} 2_6^{-2} 4_5^{+3} 8_{\text{II}}^{+0} 16_{\text{II}}^{+1} 32_{\text{II}}^{+2} 64_3^{+1} \quad (4.2)$$

zu

$$1_{\text{II}}^{+2} [2_6^{-2} 4_5^{+3}] 8_{\text{II}}^{+0} : 16_{\text{II}}^{+1} : 32_{\text{II}}^{+2} [64_3^{+1}].$$

Es gibt nun zwei mögliche Transformationen des Symbols, sodass diese Jordanzerlegungen desselben Gitters entsprechen.

Erstens stellen zwei solche Symbole das gleiche Gitter dar, wenn sie sich nur durch Änderungen der t_q bei den Typ-I-Faktoren unterscheiden, die Summen aller t_q pro Abteil jedoch kongruent sind modulo 8. Nach dieser Regel genügt es also, im 2-adischen Symbol jeweils nur einen Index pro Abteil anzugeben, der die Summe der enthaltenen t_q modulo 8 darstellt.

Zweitens sind zwei Symbole äquivalent, die durch beliebig häufige Anwendung der folgenden Schritte auseinander hervorgehen:

- Wähle $2^{k_1} < 2^{k_2} \in \{1, 2, \dots, 2^k\}$ so, dass die zugehörigen Komponenten im selben Zug liegen.

- Setzte $\epsilon_{2k_1} = -\epsilon_{2k_1}$ und $\epsilon_{2k_2} = -\epsilon_{2k_2}$.
- Definiere den Weg $W := \{(a, 2a) \mid a = k_1, 2k_1, \dots, \frac{1}{2}k_2\}$. In jedem Abteil, so-
dass $|\{(a, 2a) \in W \mid a \text{ oder } 2a \text{ im Abteil}\}|$ ungerade, ändere die Werte t_q aller
Komponenten so, dass sich die Summe dieser um genau 4 modulo 8 unterscheidet.

Beispielsweise korrespondieren die Symbole

$$1_{\text{II}}^{+2} [2^{-2} 4^{+3}]_3 8_{\text{II}}^{+0} [16^{-1}]_5$$

und

$$1_{\text{II}}^{+2} [2^{+2} 4^{+3}]_3 8_{\text{II}}^{+0} [16^{+1}]_1$$

zum gleichen Gitter. Es wurden die Vorzeichen bei 2 und 16 verändert. Dabei involvieren zwei Schritte das erste Abteil und ein Schritt das zweite Abteil, also muss der Index des zweiten Abteils um 4 geändert werden, der Index des ersten Abteils jedoch nicht.

Als Normierungsbedingung für diese Transformation können wir fordern, dass jeder Zug höchstens einmal das Vorzeichen $\epsilon_q = -1$ enthalten soll und dass dieses bei der ersten Komponente von Dimension $n_q > 0$ auftritt. Dies schließt die Beschreibung des 2-adischen Symbols ab. Es können nur noch Änderungen vorgenommen werden, die zur besseren Lesbarkeit beitragen, wie etwa das Auslassen der Komponenten von Dimension 0 und der II-Indizes. So hat beispielsweise das fertige 2-adische Symbol von (4.2) die Form

$$1^{-2} [2^{+2} 4^{+3}]_7 : 16^{+1} : 32^{+2} [64^{+1}]_3.$$

Als nächstes stellt sich andersherum die Frage, zu welchen möglichen Symbolen überhaupt Gitter existieren können. Dazu müssen folgende Bedingungen erfüllt sein.

- (i) Für alle $p \in \mathbb{P}$ gilt $\prod_{q \in \{1, p, p^2, \dots\}} \epsilon_q = \left(\frac{a}{p}\right)$, wobei $\text{Det}(L) = p^\alpha a$ und $\text{ggT}(p, a) = 1$.

(ii) Sei für $p \in \mathbb{P}$ der Wert k_p definiert als die Anzahl der Potenzen q von p , sodass q kein Quadrat ist, aber $\epsilon_q = -1$. Dann ist

$$r - s + \sum_{p>2} \left(4k_p + \sum_{q \in \{1, p, p^2, \dots\}} n_q(q-1) \right) \equiv 4k_2 + \sum_{q \in \{1, 2, 4, \dots\}} t_q \pmod{8}.$$

(iii) Für alle q mit $n_q = 0$ gilt $\epsilon_q = +1$.

(iv) Sei q eine Zweierpotenz. Dann gilt:

- $n_q = 0 \Rightarrow S_q = \text{II}$ und $\epsilon_q = +1$.
- $n_q = 1, \epsilon_q = +1 \Rightarrow t_q \equiv_8 \pm 1$.
- $n_q = 1, \epsilon_q = -1 \Rightarrow t_q \equiv_8 \pm 3$.
- $n_q = 2, S_q = \text{I}, \epsilon_q = +1 \Rightarrow t_q \equiv_8 0$ oder ± 2 .
- $n_q = 2, S_q = \text{I}, \epsilon_q = -1 \Rightarrow t_q \equiv_8 4$ oder ± 2 .
- $n_q \equiv_2 t_q$.
- n_q ungerade $\Rightarrow S_q = \text{I}$.

Erfüllt ein System von p -adischen Symbolen für $p \in \mathbb{P} \cup \{-1\}$ alle genannten Bedingungen, dann existiert ein ganzes Gitter mit diesen Symbolen. Die Gleichung (ii) bezeichnen Conway und Sloane auch als *Oddity-Formel*.

Wir können nun alle lokalen Symbole zum gesamten Geschlechtssymbol kombinieren. Dieses hat die Form

$$\text{I}_{r-s}(\dots), \quad \text{bzw.} \quad \text{II}_{r-s}(\dots).$$

Wobei I/II dem Typen S_1 entspricht, r, s der reellen Signatur und die Klammern die p -adischen Symbole für $p \in \mathbb{P}$ enthalten. Dabei werden die Komponenten zur Potenz

0 jeweils ausgelassen. Deren Invarianten lassen sich jedoch mithilfe der Dimension und Determinante von L sowie den angegebenen Bedingungen an die p -adischen Symbole bei Bedarf herleiten.

(4.3.2) Beispiele

- (i) Das Gitter $L := A_2 \times D_4$ hat Dimension 8 und Determinante $2^4 3^4$. Über \mathbb{Z}_2 hat es eine Jordanzerlegung

$$\mathbb{Z}_2 \otimes L \cong f_1 \perp \sqrt{2}f_2,$$

wobei f_1 und f_2 gerade Gitter sind, $\dim(f_2) = 4$ und $\text{Det}(f_2) = 225 \equiv_8 1$. Über \mathbb{Z}_3 hat es eine Zerlegung

$$\mathbb{Z}_3 \otimes L \cong g_1 \perp \sqrt{3}g_2,$$

wobei $\dim(g_2) = 4$ und $\text{Det}(g_2) = 1$, also ein Quadrat in \mathbb{Z}_3^* , ist. Das Geschlecht von L hat somit das Geschlechtssymbol

$$\text{II}_8(2^{+4} 3^{+4}).$$

- (ii) Ist L ein positiv-definites, gerades, n -dimensionales Gitter der Stufe $\ell \in \mathbb{P}_{>2}$ mit Determinante $\ell^{\frac{n}{2}}$, so ergibt sich aus der Oddity-Formel die Gleichung

$$\frac{n(\ell+1)}{2} \equiv_8 4k_\ell.$$

Weiterhin muss $\epsilon_1 \epsilon_3 = \left(\frac{\text{Det}(L)/\ell^{\frac{n}{2}}}{\ell} \right) = 1$ oder äquivalent $\epsilon_1 = \epsilon_\ell$ sein. Da genau dann $k_\ell = 1$ gilt, wenn $\epsilon_\ell = 1$ und sonst $k_\ell = 0$ ist, ist das zu L gehörige Geschlechtssymbol:

$$\begin{aligned} \text{II}_n \left(\ell^{+\frac{n}{2}} \right), & \quad \text{falls } \frac{n(\ell+1)}{2} \equiv_8 0, \\ \text{II}_n \left(\ell^{-\frac{n}{2}} \right), & \quad \text{falls } \frac{n(\ell+1)}{2} \equiv_8 4. \end{aligned}$$

(iii) Ähnlich können wir für 2-modulare Gitter vorgehen. Sei L ein positiv-definites, gerades, n -dimensionales Gitter der Stufe 2 mit Determinante $2^{\frac{n}{2}}$. Wie zuvor muss $\epsilon_1 = \epsilon_2$ sein und $k_2 = 1 \Leftrightarrow \epsilon_2 = 1$. Die Oddity-Formel ergibt

$$n \equiv_8 4k_2 + t_2.$$

Weiterhin können wir die Modularität von L ausnutzen. Sei

$$\mathbb{Z}_2 \otimes L \cong f_1 \perp \sqrt{2}f_2$$

die 2-adische Jordanzerlegung von L , dann hat $L^\#$ die Jordanzerlegung

$$\mathbb{Z}_2 \otimes L^\# = f_1 \perp \sqrt{2}^{-1}f_2$$

und somit

$$\mathbb{Z}_2 \otimes (\sqrt{2}L^\#) = f_2 \perp \sqrt{2}f_1.$$

Nun ist aber $\sqrt{2}L^\# \cong L$ und damit ein gerades Gitter, also muss auch f_2 gerade sein und $t_2 = 0$. Erneut ist allein anhand der Dimension das Geschlechtssymbol eindeutig festgelegt:

$$\begin{aligned} \Pi_n \left(2^{+\frac{n}{2}} \right), & \quad \text{falls } n \equiv_8 0, \\ \Pi_n \left(2^{-\frac{n}{2}} \right), & \quad \text{falls } n \equiv_8 4, \end{aligned}$$

genau wie bei ℓ -modularen Gittern für $\ell \in \mathbb{P}_{>2}$.

Jürgens beschreibt in [Jü15, Abschnitt (4.1.3)], welche Gestalt die Geschlechtssymbole der von einem Automorphismus von Primzahlordnung induzierten Fix- und Bildgitter L_1 und L_p besitzen.

(4.3.3) Satz

Sei L ein Gitter der primen Stufe $\ell \in \mathbb{P}$ mit einem Automorphismus σ von Typ $p - (n_1, n_p) - s - (k_1, k_p)$ für ein $p \in \mathbb{P}_{>2}$, wobei $\text{ggT}(p, \ell) = 1$. Wie zuvor seien außerdem $L_1 = L \cap \text{Kern}(\sigma - 1)$ und $L_p = L \cap \text{Bild}(\sigma - 1)$. Die Geschlechter von L , L_1 und L_p haben die Formen

$$L \in \text{II}_n(\ell^{\epsilon k}), \quad L_1 \in \text{II}_{n_1}(p^{\delta_1 s} \ell^{\epsilon_1 k_1}), \quad L_p \in \text{II}_{n_p}(p^{\delta_p s} \ell^{\epsilon_p k_p})$$

und für die Parameter $\epsilon, \delta_1, \epsilon_1, \delta_p, \epsilon_p$ gelten die folgenden Beziehungen:

- (i) $\epsilon_1 \epsilon_p = \epsilon$
- (ii) $\delta_1 \delta_p = (-1)^{\frac{s(p-1)}{2}}$
- (iii) $\delta_p = (-1)^{\frac{k_p}{f(\ell)} + \frac{p-1}{2}} \left(\binom{n_p / (p-1) + 1}{2} + \binom{s}{2} \right)$
- (iv) falls $\ell \neq 2$: $\epsilon_p = (-1)^{\frac{k_p}{f(\ell)} + \frac{p-1}{2}} \binom{k_p}{2}$
- (v) falls $\ell = 2$: $\epsilon_p = \delta_p \Leftrightarrow n_p + s(p-1) \equiv_8 0$.

Dabei bezeichnet $f(\ell)$ den Trägheitsgrad von $\ell \mathbb{Z}_{\mathbb{Q}(\zeta_p + \zeta_p^{-1})}$.

Die Geschlechtssymbole sind unter den gegebenen Voraussetzungen also eindeutig durch den Typen des Automorphismus festgelegt.

§ 4.4 Kneser-Nachbarschaftsmethode

Wir kommen nun zur Aufzählung aller Gitter eines gegebenen Geschlechtes. Kneser beschreibt in [Kne02, Abschnitt 28] eine Methode, welche Gitter in *Spinorgeschlechtern*

mithilfe einer Nachbar-Konstruktion aufzählt.

(4.4.1) Definition

Sei p eine Primzahl sowie L und M \mathbb{Z} -Gitter im gleichen Vektorraum. Man bezeichnet L und M als p -Nachbarn, falls $[L : L \cap M] = p = [M : L \cap M]$.

Sei \mathcal{G} ein Geschlecht und C die Menge aller Isometrie-Klassen von Gittern in \mathcal{G} . Der Graph mit Knotenmenge C und Kanten zwischen $C_1, C_2 \in C$ genau dann, wenn C_1 und C_2 p -Nachbarn für eine Primzahl p , heißt *Nachbarschafts-Graph*.

Kneser beschreibt, wie die Menge aller p -Nachbarn eines gegebenen Gittes L gebildet werden kann. Nach [SH98] ist der Nachbarschafts-Graph endlich. Außerdem gilt der folgende Satz:

(4.4.2) Satz

Sei L ein Gitter von Dimension ≥ 3 . Hat für jede Primzahl $q \in \mathbb{P}$ die Jordanzerlegung von $\mathbb{Z}_q \otimes L$ mindestens eine Komponente von Dimension ≥ 2 , so besteht der Nachbarschafts-Graph von L aus genau einer Zusammenhangskomponente.

Die recht schwache Bedingung zur Jordanzerlegung ist beispielsweise für alle Gitter von quadratfreier Stufe - also für alle von uns zu untersuchenden Geschlechter - erfüllt. Daher können wir durch sukzessive Nachbar-Bildung das gesamte Geschlecht aufzählen. Als Heuristik zur Auswahl, für welchen Knoten als nächstes die Nachbarn konstruiert werden, verwenden wir die Häufigkeit mit der ein Gitter bisher gefunden wurde; unter den am seltensten gefundenen Gittern wird zufällig eines ausgewählt. Es sind noch andere Strategien denkbar, wie beispielsweise eine einfache Breiten- oder Tiefensuche. Des

Weiteren können wir als Abbruchsbedingung das sogenannte *Maß* eines Geschlechtes benutzen (vgl. [Kne02, Abschnitt 35]).

(4.4.3) Definition

Es sei \mathcal{G} ein Geschlecht von Gittern und L_1, \dots, L_h ein Vertretersystem der Isometrie­klassen von Gittern in \mathcal{G} . Der Wert

$$\text{Maß}(\mathcal{G}) = \sum_{i=1}^h \frac{1}{|\text{Aut}(L_i)|}$$

heißt das *Maß* des Geschlechtes \mathcal{G} .

Das Maß eines Geschlechtes kann ohne tatsächliche Aufzählung mithilfe der *Maßformel* berechnet werden. Indem wir die Kehrwerte der Ordnungen der Automorphismengruppen aller bisher gefundenen Gitter während des Algorithmus aufsummieren, können wir also über die Differenz zum tatsächlichen Maß feststellen, ob wir bereits alle Isometrie­klassen gefunden haben. Zusätzlich können wir eine weitere nützliche Einschränkung machen: Haben wir bisher Gitter $L_1, \dots, L_{h'}$ gefunden und ist $m := \sum_{i=1}^{h'} \frac{1}{|\text{Aut}(L_i)|}$, so muss jedes weitere Gitter M im Geschlecht eine Automorphismengruppe mit $|\text{Aut}(M)| \geq \frac{1}{\text{Maß}(\mathcal{G}) - m}$ besitzen. Sobald das verbleibende Maß also kleiner als 1 ist, können wir Nachbarn mit zu kleiner Automorphismengruppe übergehen, da diese notwendigerweise isometrisch zu einem bereits gefundenen Gitter sein müssen. Insgesamt kommen wir so auf Algorithmus (6) zur Aufzählung eines Geschlechtes anhand eines Vertreters. Für die Bestimmung eines Vertreters zu einem gegebenen Geschlechtssymbol wird ein **MAGMA**-Programm aus der Diplomarbeit von David Lorch [Lor11] verwendet.

Algorithmus 6 Aufzählung aller Isometrieklassen eines Geschlechtes

```
1: Eingabe: Gitter  $L$  von Dimension  $\geq 3$ 
2: Ausgabe: Liste von Vertretern aller Isometrieklassen im Geschlecht von  $L$ 
3:
4:  $\mathcal{G} \leftarrow$  Geschlecht von  $L$ 
5:  $M \leftarrow \text{Ma\ss}(\mathcal{G})$ 
6:  $m \leftarrow \frac{1}{|\text{Aut}(L)|}$ 
7:  $Gen \leftarrow [L]$ 
8:  $Explored \leftarrow [false]$ 
9:  $NumFound \leftarrow [1]$ 
10: while  $m < M$  do
11:    $RareFound \leftarrow \{i \mid \neg Explored[i] \wedge NumFound[i] \leq NumFound[j] \ \forall j\}$ 
12:    $i \leftarrow$  zufälliges Element aus  $RareFound$ 
13:    $Neigh \leftarrow$  2-Nachbarn von  $Gen[i]$ 
14:    $Explored[i] \leftarrow true$ 
15:   for  $N \in Neigh$  do
16:      $MinAuto \leftarrow \frac{1}{M-m}$ 
17:     if  $|\text{Aut}(N)| < MinAuto$  then
18:       continue
19:     if  $\exists j : Gen[j] \cong N$  then
20:        $NumFound[j] \leftarrow NumFound[j] + 1$ 
21:     else
22:        $Gen \leftarrow Gen \cup [N]$ 
23:        $Explored \leftarrow Explored \cup [false]$ 
24:        $NumFound \leftarrow NumFound \cup [1]$ 
25:        $m \leftarrow m + \frac{1}{|\text{Aut}(N)|}$ 
26: return  $Gen$ 
```

Für Dimension $n \leq 2$ sind die Voraussetzungen von Satz (4.4.2) nicht erfüllt, dennoch ist die Aufzählung eines Geschlechtes leicht. Das Geschlechtssymbol legt insbesondere die Determinante d aller Gitter des Geschlechtes fest. Im Falle $n = 1$ muss jedes Gitter im Geschlecht folglich die Gram-Matrix $(d) \in \mathbb{Z}^{1 \times 1}$ besitzen. Kommen wir nun zum Fall $n = 2$: Die Hermite-Konstante γ_2 hat auf 4 Stellen abgerundet den Wert 1.1547. Für alle Gitter L des Geschlechtes muss also $\text{Min}(L) \leq 1.1548\sqrt{\text{Det}(L)}$ gelten. Sei $t := \text{Min}(L)$ und (e_1, e_2) eine Basis von L mit $b(e_1, e_1) = t$. Die Gram-Matrix von L bezüglich dieser Basis hat die Gestalt $\begin{pmatrix} t & x \\ x & y \end{pmatrix}$. Ohne Einschränkung gilt $-t < x < t$, sonst ersetze sukzessiv e_2 durch $e_2 + e_1$ für $x < -t$, bzw. durch $e_2 - e_1$ für $t < x$, denn $b(e_1, e_2 \pm e_1) = x \pm t$. Der Wert y ist dann eindeutig bestimmt durch $\text{Det}(L) = ty - x^2$. Diese Überlegungen ergeben Algorithmus (7).

