2014-06-19

Hecke Eigenvalues and Arithmetic Cohomology

William Leonard Cocke
Brigham Young University - Provo

Hecke Eigenvalues and Arithmetic Cohomology

William Leonard Cocke

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Darrin Doud, Chair
Paul Jenkins
William Lang

Department of Mathematics

Brigham Young University

June 2014

ABSTRACT

Hecke Eigenvalues and Arithmetic Cohomology

William Leonard Cocke
Department of Mathematics, BYU
Master of Science

We provide algorithms and documention to compute the cohomology of congruence subgroups of $\mathrm{SL}_3(\mathbb{Z})$ using the well-rounded retract and the Voronoi decomposition. We define the Sharbly complex and how one acts on a k-sharbly by the Hecke operators. Since the norm of a sharbly is not preserved by the Hecke operators we also examine the reduction techniques described by Gunnells and present our implementation of said techniques for $n = 3$.

Keywords: Hecke Action, Arithmetic Cohomology, Sharblies, Modular Symbols

Acknowledgments

I am grateful to Darrin Doud for his support, guidance, and profinite patience throughout the project. Special thanks to my wife Em for her support throughout our life together. I would be remiss to not acknowledge the small primate who attended all of my research meetings, listened to my drafts, and helped me clarify new ideas. I also wish to publicly thank the Mathematics Department at Brigham Young University, especially the indispensable Lonette Stoddard, who kept the department running nigh single handedly. I am presently and always indebted to our Heavenly Father who knew all the following results and graciously allowed me to discover them for myself.

ERITIS SICTUM DEUS, SCIENTES BONUM ET MALUM.

Contents

List of Tables

List of Figures

Chapter 1. Introduction

In 1987, Serre [54] made his famous modularity conjecture. Serre conjectured a correspondence the eigenvalues of Hecke operators on modular forms and Galois representations. Reciprocity conjectures such as Serre's have since been generalized to the arithmetic cohomology of congruence subgroups of $\mathrm{SL}_n(\mathbb{Z})$. However, the computation of arithmetic cohomology has proven quite difficult even given larger, faster, computing devices. Currently such cohomology groups have been computed for $n \leq 4$ since larger n yield non-simplicial cell complexes. Moreover, given a cohomology class there is not in general a feasible way to compute the Hecke action.

For the cases when $n \leq 4$ the cohomology is always simplicial and there are techniques to compute it. Because the cohomology of congruence subgroups of $\mathrm{SL}_n(\mathbb{Z})$ vanishes in sufficiently high dimension, the needed calculations are always finite. However, the action of the corresponding Hecke operators is still difficult to compute and except for the top dimension such computations are not known to terminate and indeed have only been tested for $n = 2, 3$, and 4 for one of the dimensions of cohomology in each case.

1.1 Reciprocity Laws

Reciprocity laws are one of the most surprising discoveries of the last fifty years in mathematics. In general reciprocity laws attempt to establish a correspondence between certain properties of the absolute Galois group of the rational numbers, $G_{\mathbb{Q}} = \mathrm{Gal}(\overline{\mathbb{Q}}/\mathbb{Q})$, and properties of some other collection of objects. The absolute Galois group of $\mathbb{Q}$ is a somewhat mysterious group. For example, it has infinite order and yet only two of the automorphisms which make it up can be explicitly written down: the identity automorphism and complex conjugation. Since $G_{\mathbb{Q}}$ corresponds to automorphisms which permute roots of polynomial equations, knowing the structure of $G_{\mathbb{Q}}$ would lend considerable insight into polynomials over $\mathbb{Z}$. Consequently reciprocity laws are particularly useful and interesting when the ob-

jects involved are computable and thus allow one to understand certain properties of $G_{\mathbb{Q}}$. Consider the following general examples:

Example 1.1. i) Certain homomorphisms $G_{\mathbb{Q}} \longrightarrow \mathbb{F}^{\times}$ correspond to homomorphisms of $(\mathbb{Z}/N)^{\times} \longrightarrow \mathbb{F}^{\times}$ for various N.

ii) Certain homomorphisms $G_{\mathbb{Q}} \to \mathrm{GL}_2(\overline{\mathbb{Q}_p})$ correspond to modular forms for $\Gamma_0(N)$ for various N.

iii) Certain homomorphisms $G_{\mathbb{Q}} \to \mathrm{GL}_n(\overline{\mathbb{F}_p})$ correspond to Hecke eigenclasses in $H_*(\Gamma, V)$ where $\Gamma = \Gamma_0(N) \subset \mathrm{SL}_n(\mathbb{Z})$ for V a mod p coefficient module.

In general these reciprocity laws allow us to understand some of the properties of $G_{\mathbb{Q}}$ by studying concrete objects. One of the most widely acclaimed reciprocity laws is Serre's modularity conjecture.

1.2 Serre's Conjecture and Generalizations

In 1973, Serre conjectured that odd irreducible continuous representations $\rho : G_{\mathbb{Q}} \to \mathrm{GL}_2(\overline{\mathbb{F}}_p)$ are modular; meaning that there is a resulting modular form which is an eigenfunction of the Hecke operators such that the mod p reduction of the ℓth eigenvalue equals the trace of the Frobenius of ρ. Serre gave a conjectural recipe for the character, level, and weight of the corresponding space of modular forms. For more details on Serre's conjecture see [54]. It is worth noting that Serre's conjecture has been proven by a series of celebrated results of Khare and Wintenberger [36][37][38].

Recall that the Hecke operators are a family of commuting linear operators that act on the space of cuspidal modular forms of a given weight. Consequently, the Hecke operators have common eigenspaces. These eigenspaces are spanned by eigenfunctions. The mod p

reductions of the eigenvalues of these eigenfunctions correspond to the coefficients of the characteristic polynomial of the Frobenius of ρ.

Serre's conjecture served as the foundation for other conjectures concerning homomorphisms from $G_{\mathbb{Q}}$ to groups of matrices. These generalizations of Serre's conjecture mostly take one of two forms. One area of research is the examination of two dimensional representations of the absolute Galois groups of different number fields. Research conducted by Cremona [27] focused on imaginary quadratic extensions and Dembélé[28] has examined real quadratic extensions. More recently, work by Gunnells, Hajir, and Yasaki [34] examine the case when $\mathbb{Q}$ is replaced by $\mathbb{Q}(\zeta_5)$ the cyclotomic quartic field; this case is particularly interesting since they needed to use the Sharbly complex, a technique developed to study n-dimensional representations of $G_{\mathbb{Q}}$ for $n > 3$.

The other main generalization of Serre's conjecture involves studying representations $\rho : G_{\mathbb{Q}} \to \mathrm{GL}_n(\overline{\mathbb{F}_p})$. To do this one reinterprets the correspondence in terms of the cohomology of congruence subgroups of $\mathrm{SL}_n(\mathbb{Z})$. For any n the cuspidal cohomology occurs in bands of dimension roughly half-way between 0 and a finite number called the cohomological dimension. The following table based on formulas of Schwermer [52] gives the total dimension of cohomology in terms of the dimension of a complex $\mathcal{X}$, which we will construct explicitly in chapter 2. However, the cohomology of congruence subgroups Γ vanishes after a slightly smaller number known as the virtual cohomological dimension. Schwermer also gives formulas which compute the range of cuspidal cohomology.

Table 1.1: Dimensions of Cohomology and the Cuspidal Range

n	2	3	4	5	6	7	8	9
$\dim(\mathcal{X})$	2	5	9	14	20	27	35	44
vcd Γ	1	3	6	10	15	21	28	36
top degree of H^*_{cusp}	1	3	5	8	11	15	19	24
bottom degree of H^*_{cusp}	1	2	4	6	9	12	16	20

The study of the cohomology of subgroups of $\mathrm{SL}_n(\mathbb{Z})$ yields a theory of boundary forms and cuspidal forms mirroring that of modular forms. See Schwermer [52] for more details.