Algorithmus 7 Aufzählung aller Isometrieklassen eines Geschlechtes von Dimension 2

```

1: Eingabe: Geschlechtssymbol  $S$  mit Dimension 2
2: Ausgabe: Liste von Vertretern aller Isometrieklassen im zugehörigen Geschlecht
3:
4:  $d \leftarrow$  durch  $S$  festgelegte Determinante
5:  $Gen \leftarrow []$ 
6: for  $t \in \{0, \dots, \lfloor 1.1548\sqrt{d} \rfloor\}$  do
7:   for  $x \in \{-t+1, \dots, t-1\}$  do
8:      $y := \frac{\text{Det}(L) - x^2}{t}$ 
9:     if  $y \notin \mathbb{Z}$  then
10:       continue
11:      $L \leftarrow$  Gitter mit Gram-Matrix  $\begin{pmatrix} t & x \\ x & y \end{pmatrix}$ 
12:     if  $L$  hat Geschlechtssymbol  $S$  und  $\nexists M \in Gen : L \cong M$  then
13:        $Gen \leftarrow Gen \cup [L]$ 
14: return  $Gen$ 

```

§ 4.5 Konstruktion von Obergittern

Haben wir einen Kandidaten für das Teilgitter $M := L_1 \perp L_p$ und für den Automorphismus $\sigma = \text{diag}(\sigma_1, \sigma_p)$ gefunden, so gilt es, die σ -invarianten Obergitter von M mit Index p^s zu bestimmen. Hierfür verwenden wir je nach Fall verschiedene Methoden. Zunächst ist auf die Konstruktion von Michael Jürgens in [Jü15, Abschnitt (1.4)] hinzuweisen. Hier wird eine Vorgehensweise beschrieben, um mithilfe der isotropen Teilräume des bilinearen \mathbb{F}_p -Vektorraums $(M^{\# \cdot p} / M, \bar{b})$, wobei

$$\bar{b} : M^{\# \cdot p} / M \times M^{\# \cdot p} / M \rightarrow \mathbb{F}_p, (x + M, y + M) \mapsto pb(x, y) + p\mathbb{Z}$$

die reduzierte Bilinearform auf den Faktorgruppen darstellt, sämtliche ganzen Obergitter von Index p^s zu bestimmen - unabhängig von σ . Da diese Methode wegen Nichtbeachtung der σ -invarianz potentiell noch mehr Gitter liefert als solche mit den geforderten Eigenschaften, ist sie unsere erste Wahl. In einigen Fällen scheitert sie jedoch an der steigenden Ressourcenintensivität.

Neben verwendet im Beweis von [Neb13, Theorem (5.9)] eine Vorgehensweise, welche die Struktur von M als orthogonale Summe benutzt, um die tatsächlich σ -invarianten Obergitter zu bestimmen. Diese Vorgehensweise passen wir nun auf unsere Situation an. Dazu bemerken wir zunächst die Tatsache, dass jedes ganze Obergitter $L \geq M$, sodass $[L : M]$ eine p -Potenz ist, ein Teilgitter von $M^{\# \cdot p} = L_1^{\# \cdot p} \perp L_p^{\# \cdot p}$ sein muss. Definiere nun $b_1 := b|_{V_1 \times V_1}$ und $b_p := b|_{V_p \times V_p}$ sowie $\bar{b}_1 := \bar{b}|_{(L_1^{\# \cdot p} / L_1) \times (L_1^{\# \cdot p} / L_1)}$ und $\bar{b}_p := \bar{b}|_{(L_p^{\# \cdot p} / L_p) \times (L_p^{\# \cdot p} / L_p)}$. Damit L ein ganzes Gitter wird, muss für beliebige Elemente $(x_1, x_p), (y_1, y_p) \in L$ gelten:

$$\begin{aligned} 0 + p\mathbb{Z} &\stackrel{!}{=} \bar{b}((x_1, x_p) + M, (y_1, y_p) + M) \\ &= pb_1(x_1, y_1) + pb(x_1, y_p) + pb(x_p, y_1) + pb_p(x_p, y_p) + p\mathbb{Z} \\ &= \bar{b}_1(x_1 + M, y_1 + M) + \bar{b}_p(x_p + M, y_p + M). \end{aligned}$$

Also $\overline{b_1}(x_1 + M, y_1 + M) = -\overline{b_p}(x_p + M, y_p + M)$. Demnach sind die ganzen Obergitter von Index p^s gegeben durch

$$L_\varphi := \{(x_1, x_p) \in L_1^{\#,p} \perp L_p^{\#,p} \mid \varphi(x_1 + L_1) = x_p + L_p\}$$

für die Isometrien $\varphi : (L_1^{\#,p}/L_1, \overline{b_1}) \rightarrow (L_p^{\#,p}/L_p, -\overline{b_p})$.

Nun zur σ -Invarianz eines solchen Gitters L_φ . Auf dem Dualgitter $M^\#$ ist σ ebenfalls ein Automorphismus, denn

$$x \in M^\# \Leftrightarrow b(x, M) \subseteq \mathbb{Z} \Leftrightarrow b(\sigma(x), \sigma(M)) \subseteq \mathbb{Z} \Leftrightarrow b(\sigma(x), M) \subseteq \mathbb{Z} \Leftrightarrow \sigma(x) \in M^\#,$$

also gilt auch $\sigma(M^{\#,p}) = M^{\#,p}$ und σ operiert auf $M^{\#,p}/M$ durch $\sigma(x + M) := \sigma(x) + M$. Analog operieren σ_1 auf $L_1^{\#,p}/L_1$ und σ_p auf $L_p^{\#,p}/L_p$. Damit ein Gitter L_φ nun σ -invariant ist, muss für alle $(x_1, x_p) \in L_\varphi$ auch $(\sigma_1(x_1), \sigma_p(x_p)) \in L_\varphi$ sein, also $\varphi(\sigma_1(x_1) + L_1) = \sigma_p(x_p) + L_p = \sigma_p(\varphi(x_1 + L_1))$. Dies führt zur Bedingung

$$\varphi \circ \sigma_1 = \sigma_p \circ \varphi. \quad (4.3)$$

Die Menge aller Isometrien bilden in höheren Dimensionen einen zu großen Suchraum.

Wir können allerdings einige Einschränkungen machen. Seien φ_0 und $\varphi_1 : (L_1^{\#,p}/L_1, \overline{b_1}) \rightarrow (L_p^{\#,p}/L_p, -\overline{b_p})$ Isometrien, welche Bedingung (4.3) erfüllen. Dann gilt

$$\sigma_1 \varphi_0 \varphi_1^{-1} \sigma_1^{-1} = (\sigma_1 \varphi_0)(\sigma_1 \varphi_1)^{-1} = \varphi_0 \sigma_p \sigma_p^{-1} \varphi_1^{-1} = \varphi_0 \varphi_1^{-1}.$$

Demnach ist $\varphi_0 \varphi_1^{-1}$ enthalten im Zentralisator $C_{O(L_1^{\#,p}/L_1)}(\sigma_1)$. Fixieren wir also eine Isometrie φ_0 , so erhalten wir alle weiteren durch Verkettung mit Zentralisatorelementen. **MAGMA** kann eine einzelne Isometrie $\tilde{\varphi}_0$ konstruieren, diese erfüllt jedoch nicht notwendigerweise Bedingung (4.3). Wir machen den Ansatz $\varphi_0 \stackrel{!}{=} \tilde{\varphi}_0 \circ u$ für $u \in O(L_1^{\#,p}/L_1)$. Dann muss gelten

$$\begin{aligned} \varphi_0 \circ \sigma_1 &= \sigma_p \circ \varphi_0 \\ \Leftrightarrow \tilde{\varphi}_0 \circ u \circ \sigma_1 &= \sigma_p \circ \tilde{\varphi}_0 \circ u \end{aligned}$$

$$\Leftrightarrow u \circ \sigma_1 \circ u^{-1} = \tilde{\varphi}_0 \circ \sigma_p \circ \tilde{\varphi}_0^{-1}.$$

Das Element u konjugiert also die Abbildung σ_1 zu $\tilde{\varphi}_0 \circ \sigma_p \circ \tilde{\varphi}_0^{-1}$. Ein solches Element lässt sich durch **MAGMA** bestimmen.

Eine letzte Beobachtung ist die Folgende: Sei φ eine Isometrie mit den gesuchten Eigenschaften und $c \in C_{\text{Aut}(L_1)}(\sigma_1)$, dann operiert c auf $L_1^{\#,p}/L_1$ und die Abbildung $\varphi_c := \varphi \circ c$ ist ebenfalls eine Isometrie mit Eigenschaft (4.3), da c im Zentralisator von σ_1 liegt. Diese Abbildung liefert somit ebenfalls ein σ -invariantes ganzes Obergitter L_{φ_c} von M . Definiere nun die Isometrie

$$\psi : L_1^{\#,p} \perp L_p^{\#,p} \rightarrow L_1^{\#,p} \perp L_p^{\#,p}, (x, y) \mapsto (c(x), y),$$

dann ist $\psi(L_{\varphi_c}) = L_{\varphi}$, also sind die Gitter isometrisch. Um Redundanz bei der Konstruktion zu vermeiden, genügt es daher, jeweils einen Vertreter nach den Restklassen modulo $C_{\text{Aut}(L_1)}(\sigma_1)$ zu wählen.

Insgesamt erhalten wir also Algorithmus (8):

Algorithmus 8 Konstruktion von σ -invarianten Obergittern

- 1: **Eingabe:** Gitter L_1, L_p , Automorphismen σ_1 von L_1 und σ_p von L_p , $p \in \mathbb{P}$, sodass $L_1^{\#,p}/L_1 \cong (\mathbb{F}_p)^s \cong L_p^{\#,p}/L_p$
 - 2: **Ausgabe:** Liste aller nicht-isometrischen, ganzen Obergitter von $L_1 \perp L_p$ von Index p^s
 - 3:
 - 4: $\tilde{\varphi}_0 \leftarrow$ Isometrie $(L_1^{\#,p}/L_1, \overline{b_1}) \rightarrow (L_p^{\#,p}/L_p, -\overline{b_p})$
 - 5: $u \leftarrow$ Element in $O(L_1^{\#,p}/L_1)$, sodass $u \circ \sigma_1 \circ u^{-1} = \tilde{\varphi}_0 \circ \sigma_p \circ \tilde{\varphi}_0^{-1}$
 - 6: $\varphi_0 \leftarrow \tilde{\varphi}_0 \circ u$
 - 7: $C \leftarrow$ Vertretersystem von $C_{O(L_1^{\#,p}/L_1)}(\sigma_1)/C_{\text{Aut}(L_1)}(\sigma_1)$
 - 8: $Results \leftarrow []$
 - 9: **for** $c \in C$ **do**
 - 10: $\varphi \leftarrow \varphi_0 \circ c$
 - 11: $L_\varphi \leftarrow \{(x_1, x_p) \in L_1^{\#,p} \perp L_p^{\#,p} \mid \varphi(x_1 + L_1) = x_p + L_p\}$
 - 12: **if** $\nexists M \in Results \mid L_\varphi \cong M$ **then**
 - 13: $Results \leftarrow Results \cup [L_\varphi]$
 - 14: **return** $Results$
-

§ 4.6 Konstruktion von Gittern mit großem Automorphismus

Mithilfe der Typen von Automorphismen mit Primzahlordnung kennen wir die Geschlechter - oder zumindest Dimension und Determinante - von Fix- und Bild-Gitter. In der Regel ist mindestens eines der Geschlechter zu groß, um es mithilfe der Kneser-Methode in akzeptabler Zeit aufzuzählen. Hat das L jedoch einen großen Automorphismus, so kann L_p eine Ideal-Gitter-Gestalt besitzen, wodurch die Konstruktion erleichtert

wird. Wir untersuchen zunächst, welche Form die Potenzen der Minimalpolynome von Automorphismen besitzen.

(4.6.1) Lemma

Ist V ein Vektorraum und $\sigma \in GL(V)$ mit Minimalpolynom $\mu_\sigma = \Phi_{n_1} \Phi_{n_2} \dots \Phi_{n_k}$, dann hat σ^d für $d \in \mathbb{N}$ das Minimalpolynom

$$\mu_{\sigma^d} = \text{kgV}(\Phi_{n_1/\text{ggT}(n_1,d)}, \dots, \Phi_{n_k/\text{ggT}(n_k,d)})$$

Beweis:

Sei zunächst $k = 1$, also $\mu_\sigma = \Phi_{n_1}$. Dann ist $|\langle \sigma^d \rangle| = \frac{n_1}{\text{ggT}(n_1,d)}$, also ist σ^d eine primitive Einheitswurzel mit $\mu_{\sigma^d} = \Phi_{n_1/\text{ggT}(n_1,d)}$. Für $k > 1$ können wir V zerlegen zu

$$V = \text{Kern}(\Phi_{n_1}(\sigma)) \oplus \text{Kern}(\Phi_{n_2}(\sigma)) \oplus \dots \oplus \text{Kern}(\Phi_{n_k}(\sigma)) =: V_1 \oplus \dots \oplus V_k$$

Somit hat $\sigma|_{V_i}$ jeweils das Minimalpolynom Φ_{n_i} für $i = 1, \dots, k$. Es folgt:

$$\mu_{\sigma^d} = \text{kgV}(\mu_{\sigma^d|_{V_1}}, \dots, \mu_{\sigma^d|_{V_k}}) = \text{kgV}(\Phi_{n_1/\text{ggT}(n_1,d)}, \dots, \Phi_{n_k/\text{ggT}(n_k,d)}). \quad \square$$

Sei nun L ein Gitter der Dimension n und $\sigma \in \text{Aut}(L)$ ein großer Automorphismus von Ordnung m . Dann hat das Minimalpolynom von σ die Form $\mu_\sigma = \Phi_m \Phi_{n_1} \dots \Phi_{n_k}$. Ist $\text{kgV}(n_1, \dots, n_k) < m$, so existiert eine Primzahl p mit $\text{kgV}(n_1, \dots, n_k) \mid \frac{m}{p} =: d$. Der Automorphismus σ^d hat Primzahlordnung p und liefert wie in Abschnitt (4.2) eine σ -invariante Zerlegung

$$V = \text{Bild}(\sigma^d - 1) \oplus \text{Kern}(\sigma^d - 1).$$

Nun ist allerdings $\text{Kern}(\sigma^d - 1) = \text{Kern}((\Phi_{n_1} \dots \Phi_{n_k})(\sigma))$ und somit $\text{Bild}(\sigma^d - 1) = \text{Kern}(\Phi_m(\sigma))$. Das zu σ^d gehörige Bildgitter $L_p = L \cap \text{Bild}(\sigma^d - 1) = L \cap \text{Kern}(\Phi_m(\sigma))$ ist also ein Sub-Ideal-Gitter von L . Das Fix-Gitter $L_1 = L \cap \text{Kern}(\sigma^d - 1)$ hat die Dimension $n - \varphi(m) < \frac{n}{2}$.

Wir betrachten nun das Minimalpolynom der Operation von σ auf den Faktorgruppen $L_1^{\#,p}/L_1$ und $L_p^{\#,p}/L_p$. Nach (4.2.4) waren $L_1^{\#,p}/L_1$ und $L_p^{\#,p}/L_p$ isomorph und der zugehörige Isomorphismus war die Komposition der drei Isomorphismen

$$\begin{aligned} L_1^{\#,p}/L_1 &\rightarrow L_1^{\#}/L_1^{\#,\ell}, x + L_1 \mapsto x + L_1^{\#,\ell} \\ L_1^{\#}/L_1^{\#,\ell} &\rightarrow L_p^{\#}/L_p^{\#,\ell}, y + L_1^{\#,\ell} \mapsto (\hat{y} - y) + L_p^{\#,\ell} \\ L_p^{\#}/L_p^{\#,\ell} &\rightarrow L_p^{\#,p}/L_p, x + L_p^{\#,\ell} \mapsto x + L_p, \end{aligned}$$

wobei $\hat{y} \in L^{\#}$ mit $b(x, y) = b(x, \hat{y})$ für alle $x \in L_1$. Man sieht leicht ein, dass $\sigma(\hat{y})$ eine mögliche Wahl für $\widehat{\sigma(y)}$ darstellt, da alle beteiligten Gitter und die Bilinearform invariant unter σ sind. Alle drei Isomorphismen kommutieren also mit σ . Daraus folgt, dass die Faktorgruppen auch als $\mathbb{Z}[\sigma]$ -Moduln, bzw. wegen $pL_p^{\#,p} \subseteq L_p$ und $pL_1^{\#,p} \subseteq L_1$ sogar als $\mathbb{F}_p[\sigma]$ -Moduln isomorph sind. Insbesondere hat σ auf $L_1^{\#,p}/L_1$ und $L_p^{\#,p}/L_p$ dasselbe Minimalpolynom. Wegen $(1 - \sigma^d)L_p^{\#,p} \subseteq L_p$ wird $L_p^{\#,p}/L_p$ zu einem $\mathbb{F}_p[\sigma]/(1 - \sigma^d) \cong \mathbb{F}_p[\zeta_d]$ -Modul. Das Minimalpolynom der Operationen von σ auf den Faktorgruppen ist somit ein Teiler von Φ_d , also (falls $\dim_{\mathbb{F}_p}(L_p^{\#,p}/L_p) = \dim_{\mathbb{F}_p}(L_1^{\#,p}/L_1) > 0$) gleich Φ_d . Im Falle $s > 0$ muss daher $\varphi(d) \leq s$ erfüllt sein.

Wir entwerfen nun den Algorithmus (9) zur Konstruktion solcher Gitter L .

Algorithmus 9 Konstruktion von Gittern mit großem Automorphismus

1: **Eingabe:** $n \in \mathbb{N}$, quadratfreies $\ell \in \mathbb{N}$, $m \in \mathbb{N}$ mit $\frac{n}{2} < \varphi(m) \leq n$.

2: **Ausgabe:** Liste von extremalen ℓ -modularen Gittern der Dimension n mit einem großen Automorphismus σ der Ordnung m , sodass ein $p \in \mathbb{P}$, $\text{ggT}(p, \ell) = 1$ existiert mit $\frac{\mu_\sigma}{\Phi_m} | (X^{\frac{m}{p}} - 1)$

3:

4: **if** $\varphi(m) = n$ **then return** Liste von ℓ -modularen extremalen Ideal-Gittern über $\mathbb{Q}(\zeta_m)$ nach Algorithmus (4).

5: $Results \leftarrow []$

6: $AutoTypes \leftarrow$ Liste von Aut.-Typen von Primzahlordnung nach Algorithmus (5)

7: **for** $p \in \{q \in \mathbb{P} \mid q|m, \text{ggT}(q, \ell) = 1\}$ **do**

8: $PossibleTypes \leftarrow \{p - (n - \varphi(m), \varphi(m)) - s - \dots \in AutoTypes \mid s = 0 \text{ oder } \varphi(\frac{m}{p}) \leq s\}$

9: **for** $t \in PossibleTypes$ **do**

10: $L_p_List \leftarrow$ Liste von Ideal-Gittern über $\mathbb{Q}(\zeta_m)$ mit durch t für L_p vorgegebene Dimension und Determinante nach Algorithmus (4)

11: **for** $L_p \in L_p_List$ **do**

12: $L_1_List \leftarrow$ Liste von allen Gittern mit durch t für L_1 vorgegebene Dimension und Determinante und mit quadratfreier Stufe nach Algorithmus (6)

13: **for** $L_1 \in L_1_List$ **do**

14: $M \leftarrow L_1 \perp L_p$

15: **if** $s = 0$ **then** $Results \leftarrow Results \cup [M]$

16: **for** $\sigma_p \in \{\sigma \in \text{Aut}(L_p) \mid \sigma \text{ op. auf } L_p^{\#,p}/L_p \text{ mit Mi.-Po. } \Phi_{\frac{m}{p}}\}$ **do**

17: **for** $\sigma_1 \in \{\sigma \in \text{Aut}(L_1) \mid \sigma \text{ op. auf } L_1^{\#,p}/L_1 \text{ mit Mi.-Po. } \Phi_{\frac{m}{p}}\}$ **do**

18: $\sigma \leftarrow \text{diag}(\sigma_1, \sigma_p)$

19: $L_List \leftarrow$ Liste σ -invarianter Obergitter von M mit Index p^s

20: **for** $L \in \{L \in L_List \mid L \text{ ist } \ell\text{-modular und extremal}\}$ **do**

21: $Results \leftarrow Results \cup [L]$

22: **return** Liste paarweise nicht-isometrischer Gitter in $Results$

Für $p = 2$ kann bei Zeile 12 des Algorithmus alternativ auch mit Lemma (4.2.6) vorgegangen werden: Man zählt stattdessen die möglichen U auf und erhält die Kandidaten für L_1 als die Obergitter von $\sqrt{2}U$ vom Index $p^{\frac{\varphi(n)-m-s}{2}}$ mit quadratfreier Stufe und ausreichend großem Minimum.

Für diesen Algorithmus wurden Einschränkungen an die Minimalpolynome der Automorphismen von Gittern gemacht, es ist also a priori nicht klar, welcher Grad von Vollständigkeit durch die Klassifikation mit unserem Algorithmus gegeben ist. Dennoch ist es gelungen, einige neue extremale Gitter zu konstruieren. Die Anzahlen gefundener Gitter sind in Tabelle (4.6) festgehalten. Für $(\ell, n) \in \{(7, 18), (7, 20), (1, 24), (2, 24), (5, 24), (1, 32), (3, 36)\}$ wurden dabei die Obergitter σ -invariant konstruiert und nicht allgemein mit Index p^s . Außerdem mussten in einigen Fällen aufgrund von Hardwarelimitierungen weitere Einbußen gemacht werden:

- Bei $\ell = 7, n = 20$ konnte $I_8(7^{+4})$ nicht aufgezählt werden. Es fehlten $\frac{11831}{4644864}$ zur tatsächlichen Masse; 46 Klassen wurden gefunden.
- Bei $\ell = 7, n = 22$ konnte $II_{10}(3^{+6}7^{-5})$ nicht aufgezählt werden. Es fehlten $\frac{5099082913}{1451520}$ zur Masse; 25901 Klassen wurden gefunden.
- Bei $\ell = 5, n = 28, m = 40$ und $m = 60$ konnte $II_{12}(2^{+8}5^{-6})$ nicht aufgezählt werden. Es fehlten $\frac{42166975007681999}{150493593600}$ zur Masse; 31318 Klassen wurden gefunden.
- Bei $\ell = 5, n = 28, m = 60$ konnte außerdem $II_{12}(3^{+8}5^{-6})$ nicht aufgezählt werden. Es fehlten $\frac{456495737}{1053455155200}$ zur Masse; 1148 Klassen wurden gefunden.
- Bei $\ell = 6, n = 28, m = 40$ und $m = 60$ konnte $II_{12}(2^{-6}3^{-6}5^{+4})$ nicht aufgezählt werden. Es fehlten $\frac{215972432375753515343}{14332723200}$ zur Masse; 19845 Klassen wurden gefunden.
- Bei $\ell = 6, n = 28, m = 40$ und $m = 60$ konnte $II_{12}(2^{+6}3^{+6}5^{+4})$ nicht aufgezählt werden. Es fehlten $\frac{106412107359534805169}{4954521600}$ zur Masse; 13214 Klassen wurden gefunden.

- Bei $\ell = 6$, $n = 28$, $m = 40$ konnte $\text{II}_{12}(2^{+6}3^{-6}5^{-4})$ nicht aufgezählt werden. Es fehlten $\frac{14081796664459639}{614400}$ zur Masse; 16490 Klassen wurden gefunden.
- Bei $\ell = 6$, $n = 28$, $m = 40$ konnte $\text{II}_{12}(2^{-6}3^{+6}5^{-4})$ nicht aufgezählt werden. Es fehlten $\frac{2379822753998274191}{199065600}$ zur Masse; 17705 Klassen wurden gefunden.
- Bei $\ell = 3$, $n = 30$, $m = 40$ und $m = 60$ konnte $\text{II}_{14}(2^{+8}3^{-7})$ nicht aufgezählt werden. Es fehlten $\frac{3785561625313}{7223692492800}$ zur Masse; 10030 Klassen wurden gefunden.
- Bei $\ell = 3$, $n = 30$, $m = 40$ und $m = 60$ konnte $\text{II}_{14}(2^{-8}3^{+7})$ nicht aufgezählt werden. Es fehlten $\frac{3694034783}{4246732800}$ zur Masse; 9905 Klassen wurden gefunden.
- Bei $\ell = 7$, $n = 30$ konnte $\text{II}_{14}(3^{+8}7^{+7})$ nicht aufgezählt werden. Es fehlten $\frac{921555651161350677986378219}{421382062080}$ zur Masse; 16509 Klassen wurden gefunden.
- Für $n = 32$ wurde nur für $\ell = 2, 3, 5$ gerechnet.
- Für $n = 36$ wurde nur für $\ell = 3$ gerechnet.

$\begin{array}{c} \ell \\ \backslash \\ n \end{array}$	1	2	3	5	6	7	11	14	15	23
2	—	—	1	—	—	—	—	—	—	—
4	—	1	1	—	—	—	1	1	—	1
6	—	—	1	—	—	1	1	—	—	—
8	1	1	1	1	1	1	1	1	1	—
10	—	—	1	—	—	—	1	—	—	—
12	—	2	1	1	1	—	—	1	1	—
14	—	—	1	—	—	—	—	—	—	—
16	2	1	2	—	1	3	—	—	1	—
18	—	—	1	—	—	—	—	—	—	—
20	—	1	3	—	1	—	—	—	—	—
22	—	—	$2(1^*)$	—	—	—	—	—	—	—
24	1	$8(2^*)$	1	1	$5(3^*)$	—	—	—	—	—
26	—	—	2	—	—	—	—	—	—	—
28	—	$35(25^*)$	$3(2^*)$	—	—	—	—	—	—	—
32	—	2	$67(65^*)$	—	—	—	—	—	—	—
36	—	—	—	—	—	—	—	—	—	—

Tabelle 4.1: Anzahl der durch Algorithmus (9) konstruierten extremalen stark ℓ -modularen Gitter in Dimension $n \leq 28$ und $n = 32, 36$ sowie ggf. der Anzahl der bisher unbekannten Gitter darunter

§ 4.7 Vollständigkeit der Ergebnisse

In diesem letzten Abschnitt wollen wir die Gitter genauer untersuchen, die von Algorithmus (9) nicht gefunden werden. Genauer: Wir klassifizieren die Möglichkeiten für die

charakteristischen Polynome von Gittern, die durch den Algorithmus nicht konstruiert werden können. Dazu untersuchen wir die Menge aller Kandidaten für charakteristische Polynome mithilfe der "Kompatibilität" der Dimensionen der von σ -Potenzen induzierten Fix- und Bildgitter, der Ordnungen der Teil-Automorphismen σ_1 und σ_p und mithilfe der Gleichheit induzierter Teilgitter.

Sei σ ein Automorphismus eines Gitters L in einem n -dimensionalen Vektorraums V mit Ordnung $|\sigma| = m$, dann ist

$$\chi_\sigma = \Phi_{d_1}^{c_1} \dots \Phi_{d_k}^{c_k}$$

für die Teiler $d_1 < d_2 < \dots < d_k$ von m und gewisse Potenzen c_i . Damit σ Ordnung m hat, muss daher zumindest $\text{kgV}\{d_i | c_i > 0\} = m$ gelten. Immer wenn hierbei außerdem

$$c_k = 1 \text{ und } \text{kgV}\{d_i \mid i \in \{1, \dots, k-1\} \text{ und } c_i > 0\} \mid \frac{m}{p}$$

für ein $p \in \mathbb{P}$, $\text{ggT}(p, \ell) = 1$ erfüllt ist, wird L von Algorithmus (9) konstruiert. Diesen Fällen müssen wir also von vorneherein keine Beachtung mehr schenken.

Die Dimension eines Hauptraumes $\text{Kern}(\Phi_{d_i}(\sigma))$ ist $c_i \varphi(d_i)$. Damit folgt ähnlich wie in Lemma (4.6.1):

$$\text{Dim}(\text{Kern}(\sigma^d - 1)) = \sum_{d_i \mid d} c_i \varphi(d_i)$$

für alle $d \in \mathbb{N}_0$ und $n = \sum_{i=1}^k c_i \varphi(d_i)$. Kennt man daher für Primteiler $p_1, \dots, p_t \mid m$ die Typen

$$\begin{aligned} p_1 - (n_{1,1}, \dots \\ p_2 - (n_{2,1}, \dots \\ \vdots \\ p_t - (n_{t,1}, \dots \end{aligned}$$

der Automorphismen $\sigma^{\frac{m}{p_1}}, \sigma^{\frac{m}{p_2}}, \dots, \sigma^{\frac{m}{p_t}}$, so muss der Vektor $c := (c_1, \dots, c_k) \in \mathbb{N}_0^k$ eine Lösung des Gleichungssystems $cM = (n_{1,1}, n_{2,1}, \dots, n_{t,1}, n)$ mit der Matrix

$$M \in \mathbb{N}_0^{k \times (t+1)}, \quad M_{i,j} := \begin{cases} \varphi(d_i) & , d_i \mid \frac{m}{p_j} \text{ oder } j = t+1 \\ 0 & , \text{sonst} \end{cases}$$

sein.

Als weiteres Ausschlusskriterium können wir die Ordnungen von σ_1 und σ_p betrachten. Für jeden Primteiler $p \mid m$ können wir die Ordnung des Automorphismus $\sigma_1 = \sigma|_{\text{Kern}(\sigma^{\frac{m}{p}} - 1)}$ anhand des charakteristischen Polynoms berechnen durch

$$|\sigma_1| = \text{kgV}\{d_i \mid \nu_p(d_i) < \nu_p(m) \text{ und } c_i > 0\}.$$

Analog ist für $\sigma_p = \sigma|_{\text{Bild}(\sigma^{\frac{m}{p}} - 1)}$

$$|\sigma_p| = \text{kgV}\{d_i \mid \nu_p(d_i) = \nu_p(m) \text{ und } c_i > 0\}.$$

Falls es also möglich ist, für ein p mittels Geschlechtsaufzählung alle Kandidaten für das Fix- oder Bildgitter zu konstruieren, so können wir überprüfen, ob einer der Kandidaten einen Automorphismus der passenden Ordnung hat. Ist dies nicht der Fall, so können die untersuchten Werte für c_i zu keinem charakteristischen Polynom korrespondieren.

Zu guter Letzt beobachten wir, dass in gewissen Fällen Automorphismen verschiedener Primzahlordnungen p_1, p_2 dasselbe Teilgitter induzieren. Dies ist genau dann der Fall, wenn für jeden Teiler $\Phi_{d_i} \mid \chi_\sigma$, immer *entweder* $\nu_{p_1}(d_i) = \nu_{p_1}(m)$ *oder* $\nu_{p_2}(d_i) = \nu_{p_2}(m)$ gilt. Das einfachste Beispiel hierfür wäre $\chi_\sigma = \Phi_2\Phi_3$; in diesem Fall induzieren σ^2 und σ^3 dasselbe Teilgitter in L . Induzieren zwei Automorphismen dasselbe Teilgitter M , so ist der Index $[L : M]$ sowohl eine p_1 - als auch eine p_2 -Potenz und demnach 1. Widerspricht dies den Typen von $\sigma^{\frac{m}{p_1}}$, bzw. $\sigma^{\frac{m}{p_2}}$ oder den Schranken an die Hermite-Konstanten, so lässt sich der Kandidat für das charakteristische Polynom ausschließen.

Diese Charakterisierung der möglichen charakteristischen Polynome wenden wir nun einmal beispielhaft auf 3-modulare Gitter der Dimension 24 an.

(4.7.1) Satz

Sei L ein extremales 3-modulares Gitter in einem bilinearen Vektorraum (V, b) der Dimension 24. Dann hat L keine Automorphismen der Ordnung 7 sowie der Ordnung $p \in P_{\geq 13}$. Ist $\sigma \in \text{Aut}(L)$ von Ordnung m mit $12 < \varphi(m) \leq 24$, so ist $m \in \{27, 33, 48, 54, 60, 66, 72\}$. Außerdem gelten folgende Einschränkungen:

- $m = 27 \Rightarrow \Phi_{27} | \mu_\sigma$
- $m = 33 \Rightarrow \chi_\sigma = \Phi_1^2 \Phi_3 \Phi_{11}^2$
- $m = 48 \Rightarrow \chi_\sigma = \Phi_{16} \Phi_{48}$
- $m = 54 \Rightarrow \Phi_{24} | \mu_\sigma$ und σ^{27} hat Typ $2 - (0, 24) - 0 - (0, 12)$.
- $m = 60 \Rightarrow \chi_\sigma = \Phi_4^2 \Phi_{12} \Phi_{20}^2$
- $m = 66 \Rightarrow \mu_\sigma | \Phi_2 \Phi_6 \Phi_{22} \Phi_{66}$
- $m = 72 \Rightarrow \Phi_8 | \mu_\sigma$

Beweis:

Es muss $\text{Min}(L) \geq 6$ sein. Algorithmus (5) liefert uns 50 mögliche Automorphisentypen. Zählt man alle durch die Typen festgelegten Geschlechter mit Dimensionen ≤ 12 auf, so sieht man, dass in vielen Fällen darin keine Gitter mit Minimum ≥ 6 enthalten sind, wodurch diese Typen ausgeschlossen werden können. Die verbleibenden Typen von Automorphismen der Ordnung $p \in \mathbb{P} - \{3\}$ sind:

$$\begin{array}{ll}
 2 - (12, 12) - 12 - (6, 6) & 2 - (0, 24) - 0 - (0, 12) \\
 5 - (8, 16) - 4 - (8, 4) & 5 - (8, 16) - 4 - (4, 8)
 \end{array}$$

$$5 - (0, 24) - 0 - (0, 12)$$

$$7 - (0, 24) - 0 - (0, 12)$$

$$11 - (4, 20) - 2 - (2, 10)$$

$$13 - (0, 24) - 0 - (0, 12).$$

Für den Typen $2 - (12, 12) - 12 - (6, 6)$ ist sowohl das Fix-, als auch das Bildgitter isometrisch zu $\sqrt{2}K_{12}$, wobei K_{12} das *Coxeter-Todd-Gitter* bezeichnet. Im Falle des Typen $5 - (8, 16) - 4 - (8, 4)$ liegt das Fixgitter im Geschlecht $\text{II}_8(3^{+8}5^{+4})$. Dieses Geschlecht enthält 5 Gitter mit $\text{Minimum} \geq 6$. Beim Typen $5 - (8, 16) - 4 - (4, 8)$ ist $\text{II}_8(3^{+4}5^{+4})$ das Geschlechtssymbol des Fixgitters. Dieses Geschlecht enthält 4 Gitter von $\text{Minimum} \geq 6$. Schlussendlich ist $\text{II}_4(3^{-2}11^{+2})$ das Geschlechtssymbol des Fixgitters beim Typen $11 - (4, 20) - 2 - (2, 10)$. Dieses Geschlecht enthält nur ein einziges Gitter mit $\text{Minimum} \geq 6$. Insbesondere stellt man fest, dass keine Automorphismen der Ordnung > 13 existieren. Mit dieser Einschränkung, zusammen mit der Bedingung $12 < \varphi(m) \leq 24$, muss m a priori in der Menge

$$\{25, 27, 32, 33, 40, 44, 45, 48, 50, 54, 60, 66, 72, 84, 90\}$$

liegen. Für diese Fälle zählen wir mithilfe der Automorphismen mit dem in diesem Abschnitt beschriebenen Verfahren die möglichen charakteristischen Polynome der möglicherweise nicht von Algorithmus (9) gefundenen Gitter auf und erhalten so 400 verschiedene Polynome. Die Möglichkeiten $m = 25, 32, 40, 44$ und 50 können so bereits ausgeschlossen werden. Für den Rest machen wir nun eine Fallunterscheidung:

$m = 27 = 3^3$: Damit σ Ordnung 27 hat, muss Φ_{27} das Minimalpolynom teilen. Weitere Einschränkungen können wir nicht machen.

$m = 33 = 3 \cdot 11$: Hier zeigt eine Betrachtung der Polynome, dass die einzigen Möglichkeiten für das Minimalpolynom μ_σ entweder $\Phi_3\Phi_{11}$ oder $\Phi_1\Phi_3\Phi_{11}$ sind. Im Falle $\mu_\sigma = \Phi_3\Phi_{11}$ hat das von σ^{11} induzierte Teilgitter

$$L \cap \text{Kern}(\sigma^{11} - 1) \perp L \cap \text{Bild}(\sigma^{11} - 1) = L \cap \text{Kern}(\Phi_{11}(\sigma)) \perp L \cap \text{Kern}(\Phi_3)(\sigma))$$

nach Lemma (4.2.1) einen 3-Potenz-Index in L . Allerdings induziert σ^3 dasselbe Teilgitter, daher hat es gleichzeitig Index 11^2 in L . Dies ist ein Widerspruch, also kann der Fall nicht eintreten. Die einzige Möglichkeit für das Minimalpolynom ist daher $\mu_\sigma = \Phi_1\Phi_3\Phi_{11}$ bei charakteristischem Polynom $\chi_\sigma = \Phi_1^2\Phi_3\Phi_{11}$.

$m = 45 = 3^2 \cdot 5$. Hier teilt immer Φ_9 das Minimalpolynom μ_σ . Demnach hat $\sigma|_{\text{Kern}(\sigma^9-1)}$ die Ordnung 9. Keines der 9 möglichen Fixgitter von σ^9 hat jedoch einen Automorphismus der Ordnung 9, also kann auch dieser Fall nicht auftreten.

$m = 48 = 2^4 \cdot 3$: Das einzig mögliche charakteristische Polynom in diesem Falle ist nach unserer Aufzählung $\chi_\sigma = \Phi_{16}\Phi_{48}$.

$m = 54 = 2 \cdot 3^3$: Das Minimalpolynom wird immer von Φ_{54} geteilt. Und der Typ von σ^{27} ist in jedem Falle $2 - (0, 24) - 0 - (0, 12)$, mehr lässt sich allerdings nicht aussagen.