Most of the work involving the generalization to n-dimensional representations of $\mathbb{G}_Q$ has focused on calculating the top dimensional cohomology for $n = 3$. Various computations for $n = 3$ supporting a conjectural correspondence between Galois representations and Hecke eigenvalues for the top dimensional cohomology have been conducted by Avner Ash and his collaborators [1][2][5][10][11][17][18][19][59].

For $n = 4$ the top dimensional cohomology occurs in dimension 6, while the cuspidal cohomology appears in dimensions 4 and 5. A series of papers by Ash, Gunnells, and McConnell establish the needed machinery and provide computational results for $H^5(\Gamma_0(N), \mathbb{C})$ for $\Gamma_0(N)$ a congruence subgroup of $\mathrm{SL}_4(\mathbb{Z})$ [12][13][14][15][16]. The next section examines the details of the generalization of Serre's conjecture to n-dimensional Galois representations.

1.3 The Ash-Doud-Pollack-Sinnott Conjecture

For a continuous representation

$$\rho : G_{\mathbb{Q}} \to \mathrm{GL}_n(\overline{\mathbb{F}}_p)$$

unramified outside of pN, we can define $\det(I - \rho(\mathrm{Frob}_\ell)T)$ for $\ell \not| pN$ since the determinant is conjugacy invariant. Let $v \in H_*(\Gamma_0(N), \mathbb{C})$ be a Hecke eigenclass, where

$$\Gamma_0(N) = \{\gamma \in \mathrm{SL}_n(\mathbb{Z}) : \text{ the bottom row of } \gamma \equiv (0, \ldots, 0, *) \pmod{N}.$$

In such a situation we say that ρ is attached to v if for every $\ell \not| pN$ we have that

$$P_{v,\ell}(T) = \sum_{k=0}^{n} (-1)^k a_{\ell,k} \ell^{k(k-1)/2} T^k = \det(I - \rho(\mathrm{Frob}_\ell)T),$$

where $a_{\ell,k}$ is an eigenvalue of the Hecke operator $T_{\ell,k}$.

Based on Serre's conjecture Ash conjectured the following in [3].

Conjecture 1.2. *For all simultaneous eigenvectors v of the Hecke operators there is an attached ρ and such a representation must be* odd, *meaning that complex conjugation is*

mapped to a matrix γ such that γ is conjugate to a matrix with alternating 1's and -1's down the diagonal. Consequently, the trace of complex conjugation is either $1, 0$ or -1.

The above conjecture was proven by Peter Scholze [50], in a widely acclaimed tour de force, conditional on some results in operator theory. Ash also developed a converse conjecture in his work with Sinnott [20] and in his work with Doud and Pollack [10].

Conjecture 1.3. *For all odd continuous $\rho : G_{\mathbb{Q}} \to GL_n(\overline{\mathbb{F}_p})$ there exists a Hecke eigenclass v with ρ attached.*

We will call this conjecture the ADPS conjecture. The strong form of the conjecture gives specific recipes for the weight, level, and character attached to the representations. The aforementioned papers of Ash and Sinnott [20], and Ash, Doud, and Pollack [10], give computational evidence for this conjecture in the case $n = 3$ for which the eigenclass belongs to the top dimensional cohomology. Moreover in certain cases the ADPS conjecture has been proven:

i) For $n = 2$ it is none other than Serre's conjecture.

ii) Ash proved that the ADPS conjecture holds if the representation is a sum of one-dimensional representations with certain other restrictions [6].

iii) For sums of a one-dimensional character and a two-dimensional odd representation [9] work by Ash and Doud proves the conjecture in squarefree level.

iv) In [10] Ash, Doud, and Pollack prove that the ADPS conjecture holds for certain irreducible symmetric square representations.

v) In [8] Ash and Doud prove that for certain n-dimensional highly reducible representations the conjecture holds.

1.4 Computation for Cohomology and Hecke Operators

To generate evidence for the ADPS conjecture, one encounters two distinct computational challenges. First, one must compute the cohomology of congruence subgroups of $\mathrm{SL}_n(\mathbb{Z})$. Second, one must act on the cohomology of such subgroups by the Hecke operators. The first problem has historically been the easier of the two to solve.

One way to calculate the cohomology of congruence subgroups $\Gamma_0(N) \subset \mathrm{SL}_n(\mathbb{Z})$ is to first design algorithms to calculate the cohomology of $\mathrm{SL}_n(\mathbb{Z})$ and then apply Shapiro's lemma [61]. However, the action of the Hecke operators is not easily computable on the general constructions of homology for $\mathrm{SL}_n(\mathbb{Z})$. For the top dimensional cohomology this conundrum was resolved by Manin [44] with his discovery of modular symbols.

Most research has focused on cuspidal cohomology, and since for $n = 3$ and $n = 4$ there is a Leftschetz duality between the the cuspidal cohomology occuring in H^3 and H^2 for $n = 3$ and H^5 and H^4 for $n = 4$, previous computations have focused exclusively on H^3 with arbitrary coefficient modules for $n = 3$ and $H^5(\Gamma_0(N), \mathbb{F}_p)$ for $n = 4$.

As part of this thesis we wrote a program which calculates $H^2(\Gamma, \mathbb{C})$ and $H^3(\Gamma, \mathbb{C})$ for $\Gamma = \Gamma_0(N)$ a congruence subgroup of $\mathrm{SL}_3(\mathbb{Z})$. These calculations could provide computational evidence for the theorems of Ash and Doud in [7]. Moreover, such computations should also generate new examples of boundary cohomology that have not previously been studied. However in all cases we computed we found only cuspidal cohomology, which is dual to the cuspidal cohomology of $H^3(\Gamma, \mathbb{C})$. It is known that there are boundary classes corresponding to the cohomology with nontrivial coefficient modules and nontrivial characters; it was not previously known that for $N < 65$ there are no boundary classes in $H^2(\Gamma, \mathbb{C})$. We tested our calculations by computing the Hecke action on the top dimensional cohomology and the eigenvalues agreed with those previous calculated by Ash and others.

In order to calculate the cohomology one degree below the cohomological dimension we used techniques developed by McConnell [45] which allow one to explicitly compute the cohomology for $n < 5$; these techniques use the Voronoi complex as studied by Voronoi in

his work on perfect quadratic forms [60].

As part of our research we also wrote a program to test an experimental technique of Gunnells [33] which aims to calculate the Hecke action on cohomology classes one below the cohomological dimension. Gunnells' technique is not known to terminate and our experiments provide additional evidence that it does so. However, we did not test Gunnells methods on any cohomology classes, since we found no new classes to test.

All of our calculations were done in C++, but we verified many of our calculations using PARI [58].

In this thesis we first describe how to calculate $H^2(\Gamma_0(N), \mathbb{C})$ and $H^3(\Gamma_0(N), \mathbb{C})$ for $\Gamma_0(N)$ a congruence subgroup of $\mathrm{SL}_3(\mathbb{Z})$. For computational purposes we will show that the cells of the related retract correspond to orbits of $\mathbb{P}^2(\mathbb{Z}/N\mathbb{Z})$ under the right group actions. Then we can use the orbits to construct the boundary maps which yield the cohomology (chapter 2). Then for each cohomology class there is a corresponding chain of orbits of $\mathbb{P}^2(\mathbb{Z}/N\mathbb{Z})$ and we use this correspondence to map our cohomology into the Sharbly complex, a complex developed by Lee and Szczarba [42] to study cohomology of $\mathrm{SL}_n(\mathbb{Z})$ (chapter 3). We map into the Sharbly complex because the Sharbly complex has an easily computed Hecke action (chapter 4). However, after acting by the Hecke operators, the sharbly chain needs to be modified before we can map it back into our cell complex (chapter 5). We then present an example of this process (chapter 6).