$m = 60 = 2^2 \cdot 3 \cdot 5$: Hier zeigt die Auflistung der Polynome, dass immerzu Φ_{20} das Minimalpolynom μ_σ teilen muss. Da σ^{30} ein Automorphismus der Ordnung 2 ist, hat σ^{30} den Typen $2 - (12, 12) - 12 - (6, 6)$ oder $2 - (0, 24) - 0 - (0, 12)$. Eine genaue Betrachtung der möglichen charakteristischen Polynome zeigt in jedem Fall $\Phi_{20}|\mu_\sigma$. Damit ist $\sigma|_{\text{Bild}(\sigma^{30}-1)}$ ein Automorphismus des von σ^{30} induzierten Bildgitters, dessen Ordnung ein Vielfaches von 20 ist. Das skalierte Coxeter-Todd Gitter $\sqrt{2}K_{12}$ hat jedoch keinen Automorphismus der Ordnung 20, also muss σ^{30} den Typen $2 - (0, 24) - 0 - (0, 12)$ haben. Für alle Teiler $\Phi_a|\mu_\sigma$ gilt daher $a \nmid 30$, also $4|a$. Die einzigen Möglichkeiten für das Minimalpolynom von σ sind insgesamt $\Phi_{12}\Phi_{20}$ und $\Phi_4\Phi_{12}\Phi_{20}$. Im ersten Falle ist erneut der Index des von σ^{20} induzierten Teilgitters $L \cap \text{Kern}(\Phi_{20}(\sigma)) \perp L \cap \text{Kern}(\Phi_{12}(\sigma))$ in L nach Lemma (4.2.1) eine 3-Potenz. Das gleiche Teilgitter wird allerdings auch von σ^{12} induziert und hat somit Index 5^4 in L . Dies ist ein Widerspruch, also kann dieser Fall nicht auftreten. Das einzig mögliche charakteristische Polynom bleibt damit $\Phi_4^2\Phi_{12}\Phi_{20}^2$.

Im Falle $m = 66$ muss Φ_{22} das Minimalpolynom μ_σ teilen, wie eine Betrachtung der aufgezählten Möglichkeiten für das charakteristische Polynom zeigt. Die Ordnung des Automorphismus $\sigma|_{\text{Bild}(\sigma^{33}-1)}$ ist also ein Vielfaches von 22, jedoch hat $\sqrt{2}K_{12}$ auch keinen Automorphismus der Ordnung 22, also muss σ^{33} den Typen $2 - (0, 24) - 0 - (0, 12)$ haben. Damit gilt $\mu_\sigma | \Phi_2 \Phi_6 \Phi_{22} \Phi_{66}$.

$m = 72 = 2^3 \cdot 3^2$: Man kann beobachten, dass immerzu Φ_8 das Minimalpolynom teilt, mehr lässt sich nicht aussagen.

$m = 84 = 2^2 \cdot 3 \cdot 7$: Es gibt nur zwei mögliche charakteristische Polynome. Diese sind $\Phi_{28} \Phi_{42}$ und $\Phi_{21} \Phi_{28}$. Da $\text{Kern}(\sigma^{42} - 1)$ in jedem Fall nicht trivial ist, hat σ^{42} Typ $2 - (12, 12) - 12 - (6, 6)$. Der Automorphismus σ^{28} der Ordnung 3 induziert allerdings das gleiche Teilgitter wie σ^{42} , also muss der Index gleichzeitig eine 3-Potenz und 2^{12} sein. Dies ist ein Widerspruch, also kann der Fall nicht eintreten.

$m = 90 = 2 \cdot 3^2 \cdot 5$: Es ist $\text{Kern}(\sigma^{18})$ stets nicht-trivial, also hat σ^{18} Typ $5 - (8, 16) - 4 - (4, 8)$ oder $5 - (8, 16) - 4 - (4, 8)$. Keiner der Kandidaten für die Fix-Gitter hat jedoch Automorphismen mit passender Ordnung, also kann auch dieser Fall nicht auftreten. \square

Die Methoden zur Einschränkung der Möglichkeiten für die charakteristischen Polynome wurde ebenfalls in Magma implementiert. Eine Anwendung auf alle l, n und m lässt uns bereits viele mögliche Automorphismenordnungen ausschließen. Welche Ordnungen nach unseren Einschränkungen jeweils nicht möglich sind, kann man den Ergebnissen der Subideal-Gitter-Klassifikation in Anhang (6.2) entnehmen. Besonders stark sind die folgenden Ergebnisse:

(4.7.2) Satz

- Alle bisher unbekannten 5-modularen Gitter der Dimension 20 haben keine großen Automorphismen.
- Alle bisher unbekannten 7-modularen Gitter der Dimension 20 haben keine großen Automorphismen.
- Falls der Algorithmus alle Gitter mit den gegebenen Voraussetzungen gefunden hat, existieren keine weiteren 7-modularen Gitter der Dimension 22 mit großem Automorphismus.
- Alle bisher unbekannten 11-modularen Gitter der Dimension 22 haben keine großen Automorphismen.

5 Zusammenfassung und Ausblick

Nach einem einführenden Kapitel zum Begriff eines modularen Gitters und zur Dichte eines Gitters haben wir die Theorie der Ideal-Gitter dargelegt, welche in einem Algorithmus von Jürgens in [Jü15] bereits klassifiziert werden konnten. Wir haben die Schritte des Algorithmus von Jürgens genau beschrieben und erfolgreich in **MAGMA** implementiert. So konnten die Ergebnisse von Jürgens rekonstruiert und auf einige - bisher nicht von Jürgens aufgezählte - Dimensionen erweitert werden.

In Kapitel 4 haben wir zunächst die von Jürgens und Nebe beschriebene Theorie zu den Typen von Gitterautomorphismen mit Primzahlordnung aufgegriffen sowie in gewissen Fällen verallgemeinert. Anschließend wurde der Begriff eines Gittergeschlechtes nach [CS93] eingeführt und das Kneser'sche Nachbarschaftsverfahren erläutert, mit dessen Hilfe wir relevante Geschlechter aufzählen können. Des Weiteren haben wir Methoden von Jürgens und Nebe auf unsere Situation angepasst, mit deren Hilfe wir ganzzahlige (σ -invariante) Obergitter konstruieren können. Alle beschriebenen Aussagen und Methoden führten wir nun zu einem finalen Algorithmus zusammen, der gewisse extremale modulare Gitter mit großen Automorphismen konstruieren kann. Mit diesem Algorithmus konnten wir insgesamt 98 neue extremale modulare Gitter konstruieren. Dennoch unterliegt der Algorithmus in seiner aktuellen Fassung noch einigen Einschränkungen. Besonders die Tatsache, dass wir im Zuge dieser Arbeit noch sehr wenig über Automorphismen von Primzahlordnung p aussagen konnten, sodass p ein Teiler der Stufe

des Gitters ist, stellt eine große Erschwerung dar und verhindert in vielen Fällen eine vollständigere Klassifikation. Im letzten Abschnitt der Arbeit haben wir schließlich die charakteristischen Polynome all solcher Gitter klassifiziert, die durch den bestehenden Algorithmus gegebenenfalls *nicht* gefunden werden können.

Ein Ansatzpunkt für weitere Forschung zu diesem Thema ist insbesondere die Untersuchung von Automorphismen mit nicht zur Stufe teilerfremder Ordnung. Eine Charakterisierung solcher Automorphismen könnte den Algorithmus potentiell deutlich verbessern. Des Weiteren ist eine Optimierung der Teilschritte von Interesse. Besonders die Konstruktion eines Repräsentanten sowie die vollständige Aufzählung von Geschlechtern versagt in hohen Dimensionen schnell an Hardware-Limitierungen. Zumindest in Spezialfällen ist eventuell eine effizientere Berechnung möglich. Zuletzt könnte möglicherweise die Klassifikation der nicht-gefundenen Gitter allgemein oder in Spezialfällen verfeinert werden.

6 Anhang

§ 6.1 Ergebnisse der Ideal-Gitter-Klassifikation

ℓ	Dim	Gesamtzahl(extremal)	K	Minimum									
				2	4	6	8	10	12	14	16	18	
1	8	1(1)	$\mathbf{Q}(\zeta_{15})$	1	—	—	—	—	—	—	—	—	
	16	1(1)	$\mathbf{Q}(\zeta_{40})$	1	—	—	—	—	—	—	—	—	
	24	4(1)	$\mathbf{Q}(\zeta_{35})$	—	1	—	—	—	—	—	—	—	
			$\mathbf{Q}(\zeta_{45})$	1	—	—	—	—	—	—	—		
			$\mathbf{Q}(\zeta_{54})$	1	—	—	—	—	—	—	—		
			$\mathbf{Q}(\zeta_{75})$	1	—	—	—	—	—	—	—		
	32	7(5)	$\mathbf{Q}(\zeta_{51})$	—	2	—	—	—	—	—	—	—	
			$\mathbf{Q}(\zeta_{68})$	1	1	—	—	—	—	—	—	—	
			$\mathbf{Q}(\zeta_{80})$	1	1	—	—	—	—	—	—	—	
			$\mathbf{Q}(\zeta_{120})$	—	1	—	—	—	—	—	—	—	
2	4	1(1)	$\mathbf{Q}(\zeta_8)$	1	—	—	—	—	—	—	—		
	8	1(1)	$\mathbf{Q}(\zeta_{16})$	1	—	—	—	—	—	—	—		
	12	1(1)	$\mathbf{Q}(\zeta_{36})$	1	—	—	—	—	—	—	—		

ℓ	Dim	Gesamtzahl(extremal)	K	Minimum								
				2	4	6	8	10	12	14	16	18
2	16	2(1)	$Q(\zeta_{32})$	1	—	—	—	—	—	—	—	—
			$Q(\zeta_{40})$	—	1	—	—	—	—	—	—	—
	20	1(1)	$Q(\zeta_{33})$	—	1	—	—	—	—	—	—	—
	24	2(1)	$Q(\zeta_{56})$	—	1	—	—	—	—	—	—	—
			$Q(\zeta_{72})$	1	—	—	—	—	—	—	—	—
	32	13(4)	$Q(\zeta_{51})$	—	—	3	—	—	—	—	—	—
			$Q(\zeta_{64})$	1	1	—	—	—	—	—	—	—
			$Q(\zeta_{68})$	—	3	—	—	—	—	—	—	—
			$Q(\zeta_{80})$	—	1	1	—	—	—	—	—	—
			$Q(\zeta_{96})$	—	1	—	—	—	—	—	—	—
			$Q(\zeta_{120})$	—	2	—	—	—	—	—	—	—
	36	6(3)	$Q(\zeta_{57})$	—	—	3	—	—	—	—	—	—
			$Q(\zeta_{76})$	—	1	—	—	—	—	—	—	—
			$Q(\zeta_{108})$	1	1	—	—	—	—	—	—	—
3	4	1(1)	$Q(\zeta_{12})$	1	—	—	—	—	—	—	—	—
	6	1(1)	$Q(\zeta_9)$	1	—	—	—	—	—	—	—	—
	8	1(1)	$Q(\zeta_{24})$	1	—	—	—	—	—	—	—	—
	12	2(1)	$Q(\zeta_{21})$	—	1	—	—	—	—	—	—	—
			$Q(\zeta_{36})$	1	—	—	—	—	—	—	—	—
	16	3(2)	$Q(\zeta_{40})$	—	1	—	—	—	—	—	—	—
			$Q(\zeta_{48})$	1	—	—	—	—	—	—	—	—
			$Q(\zeta_{60})$	—	1	—	—	—	—	—	—	—
	18	1(—)	$Q(\zeta_{27})$	1	—	—	—	—	—	—	—	—
	24	7(1)	$Q(\zeta_{39})$	—	—	1	—	—	—	—	—	—
			$Q(\zeta_{52})$	—	1	—	—	—	—	—	—	—
			$Q(\zeta_{56})$	—	2	—	—	—	—	—	—	—
			$Q(\zeta_{72})$	1	1	—	—	—	—	—	—	—

ℓ	Dim	Gesamtzahl(extremal)	K	Minimum								
				2	4	6	8	10	12	14	16	18
3	32	13(7)	$Q(\zeta_{80})$	—	2	4	—	—	—	—	—	—
			$Q(\zeta_{96})$	1	—	1	—	—	—	—	—	—
			$Q(\zeta_{120})$	—	3	2	—	—	—	—	—	—
	36	8(—)	$Q(\zeta_{57})$	—	1	2	—	—	—	—	—	—
			$Q(\zeta_{63})$	—	1	1	—	—	—	—	—	—
			$Q(\zeta_{76})$	—	—	1	—	—	—	—	—	—
			$Q(\zeta_{108})$	1	—	1	—	—	—	—	—	—
5	8	1(1)	$Q(\zeta_{15})$	—	1	—	—	—	—	—	—	—
	12	1(1)	$Q(\zeta_{21})$	—	1	—	—	—	—	—	—	—
	16	1(—)	$Q(\zeta_{40})$	—	1	—	—	—	—	—	—	—
	24	5(1)	$Q(\zeta_{35})$	—	—	—	1	—	—	—	—	—
			$Q(\zeta_{45})$	—	1	—	—	—	—	—	—	—
			$Q(\zeta_{56})$	—	1	—	—	—	—	—	—	—
			$Q(\zeta_{72})$	—	—	1	—	—	—	—	—	—
			$Q(\zeta_{84})$	—	1	—	—	—	—	—	—	—
	32	10(—)	$Q(\zeta_{80})$	—	1	—	1	—	—	—	—	—
			$Q(\zeta_{96})$	—	1	—	—	—	—	—	—	—
			$Q(\zeta_{120})$	—	—	2	5	—	—	—	—	—
	36	8(—)	$Q(\zeta_{63})$	—	1	—	7	—	—	—	—	—
6	8	1(1)	$Q(\zeta_{24})$	—	1	—	—	—	—	—	—	—
	12	1(1)	$Q(\zeta_{28})$	—	1	—	—	—	—	—	—	—
	16	2(1)	$Q(\zeta_{40})$	—	—	1	—	—	—	—	—	—
			$Q(\zeta_{48})$	—	1	—	—	—	—	—	—	—
	20	1(1)	$Q(\zeta_{33})$	—	—	1	—	—	—	—	—	—
	24	5(2)	$Q(\zeta_{56})$	—	1	—	1	—	—	—	—	—
			$Q(\zeta_{72})$	—	1	1	—	—	—	—	—	—
			$Q(\zeta_{84})$	—	—	—	1	—	—	—	—	—

ℓ	Dim	Gesamtzahl(extremal)	K	Minimum								
				2	4	6	8	10	12	14	16	18
6	32	12(−)	$\mathbf{Q}(\zeta_{80})$	−	−	1	5	−	−	−	−	−
			$\mathbf{Q}(\zeta_{96})$	−	1	−	1	−	−	−	−	−
			$\mathbf{Q}(\zeta_{120})$	−	−	−	4	−	−	−	−	−
7	6	1(1)	$\mathbf{Q}(\zeta_7)$	−	1	−	−	−	−	−	−	−
	8	1(1)	$\mathbf{Q}(\zeta_{24})$	−	1	−	−	−	−	−	−	−
	12	1(−)	$\mathbf{Q}(\zeta_{28})$	−	1	−	−	−	−	−	−	−
	16	4(3)	$\mathbf{Q}(\zeta_{40})$	−	−	1	−	−	−	−	−	−
			$\mathbf{Q}(\zeta_{48})$	−	1	1	−	−	−	−	−	−
			$\mathbf{Q}(\zeta_{60})$	−	−	1	−	−	−	−	−	−
	20	1(−)	$\mathbf{Q}(\zeta_{44})$	−	−	1	−	−	−	−	−	−
	24	8(−)	$\mathbf{Q}(\zeta_{56})$	−	1	2	2	−	−	−	−	−
			$\mathbf{Q}(\zeta_{72})$	−	1	1	−	−	−	−	−	−
	32	19(−)	$\mathbf{Q}(\zeta_{80})$	−	−	1	1	2	−	−	−	−
			$\mathbf{Q}(\zeta_{96})$	−	1	2	3	−	−	−	−	−
			$\mathbf{Q}(\zeta_{120})$	−	−	2	7	−	−	−	−	−
11	4	1(1)	$\mathbf{Q}(\zeta_{12})$	−	1	−	−	−	−	−	−	−
	8	2(1)	$\mathbf{Q}(\zeta_{15})$	−	−	1	−	−	−	−	−	−
			$\mathbf{Q}(\zeta_{24})$	−	1	−	−	−	−	−	−	−
	10	1(1)	$\mathbf{Q}(\zeta_{11})$	−	−	1	−	−	−	−	−	−
	12	1(−)	$\mathbf{Q}(\zeta_{36})$	−	1	−	−	−	−	−	−	−
	16	5(−)	$\mathbf{Q}(\zeta_{40})$	−	−	1	1	−	−	−	−	−
			$\mathbf{Q}(\zeta_{48})$	−	1	−	−	−	−	−	−	−
			$\mathbf{Q}(\zeta_{60})$	−	−	−	2	−	−	−	−	−
	20	2(−)	$\mathbf{Q}(\zeta_{33})$	−	−	−	1	−	−	−	−	−
			$\mathbf{Q}(\zeta_{44})$	−	−	1	−	−	−	−	−	−

ℓ	Dim	Gesamtzahl(extremal)	K	Minimum								
				2	4	6	8	10	12	14	16	18
11	24	7(−)	$\mathbf{Q}(\zeta_{35})$	−	−	−	1	−	1	−	−	−
			$\mathbf{Q}(\zeta_{45})$	−	−	1	−	−	−	−	−	−
			$\mathbf{Q}(\zeta_{56})$	−	−	−	1	−	−	−	−	−
			$\mathbf{Q}(\zeta_{72})$	−	1	−	1	−	−	−	−	−
			$\mathbf{Q}(\zeta_{84})$	−	−	1	−	−	−	−	−	−
	32	42(−)	$\mathbf{Q}(\zeta_{80})$	−	−	1	1	1	1	−	−	−
			$\mathbf{Q}(\zeta_{96})$	−	1	−	−	1	−	−	−	−
			$\mathbf{Q}(\zeta_{120})$	−	−	1	13	18	4	−	−	−
	36	2(−)	$\mathbf{Q}(\zeta_{108})$	−	1	−	−	1	−	−	−	−
14	4	1(1)	$\mathbf{Q}(\zeta_8)$	−	1	−	−	−	−	−	−	−
	8	2(1)	$\mathbf{Q}(\zeta_{16})$	−	1	−	−	−	−	−	−	−
			$\mathbf{Q}(\zeta_{24})$	−	−	1	−	−	−	−	−	−
	12	1(1)	$\mathbf{Q}(\zeta_{28})$	−	−	−	1	−	−	−	−	−
	16	5(−)	$\mathbf{Q}(\zeta_{32})$	−	1	−	−	−	−	−	−	−
			$\mathbf{Q}(\zeta_{40})$	−	−	1	−	−	−	−	−	−
			$\mathbf{Q}(\zeta_{48})$	−	−	1	1	−	−	−	−	−
			$\mathbf{Q}(\zeta_{60})$	−	−	1	−	−	−	−	−	−
	24	8(−)	$\mathbf{Q}(\zeta_{56})$	−	−	−	4	−	2	−	−	−
			$\mathbf{Q}(\zeta_{72})$	−	−	1	1	−	−	−	−	−
	32	21(−)	$\mathbf{Q}(\zeta_{64})$	−	1	−	−	2	−	−	−	−
			$\mathbf{Q}(\zeta_{80})$	−	−	−	−	−	2	1	−	−
			$\mathbf{Q}(\zeta_{96})$	−	−	1	1	2	2	−	−	−
			$\mathbf{Q}(\zeta_{120})$	−	−	−	4	−	5	−	−	−
	36	36(−)	$\mathbf{Q}(\zeta_{57})$	−	−	−	−	3	25	8	−	−