Chapter 2. Calculating Cohomology

We will discuss two cell complexes that have been used to calculate the cohomology of congruence subgroups of $\mathrm{SL}_3(\mathbb{Z})$. One complex was utilized by Voronoi in his study of quadratic forms [60]; albeit Voronoi was not explicitly aware of its cohomological application at the time. A dual complex, known as the well-rounded retract, was discovered by Ash and utilized to calculate the homology of $\mathrm{SL}_3(\mathbb{Z})$. Chronologically the well-rounded retract was

utilized for reciprocity calculations beginning in 1975 [56], while the computational value of the Voronoi complex was not exploited until 2000 [33]. Although our computations follow the work of Voronoi, we summarize the work of Ash and Voronoi separately for historical reasons. Then we explain how one explicitly uses the information from the Voronoi celluation to actually compute cohomology. The methods mentioned generalize to higher dimensions, but in what follows we will often focus on $n = 3$.

2.1 Ash's Approach

Ash's computations of cohomology start with the cone of positive-definite, symmetric, real, $n \times n$ matrices; for convenience we let C denote this cone, and we will show that there is a smaller space called the well-rounded retract $\mathcal{W}$ inside of C which can used to calculate homology. In what follows it is assumed that $n \geq 2$ and is fixed. Naturally C relates to quadratic forms and the theory was historically developed for the study and classification of quadratic forms. Recall that for a symmetric $n \times n$ matrix A we can symbolically define a quadratic form on n variables by $X^T AX$, where X is a column vector containing the variables.

The following definitions can be found in Ash [2]. There are similar definitions in the work of Voronoi [60] and we shall later see that these definitions provide the duality of the two complexes.

Definition 2.1. The *arithmetic minimum* of $A \in C$ is defined to be

$$m(A) = \min\{x^T Ax : x \in \mathbb{Z}^n \backslash \{0\}\}.$$

Since A is positive-definite, we note that $m(A)$ exists. If A is integral than $m(A)$ is a positive integer.

Definition 2.2. The set of *minimal vectors* of $A \in C$ are

$$M(A) = \left\{ x \in \mathbb{Z}^n : x^T A x = m(A) \right\}.$$

In words, $M(A)$ is the set of vectors which achieve the arithmetic minimum of A. Alternatively, we know from linear algebra that any $A \in C$ can be written as $A = B^T B$ for some invertible matrix B. Then

$$m(A) = \min\{x^T B^T B x : x \in \mathbb{Z}^n \backslash \{0\}\} = \min\{||Bx|| : x \in \mathbb{Z}^n\}$$

is the length of the shortest vector of the lattice $B\mathbb{Z}^n$. Similarly

$$M(A) = \{x \in \mathbb{Z}^n : ||Bx|| = m(A)\},$$

which is the set of minimal vectors of $B\mathbb{Z}^n$. Since each lattice has only a finite set of minimal vectors we see that $|M(A)|$ is finite. We will discuss a few basic details about lattices in 5.2 where we will also discuss the concept of a reduced lattice and LLL-reduction. $M(A)$ will be invariant under multiplication by -1 since $(-x)^T A(-x) = x^T A x$.

It should be noted that a vector in $M(A)$ will always be primitive, meaning that the greatest common divisor of the entries is 1.

The following definition is crucial to the construction of the well-rounded retract $\mathcal{W}$.

Definition 2.3. $A \in C$ is *well-rounded* if $M(A)$ spans $\mathbb{R}^n$.

Example 2.4. i) Let I_n be the $n \times n$ identity matrix. Clearly $I_n \in C$. We see that $m(I_n) = 1$, and $M(I_n) = \{\pm e_1, \pm e_2, \ldots, \pm e_n\}$, where e_i is the standard ith unit vector. I_n is well-rounded.

ii) Let

$$J = \begin{pmatrix} 1 & 1/4 \\ 1/4 & 1 \end{pmatrix} \in C.$$

Then for

$$X = (x, y)^T, X^T JX = x^2 + \frac{1}{2}xy + y^2.$$

Since $(1, 0) \to 1$ we have that $m(J) = 1$ and solving the equation $x^2 + 1/2xy + y^2 = 1$ for integer points gives us that $M(J) = \{\pm e_1, \pm e_2\}$. We thus see that $m(A)$ and $M(A)$ do not uniquely determine A since $m(I_2) = m(J)$ and $M(I_2) = M(J)$. This will be an important property in the rest of the chapter.

iii) Let

$$K = \begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix} \in C.$$

Then for

$$X = (x, y, z)^T, X^T KX = 2(x^2 + y^2 + z^2 - xy - yz).$$

A direct approach shows us that $m(K) = 2$ with $M(K) = \{\pm e_1, \pm e_2, \pm e_3, \pm(e_1 + e_2), \pm(e_2 + e_3), \pm(e_1 + e_2 + e_3)\}$. K is also well-rounded.

iv) Let

$$N = \begin{pmatrix} 3 & 1 \\ 1 & 2 \end{pmatrix} \in C.$$

Then for

$$X = (x, y)^T, X^T NX = 3x^2 + 2xy + 2y^2.$$

Then $m(N) = 2, M(N) = \{\pm e_2\}$. Therefore N is not well-rounded.

What makes C useful for our desired computations is that it has a natural $\mathrm{GL}_n(\mathbb{R})$ action; moreover the inherited action of $\mathrm{GL}_n(\mathbb{Z})$ will take well-rounded matrices to well-rounded

matrices. For a matrix $G \in \mathrm{GL}_n(\mathbb{R})$ and $A \in C$ define $G \cdot A = GAG^T$. Since

$$X^T(GAG^T)X = (G^TX)^TA(G^TX) \tag{2.1}$$

we have that $G \cdot A$ is positive definite; taking the transpose shows it is symmetric and hence $G \cdot A$ is in C. The stabilizer of I_n is $\{G \in \mathrm{GL}_n(\mathbb{R}) : G^TG = I\} = \mathrm{O}(n, \mathbb{R})$ by definition.

Lemma 2.5. *The action of $GL_n(\mathbb{R})$ on C is transitive.*

Proof. As mentioned above it is a standard fact from linear algebra that for $A \in C$ there is a $B \in \mathrm{GL}_n(\mathbb{R})$ such that $A = B^TB$. For $A, D \in C$ let $A = B_1^TB_1, D = B_2^TB_2$ for some $B_1, B_2 \in \mathrm{GL}_n(\mathbb{R})$. Then

$$\left(B_1{}^{-1}B_2\right)^T \cdot A = B_2^TB_1^{-1^T}B_1^TB_1B_1^{-1}B_2 = D.$$

□

Hence the $\mathrm{GL}_n(\mathbb{R})$ action on C is transitive, and C is diffeomorphic to the symmetric space $\mathrm{GL}_n(\mathbb{R})/O(n, \mathbb{R})$.

Following Ash [2] we let $C_1 = \{A \in C : m(A) = 1\}$. Then we set

$$\mathcal{X} \cong \{A \in C_1 : A \text{ is well-rounded}\}.$$

$\mathcal{X}$ can be thought of as the set of rays of well-rounded matrices; meaning every ray of well-rounded matrices has a unique representative in $\mathcal{X}$.

Because A is well-rounded if and only if $M(A)$ has rank n, restricting the action in equation 2.1 to integral matrices shows us that $M(G \cdot A)$ will also be well-rounded. Hence the $\mathrm{GL}_n(\mathbb{Z})$ action will take well-rounded matrices to well-rounded matrices. Since the actions of $\mathrm{GL}_n(\mathbb{R})$ and $\mathrm{GL}_n(\mathbb{Z})$ commute with homotheties, we see that the $\mathrm{GL}_n(\mathbb{Z})$ action on C induces a $\mathrm{GL}_n(\mathbb{Z})$ action on $\mathcal{X}$.

For our calculations we do not work with the full general linear group, but instead consider subgroups of $\mathrm{SL}_n(\mathbb{R})$. In an analogous way to the above work we see that the $\mathrm{SL}_n(\mathbb{R})$ action is transitive on C and the stabilizer of any point is isomorphic to $\mathrm{SO}_n(R)$; this allows us to identify $\mathcal{X}$ with the global Riemannian symmetric space $\mathrm{SL}_n(\mathbb{R})/\mathrm{SO}_n(R)$. This space is a contractible smooth manifold of real dimension $d = n(n+1)/2 - 1 = n(n-1)/2$. For $n = 2$ the reader might be familiar with the standard representation of this space via the upper half-plane. (See Gelfand [32] and Example 2.11 for more details.)

Any subgroup $\Gamma \leq \mathrm{SL}_n(\mathbb{R})$ inherits the action on C. Let $\Gamma = \Gamma_0(N)$, the congruence subgroup of $\mathrm{SL}_n(\mathbb{Z})$ consisting of matrices whose bottom row is congruent to $(0, \ldots, 0, *) \pmod N$.

For $\gamma \in \Gamma$, we have that $\gamma\mathbb{Z}^n = \mathbb{Z}^n$, since γ is an integral matrix of determinant 1. Thus C_1 and $\mathcal{X}$ are Γ-invariant. Relying on work from Ash [2] and Soulé [56], we know that there exists a Γ-equivariant deformation retract of C_1 onto $\mathcal{X}$. Furthermore, one can construct such a retract explicitly. To do so, let E be a subset of $\mathbb{Z}^n$. We define a function σ from the power set of $\mathbb{Z}^n$ to the power set of $\mathcal{X}$ as follows.

Definition 2.6. Let $\sigma(E) = \{A \in \mathcal{X} : M(A) = E\}$.

Example 2.7. i) Let E be any finite subset of $\mathbb{Z}^n$ containing the zero vector $\mathbf{0}$. Then since $A((\mathbf{0})) = \mathbf{0}$ we have that $\sigma(E) = \emptyset$ and $|\sigma(E)| = 0$.

ii) Let $n = 2$ and let $E = \{\pm e_1, \pm e_2\}$. Let $A = \begin{pmatrix} x & y \\ y & z \end{pmatrix} \in C$. Since $M(A) = E$ and $m(A) = 1$ we have that $e_1^T A e_1 = x = 1$ and $e_2^T A e_2 = z = 1$. Noting that $\pm(e_1 - e_2) \notin M(A)$, we have $(e_1 - e_2)^T A(e_1 - e_2) = 2 - 2y > 1$ and $(e_1 + e_2)^T A(e_1 + e_2) = 2y + 2 > 1$ which means that $|y| < 1/2$. Thus

$$\sigma(E) = \left\{ \begin{pmatrix} 1 & y \\ y & 1 \end{pmatrix} : -\frac{1}{2} < y < \frac{1}{2} \right\}$$

and $|\sigma(E)|$ is infinite.

iii) Let $n = 2$ and let $E = \{\pm e_1, \pm e_2, \pm(e_1 + e_2)\}$. Let $A = \begin{pmatrix} x & y \\ y & z \end{pmatrix} \in C$. Then as above $x = z = 1$ and we have $(e_1 + e_2)^T A(e_1 + e_2) = 2y + 2 = 1 \Rightarrow y = -\frac{1}{2}$. In this case we have

$$\sigma(E) = \left\{ \begin{pmatrix} 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 \end{pmatrix} \right\}$$

and $|\sigma(E)| = 1$.

iv) If E doesn't span $\mathbb{R}^n$, or is not invariant under -1 then $\sigma(E) = \emptyset$. If E contains a non-primitive point then $\sigma(E) = \emptyset$. Furthermore the work of Voronoi [60] shows that if $|E|$ is greater than $2^n - 1$ then $\sigma(E) = \emptyset$.

Remark. It would be very interesting to have an absolute criterion on a subset K of $\mathbb{Z}^n$ that determines whether $\sigma(K)$ is nonempty.

In the above example we have seen sets E such that $|\sigma(E)| = 0, 1$, or ∞. The next lemma implies that these are the only possibilities for $|\sigma(E)|$.

Lemma 2.8. *$\sigma(E)$ is convex.*

Proof. If $|\sigma(E)| \leq 1$ this is obviously true. Assume $A, B \in \sigma(E)$ and $A \neq B$. Then $X^T AX = X^T BX = 1$ for all $X \in E$. Then $X^T \left((1-\alpha)A + \alpha B\right) X = X^T(1-\alpha)AX + X^T \alpha BX = 1 - \alpha + \alpha = 1$ and thus $((1-\alpha)A + \alpha B) \in E$. □

Proposition 2.9. (**Ash** [2].) *$\sigma(E)$ is bounded in the vector space V of all $n \times n$ symmetric matrices.*

Proof. Let $A = B^T B \in \sigma(E)$ for some $B \in \mathrm{GL}_n(\mathbb{R})$. As found after definition 2.2 $m(A)$ is equal to the length of the shortest vector of $B\mathbb{Z}^n$. Now since A is well-rounded we have that $M(A)$ spans $\mathbb{R}^n$. Thus the minimal vectors of $B\mathbb{Z}^n$ being $BM(A)$ must span $\mathbb{R}^n$. Consequently the lattice $B\mathbb{Z}^n$ is determined by $BM(A)$.

Since $A \in \mathcal{X}$, we have that $m(A)$ is 1 and thus $BM(A)$ is contained in the unit sphere of $\mathbb{R}^n$. We know that $BM(A)$ must have cardinality equal to $|E|$ and span $\mathbb{R}^n$. Since there are only a compact set of such possibilities (the n-sphere), we see that $\sigma(E)$ is relatively compact and thus bounded. $\square$

Having shown $\sigma(E)$ is a convex, bounded subset of V, we see that its closure is a cell as defined by Hatcher [35]. The well-rounded retract $\mathcal{W}$ is the cell complex obtained by decomposing $\mathcal{X}$ into the union of the closures of the $\sigma(E)$. By construction Γ acts on $\mathcal{X}$ cellularly, since $\sigma(\gamma E) = \gamma^T \sigma(E)\gamma = \gamma \cdot \sigma(E)$.

Theorem 2.10. (**Ash** [2].) *The retract $\mathcal{W}$ has the structure of a connected locally finite regular cell complex in which the closed cells are exactly the nonempty sets of the form $\sigma(E)$ for all finite subsets $E \subset \mathbb{Z}^n$.*

The construction of the well-rounded retract $\mathcal{W}$ has only explicitly been carried out for $n \leq 5$. For a given n there are only finitely many $\mathrm{SL}_n(\mathbb{Z})$-orbits of cells. Moreover, Ash showed that the virtual cohomological dimension of the cell complex is equal to $\frac{n(n-1)}{2}$.