ℓ	Dim	Gesamtzahl(extremal)	K	Minimum								
				2	4	6	8	10	12	14	16	18
15	8	1(1)	$\mathbf{Q}(\zeta_{24})$	—	—	1	—	—	—	—	—	—
	16	3(1)	$\mathbf{Q}(\zeta_{40})$	—	—	—	—	1	—	—	—	—
			$\mathbf{Q}(\zeta_{48})$	—	—	1	—	—	—	—	—	—
			$\mathbf{Q}(\zeta_{60})$	—	—	—	1	—	—	—	—	—
	24	5(—)	$\mathbf{Q}(\zeta_{56})$	—	—	—	1	—	—	—	—	—
			$\mathbf{Q}(\zeta_{72})$	—	—	1	—	—	1	—	—	—
			$\mathbf{Q}(\zeta_{84})$	—	—	—	—	—	2	—	—	—
	32	23(—)	$\mathbf{Q}(\zeta_{80})$	—	—	—	—	2	3	1	—	—
			$\mathbf{Q}(\zeta_{96})$	—	—	1	—	1	—	—	—	—
			$\mathbf{Q}(\zeta_{120})$	—	—	—	4	1	9	1	—	—
	36	4(—)	$\mathbf{Q}(\zeta_{76})$	—	—	—	—	2	1	—	1	—
23	4	1(1)	$\mathbf{Q}(\zeta_{12})$	—	—	1	—	—	—	—	—	—
	8	3(—)	$\mathbf{Q}(\zeta_{24})$	—	—	1	2	—	—	—	—	—
	12	1(—)	$\mathbf{Q}(\zeta_{36})$	—	—	1	—	—	—	—	—	—
	16	5(—)	$\mathbf{Q}(\zeta_{40})$	—	—	—	—	1	—	—	—	—
			$\mathbf{Q}(\zeta_{48})$	—	—	1	2	—	—	—	—	—
			$\mathbf{Q}(\zeta_{60})$	—	—	—	—	—	1	—	—	—
	22	2(—)	$\mathbf{Q}(\zeta_{23})$	—	—	—	—	—	2	—	—	—

ℓ	Dim	Gesamtzahl(extremal)	K	Minimum								
				2	4	6	8	10	12	14	16	18
23	24	14(−)	$\mathbb{Q}(\zeta_{39})$	−	−	−	−	1	1	−	1	−
			$\mathbb{Q}(\zeta_{52})$	−	−	−	−	1	2	−	−	−
			$\mathbb{Q}(\zeta_{56})$	−	−	−	−	−	−	−	1	−
			$\mathbb{Q}(\zeta_{72})$	−	−	1	2	1	2	−	−	−
			$\mathbb{Q}(\zeta_{84})$	−	−	−	−	−	1	−	−	−
	32	20(−)	$\mathbb{Q}(\zeta_{80})$	−	−	−	−	1	−	1	1	1
			$\mathbb{Q}(\zeta_{96})$	−	−	1	2	−	−	2	2	−
			$\mathbb{Q}(\zeta_{120})$	−	−	−	−	1	3	1	4	−
	36	2(−)	$\mathbb{Q}(\zeta_{108})$	−	−	1	−	−	−	1	−	−

Tabelle 6.1: Anzahlen der Ideal-Gitter der Stufen $\ell \in \{1, 2, 3, 5, 6, 7, 11, 14, 15, 23\}$ und Determinante $\ell^{\frac{n}{2}}$ mit Dimensionen ≤ 36 nach zugehörigem Kreisteilungskörper K und Minimum.

§ 6.2 Ergebnisse der Subideal-Gitter-Klassifikation

Es folgt eine Tabelle mit den Anzahlen der mit Algorithmus (9) gefundenen (ggf. bisher unbekannten) extremalen, stark modularen Gitter mit großem Automorphismus der Ordnung m für Werte ℓ und n , bei denen die Gesamtzahl extremaler Gitter bisher nicht bekannt ist. Dabei gibt "Vollst." an, ob mit Sicherheit alle Gitter mit Automorphismen der Ordnung m so konstruiert wurden und "Vollst. groß", ob mit Sicherheit alle Gitter mit großen Automorphismen der Ordnung m konstruiert wurden. Werte für m , sodass nach Anwendung der Methode sicher keine Automorphismen der Ordnung m existieren können, wurden aus Platzgründen weggelassen. Teilweise sind die Anzahlen mit "!!" markiert, dies bedeutet, dass in Algorithmus (9) ein Geschlecht für das Fixgitter nicht vollständig aufgezählt wurde und daher möglicherweise nicht alle Gitter mit den richtigen Voraussetzungen vom Algorithmus gefunden wurden.

Dim	ℓ	Anzahl(unbekannte)	m	Anzahl	Vollst.	Vollst. groß	σ -inv. Oberg.
12	14	1(0)	16	0	×	×	×
			20	0	×	✓	×
			24	1	×	×	×
			30	0	×	✓	×
			21	0	×	✓	×
			28	1	×	✓	×
			42	0	×	✓	×
	15	1(0)	15	0	×	×	×
			20	0	×	×	×
			24	1	×	×	×
			30	0	×	×	×
			36	0	×	✓	×
14	11	0(0)	11	0	×	×	×
16	6	1(0)	21	0	×	✓	×
			28	0	×	✓	×
			36	0	×	×	×
			42	0	×	✓	×
			40	1	×	✓	×
			48	0	×	✓	×
			60	0	×	✓	×
	7	3(1)	21	0	×	×	×
			28	0	×	×	×
			42	0	×	×	×
			40	1	✓	✓	×
			48	1	✓	✓	×
			60	1	✓	✓	×

Dim	ℓ	Anzahl(unbekannte)	m	Anzahl	Vollst.	Vollst. groß	σ -inv. Oberg.
16	14	0(0)	21	0	×	×	×
			28	0	×	×	×
			36	0	×	✓	×
			42	0	×	×	×
			40	1	×	✓	×
			48	1	×	✓	×
			60	1	×	✓	×
	15	1(0)	21	0	×	✓	×
			36	0	×	×	×
			42	0	×	✓	×
			40	1	×	✓	×
			48	1	×	✓	×
			60	1	×	✓	×
18	11	0(0)	11	0	×	×	×
			40	1	✓	✓	×
			48	1	✓	✓	×
20	3	3(0)	21	0	×	✓	×
			28	0	×	✓	×
			36	0	×	×	×
			42	0	×	✓	×
			40	0	×	✓	×
			48	0	×	✓	×
			60	3	×	✓	×
			27	0	×	×	×
			54	0	×	×	×

Dim	ℓ	Anzahl(unbekannte)	m	Anzahl	Vollst.	Vollst. groß	σ -inv. Oberg.
20	5	3(0)	36	0	×	✓	×
			40	0	×	×	×
			60	3	×	×	×
	6	1(0)	21	0	×	✓	×
			28	0	×	✓	×
			36	0	×	×	×
			42	0	×	✓	×
			32	0	×	×	×
			40	0	×	✓	×
			48	0	×	×	×
			60	0	×	✓	×
			27	0	×	×	×
			54	0	×	×	×
			33	1	×	✓	×
			44	0	×	✓	×
			66	1	×	✓	×
	7	0(0)	21	0	×	×	×
			28	0	×	×	×
			42	0	×	×	×
			48	0	×	✓	×
	14	0(0)	21	0	×	✓	×
			28	0	×	×	×
			36	0	×	✓	×
			42	0	×	✓	×
			32	0	×	×	×
			48	0	×	✓	×

Dim	ℓ	Anzahl(unbekannte)	m	Anzahl	Vollst.	Vollst. groß	σ -inv. Oberg.
20	15	0(0)	21	0	×	✓	×
			28	0	×	✓	×
			36	0	×	✓	×
			42	0	×	✓	×
			40	0	×	✓	×
			48	0	×	✓	×
			60	0	×	✓	×
			27	0	×	×	×
22	3	2(1)	13	1	✓	✓	×
			21	0	×	✓	×
			26	0	×	✓	×
			28	0	×	✓	×
			36	0	×	×	×
			42	0	×	✓	×
			40	0	×	✓	×
			48	0	×	✓	×
			60	0	×	✓	×
			27	0	×	×	×
			54	0	×	×	×
			33	1	✓	✓	×
			66	1	✓	✓	×
	7	0(0)	21	0!!	×	×	×
			28	0	×	×	×
			36	0	×	✓	×
			42	0!!	×	×	×
	11	0(0)	36	0	×	✓	×
			60	0	×	✓	×
			44	0	×	✓	×

Dim	ℓ	Anzahl(unbekannte)	m	Anzahl	Vollst.	Vollst. groß	σ -inv. Oberg.
24	2	8(2)	32	0	×	×	×
			40	7	×	×	×
			48	1	×	×	✓
			60	7	×	×	×
			54	0	×	✓	×
			44	0	×	✓	×
			66	0	×	✓	×
			35	0	×	✓	×
			45	0	×	✓	×
			56	1	×	✓	×
			70	0	×	✓	×
			72	1	×	✓	×
			84	1	×	✓	×
			90	0	×	✓	×
	3	1(0)	48	0	×	×	×
			60	0	×	×	×
			27	0	×	×	×
			54	0	×	×	×
			33	0	×	✓	×
			66	0	×	✓	×
			39	1	✓	✓	×
			52	1	✓	✓	×
			72	0	×	✓	×
			78	1	✓	✓	×

Dim	ℓ	Anzahl(unbekannte)	m	Anzahl	Vollst.	Vollst. groß	σ -inv. Oberg.
24	5	1(0)	40	1	×	×	✓
			60	1	×	×	×
			25	0	×	×	×
			50	0	×	×	×
			35	1	×	✓	×
			45	0	×	✓	×
			56	1	×	✓	×
			70	1	×	✓	×
			72	0	×	✓	×
			84	0	×	✓	×
			90	0	×	✓	×

§ 6.3 MAGMA-Implementierungen von Hilfsfunktionen

Es folgt der Quellcode zu folgenden Hilfsfunktionen in der Reihenfolge des Quellcodes:

- Das komplex-konjugierte eines \mathbb{Z}_K -Ideals berechnen.
- Eine Liste von Gittern nach Isometrie reduzieren.
- Das Minimum ausgeben, was ein ℓ -modulares Gitter der Dimension n mindestens haben muss.
- Das Geschlechtssymbol eines positiv definiten Gitters quadratfreier Stufe bestimmen.
- Ein Gitter des MAGMA-Typs `LatNF` zum Typen `Lat` konvertieren.

- Testen ob ein Gitter L ℓ -modular ist.
- Testen ob ein Gitter L stark ℓ -modular ist.

```

1  load "hu.m"; // Program by David Lorch for constructing lattices
   with given elementary divisors
2
3  HermiteBounds := [1, 1.1547, 1.2599, 1.1412, 1.5157, 1.6654,
   1.8115, 2, 2.1327, 2.2637, 2.3934, 2.5218, 2.6494, 2.7759,
   2.9015, 3.0264, 3.1507, 3.2744, 3.3975, 3.5201, 3.6423, 3.7641,
   3.8855, 4.0067, 4.1275, 4.2481, 4.3685, 4.4887, 4.6087, 4.7286,
   4.8484, 4.9681, 5.0877, 5.2072, 5.3267, 5.4462];
4
5
6  function IdealConjugate(I, K)
7  // Input: Z_K-Ideal I; Field K
8
9  // Output: Z_K-Ideal which is the complex conjugate of I
10
11  gens := [];
12  for g in Generators(I) do
13    Append(~gens, ComplexConjugate(K ! g));
14  end for;
15
16  return ideal<Integers(K)|gens>;
17
18 end function;
19
20
21 function ReduceByIsometry(Lattices)
22 // Input: List of lattices
23
24 // Output: Reduced list for which the elements are pairwise non-
   isometric
25
26 LatticesReduced := [];
27 Minima := [* *];
28 NumShortest := AssociativeArray();
29 SizeAuto := AssociativeArray();
30
31
32 for i in [1..#Lattices] do
33   L := Lattices[i];
34
35   min_computed := false;
36   minimum := 0;
37
38   shortest_computed := false;
39   shortest := 0;
40
41   auto_computed := false;
42   auto := 0;
43
44   for j in [1..#LatticesReduced] do
45     M := LatticesReduced[j];
46
47     if not min_computed then
48       min_computed := true;
49       minimum := Min(L);

```

```

50     end if;
51
52     if not IsDefined(Minima, j) then
53         Minima[j] := Min(M);
54     end if;
55
56     if minimum ne Minima[j] then
57         continue;
58     end if;
59
60
61     if not shortest_computed then
62         shortest_computed := true;
63         shortest := #ShortestVectors(L);
64     end if;
65
66     if not IsDefined(NumShortest, j) then
67         NumShortest[j] := #ShortestVectors(M);
68     end if;
69
70     if shortest ne NumShortest[j] then
71         continue;
72     end if;
73
74
75     if not auto_computed then
76         auto_computed := true;
77         auto := #AutomorphismGroup(L);
78     end if;
79
80     if not IsDefined(SizeAuto, j) then
81         SizeAuto[j] := #AutomorphismGroup(M);
82     end if;
83
84     if auto ne SizeAuto[j] then
85         continue;
86     end if;
87
88
89     if IsIsometric(L, M) then
90         continue i;
91     end if;
92 end for;
93
94 Append(~LatticesReduced, Lattices[i]);
95
96 NewIndex := #LatticesReduced;
97 if min_computed then
98     Minima[NewIndex] := minimum;
99 end if;
100
101 if shortest_computed then
102     NumShortest[NewIndex] := shortest;
103 end if;
104
105 if auto_computed then

```

```

106     SizeAuto[NewIndex] := auto;
107     end if;
108
109     end for;
110
111     return LatticesReduced;
112 end function;
113
114
115 function ExtremalMinimum(l, n)
116 // Input: Square-free l in N; n in N
117
118 // Output: Minimum that a l-modular lattice of dimension n must
119 // have at least
120
121 if l eq 1 then k := 24;
122 elif l eq 2 then k := 16;
123 elif l eq 3 then k := 12;
124 elif l eq 5 then k := 8;
125 elif l eq 6 then k := 8;
126 elif l eq 7 then k := 6;
127 elif l eq 11 then k := 4;
128 elif l eq 14 then k := 4;
129 elif l eq 15 then k := 4;
130 elif l eq 23 then k := 2;
131 end if;
132
133 return 2 + 2*Floor(n/k);
134 end function;
135
136 function GenSymbol(L)
137 // Input: Positive definite Numberfield Lattice L of square-free
138 // level
139
140 // Output: Genus symbol of L in the form [S_1, n, <2, n_2,
141 // epsilon_2, S_2, t_2>, <3, n_3, epsilon_3>, <5,...>, ...] for all
142 // primes dividing Det(L)
143 Symbol := [* *];
144
145 Rat := RationalAsNumberField();
146 Int := Integers(Rat);
147
148 LNF := NumberFieldLatticeWithGram(Matrix(Rat, GramMatrix(L)));
149 _, Grams2, Vals2 := JordanDecomposition(LNF, ideal<Int|2>);
150
151 // Checks if all diagonal entries of the 1-component of the 2-
152 // adic jordan decomposition are even
153 if Vals2[1] ne 0 or (Vals2[1] eq 0 and &and([Valuation(Rationals
154 () ! (Grams2[1][i][i]), 2) ge 1 : i in [1..NumberOfRows(Grams2
155 [1])])) then
156     Append(~Symbol, 2);
157 else
158     Append(~Symbol, 1);
159 end if;

```

```

154
155 Append(~Symbol, Dimension(L));
156
157
158 for p in PrimeDivisors(Integers() ! (Determinant(L))) do
159   _, Gramsp, Valsp := JordanDecomposition(LNF, ideal<Int|p>);
160
161   if Valsp[1] eq 0 then
162     G := Matrix(Rationals(), 1/p * Gramsp[2]);
163   else
164     G := Matrix(Rationals(), 1/p * Gramsp[1]);
165   end if;
166
167   sym := <p, NumberOfRows(G)>;
168
169   det := Determinant(G);
170   det := Integers() ! (det * Denominator(det)^2);
171
172   if p eq 2 then
173     if IsDivisibleBy(det+1, 8) or IsDivisibleBy(det-1, 8) then
174       Append(~sym, 1);
175     else
176       Append(~sym, -1);
177     end if;
178
179     if &and([Valuation(Rationals() ! (G[i][i]), 2) ge 1 : i in
180 [1..sym[2]]) then
181       Append(~sym, 2);
182     else
183       Append(~sym, 1);
184     end if;
185
186     if sym[4] eq 2 then
187       Append(~sym, 0);
188     else
189       Append(~sym, Integers() ! (Trace(G)* Denominator(Trace
190 (G))^2) mod 8);
191     end if;
192   else
193     Append(~sym, LegendreSymbol(det, p));
194   end if;
195
196   Append(~Symbol, sym);
197 end for;
198
199 return Symbol;
200
201
202 function ToZLattice(L)
203 // Input: Numberfield lattice L
204
205 // Output: L as Z-lattice
206 B:= Matrix(ZBasis(L`Module));

```



```

207   G:= B * L`Form * InternalConjugate(L, B);
208   Form:= Matrix( Ncols(G), [ AbsoluteTrace(e) : e in Eltseq(G) ]
209 );
209   Form:=IntegralMatrix(Form);
210
211   LZ := LatticeWithGram(LLGram(Matrix(Integers(),Form)));
212
213   return LZ;
214 end function;
215
216
217 function IsModular(L, l)
218 // Input: Lattice L; l in N
219
220 // Output: true iff L is a l-modular lattice
221
222 return IsIsometric(L, LatticeWithGram(l*GramMatrix(Dual
(L:Rescale:=false))));
223
224 end function;
225
226 function IsStronglyModular(L,l)
227 // Input: Lattice L; l in N
228
229 // Output: true iff L is a strongly l-modular lattice
230
231 return &and[IsIsometric(L, LatticeWithGram(m*GramMatrix
(PartialDual(L, m : Rescale:=false))) : m in [m : m in Divisors
(l) | Gcd(m, Integers() ! (l/m)) eq 1]];
232
233 end function;

```

§ 6.4 MAGMA-Implementierungen der Ideal-Gitter-Algorithmen

Es folgt der Quellcode zu den Algorithmen aus Kapitel 3. Die implementierten Funktionen sind in der Reihenfolge des Quellcodes:

- Alle Teiler eines \mathbb{Z}_K -Ideals \mathcal{I} von festgelegter Norm berechnen. Siehe Algorithmus (1).
- Einen total-reellen Erzeuger eines \mathbb{Z}_K -Ideals \mathcal{I} bestimmen. Siehe Algorithmus (2).
- Matrix, deren Einträge die Vorzeichen der reellen Einbettung der Grundeinheiten kodieren sowie eine Liste aller total-positiven Elemente in \mathbb{Z}_{K+}^* reduziert nach $\{\lambda\bar{\lambda} \mid \lambda \in \mathbb{Z}_K^*\}$ und eine Liste von Erzeugern einer Untergruppe von \mathbb{Z}_{K+}^* mit ungeradem Index bestimmen. Siehe Algorithmus (3)
- Das Minimalpolynom der Operation eines Automorphismus σ von einem Gitter L auf dem \mathbb{F}_p -Vektorraum $L^{\#p}/L$ berechnen
- Liste aller total-positiven Erzeuger eines \mathbb{Z}_K -Ideals \mathcal{I} reduziert nach $\{\lambda\bar{\lambda} \mid \lambda \in \mathbb{Z}_K^*\}$ bestimmen. Siehe ebenfalls Algorithmus (3).
- Liste von Vertretern der Klassengruppe von K modulo der Operation der Galoisgruppe $\text{Gal}(K/\mathbb{Q})$ bestimmen. Siehe Abschnitt (3.3).
- Aus einem \mathbb{Z}_K -Ideal \mathcal{I} und einem total-positiven Element α das Gitter, welches durch Auffassung von \mathcal{I} als \mathbb{Z} -Gitter mit Bilinearform $b(x, y) := \text{Spur}(\alpha x \bar{y})$ entsteht.
- Alle Ideal-Gittern über gegebenem CM-Körper mit vorgegebener Determinante und quadratfreier Stufe aufzählen. Siehe Algorithmus (4).