Example 2.11. Consider $n = 2$. There are two $\mathrm{SL}_2(\mathbb{Z})$ orbits of cells:

i) The 0-dimensional orbit is the orbit of

$$\tau = \left\{ \begin{pmatrix} 1 & 1/2 \\ 1/2 & 1 \end{pmatrix} \right\}.$$

We note that $\tau = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \cdot \sigma(E)$ from example 2.7 iii) since

$$\begin{pmatrix} 0 & -1 \\ -1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1/2 \\ 1/2 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & -1/2 \\ -1/2 & 1 \end{pmatrix}.$$

ii) The top dimension is spanned by a single $\mathrm{SL}_2(\mathbb{Z})$-orbit; this 1-dimensional orbit is the orbit of

$$\sigma = \left\{ \begin{pmatrix} 1 & y \\ y & 1 \end{pmatrix} : |y| < \frac{1}{2} \right\} = \sigma(\pm\{e_1, e_2\}).$$

We can see the details of this cell complex by first realizing $\mathfrak{h} = \{z \in \mathbb{C} : \mathrm{Im}(z) > 0\}$ as the homogeneous space for $\mathrm{SPos}_2(\mathbb{R})/\mathrm{SO}_2(\mathbb{R})$ where $\mathrm{SPos}_2(\mathbb{R})$ is $C \cap \mathrm{SL}_2(\mathbb{R})$. To do this we take the map $f : \mathrm{SPos}_2(\mathbb{R})/\mathrm{SO}_2(\mathbb{R}) \to \mathfrak{h}$ given by $f\left(\begin{pmatrix} a & b \\ b & c \end{pmatrix} \mathrm{SO}_2(\mathbb{R})\right) = \frac{1}{a}(-b+i)$.

Alternatively $\begin{pmatrix} a & b \\ b & c \end{pmatrix} = \frac{1}{a}\begin{pmatrix} 1 & -x \\ -x & y \end{pmatrix}$ for suitable x, y. Then $f\left(\begin{pmatrix} a & b \\ b & c \end{pmatrix} \mathrm{SO}_2(\mathbb{R})\right) = x + iy$.

Thus after scaling by the determinant we can see that $f(\sigma)$ is the arc of the unit circle between $-\frac{1}{2} + \frac{\sqrt{3}}{2}i$ and $\frac{1}{2} + \frac{\sqrt{3}}{2}i$. Analogously $f(\tau)$ is the point $-\frac{1}{2} + \frac{\sqrt{3}}{2}i$. The other cells are all contained in the same $\mathrm{SL}_2(\mathbb{Z})$-orbit, where $\mathrm{SL}_2(\mathbb{Z})$ acts on $\mathfrak{h}$ by fractional linear transformation. The exposition by Fuchs [31] shows that the map f is $\mathrm{SL}_2(\mathbb{Z})$-equivariant. A portion of this orbit is presented below in red.

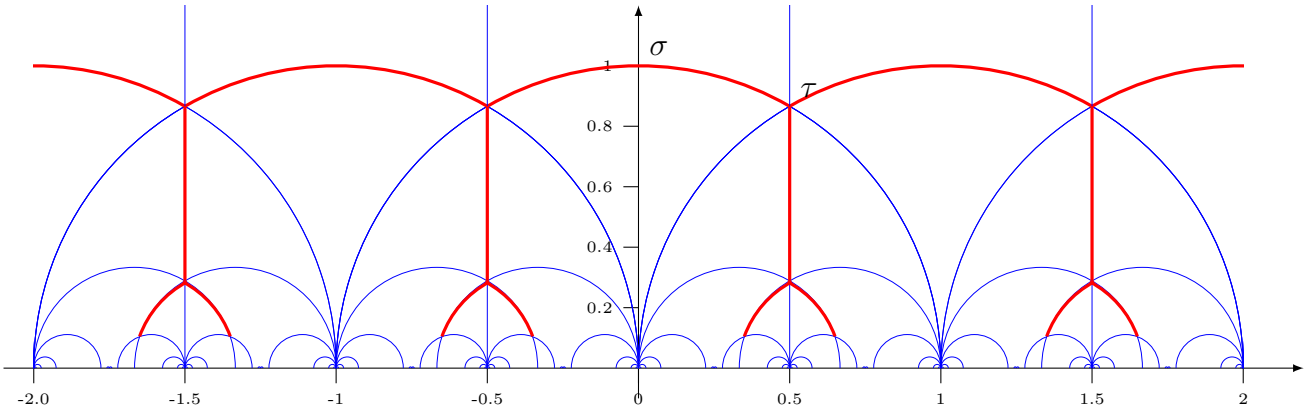

Figure 2.1: A Portion of the $\mathrm{SL}_2(\mathbb{Z})$ orbit of $\overline{\sigma}$.

Example 2.12. For $n = 3$ there are five $\mathrm{SL}_3(\mathbb{Z})$ orbits of cells:

i) The 0-dimensional orbit is the orbit of

$$\mu = \sigma\left(\pm\{e_1, e_2, e_3, e_1+e_2, e_1+e_2+e_3, e_1+e_3\}\right).$$

ii) The 1-dimensional orbit is the orbit of

$$\omega_2 = \sigma\left(\pm\{e_1, e_2, e_3, e_1+e_2, e_1+e_2+e_3\}\right).$$

iii) There are two 2-dimensional orbits, one is the orbit of

$$\tau_2 = \sigma\left(\pm\{e_1, e_2, e_3, e_1+e_2\}\right),$$

and the other is the orbit of

$$\tau_3 = \sigma\left(\pm\{e_1, e_2, e_3, e_1+e_2+e_3\}\right).$$

ib) There is one top dimensional orbit of dimension 3, it is the orbit of

$$\sigma_0 = \sigma\left(\pm\{e_1, e_2, e_3\}\right).$$

For $n = 3$ Ash used the above cells to determine the top dimensional homology of congruence subgroups of $\mathrm{SL}_3(\mathbb{Z})$ and the corresponding Hecke action [2]. While one can extend his results to calculate any level of the homology, the Hecke action does not extend in an obvious way.

Clearly there is some work to be done to determine $\sigma(E)$ for a given set E. The Voronoi celluation will circumvent this problem and we can work directly with E.

2.2 Voronoi's Work on Perfect Forms

There is a complex discovered by Voronoi which is dual to the well-rounded retract and has recently gained importance in its recent use to calculate cohomology of congruence subgroups over extensions of $\mathbb{Q}$. This section examines some of Voronoi's work on perfect forms as discussed in [60] and shows how his work leads to a cell complex $\mathcal{R}$, which can be used to calculate the cohomology of congruence subgroups. We will continue with the notation from the previous section, but will often refer to $A, \phi \in C$ as quadratic forms.

Definition 2.13. A quadratic form $A \in C$ is perfect if $m(A)$ and $M(A)$ uniquely determine A.

Example 2.14. We let I_n, J be defined as in example 2.4.

i) I_n is not perfect since $m(I_n) = m(J_n) = 1$ and $M(I_n) = m(J_n) = \{e_1, \ldots, e_n\}$ where J_n is the $n \times n$ matrix containing J in the upper left corner, 1's down the remaining diagonal and 0 elsewhere.

ii) Let $H = \begin{pmatrix} 1 & 1/2 \\ 1/2 & 1 \end{pmatrix}$. Then $m(H) = 1$ and $M(H) = \{\pm e_1, \pm e_2, \pm(e_1 - e_2)\}$. H is perfect, since $e_i^T \begin{pmatrix} x & y \\ y & z \end{pmatrix} e_i = 1$ implies that $x = z = 1$ and $(e_1 - e_2)^T \begin{pmatrix} 1 & y \\ y & 1 \end{pmatrix} (e_1 - e_2) = 1 \Rightarrow y = \frac{1}{2}$.

For cohomology purposes we are interested in the number of classes of perfect forms under the action of $\mathrm{GL}_n(\mathbb{Z})$ since these will ultimately contribute to the number of cells in the Voronoi celluation. Voronoi classified all classes of perfect forms for $n \leq 5$ [60]. We will write N_{perf} for the number of equivalence classes of perfect quadratic forms under the action of $\mathrm{GL}_n(\mathbb{Z})$. Table 2.2 displays the value of N_{perf} for $n \leq 8$ as calculated by Voronoi for $n \leq 5$

[60], by Barnes for $n = 6$ [21], by Jaquet and Chiffelle for $n = 7$, and by Sikirić, Schürman, and Vallentin [55] for $n = 8$.

Table 2.1: Number of Classes of Perfect Forms for Dimensions 2-8

Dimension	2	3	4	5	6	7	8
N_{perf}	1	1	2	3	7	33	10916

The algorithms needed to compute the number of classes of perfect forms are quite interesting in of and themselves, see Schürmann [51] for more information. Following McConnell [45] we will use the minimal vectors of a perfect quadratic form to construct the cell complex $\mathcal{R}$. McConnell proves that $\mathcal{R}$ is dual to $\mathcal{W}$ and thus we can use $\mathcal{R}$ or $\mathcal{W}$ to calculate the cohomology of Γ.

Let $\mathcal{P} \subset \mathbb{Z}^n$ be the set of primitive points. Recall that $(x_1, \ldots, x_n) \in \mathbb{Z}^n$ is primitive if and only if $(x_1, \ldots, x_n) = 1$. For any $x \in \mathcal{P}$ we can construct a positive semi-definite form ω_x on $\mathbb{R}^n$ by setting

$$\omega_x(y) = \langle x, y \rangle^2,$$

for all $y \in \mathbb{R}^n$.

For a perfect form ϕ with $M(\phi) = \{x_1, \ldots, x_n\}$ define

$$R_\phi = \left\{ \theta \in C : \theta = \sum_{i=1}^{s} \rho_i \omega_{x_i}, \rho \in \mathbb{R}^+ \cup \{0\} \right\}.$$

By construction R_ϕ is a convex cone in C', the space of all quadratic forms, with finitely many faces. The dimension of R_ϕ is equal to the dimension of C which is $\frac{n^2+n}{2}$. The interior of R_ϕ lies entirely in C although its boundary will include some positive semi-definite forms.

Then we can construct a cell complex by taking cells to be subsets $F \subset C$ such that F corresponds to the interior of some face Z of R_ϕ for a perfect form ϕ. We call this cell complex the Voronoi celluation and denote it as $\mathcal{R}$. We are interested in $\mathcal{R}$ for two reasons.

First we have the following theorem of McConnell:

Theorem 2.15. (**McConnell** [45].) *Given $S \in \mathcal{W}$, there is a unique $F \in \mathcal{R}$ such that the minimal vectors which determine the forms bounding F are exactly the minimal vectors that determine S. The map $S \to F$ is a canonical bijection $\mathcal{W} \to \mathcal{R}$ which is inclusion-reversing on the face relations.*

The second reason we are interested in $\mathcal{R}$ is the existence of explicit algorithms to calculate $\mathcal{R}$ for low dimensional cells; this algorithm has been implemented most recently in the exciting work of Ash, Gunnells, and McConnell [12]. The steps needed to implement this algorithm are examined in the rest of this chapter. For convenience we will refer to the cells in $\mathcal{R}$ by the names of the corresponding cell in $\mathcal{W}$. We will also write $\sigma = \pm\{a_1, \ldots, a_n\}$ to represent the face σ whose vertices are the a_i. Essentially we will refer to a cell in either $\mathcal{W}$ or $\mathcal{R}$ by its minimal vectors; technically this means we are writing $\sigma(\pm\{e_1, e_2\}) = \{e_1, e_2\}$.

Example 2.16. For $n = 2$ there is one $\mathrm{SL}_2(\mathbb{Z})$-orbit of perfect quadratic forms. We will work with the form given by the matrix H from example 2.14. $M(H) = \underbrace{\pm\{e_1, e_2, e_1 - e_2\}}_{\sigma}$. Since we are inherited interesting in well-rounded forms, we want to restrict our attention to subsets of $M(H)$ that span $\mathbb{R}^2$. The following are the only subsets of $M(H)$ that span $\mathbb{R}^2$: $\underbrace{\pm\{e_1, e_2\}}_{\tau}, \underbrace{\pm\{e_1, e_1 - e_2\}}_{\tau_1}, \underbrace{\pm\{e_2, e_1 - e_2\}}_{\tau_2}$ all lie in the same $\mathrm{SL}_2(\mathbb{Z})$ orbit, namely

$$\begin{pmatrix} 1 & 1 \\ -1 & 0 \end{pmatrix} \cdot \tau = \tau_1 \text{ and } \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix} \cdot \tau = \tau_2.$$

Using the transformation matrix in example 2.11 i) we get the same decomposition of $\mathcal{W}$ into $\mathrm{SL}_2(\mathbb{Z})$-orbits as before.

Example 2.17. For $n = 3$ there is also one $\mathrm{SL}_2(\mathbb{Z})$-orbit of perfect quadratic forms given

by the form corresponding to the matrix

$$K = \begin{pmatrix} 1 & -1/2 & -1/2 \\ -1/2 & 1 & 0 \\ = 1/2 & 0 & 1 \end{pmatrix}.$$

We note that $M(K) = \pm\{e_1, e_2, e_3, e_1 + e_2, e_1 + e_3, e_1 + e_2 + e_3\}$ which corresponds to the top dimensional cell which is σ_0. The other cells are labeled in a similar way.
Thus $\tau_2 \sim \pm\{e_1, e_2, e_3, e_1 + e_2\}, \tau_3 \sim \pm\{e_1, e_2, e_3, e_1 + e_2 + e_3\}$, etc.

2.3 Calculating the Cohomology for $\Gamma_0(N)$

The methods above prove the existence of the well-rounded retract $\mathcal{W}$ and the Voronoi celluation $\mathcal{R}$ which can be used to compute the cohomology of $\mathrm{SL}_n(\mathbb{Z})$. We wish to implement these complexes to compute the cohomology of congruence subgroups $\Gamma_0(N) \subset \mathrm{SL}_n(\mathbb{Z})$. The following sections are primarily concerned with the calculation of cohomology for the congruence subgroup $\Gamma_0(N)$.

We follow the exposition from [12]. We write $\mathcal{W}_T$ for the $\mathrm{SL}_3(\mathbb{Z})$-orbit of the cell of type T in $\mathcal{W}$. Since we are dealing with congruence subgroups of $\mathrm{SL}_3(\mathbb{Z})$, we fix $n = 3$ in what follows. However, all of the calculations are analogous to those for $n = 4$ and indeed could potentially be modified to work for higher n.

Recall that there are five $\mathrm{SL}_3(\mathbb{Z})$-orbits of cells in the well-rounded retract for $n = 3$. In the Voronoi retract we are interested in the faces of the polyhedron formed from the minimal vectors of the equivalence class of the perfect ternary form. Thus under the correspondence between $\mathcal{W}$ and $\mathcal{R}$ we identify a cell with its minimal vectors. Without loss of generality we will identify the cell with its set of minimal vectors; although technically the cell is the set of $A \in \mathcal{X}$ such that $M(A)$ is the set of minimal vectors. For example σ_0 as defined in example 2.12 corresponds to the cell $\pm\{e_1, e_2, e_3\} = \sigma_0 \in \mathcal{R}$. Recall that there is only one orbit of top dimensional cells in $\mathcal{W}$ for $n = 3$. However, under the action of the subgroup

$\Gamma_0(N) \subset \mathrm{SL}_3(\mathbb{Z})$ the single orbit might decompose into multiple orbits.