- Liste von allen ℓ -modularen Gittern in Dimension n erstellen, welche einen Automorphismus σ besitzen mit $\mu_\sigma = \Phi_m$ und $\varphi(m) = n$. Siehe Abschnitt (3.5).
- Schleife, welche die letzte Funktion für verschiedene n und ℓ auswertet und die Ergebnisse speichert.

```

1  load "Utility.m";
2
3  function DivisorsWithNorm(I, n)
4  // Input: Z_K-Ideal I; norm n in Z
5
6  // Output: List of divisors of I with norm n
7
8      norm := Integers() ! Norm(I);
9
10     if n eq 1 then return [I*I^(-1)]; end if;
11     if not IsDivisibleBy(norm, n) then return []; end if;
12     if norm eq n then return [I]; end if;
13
14     Fact := Factorization(I);
15
16     p1 := Fact[1][1];
17     s1 := Fact[1][2];
18     np := Integers() ! Norm(p1);
19
20     Results := [];
21
22     for j in [0..s1] do
23         if IsDivisibleBy(n, np^j) then
24             B := DivisorsWithNorm(I*p1^(-s1), Integers() ! (n / np^j));
25
26             for J in B do
27                 Append(~Results, p1^j*J);
28             end for;
29         end if;
30     end for;
31
32     return Results;
33
34 end function;
35
36
37 function TotallyRealGenerator(I, K, Kpos)
38 // Input: Z_K-Ideal I; Field K; Field Kpos
39
40 // Output: Boolean that indicates success; totally real generator
41 // of I cap Kpos
42
43     ZK := Integers(K);
44     ZKpos := Integers(Kpos);
45
46     Ipos:=ideal<ZKpos|1>;
47     Split:=[];
48
49     Fact := Factorization(I);
50
51     for i in [1..#Fact] do
52         if i in Split then continue; end if;
53
54         pi:=Fact[i][1];
55         si:=Fact[i][2];

```

```

55     piConj := IdealConjugate(pi,K);
56
57     p:=MinimalInteger(pi);
58
59     pFact:=Factorization(ideal< ZKpos | p >);
60
61     for qj in [fact[1] : fact in pFact] do
62         if ideal<ZK | Generators(qj)> subset pi then
63             a := qj;
64             break;
65         end if;
66     end for;
67
68     aZK := ideal<ZK|Generators(a)>;
69
70     if aZK eq pi^2 then
71
72         if not IsDivisibleBy(si, 2) then return false, _; end if;
73         Ipos *:= a^(Integers() ! (si/2));
74
75     elif aZK eq pi then
76
77         Ipos *:= a^si;
78
79     elif aZK eq pi*piConj then
80
81         if Valuation(I, pi) ne Valuation(I, piConj) then return
false, _; end if;
82         Ipos *:= a^si;
83         for j in [1..#Fact] do
84             pj := Fact[j][1];
85             if pj eq piConj then
86                 Append(~Split, j);
87                 break;
88             end if;
89         end for;
90     end if;
91 end for;
92
93 return IsPrincipal(Ipos);
94
95 end function;
96
97
98 function EmbeddingMatrix(K, Kpos)
99 // Input: Field K; Field Kpos
100
101 // Output: Matrix M whose entries give the signs of the
embeddings of the fundamental units; List U of all totally
positive units in ZKpos modulo norms; List of generators of a
subgroup of Z_Kpos^* of odd index
102
103 ZKpos := Integers(Kpos);
104
105 t := #Basis(ZKpos);
106

```

```

107 G, mG := pFundamentalUnits(ZKpos, 2);
108 FundUnits := [mG(G.i) : i in [1..t]];
109
110 M := ZeroMatrix(GF(2), t, t);
111
112 for i in [1..t] do
113   Embeds := RealEmbeddings(FundUnits[i]);
114   for j in [1..t] do
115     if Embeds[j] < 0 then
116       M[i][j] := 1;
117     end if;
118   end for;
119 end for;
120
121 U := [];
122 for a in Kernel(M) do
123   e := ZKpos ! &*[FundUnits[i]^(Integers() ! a[i]) : i in
124 [1..t]];
125   Append(~U, e);
126 end for;
127
128 ZRel := Integers(RelativeField(Kpos, K));
129
130 Units := [];
131 for u in U do
132   for w in Units do
133     if NormEquation(ZRel, ZRel ! (u/w)) then
134       continue u;
135     end if;
136   end for;
137   Append(~Units, u);
138 end for;
139
140 return M, Units, FundUnits;
141
142 end function;
143
144
145 function TotallyPositiveGenerators(alpha, K, Kpos, M, U,
FundUnits)
146 // Input: alpha in ZKpos; Field K; Field Kpos; Embedding-Matrix
M; List U of all totally-positive units in ZKpos modulo norms;
List FundUnits of generators of a subgroup of  $Z_{Kpos}^*$  of odd
index
147
148 // Output: Boolean that indicates success; List of all totally-
positive generators of  $\alpha \cdot ZK$  modulo norms
149
150 t := #Basis(Kpos);
151 V := ZeroMatrix(GF(2), 1, t);
152
153 Embeds := RealEmbeddings(alpha);
154 for i in [1..t] do
155   if Embeds[i] < 0 then
156     V[1][i] := 1;

```

```

157     end if;
158 end for;
159
160 solvable, x := IsConsistent(M,V);
161 if not solvable then
162     return false, _;
163 end if;
164
165 g := Integers(Kpos) ! &*[FundUnits[i]^(Integers() ! x[1][i]) :
i in [1..t]];
166
167 return true, [alpha*g*u : u in U];
168
169 end function;
170
171
172 function ClassesModGalois(K)
173 // Input : Field K
174
175 // Output : List of representatives of the class group of  $\mathbb{Z}_K$ 
modulo the action of the Galois-group of  $K/\mathbb{Q}$ 
176
177 ZK := Integers(K);
178 Cl, mCl := ClassGroup(ZK : Proof:="GRH");
179
180 ClModGal:=[];
181 for a in Cl do
182     A:=mCl(a);
183     for f in Automorphisms(K) do
184         if Inverse(mCl)(ideal<ZK | [f(x) : x in Generators(A)]>) in
ClModGal then
185             continue a;
186         end if;
187     end for;
188     Append(~ClModGal,a);
189 end for;
190
191 return [mCl(g) : g in ClModGal];
192
193 end function;
194
195
196
197 function LatFromIdeal(J, alpha, K)
198 // Input: ZK-Ideal J; Totally positive element alpha Kpos; Field K
199
200 // Output: Z-Lattice with elements J and inner product (x,y) := Tr
(alpha*x*Conj(y))
201
202 n := #Basis(K);
203 z := PrimitiveElement(K);
204
205 GeneratorMatrix := KMatrixSpace(Rationals(), #Generators(J)*n,
n) ! 0;
206

```

```

207   for i in [1..#Generators(J)] do
208     g := K ! (Generators(J)[i]);
209
210     for j in [1..n] do
211       GeneratorMatrix[(i-1)*n + j] := Vector(Rationals(), n,
212       Eltseq(g*z^(j-1)));
213     end for;
214   end for;
215
216   BaseVecs := Basis(Lattice(GeneratorMatrix));
217
218   ZBase := [];
219   for i in [1..n] do
220     b := K ! 0;
221     for j in [1..n] do
222       b += BaseVecs[i][j]*z^(j-1);
223     end for;
224     Append(~ZBase, b);
225   end for;
226
227   InnProd := KMatrixSpace(Rationals(), n, n) ! 0;
228   for i in [1..n] do
229     for j in [1..n] do
230       InnProd[i][j] := Trace(K ! (alpha * z^(i-j)));
231     end for;
232   end for;
233
234   L := LatticeWithBasis(KMatrixSpace(Rationals(), n, n) ! Matrix
235   (BaseVecs), InnProd);
236   L := LatticeWithGram(LLGram(GramMatrix(L)));
237   return L;
238
239 end function;
240
241
242 function IdeallLattices(d, K, Kpos, A, M, U, FundUnits, Reduce)
243 // Input: d in N; Field K; Field Kpos; Class Group of K mod
244 // Galois-Group A; Embedding-Matrix M; List of totally-positive
245 // units U; List FundUnits of generators of a subgroup of  $\mathbb{Z}_{Kpos}^*$ 
246 // of odd index; Boolean Reduce that indicates, whether the list
247 // shall be reduced by isometry.
248
249 // Output: List of all even ideal-lattices over K of square-free
250 // level and determinant d
251
252   ZK := Integers(K);
253   InvDiff := Different(ZK)^(-1);
254
255   l := &*(PrimeDivisors(d));
256
257   B := DivisorsWithNorm(ideal<ZK|l>, d);
258
259   Results := [];

```



```

256   for I in A do
257       for b in B do
258           J := (I*IdealConjugate(I,K))(-1)*InvDiff*b;
259
260           x, alphaPrime := TotallyRealGenerator(J, K, Kpos);
261
262           if x then
263               y, TotPos := TotallyPositiveGenerators(alphaPrime, K,
Kpos, M, U, FundUnits);
264               if y then
265                   for alpha in TotPos do
266                       L := LatFromIdeal(I, alpha, K);
267                       if IsEven(L) then
268                           Append(~Results, L);
269                       end if;
270                   end for;
271               end if;
272           end if;
273       end for;
274   end for;
275
276   if Reduce then Results := ReduceByIsometry(Results); end if;
277
278   return Results;
279
280 end function;
281
282
283 function ModIdLat(l, n)
284 // Input: square-free l in N; n in N
285
286 // Output: List of all l-modular lattices of dimension n that are
ideal lattices over some cyclotomic field reduced by isometry
287
288 det := l^(Integers() ! (n/2));
289
290 Lattices := [];
291
292 for m in [m : m in EulerPhiInverse(n) | m mod 4 ne 2] do
293     K<z> := CyclotomicField(m);
294     Kpos := sub<K | z + z(-1)>;
295
296     A := ClassesModGalois(K);
297     M, U, FundUnits := EmbeddingMatrix(K, Kpos);
298     Lattices cat:= IdealLattices(det, K, Kpos, A, M, U,
FundUnits, false);
299 end for;
300
301 Lattices := ReduceByIsometry(Lattices);
302
303 PrintFileMagma(Sprintf("IdealLattices/%o-Modular/%o-
Dimensional", l, n), Lattices : Overwrite := true);
304
305 return Lattices;
306

```

```

307 end function;
308
309 procedure MainLoop()
310     for n := 2 to 36 by 2 do
311         for l in [1,2,3,5,6,7,11,14,15,23] do
312             printf "dim = %o, l = %o\n", n, l;
313             Results := ModIdLat(l, n);
314             ModList := [L : L in Results | IsModular(L, l)];
315             StrongModList := [L : L in Results | IsStronglyModular
(L,l)];
316             PrintFileMagma(sprintf("SubidealLattices/%o-Modular/%o-
Dimensional", l, n), Results : Overwrite := true);
317             PrintFileMagma(sprintf("SubidealLattices/%o-Modular/%o-
DimensionalModular", l, n), ModList : Overwrite := true);
318             PrintFileMagma(sprintf("SubidealLattices/%o-Modular/%o-
DimensionalStronglyModular", l, n), StrongModList : Overwrite :=
true);
319
320             if #Results gt 0 then
321                 printf "\n\n-----n = %o, l = %o: %o lattices found! %
o of them are modular and %o are strongly modular-----\n\n",
n, l, #Results, #ModList, #StrongModList;
322             end if;
323         end for;
324     end for;
325 end procedure;

```

§ 6.5 MAGMA-Implementierungen der Subideal-Gitter-Algorithmen

Es folgt der Quellcode zu den Algorithmen aus Kapitel 4 bis einschließlich Abschnitt (4.6). Die implementierten Funktionen sind in der Reihenfolge des Quellcodes:

- Mithilfe von Hermite-Schranken die möglichen Typen von Automorphismen von Primzahlordnung gerader Gitter mit quadratfreier Stufe ℓ , Determinante ℓ^k und Minimum $\geq t$ aufzählen. Siehe Algorithmus (5).
- Mit dem Kneser'schen Nachbarverfahren die Vertreter der Isometrieklassen aller Gitter in dem Geschlecht eines Vertreters bestimmen. Siehe Algorithmus (6).
- Anhand von vorgegebener Dimension und Determinante Vertreter aller Isometrieklassen von (wenn gewünscht geraden) Gittern mit dieser Dimension und Determinante und mit quadratfreier Stufe aufzählen.
- Zu einem vorgegebenen Geschlechtssymbol Vertreter aller Isometrieklassen von Gittern mit diesem Geschlechtssymbol und quadratfreier Stufe aufzählen.
- Mit der MAGMA-Methode `Sublattices` σ -invariante Obergitter von Index p^s für einen Gitterautomorphismus σ bestimmen.
- Mit der in Abschnitt (4.5) beschriebenen Methode Obergitter von $L_1 \perp L_p$ invariant unter $\text{diag}(\sigma_1, \sigma_p)$ bestimmen. Siehe Algorithmus (8).
- Mit der Methode von Michael Jürgens alle Obergitter von L mit Index p^s bestimmen.
- Minimalpolynom der Operation eines Automorphismus σ von einem Gitter L auf dem \mathbb{F}_p -Vektorraum $L^{\#p}/L$ bestimmen. Siehe Abschnitt (4.6).

- Liste aller extremalen ℓ -modularen Gitter in Dimension n bestimmen, die einen großen Automorphismus der Ordnung m haben, sodass m einen Primteiler p mit $\text{ggT}(p, \ell) = 1$ und $\frac{\mu_\sigma}{\Phi_m} | (X^{\frac{m}{p}} - 1)$ besitzt. Siehe Algorithmus (9).
- Schleife, welche die letzte Funktion für verschiedene n und ℓ auswertet und die Ergebnisse speichert.

```

1  load "Utility.m";
2  load "IdeallLattices.m";
3
4
5  function AutomorphismTypes(l, k, n, t)
6  // Input: Square-free l in N, k in N, n in N, t in N
7
8  // Output: List of all possible types of automorphisms of prime
           order for even lattices of level l with determinant l^k,
           dimension n and minimum greater or equal to t
9      Results := [];
10
11      lFactors := PrimeDivisors(l);
12
13      for p in PrimesUpTo(n+1) do
14          if p in lFactors then continue; end if;
15
16          K<z> := CyclotomicField(p);
17          Kpos := sub<K|z+1/z>;
18
19          f := [];
20
21          for q in lFactors do
22              if p le 3 then
23                  Append(~f, 1);
24              else
25                  Append(~f, InertiaDegree(Factorization(ideal<Integers
(Kpos) | q>)[1][1]));
26              end if;
27          end for;
28
29          for np in [i*(p-1) : i in [1..Floor(n/(p-1))]] do
30
31              n1 := n - np;
32              for s in [0..Min(n1, Integers() ! (np/(p-1)))] do
33                  if not IsDivisibleBy(s - Integers() ! (np / (p-1)), 2)
then continue s; end if;
34                  if p eq 2 and not IsDivisibleBy(s, 2) then continue s;
end if;
35
36                  if l eq 1 then
37                      if n1 gt 0 then
38                          Gamma1 := t/p^(s/n1);
39                          if Gamma1 gt HermiteBounds[n1] + 0.1 then continue;
end if;
40                      end if;
41
42                      if np gt 0 then
43                          Gammap := t/p^(s/np);
44                          if Gammap gt HermiteBounds[np] + 0.1 then continue;
end if;
45                      end if;
46                      type := <p, n1, np, s>;
47
48                      Append(~Results, type);
49                  else

```

```

50         for kp in CartesianProduct([[2*f[i]*j : j in [0..Floor
(Min(np,k)/(2*f[i]))]] : i in [1..#f]]) do
51
52             k1 := [k - kp[i] : i in [1..#kp]];
53
54             for i in [1..#kp] do
55                 if k1[i] gt Min(n1,k) then continue kp; end if;
56                 if not IsDivisibleBy(k1[i] - k, 2) then continue
kp; end if;
57                 if not IsDivisibleBy(kp[i], 2) then continue kp;
end if;
58             end for;
59
60             if n1 gt 0 then
61                 Gamma1 := p^s;
62                 for i in [1..#lFactors] do
63                     Gamma1 *= lFactors[i]^k1[i];
64                 end for;
65                 Gamma1 := t / Gamma1^(1/n1);
66
67                 if Gamma1 gt HermiteBounds[n1] + 0.1 then
continue; end if;
68             end if;
69
70             if np gt 0 then
71                 Gammap := p^s;
72                 for i in [1..#lFactors] do
73                     Gammap *= lFactors[i]^kp[i];
74                 end for;
75                 Gammap := t / Gammap^(1/np);
76
77                 if Gammap gt HermiteBounds[np] + 0.1 then
continue; end if;
78             end if;
79
80             if p eq 2 then
81                 if n1 gt 0 then
82                     Gamma1 := 1;
83                     for i in [1..#lFactors] do
84                         Gamma1 *= lFactors[i]^k1[i];
85                     end for;
86                     Gamma1 := t/2 / Gamma1^(1/n1);
87
88                     if Gamma1 gt HermiteBounds[n1] + 0.1 then
continue; end if;
89                 end if;
90
91                 if np gt 0 then
92                     Gammap := 1;
93                     for i in [1..#lFactors] do
94                         Gammap *= lFactors[i]^kp[i];
95                     end for;
96                     Gammap := t/2 / Gammap^(1/np);
97
98                     if Gammap gt HermiteBounds[np] + 0.1 then
continue; end if;

```

```

99         end if;
100     end if;
101
102     type := <p, n1, np, s>;
103     for i in [1..#lFactors] do
104         Append(~type, lFactors[i]);
105         Append(~type, k1[i]);
106         Append(~type, kp[i]);
107     end for;
108
109     Append(~Results, type);
110 end for;
111 end if;
112 end for;
113 end for;
114 end for;
115
116 return Results;
117
118 end function;
119
120
121 function EnumerateGenusOfRepresentative(L)
122 // Input: Lattice L, t in N
123
124 // Output: List of all representatives of isometry-classes in the
125 // genus of L
126
127 "Enumerate genus of representative";
128 try return eval Read(Sprintf("GenusSymbols/Gen_%o", GenSymbol
129 (L))); catch e; end try;
130
131 try
132     Gen := eval Read(Sprintf("GenusSymbols/Gen_%o_partial",
133 GenSymbol(L)));
134     printf "Only using partial genus for %o!\n", GenSymbol(L);
135     return Gen;
136 catch e; end try;
137
138 if Dimension(L) le 6 then
139     Gen := GenusRepresentatives(L);
140     ZGen := [];
141     for M in Gen do
142         if Type(M) eq Lat then
143             Append(~ZGen, LLL(M));
144         else
145             Append(~ZGen, LatticeWithGram(LLLGram(Matrix(Rationals()),
146 GramMatrix(SimpleLattice(M)))));
147         end if;
148     end for;
149     PrintFileMagma(Sprintf("GenusSymbols/Gen_%o", GenSymbol(L)),
150 ZGen : Overwrite := true);
151     return ZGen;
152 end if;
153
154 M := Mass(L);
155 Gen := [L];