To calculate the cohomology of $\Gamma_0(N)$ we need to understand how any $\mathrm{SL}_3(\mathbb{Z})$-orbit of cells in $\mathcal{W}$ decomposes into $\Gamma_0(N)$-orbits of cells. We first note that for a fixed $\mathrm{SL}_3(\mathbb{Z})$-orbit of cells $\mathcal{W}_T$ we have a correspondence between

$$\{\sigma \in W_T\} \leftrightarrow \{\gamma Stab(T) : \gamma \in \mathrm{SL}_3(\mathbb{Z})\}$$

given by $\sigma \to \gamma Stab(T)$ such that $\sigma = \gamma\sigma_T$. The above identification will allow us to determine how the $\mathrm{SL}_3(\mathbb{Z})$-orbits of cells break up into suborbits under the action of $\Gamma_0(N)$ by looking at $Stab(T)$-orbits of the finite projective space $\mathbb{P}^2(\mathbb{Z}/N\mathbb{Z})$. We define the space $\mathbb{P}^{n-1}(\mathbb{Z}/N\mathbb{Z})$ in the following section; in that section we also show give two lemmas that combined with the choice of orientations will allow us to actually calculate the cohomology of $\Gamma_0(N)$ using a spectral sequence.

2.4 $\mathbb{P}^{n-1}(\mathbb{Z}/N\mathbb{Z})$.

$\mathbb{P}^{n-1}(\mathbb{Z}/N\mathbb{Z})$ consists of primitive n-tuples modulo homotheties. Naturally, this space is defined as a set of equivalence classes of affine space. Recall that $\mathbb{A}^n(\mathbb{Z}/N\mathbb{Z})$ is the set of n-tuples with entries in $\mathbb{Z}/N\mathbb{Z}$. A tuple $(a_1, \ldots, a_n)$ is called *primitive* in $\mathbb{A}(\mathbb{Z}/N\mathbb{Z})$ if $\gcd(a_1, \ldots, a_n, N) = 1$. Let $\mathbb{A}^n(\mathbb{Z}/N\mathbb{Z})^\dagger$ be the subset of $\mathbb{A}^n(\mathbb{Z}/N\mathbb{Z})$ consisting of primitive tuples.

Definition 2.18. $\mathbb{P}^{n-1}(\mathbb{Z}/N\mathbb{Z})$ is $\mathbb{A}^n(\mathbb{Z}/N\mathbb{Z})^\dagger$ modulo *homotheties*, meaning two tuples $(a_1, \ldots, a_n), (\alpha_1, \ldots, \alpha_n) \in \mathbb{A}^n(\mathbb{Z}/N\mathbb{Z})^\dagger$ are in the same equivalence class in $\mathbb{P}^{n-1}(\mathbb{Z}/N\mathbb{Z})$ if and only if there is a $u \in U(\mathbb{Z}/N\mathbb{Z})$, that is to say u is a unit in $\mathbb{Z}/N\mathbb{Z}$, with

$$u(a_1, \ldots, a_n) = (\alpha_1, \ldots, \alpha_n).$$

Without loss of generality we will treat the entries of a tuple $(a) = (a_1, \ldots, a_n) \in \mathbb{A}^n(\mathbb{Z}/N\mathbb{Z})$ as integers that are reduced modulo N to their smallest nonnegative representa-

tive; this notation will not only save time, but also affords itself to computer implementation. We will also write $[a] \in \mathbb{P}^{n-1}(\mathbb{Z}/N\mathbb{Z})$ for the homotheties class of tuple a.

To aid in computations we will want to determine a canonical representative of each class in $\mathbb{P}^{n-1}(\mathbb{Z}/N\mathbb{Z})$. To do this we will use the natural lexicographical ordering of $\mathbb{A}^n(\mathbb{Z}/N\mathbb{Z})$. In this ordering we first scale a tuple to its smallest nonzero representative and then order by the standard ordering on the positive integers. If N is prime, then simply scaling the first nonzero entry to 1 determines a unique representative of every class; moreover this representative will be minimal with respect to the standard lexicographical ordering. We want to generalize these two properties for our choice of representative of a class in $\mathbb{P}^{n-1}(\mathbb{Z}/N\mathbb{Z})$ for a generic N. First we prove the following lemma.

Theorem 2.19. *Any class in $\mathbb{P}^{n-1}(\mathbb{Z}/N\mathbb{Z})$ contains a tuple $(a_1 \ldots, a_n)$, for which the smallest positive representative of the first nonzero a_i is a divisor of N.*

Proof. Let $(a_1, \ldots, a_n) \in \mathbb{A}^n$. Without loss of generality assume that a_i is the first nonzero entry in the tuple. Then if $(a_i, N) = d$ we can find nonzero integers r and s such that

$$a_i r + Ns = d.$$

Dividing by d we have

$$\frac{a_i}{d} r + \frac{N}{d} s = 1.$$

Thus we see that r is a unit modulo N/d. Then we can apply a theorem of Bass [41] which states that subrings of semilocal rings are small and hence r lifts to a unit modulo N.

Alternatively, consider the case that N is a prime power, say $N = p^n$. Then for any divisor d of N, we know the units of $\mathbb{Z}/(N/d)\mathbb{Z}$ are the elements coprime to p; these are clearly units in $\mathbb{Z}/N\mathbb{Z}$. Then since we can lift the units modulo each prime, the Chinese Remainder Theorem allows us to lift units in $\mathbb{Z}/d\mathbb{Z}$ to units of $\mathbb{Z}/N\mathbb{Z}$ for $d|N$.

Thus for any equivalence class in $\mathbb{P}^{n-1}(\mathbb{Z}/N\mathbb{Z})$ we can construct a representative $(a_1, \ldots, a_n)$ for each class such that the first nonzero a_i is a divisor of N. Since $(a_i, N) = d$, we have that

$d \leq a_i$; therefore a unit multiple of the tuple $(a_1, \ldots, a_n)$ such that the first nonzero entry is a divisor of N will be the minimal element of the equivalence class of $(a_1, \ldots, a_n)$. $\square$

For computational purposes we want to store an equivalence class $[a] \in \mathbb{P}^{n-1}$ by its minimal element: the element minimal with respect to the lexicographical ordering on $\mathbb{A}^n$ denoted $[a]_{min}$. As stated $[a]_{min}$ will have the form $(a_1, \ldots, a_n)$ where the first nonzero $a_i | N$. There are a total of

$$(d(N)-1)N^{n-1} + (d(N)-1)N^{n-2} + \cdots + (d(N)-1)N + d(N) - 1$$

tuples whose first nonzero entry is a divisor of N; here $d(N)$ is the number of divisors of N ($d(4) = 3$). If N is prime then every such tuple will determine a unique class in $\mathbb{P}^{n-1}$. However, when N is composite some of the tuples with this form will not be primitive and some will be collinear and hence determine the same class in $\mathbb{P}^{n-1}$.

In general, when N is not prime, there are some primitive tuples where the stabilizer of the first entry changes the following entries; see below for examples of non-reduced primitive tuples. In Appendix B we present a simple algorithm which first reduces a tuple and then acts by the stabilizer of the first nonzero element to produce a minimal representative of the input tuple. However, the simplicity of the complete ordering makes up for the need to remove redundant elements. For example using the entire list of tuples of the form $(a_1, \ldots, a_n)$ where the first nonzero $a_i | N$ allows us to assign an integer to every class in a canonical way. Thus we avoid the need for search tables and are able to save working memory by only storing integers. While formulas can be found for the number of classes in $\mathbb{P}^{n-1}(\mathbb{Z}/N\mathbb{Z})$ we found that the complexity of such enumeration algorithms outweighed the benefits. The tuple to integer conversion algorithm and code we constructed can be found in Appendix B.

Since we are storing projective tuples by their minimal element we must take caution that when acting on $[a]_{min}$ by elements of $\overline{\Gamma}$ we must reduce the output to preserve the minimal representation. Our reduction algorithm can also be found in Appendix B.