```

```

150     Explored := [false];
151     NumFound := [1];
152     Minima := [Minimum(L)];
153     NumShortest := [#ShortestVectors(L)];
154     SizeAuto := [#AutomorphismGroup(L)];
155     m := 1 / SizeAuto[1];
156
157     p := 2;
158
159     t0 := Realtime();
160
161     while m lt M do
162         //printf "So far %o classes found. Difference to actual mass
is %o. \n", #Gen, M-m;
163         if Realtime(t0) ge 120*60 then
164             printf "2 hours have elapsed and not the whole genus was
explored. Remaining difference to actual mass is %o. %o classes
were found so far. The symbol is %o.\n", M-m, #Gen, GenSymbol(L);
165             PrintFileMagma(Sprintf("GenusSymbols/Gen_%o_partial",
GenSymbol(L)), Gen : Overwrite := true);
166             return Gen;
167         end if;
168
169         RareFound := [];
170         MinCount := Infinity();
171
172         if &and(Explored) then
173             "All explored. Going to next prime.";
174             Explored := [false : x in Explored];
175             p := NextPrime(p);
176             if p ge 5 and Dimension(L) ge 8 then
177                 printf "Prime too large, cannot continue constructing
neighbours. Remaining difference to actual mass is %o. %o classes
were found so far. The symbol is %o.\n", M-m, #Gen, GenSymbol(L);
178                 PrintFileMagma(Sprintf("GenusSymbols/Gen_%o_partial",
GenSymbol(L)), Gen : Overwrite := true);
179                 return Gen;
180             end if;
181             if p ge 3 and Dimension(L) ge 12 then
182                 printf "Prime too large, cannot continue constructing
neighbours. Remaining difference to actual mass is %o. %o classes
were found so far. The symbol is %o.\n", M-m, #Gen, GenSymbol(L);
183                 PrintFileMagma(Sprintf("GenusSymbols/Gen_%o_partial",
GenSymbol(L)), Gen : Overwrite := true);
184                 return Gen;
185             end if;
186         end if;
187
188         for i in [1..#Gen] do
189             if not Explored[i] then
190                 if NumFound[i] lt MinCount then
191                     RareFound := [i];
192                     MinCount := NumFound[i];

```



```

193         elif NumFound[i] eq MinCount then
194             Append(~RareFound, i);
195         end if;
196     end if;
197 end for;
198
199 i := RareFound[Random(1, #RareFound)];
200
201 Neigh := [CoordinateLattice(N) : N in Neighbours(Gen[i], p)];
202 Explored[i] := true;
203
204 for N in Neigh do
205
206     auto := #AutomorphismGroup(N);
207     if auto lt 1/(M-m) then continue; end if;
208
209     minimum := Minimum(N);
210     shortest := #ShortestVectors(N);
211
212     for j in [1..#Gen] do
213
214         if minimum ne Minima[j] then
215             continue j;
216         end if;
217
218         if shortest ne NumShortest[j] then
219             continue j;
220         end if;
221
222         if auto ne SizeAuto[j] then
223             continue j;
224         end if;
225
226         if IsIsometric(N, Gen[j]) then
227             NumFound[j] += 1;
228             continue N;
229         end if;
230     end for;
231
232     Append(~Gen, N);
233     Append(~Explored, false);
234     Append(~NumFound, 1);
235     Append(~Minima, minimum);
236     Append(~NumShortest, shortest);
237     Append(~SizeAuto, auto);
238     m += 1/auto;
239     if m eq M then
240         break N;
241     end if;
242 end for;
243 end while;
244
245 PrintFileMagma(Sprintf("GenusSymbols/Gen_%o", GenSymbol(L)),
Gen : Overwrite := true);
246

```

```

247     return Gen;
248
249 end function;
250
251
252 function EnumerateGenusDeterminant(det, n, even)
253 // Input: det in N; n in N; boolean even that indicates whether
254 //         only even lattices shall be enumerated
255 // Output: Representatives of all isometry-classes belonging to a
256 //         genus of integral lattices with determinant det, dimension n, and
257 //         square-free level
258
259 if n eq 0 then
260     return [LatticeWithGram(Matrix(Rationals(), 0, 0, []))];
261 end if;
262
263 if n eq 1 then
264     L := LatticeWithGram(Matrix(Rationals(), 1, 1, [det]));
265     Symbol := GenSymbol(L);
266     if even and not Symbol[1] eq 2 then return []; end if;
267     if not IsSquarefree(Level(L)) then return []; end if;
268     if even and IsDivisibleBy(Determinant(L), 2) then
269         if not Symbol[3][4] eq 2 then return []; end if;
270     end if;
271     return [L];
272 end if;
273
274 if n eq 2 then
275     Results := [];
276
277     for m in [1..Floor(1.155*Sqrt(det))] do
278         for a in [-m+1..m-1] do
279
280             if not IsDivisibleBy(det + a^2, m) then continue; end if;
281             b := Integers() ! ((det + a^2) / m);
282
283             if b lt m then continue; end if;
284             if even and not IsEven(b) then continue; end if;
285
286             Mat := Matrix(Rationals(), 2, 2, [m,a,a,b]);
287             if not IsPositiveDefinite(Mat) then continue; end if;
288
289             L := LatticeWithGram(Mat);
290
291             if not IsSquarefree(Level(L)) then continue; end if;
292
293             Symbol := GenSymbol(L);
294             if even and not Symbol[1] eq 2 then continue; end if;
295             if even and IsDivisibleBy(Determinant(L), 2) then
296                 if not Symbol[3][4] eq 2 then continue; end if;
297             end if;
298
299             Append(~Results, L);
300         end for;
301     end for;

```

```

299     end for;
300
301     return ReduceByIsometry(Results);
302 end if;
303
304
305 Rat := RationalAsNumberField();
306 Int := Integers(Rat);
307
308 primes := PrimeBasis(det);
309 exps := [Valuation(det, p) : p in primes];
310
311 Ideallist := [];
312 if not 2 in primes then
313     Append(~Ideallist, <ideal<Int|2>, [[0,n]]>);
314 end if;
315
316 for i in [1..#primes] do
317     p := primes[i];
318     e := Abs(exps[i]);
319     if n eq e then
320         Append(~Ideallist, <ideal<Int|p>, [[1,e]]>);
321     elif e eq 0 then
322         Append(~Ideallist, <ideal<Int|p>, [[0,n]]>);
323     else
324         Append(~Ideallist, <ideal<Int|p>, [[0,n-e],[1,e]]>);
325     end if;
326 end for;
327
328 "Constructing representatives";
329 try
330     Rep := LatticesWithGivenElementaryDivisors(Rat, n, Ideallist);
331 catch e
332     print "Error while trying to construct a representative.
Ideallist:";
333     Ideallist;
334     return [];
335 end try;
336
337 Results := [];
338
339 for L in Rep do
340
341     LZ := ToZLattice(L);
342     if IsSquarefree(Level(LZ)) then
343         Symbol := GenSymbol(LZ);
344         if even and not Symbol[1] eq 2 then continue L; end if;
345         if even and IsDivisibleBy(det, 2) then
346             if not Symbol[3][4] eq 2 then continue L; end if;
347         end if;
348
349         Gen := EnumerateGenusOfRepresentative(LZ);
350         Results cat:= Gen;
351     end if;
352 end for;

```

```

353
354     return Results;
355
356 end function;
357
358
359 function EnumerateGenusSymbol(Symbol)
360 // Input: Genus-symbol Symbol of positive definite lattices of
361 // square-free level; t in N
362
363 // Output: Representatives of all isometry-classes belonging to
364 // the genus
365
366     try return eval Read(Sprintf("GenusSymbols/Gen_%o", Symbol));
367 catch e; end try;
368     try
369         Gen := eval Read(Sprintf("GenusSymbols/Gen_%o_partial",
370 Symbol));
371         printf "Only using partial genus for %o!\n", Symbol;
372         return Gen;
373     catch e; end try;
374
375     n := Symbol[2];
376
377     if n eq 0 then
378         return [LatticeWithGram(Matrix(Rationals(),0,0,[]))];
379     end if;
380
381     if n eq 1 then
382         det := &*[Symbol[i][1]^Symbol[i][2] : i in [3..#Symbol]];
383         L := LatticeWithGram(Matrix(Rationals(), 1, 1, [det]));
384         if GenSymbol(L) eq Symbol then
385             return [L];
386         end if;
387         return [];
388     end if;
389
390     if n eq 2 then
391         det := &*[Symbol[i][1]^Symbol[i][2] : i in [3..#Symbol]];
392
393         for m := 2 to Floor(1.155*Sqrt(det)) by 2 do
394             for a in [-m+1..m-1] do
395
396                 if not IsDivisibleBy(det + a^2, m) then continue; end if;
397                 b := Integers() ! ((det + a^2) / m);
398
399                 if b lt m then continue; end if;
400                 if not IsEven(b) then continue; end if;
401
402                 Mat := Matrix(Rationals(), 2, 2, [m,a,a,b]);
403                 if not IsPositiveDefinite(Mat) then continue; end if;
404
405                 L := LatticeWithGram(Mat);
406
407                 if not IsSquarefree(Level(L)) then continue; end if;

```

```

404
405         if Symbol eq GenSymbol(L) then
406             return EnumerateGenusOfRepresentative(L);
407         end if;
408     end for;
409 end for;
410
411     return [];
412
413 end if;
414
415 Rat := RationalsAsNumberField();
416 Int := Integers(Rat);
417
418 IdealList := [];
419 if Symbol[3][1] ne 2 then
420     Append(~IdealList, <ideal<Int|2>, [[0,n]]>);
421 end if;
422
423 for i in [3..#Symbol] do
424     p := Symbol[i][1];
425     np := Symbol[i][2];
426
427     if n eq np then
428         Append(~IdealList, <ideal<Int|p>, [[1,np]]>);
429     elif np eq 0 then
430         Append(~IdealList, <ideal<Int|p>, [[0,n]]>);
431     else
432         Append(~IdealList, <ideal<Int|p>, [[0,n-np],[1,np]]>);
433     end if;
434 end for;
435
436 "Constructing representatives";
437 try
438     Rep := LatticesWithGivenElementaryDivisors(Rat, n, IdealList);
439 catch e
440     print "Error while trying to construct a representative.
IdealList:";
441     IdealList;
442     return [];
443 end try;
444
445 for L in Rep do
446     LZ := ToZLattice(L);
447     if GenSymbol(LZ) eq Symbol then
448         Gen := EnumerateGenusOfRepresentative(LZ);
449         return Gen;
450     end if;
451 end for;
452
453 return [];
454
455 end function;
456
457

```

```

458 function SuperLatticesMagma(L, p, s, sigma)
459 // Input: Lattice L; Prime p; s in N; Automorphism sigma of L
460
461 // Output: All even sigma-invariant superlattices of L with index
         p^s using magmas method
462
463
464     LD := PartialDual(L,p:Rescale:=false);
465
466     G := MatrixGroup<NumberOfRows(sigma), Integers() | sigma >;
467     den1 := Denominator(BasisMatrix(LD));
468     den2 := Denominator(InnerProductMatrix(LD));
469
470     A := LatticeWithBasis(G, Matrix(Integers(), den1*BasisMatrix
         (LD)), Matrix(Integers(), den2^2*InnerProductMatrix(LD)));
471
472     SU := [];
473     SU := Sublattices(A, p : Levels := s, Limit := 100000);
474
475     if #SU eq 100000 then "List of sublattices is probably not
         complete."; end if;
476
477     Results := [];
478
479     for S in SU do
480         M := 1/den1 * 1/den2 * S;
481
482         if Determinant(M)*p^(2*s) eq Determinant(L) then
483             Append(~Results, M);
484         end if;
485     end for;
486
487     return [L : L in Results | IsEven(L)];
488 end function;
489
490
491
492 function SuperLattices(L1, Lp, p, sigma1, sigma2)
493 // Input: Lattice L1; Lattice Lp; Prime p; Automorphism sigma of L
494
495 // Output: All even superlattices of L1 + Lp invariant under diag
         (sigma1, sigma2) with index p^s using isometry-method
496
497     M := OrthogonalSum(L1, Lp);
498
499     L1Quot, phi1 := PartialDual(L1,p : Rescale:=false) / L1;
500     LpQuot, phi2 := PartialDual(Lp,p : Rescale:=false) / Lp;
501
502     m := #Generators(L1Quot);
503
504     phi1Inv := Inverse(phi1);
505     phi2Inv := Inverse(phi2);
506
507     G1 := ZeroMatrix(GF(p),m,m);
508     G2 := ZeroMatrix(GF(p),m,m);

```

```

509     for i in [1..m] do
510         for j in [1..m] do
511             G1[i,j] := GF(p) ! (p*InnerProduct(philInv(L1Quot.i),
philInv(L1Quot.j)));
512             Gp[i,j] := GF(p) ! (-p*InnerProduct(philInv(LpQuot.i),
philInv(LpQuot.j)));
513         end for;
514     end for;
515
516     V1 := KSpace(GF(p), m, G1);
517     Vp := KSpace(GF(p), m, Gp);
518
519     O1 := IsometryGroup(V1);
520
521     sigma1Quot := ZeroMatrix(GF(p),m,m);
522     for i in [1..m]
do
523         sigma1Quot[i] := Vector(GF(p), Eltseq(phil(philInv
(L1Quot.i)*Matrix(Rationals(),sigma1))));
524     end for;
525
526     sigma2Quot := ZeroMatrix(GF(p),m,m);
527     for i in [1..m]
do
528         sigma2Quot[i] := Vector(GF(p), Eltseq(phil(philInv
(LpQuot.i)*Matrix(Rationals(),sigma2))));
529     end for;
530
531     CL1Quot := Centralizer(O1, O1 ! sigma1Quot);
532
533     CL1 := Centralizer(AutomorphismGroup(L1), sigma1);
534
535     CL1ProjGens := [];
536     for g in Generators(CL1) do
537         gProj := ZeroMatrix(GF(p),m,m);
538         for i in [1..m] do
539             gProj[i] := Vector(GF(p), Eltseq(phil(philInv
(L1Quot.i)*Matrix(Rationals(), g))));
540         end for;
541         Append(~CL1ProjGens, gProj);
542     end for;
543
544     CL1Proj := MatrixGroup<m, GF(p) | CL1ProjGens>;
545
546     _, psi := IsIsometric(V1,Vp);
547
548     psi := MatrixOfIsomorphism(psi);
549     _, u := IsConjugate(O1, O1 ! sigma1Quot, O1 !
(psi*sigma2Quot*psi^(-1)));
550
551     phi0 := u*psi;
552
553     U, mapU := CL1Quot / CL1Proj;
554
555     LphiList := [];
556     for u in U do

```

```

557     phi := Inverse(mapU)(u)*phi0;
558
559     Gens := [];
560     for i in [1..m] do
561         x := phiInv(L1Quot.i);
562         y := phiInv(LpQuot ! Eltseq(phi[i]));
563         Append(~Gens, Eltseq(x) cat Eltseq(y));
564     end for;
565
566     Lphi := ext<M | Gens>;
567     Append(~LphiList, LatticeWithGram(LLLGram(GramMatrix(Lphi))));
568 end for;
569
570 return [L : L in LphiList | IsEven(L)];
571 end function;
572
573
574
575 function SuperLatticesJuergens(L, p, s)
576 // Input: Lattice L; Prime p; s in N; t in N
577
578 // Output: All even superlattices of L with index p^s using
579 // juergens method
580
581 if s eq 0 then
582     return [L];
583 end if;
584
585 T, mapT := PartialDual(L, p:Rescale:=false) / L;
586 mapTinv := Inverse(mapT);
587
588 m := #Generators(T);
589 G := GramMatrix(L);
590 G_F := MatrixAlgebra(GF(p), m) ! 0;
591
592 for i:=1 to m do
593     for j:=1 to m do
594         G_F[i, j] := GF(p) ! (p*InnerProduct(mapTinv(T.i), mapTinv
595 (T.j)));
596     end for;
597 end for;
598
599 V := KSpace(GF(p), m, G_F);
600 if not s le WittIndex(V) then
601     return [];
602 end if;
603
604 M1 := MaximalTotallyIsotropicSubspace(V);
605 M := sub< M1 | Basis(M1)[1..s] >;
606
607 O := IsometryGroup(V);
608 Aut := AutomorphismGroup(L:Decomposition:=true);
609
610 Gens := [];
611 for g in Generators(Aut) do

```



```

610     g_F:=MatrixAlgebra(GF(p),m)!0;
611     for i:=1 to m do
612         g_F[i]:=V!Vector(Eltseq(mapT(mapTinv(T!Eltseq
(V.i))*Matrix(Rationals(),g))));
613     end for;
614     Append(~Gens,g_F);
615 end for;
616
617 O_L:=sub< 0 | Gens>;
618 mapS,S,Kernel:=OrbitAction(O_L,Orbit(0,M));
619 Set:=[Inverse(mapS)(i[2]) : i in OrbitRepresentatives(S)];
620 SuperLat := [CoordinateLattice(ext< L | [mapTinv(T!Eltseq
(x)) : x in Basis(W)] >) : W in Set];
621
622 return [L : L in SuperLat | IsEven(L)];
623
624 end function;
625
626
627 function MiPoQuotient(sigma, L, p);
628 // Input : Automorphism sigma of L; Lattice L
629
630 // Output: Minimal polynomial of the operation of sigma on the
partial dual quotient L^(#, p) / L
631
632 sigma := Matrix(Rationals(), sigma);
633 L := CoordinateLattice(L);
634 LD := PartialDual(L, p : Rescale := false);
635 _,phi := LD / L;
636 MiPo := PolynomialRing(GF(p)) ! 1;
637
638 B := [];
639
640 for i in [1..Rank(LD)] do
641
642     b := LD.i;
643     if b in sub<LD|L,B> then
644         continue;
645     end if;
646     Append(~B,b);
647
648     dep := false;
649     C := [Eltseq(phi(b))];
650     while not dep do
651         b := b*sigma;
652         Append(~C, Eltseq(phi(b)));
653         Mat := Matrix(GF(p),C);
654         if Dimension(Kernel(Mat)) gt 0 then
655             dep := true;
656             coeff := Basis(Kernel(Mat))[1];
657             coeff /= coeff[#C];
658             coeff := Eltseq(coeff);
659             MiPo := LCM(MiPo, Polynomial(GF(p), coeff));
660         else
661             Append(~B, b);
662         end if;

```

```

663         end while;
664     end for;
665
666     return MiPo;
667
668 end function;
669
670
671 function ConstructLattices(l, n)
672 // Input: Square-free l; n in N
673
674 // Output: List of all extremal l-modular lattices that have a
        large automorphism sigma of order m, such that there is a prime
        divisor p of m with ggT(p,l) = 1 and mu_sigma / Phi_m | (x^(m/p)
        - 1)
675     Results := [];
676
677     min := ExtremalMinimum(l,n);
678
679     AutoTypes := AutomorphismTypes(l, Integers() ! (n/2), n, min);
680
681     for phim in [Integers() ! (n/2)+1 .. n] do
682
683         n1 := n - phim;
684         np := phim;
685
686         for m in EulerPhiInverse(phim) do
687
688             printf "m = %o\n", m;
689
690             for p in PrimeDivisors(m) do
691                 if Gcd(p,l) ne 1 then continue; end if;
692                 d := Integers() ! (m/p);
693                 PossibleTypes := [type : type in AutoTypes | type[1] eq p
and type[2] eq n1 and type[3] eq np and (type[4] eq 0 or EulerPhi
(d) le type[4])];
694
695
696                 for type in PossibleTypes do
697                     s := type[4];
698
699                     detp := p^s;
700                     for i := 5 to #type by 3 do
701                         detp *= type[i]^type[i+2];
702                     end for;
703
704                     // Enumerate ideal-lattices over K(zeta_m) with given
determinant
705                     K<z> := CyclotomicField(m);
706                     Kpos := sub<K | z + z^(-1)>;
707
708                     A := ClassesModGalois(K);
709                     M, U, FundUnits := EmbeddingMatrix(K, Kpos);
710                     LpList := IdealLattices(detp, K, Kpos, A, M, U,
FundUnits, false);

```

```

711         LpList := [L : L in LpList | Minimum(L) ge min];
712         LpList := ReduceByIsometry(LpList);
713
714     if n1 eq 0 then
715         Results cat:= LpList;
716         continue type;
717     end if;
718
719     for Lp in LpList do
720         if s ne 0 then
721             sigmapList := [c[3] : c in ConjugacyClasses
722 (AutomorphismGroup(Lp)) | MiPoQuotient(c[3], Lp, p) eq Polynomial
723 (GF(p), CyclotomicPolynomial(d))];
724             if #sigmapList eq 0 then
725                 continue Lp;
726             end if;
727         end if;
728
729         "Enumerate candidates for L_1";
730
731         K<z> := CyclotomicField(p);
732         Kpos := sub<K|z+1/z>;
733
734         if p eq 2 then
735             // In this case use the sublattice U of L_1 with U^
736             {#,2} = U
737             det1U := 1;
738             for i := 5 to #type by 3 do
739                 det1U *:= type[i]^type[i+1];
740             end for;
741
742             "Enumerate sublattices U";
743             UList := EnumerateGenusDeterminant(det1U, n1,
744 false);
745
746             "Construct L1 as superlattice for U";
747             L1List := &cat[SuperLatticesJuergens
748 (LatticeWithGram(2*GramMatrix(U)), p, Integers() ! ((n1 -
749 s)/2)) : U in UList | Minimum(U) ge Ceiling(min/2)];
750             L1List := [L : L in L1List | IsEven(L) and Minimum
751 (L) ge min];
752
753             elif IsPrime(l) then
754                 // In this case the genus symbol of L_1 is known
755                 and allows for a more efficient enumeration
756                 k1 := type[6];
757                 kp := type[7];
758
759                 if p le 3 then
760                     f := 1;
761                 else
762                     f := InertiaDegree(Factorization(ideal<Integers
763 (Kpos) | l>)[1][1]);
764                 end if;

```

```

758      deltap := (-1)^(Integers() ! (kp/f + (p-1)/2 *
(Binomial(Integers() ! (np / (p-1) + 1), 2) + Binomial(s, 2))));
759      delta1 := deltap * (-1)^(Integers() ! (s*(p-1)/2));
760
761      if l eq 2 then
762          if IsDivisibleBy(np + s*(p-1), 8) then
763              epsilonp := deltap;
764          else
765              epsilonp := -deltap;
766          end if;
767
768          if IsDivisibleBy(n, 8) then
769              epsilon := 1;
770          else
771              epsilon := -1;
772          end if;
773      else
774          epsilonp := (-1)^(Integers() ! (kp / f +
(l-1)/2*Binomial(kp,2)));
775
776          if IsDivisibleBy(n*(l+1), 16) then
777              epsilon := 1;
778          else
779              epsilon := -1;
780          end if;
781      end if;
782
783      epsilon1 := epsilonp*epsilon;
784
785      Sym1 := [* 2, n1 *];
786      if l eq 2 then
787          Append(~Sym1, <2, k1, epsilon1, 2, 0>);
788          Append(~Sym1, <p, s, delta1>);
789      else
790          if l lt p then
791              Append(~Sym1, <l, k1, epsilon1>);
792              Append(~Sym1, <p, s, delta1>);
793          else
794              Append(~Sym1, <p, s, delta1>);
795              Append(~Sym1, <l, k1, epsilon1>);
796          end if;
797      end if;
798
799      "Enumerate genus symbol";
800
801      L1List := [L : L in EnumerateGenusSymbol(Sym1) |
IsEven(L) and Minimum(L) ge min];
802
803      else
804
805          det1 := p^s;
806          for i := 5 to #type by 3 do
807              det1 := type[i]^type[i+1];
808          end for;

```

```

809
810         "Enumerate genus by determinant";
811
812         L1List := [L : L in EnumerateGenusDeterminant
163 (det1, n1, true) | IsEven(L) and Minimum(L) ge min];
813
814         end if;
815
816         for L1 in L1List do
817
818             M := CoordinateLattice(OrthogonalSum(L1,Lp));
819
820             if s eq 0 then
821                 if Minimum(M) ge min then
822                     Append(~Results, M);
823                 end if;
824                 continue L1;
825             end if;
826
827             sigmalList := [c[3] : c in ConjugacyClasses
163 (AutomorphismGroup(L1)) | MiPoQuotient(c[3], L1, p) eq Polynomial
163 (GF(p), CyclotomicPolynomial(d)) and Degree(MinimalPolynomial(c
163 [3])) le EulerPhi(d)];
828             if #sigmalList eq 0 then
829                 continue L1;
830             end if;
831
832             "Constructing superlattices";
833
834             if <l,n> in [] then
835                 for sigmal in sigmalList do
836                     for sigmap in sigmapList do
837                         LList cat:= SuperLatticesMagma(M, p, s,
163 DiagonalJoin(sigmal, sigmap));
838                     end for;
839                 end for;
840                 elif <l,n> in
163 [<7,18>,<7,20>,<1,24>,<2,24>,<5,24>,<1,32>] then
841                     LList := [];
842                     for sigmal in sigmalList do
843                         for sigmap in sigmapList do
844                             LList cat:= SuperLattices(CoordinateLattice
163 (L1), CoordinateLattice(Lp), p, sigmal, sigmap);
845                         end for;
846                     end for;
847                 else
848                     LList := SuperLatticesJuergens(M,p,s);
849                 end if;
850
851             Results cat:= [L : L in LList | Minimum(L) ge min];
852
853         end for;
854     end for;
855 end for;

```

```

856         end for;
857     end for;
858 end for;
859
860     return ReduceByIsometry(Results);
861
862 end function;
863
864
865 procedure MainLoop(nMin, nMax : lList :=
[1,2,3,5,6,7,11,14,15,23])
866     for n := nMin to nMax by 2
867     do
868         for l in lList do
869             if l eq 1 and not IsDivisibleBy(n,8) then continue; end if;
870             if l eq 2 and n eq 2 then continue; end if;
871             if l eq 11 and n in [20,24,28,30,32,34,36] then continue;
872             end if;
873             if l eq 23 and n ge 6 then continue; end if;
874
875             printf "dim = %0, l = %0\n", n, l;
876             Results := ConstructLattices(l, n);
877             ModList := [L : L in Results | IsModular(L, l)];
878             StrongModList := [L : L in Results | IsStronglyModular
(L,l)];
879             PrintFileMagma(Sprintf("SubidealLattices/%0-Modular/%0-
Dimensional", l, n), Results : Overwrite := true);
880             PrintFileMagma(Sprintf("SubidealLattices/%0-Modular/%0-
DimensionalModular", l, n), ModList : Overwrite := true);
881             PrintFileMagma(Sprintf("SubidealLattices/%0-Modular/%0-
DimensionalStronglyModular", l, n), StrongModList : Overwrite :=
true);
882
883             if #Results gt 0 then
884                 printf "\n\n-----n = %0, l = %0: %0 lattices found! %
o of them are modular and %0 are strongly modular-----\n\n",
n, l, #Results, #ModList, #StrongModList;
885             end if;
886         end for;
887     end for;
888 end procedure;

```

§ 6.6 MAGMA-Implementierungen zur Aufzählung der charakteristischen Polynome

Es folgt der Quellcode zu der Methode aus Abschnitt (4.7). Die implementierten Funktionen sind in der Reihenfolge des Quellcodes:

- Einschränken der Möglichkeiten für Automorphisentypen extremaler modularer Gitter, indem versucht wird, Geschlechter von Bild- und Fixgitter mit Dimension ≤ 12 aufzuzählen.
- Liste aller möglichen charakteristischen Polynome, die zu möglichen Automorphisentypen passende Partitionen des Raumes ergeben. Siehe Abschnitt (4.7).

```

1  load "SubidealLattices.m";
2
3  function RestrictAutomorphismTypes(l,n)
4  // Input: l in N; n in N
5
6  // Output: Restricts the possible automorphism types for extremal
7  modular lattices as much as possible
8
9  min := ExtremalMinimum(l,n);
10
11  Types := AutomorphismTypes(l, Integers() ! (n/2), n, min);
12
13  RestrictedTypes := [];
14
15  for type in Types do
16      type;
17      p := type[1];
18      n1 := type[2];
19      np := type[3];
20      s := type[4];
21
22      if p ne 2 and IsPrime(l) then
23          k1 := type[6];
24          kp := type[7];
25
26          K<z> := CyclotomicField(p);
27          Kpos := sub<K | z + z^(-1)>;
28
29          f := InertiaDegree(Factorization(ideal<Integers(Kpos) | l>
30 [1][1]));
31          deltap := (-1)^(Integers() ! (kp/f + (p-1)/2 * (Binomial
32 (Integers() ! (np / (p-1) + 1), 2) + Binomial(s, 2))));
33          delta1 := deltap * (-1)^(Integers() ! (s*(p-1)/2));
34
35          if l eq 2 then
36              if IsDivisibleBy(np + s*(p-1), 8) then
37                  epsilonp := deltap;
38              else
39                  epsilonp := -deltap;
40              end if;
41
42              if IsDivisibleBy(n, 8) then
43                  epsilon := 1;
44              else
45                  epsilon := -1;
46              end if;
47          else
48              epsilonp := (-1)^(Integers() ! (kp / f + (l-1)/2*Binomial
49 (kp,2)));
50
51              if IsDivisibleBy(n*(l+1), 16) then
52                  epsilon := 1;
53              else
54                  epsilon := -1;
55              end if;

```



```

53     end if;
54
55     epsilon1 := epsilonp*epsilon;
56
57     Sym1 := [* 2, n1 *];
58     Symp := [* 2, np *];
59     if l eq 2 then
60         Append(~Sym1, <2, k1, epsilon1, 2, 0>);
61         Append(~Sym1, <p, s, delta1>);
62         Append(~Symp, <2, kp, epsilonp, 2, 0>);
63         Append(~Symp, <p, s, deltap>);
64     else
65         if l lt p then
66             Append(~Sym1, <l, k1, epsilon1>);
67             Append(~Sym1, <p, s, delta1>);
68             Append(~Symp, <l, kp, epsilonp>);
69             Append(~Symp, <p, s, deltap>);
70         else
71             Append(~Sym1, <p, s, delta1>);
72             Append(~Sym1, <l, k1, epsilon1>);
73             Append(~Symp, <l, kp, epsilonp>);
74             Append(~Symp, <p, s, deltap>);
75         end if;
76     end if;
77
78     if n1 le 12 and n1 gt 0 then
79         List := [L : L in EnumerateGenusSymbol(Sym1) | Minimum(L)
ge min];
80         if #List eq 0 then
81             continue type;
82         end if;
83     end if;
84
85     if np le 12 and np gt 0 then
86         List := [L : L in EnumerateGenusSymbol(Symp) | Minimum(L)
ge min];
87         if #List eq 0 then
88             continue type;
89         end if;
90     end if;
91
92     else
93
94         if n1 le 12 and n1 gt 0 then
95             det1 := p^s;
96             for i := 5 to #type by 3 do
97                 det1 *:= type[i]^type[i+1];
98             end for;
99
100             List := [L : L in EnumerateGenusDeterminant(det1, n1,
true) | Minimum(L) ge min];
101             if #List eq 0 then
102                 continue type;
103             end if;
104         end if;
105

```

```

106     if np le 12 and np gt 0 then
107         detp := p^s;
108         for i := 5 to #type by 3 do
109             detp *:= type[i]^type[i+2];
110         end for;
111
112         List := [L : L in EnumerateGenusDeterminant(detp, np,
true) | Minimum(L) ge min];
113         if #List eq 0 then
114             continue type;
115         end if;
116     end if;
117
118 end if;
119
120 Append(~RestrictedTypes, type);
121 end for;
122
123 return RestrictedTypes;
124
125 end function;
126
127
128 function PossibleCharPos(l, n)
129 // Input: l in N; n in N
130
131 // Output: List of all characteristic polynomials of lattices
possibly not found by the subideal-lattice algorithm. Format:
[[<d_1,c_1>,...,<d_k,c_k>], ...] for the exponents c_i > 0 of the
Phi_(d_l) for the divisors d_l
132
133 Types := RestrictAutomorphismTypes(l,n);
134
135 Results := [];
136
137 for phim in [Integers() ! (n/2)+1..n] do
138     for m in EulerPhiInverse(phim) do
139         Div := Sort(Divisors(m));
140         Phi := [EulerPhi(d) : d in Div];
141         k := #Div;
142
143         pList := [p : p in PrimeDivisors(m) | Gcd(p,l) eq 1];
144         FixDimLists := [];
145         for p in pList do
146             FixDims := [];
147             for type in Types do
148                 if type[1] eq p then
149                     FixDim := type[2];
150                     if not FixDim in FixDims then
151                         Append(~FixDims, FixDim);
152                     end if;
153                 end if;
154             end for;
155             if #FixDims eq 0 then
156                 continue m;
157             end if;

```

```

158     Append(~FixDimLists, FixDims);
159 end for;
160
161 t := #pList;
162
163 M := ZeroMatrix(Integers(), k, t+1);
164 for i in [1..k] do
165     for j in [1..t] do
166         if IsDivisibleBy(Integers() ! (m/pList[j]), Div[i]) then
167             M[i,j] := Phi[i];
168         end if;
169     end for;
170     M[i, t+1] := Phi[i];
171 end for;
172
173 if t gt 0 then
174     TypeChoice := CartesianProduct([1..#List]: List in
FixDimLists]);
175
176     for IndexList in TypeChoice do
177         N := ZeroMatrix(Integers(), 1, t+1);
178         MaxDim := [];
179         for i in [1..t] do
180             N[1][i] := FixDimLists[i][IndexList[i]];
181         end for;
182         N[1][t+1] := n;
183
184         MaxDim := [Floor(n/EulerPhi(d)) : d in Div];
185         for i in [1..k] do
186             for j in [1..t] do
187                 if IsDivisibleBy(Integers() ! (m / pList[j]), Div
[i]) then
188                     MaxDim[i] := Minimum(MaxDim[i], Floor(N[1][j] /
Phi[i]));
189                 else
190                     MaxDim[i] := Minimum(MaxDim[i], Floor((n-N[1]
[j]) / Phi[i]));
191                 end if;
192             end for;
193         end for;
194
195         C := CartesianProduct([0..MaxDim[i]] : i in [1..k]);
196
197         for c in C do
198             v := Matrix(Integers(), 1, k, [x : x in c]);
199             if v*M eq N then
200                 if &or[c[i] gt 0 : i in [1..k-1]] and Lcm([Div[i] :
i in [1..k-1] | c[i] gt 0]) eq m then
201                     ExpList := [<Div[i], c[i]> : i in [1..k] | c[i] gt
0];
202                     Append(~Results, ExpList);
203                 end if;
204             end if;
205         end for;

```

```

206         end for;
207     else
208         C := CartesianProduct([[0..Floor(n/EulerPhi(d))] : d in
Div]);
209         N := Matrix(Integers(), 1, 1, [n]);
210         for c in C do
211             v := Matrix(Integers(), 1, k, [x : x in c]);
212             if v*M eq N then
213                 if Lcm([Div[i] : i in [1..k] | c[i] gt 0]) eq m then
214                     ExpList := [<Div[i], c[i]> : i in [1..k] | c[i] gt
0];
215                     Append(~Results, ExpList);
216                 end if;
217             end if;
218         end for;
219     end if;
220 end for;
221 end for;
222
223 return Results;
224
225 end function;

```

§ 6.7 Eidesstattliche Versicherung

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Masterarbeit mit dem Titel *Extremale Gitter mit großen Automorphismen* selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Aachen, im September 2018

Belehrung:

§ 156 StGB: Falsche Versicherung an Eides Statt

Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe von bis zu drei Jahren oder mit Geldstrafe bestraft.

§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt

(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe von bis zu einem Jahr oder Geldstrafe ein.

(2) Strafflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend

Die vorstehende Belehrung habe ich zur Kenntnis genommen:

Aachen, im September 2018

7 Literaturverzeichnis

- [BFS05] Eva Bayer Fluckiger and Ivan Suarez. Modular lattices over cyclotomic fields. *Journal of Number Theory*, 114:394–411, 2005.
- [CE03] Henry Cohn and Noam Elkies. New upper bounds on sphere packings I. *Annals of Mathematics*, 157:689–714, 2003.
- [CS93] J. H. Conway and N. J. A. Sloane. *Sphere packings, lattices and groups*, volume 290 of *Grundlehren der mathematischen Wissenschaften*. Springer, 3rd edition, 1993.
- [Jü15] Michael Jürgens. *Nicht-Existenz und Konstruktion extremaler Gitter*. PhD thesis, Technische Universität Dortmund, März 2015.
- [Kne02] M. Kneser. *Quadratische Formen*. Springer, 2002.
- [Lor11] David Lorch. Einklassige Geschlechter positiv definiter dreidimensionaler Gitter, 2011.
- [Mol11] Richard A. Mollin. *Algebraic number theory*. CRC Press, 2nd edition, 2011.
- [Neb13] Gabriele Nebe. On automorphisms of extremal even unimodular lattices. *International Journal of Number Theory*, 09:1933–1959, 2013.
- [Neu92] Jürgen Neukirch. *Algebraische Zahlentheorie*. Springer, 1992.

- [NS] Gabriele Nebe and N. J. A. Sloane. A Catalogue of Lattices. <http://www.math.rwth-aachen.de/~Gabriele.Nebe/LATTICES/>. Aufgerufen: 10.08.2018.
- [Que95] H. G. Quebbemann. Modular Lattices in Euclidean Spaces. *Journal of Number Theory*, 54:190–202, 1995.
- [SH98] Rudolf Scharlau and Boris Hemkemeier. Classification of integral lattices with large class number. *Mathematics of computation*, 67(222):737–749, April 1998.