Example 2.20. The following tuples are in $\mathbb{A}^3$ for various N.

i) $(1,1,1)$ is a minimal representative of its class for all N.

ii) $(2,2,2)$ is not a minimal representative of its class for any N; if $2|N$ then the tuple is not primitive, otherwise it is not reduced.

iii) $(3,4,2)$ is not a minimal representative for $N = 6$, since $5 \cdot (3,4,2) = (3,2,4)$ is contained in the same class and $(3,2,4)$ comes before $(3,4,2)$ in our ordering.

The next two lemmas reveal how the set $\mathbb{P}^{n-1}(\mathbb{Z}/N\mathbb{Z})$ will allow us to calculate the cohomology of $\Gamma_0(N)$.

Lemma 2.21. *The bottom row map $\mathfrak{b} : SL_n(\mathbb{Z}) \to \mathbb{P}^{n-1}(\mathbb{Z}/N\mathbb{Z})$ gives a bijection between $\Gamma_0(N)\backslash SL_n(\mathbb{Z})$ and $\mathbb{P}^{n-1}$. Moreover, this map respects the action of $SL_n(\mathbb{Z})$ on the right.*

Proof. From the rules of determinants it is clear that the image of $\mathfrak{b}$ will consist of primitive tuples and thus maps into $\mathbb{P}^{n-1}(\mathbb{Z}/N\mathbb{Z})$. Moreover, any class in $\mathbb{P}^{n-1}(\mathbb{Z}/N\mathbb{Z})$ has a primitive representation, which is the bottom row of some matrix in $\mathrm{SL}_n(\mathbb{Z})$; in section 5.3 we produce an algorithm which outputs such a matrix for $n = 3$ using a constructive argument due to Schenkman [49].

Clearly the only matrices sent to the class containing $(0,0,\ldots,0,1)$ are those in $\Gamma_0(N)$. Moreover working modulo N it is apparent that the left cosets of $\Gamma_0(N)$ will have the same image under the bottom row map. Thus, we have an injective and surjective map from $\Gamma_0(N)\backslash\mathrm{SL}_n(\mathbb{Z})$ to $\mathbb{P}^{n-1}(\mathbb{Z}/N\mathbb{Z})$.

The action of $\mathrm{SL}_n(\mathbb{Z})$ is respected since the action of $\mathrm{SL}_n(\mathbb{Z})$ on $\mathbb{P}^{n-1}(\mathbb{Z}/N\mathbb{Z})$ can be thought of a multiplying a row vector on the right by a matrix. □

Lemma 2.22. *The set of $\Gamma_0(N)$-orbits of cells in $\mathcal{W}_T$ correspond to the orbits of the right action of $Stab(T)$-action on $\mathbb{P}^{n-1}$.*

Proof. We note that $\Gamma_0(N)\backslash\mathcal{W}_T = \Gamma\backslash\mathrm{SL}_n(\mathbb{Z})/Stab(T)$ from the previous section. Then by lemma 2.21 we have that $\Gamma_0(N)\backslash\mathrm{SL}_n(\mathbb{Z})/Stab(T) \cong \mathbb{P}^{n-1}(\mathbb{Z}/N\mathbb{Z})/\mathrm{Stab}(T)$. □

Hence we have reduced the problem of how the $\mathrm{SL}_n(\mathbb{Z})$-orbits decompose when we restrict the action to $\Gamma_0(N)$ into a problem about $Stab(T)$-orbits of $\mathbb{P}^{n-1}(\mathbb{Z}/N\mathbb{Z})$. In order to calculate the orbits of the stabilizer we must first calculate the stabilizers for the standard cells in the well-rounded retract for $n = 3$. We are primarily interested in the cohomology one below the top dimension; consequently, we need only consider the cells with dimensions $1, 2$, and 3.

2.5 Calculating the Stabilizers

We briefly recall our decision to identify a cell with its minimal vectors; this will actually make the action of the stabilizer easier since $\gamma \cdot \sigma(E) = \gamma^T \sigma(E)\gamma = \sigma(\gamma E)$. Thus by considering our cells as subsets of $\mathbb{Z}^3$ the action of $\mathrm{SL}_3(\mathbb{Z})$ becomes matrix multiplication. Elements of $\mathbb{Z}^3$ can be acted on the left or on the right by 3×3 matrices depending on whether we are viewing the vector as a column or a row. For our purposes we want to view the cells as columns and thus the stabilizers listed below are for the left action of $\mathrm{SL}_3(\mathbb{Z})$. One way to work with a given cell would be to consider it as the columns of a $3 \times 2k$ matrix modulo permutation of the columns, where k is determined by the dimension of the cell. Given a fixed ordering of the elements of a cell, each element of the stabilizer will possibly reorder the elements. The orientation character on the stabilizer of a given cell (discussed in section 2.6), is the sign of the permutation action on the rays constituting the elements of the cell.

For $n = 3$ the cells are rather small and can be found after theorem 2.10, recall that we let e_1, e_2, e_3 be the standard unit vectors and $t_1 = e_1 + e_2, t_3 = e_1 + e_2 + e_3$. Hence $\sigma_0 = \pm\{e_1, e_2, e_3\}$, $\tau_2 = \pm\{e_1, e_2, e_3, t_2\}$, $\tau_3 = \pm\{e_1, e_2, e_3, t_3\}$ and $\omega_2 = \pm\{e_1, e_2, e_3, t_2, t_3\}$.

Lemma 2.23. *To calculate the stabilizer of a cell that contains the standard unit vectors, it suffices to utilize a finite computer search through all matrices whose column sets are a subset of the given cell.*

Proof. This suffices, since an element of $\mathrm{GL}_3(\mathbb{Z})$ that stabilizes a cell σ containing the stan-

dard unit vectors must take the unit vectors to elements of the cell. Thus, the columns must be elements of the cell. □

Because all of our cells contain the standard unit vectors, this observation motivated a simple computer program that searches through all possible matrices containing the elements of a given cell, computes the determinant, and checks that the image is exactly the desired cell; the program can be found in Appendix E.1. We ran our program once and then stored the data in a text file for future reference. The cohomology calculations simply read the appropriate stabilizers from the text file. A hard copy of the data can be found in Appendix A. As part of the data we hand computed the orientations of the matrices.

The stabilizer program mentioned above produced an output that agrees with a proof by Ash in [2] that the stabilizer of σ_0 is the group of order 24 generated by

$$Stab(\sigma_0) = \left\langle \underbrace{\begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{h_1}, \underbrace{\begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}}_{h_2} \right\rangle.$$

A quick calculation in Magma [24] shows us that this group is isomorphic to S_4, the symmetric group of order 24. We used Magma to identify the output of the stabilizer program for each cell. For the other standard cells we found that

$$Stab(\tau_2) = \left\langle \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{pmatrix}, \begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \right\rangle,$$

has order 12 and is isomorphic to D_{12}, the dihedral group of order 12.

$$Stab(\tau_3) = \left\langle \begin{pmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & -1 \end{pmatrix}, \begin{pmatrix} 0 & 0 & -1 \\ 0 & -1 & 0 \\ -1 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 1 & -1 & 0 \\ 1 & 0 & -1 \\ 1 & 0 & 0 \end{pmatrix} \right\rangle,$$

has order 24 and is isomorphic to S_4.

$$Stab(\omega_2) = \left\langle \begin{pmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & -1 \end{pmatrix}, \begin{pmatrix} -1 & 0 & 0 \\ -1 & 0 & 1 \\ -1 & 1 & 0 \end{pmatrix} \right\rangle,$$

has order 8 and is isomorphic to D_8.

In Appendix A we record the stabilizers for the above cells for $n = 3$.

2.6 Orientation of the Orbits

Part of our calculations for cohomology will depend on tracking the orientation of each our cells and the orientations on the suborbits of $\mathbb{P}^{n-1}(\mathbb{Z}/N\mathbb{Z})$ under the stabilizer. The orientation of a cell can be thought of as a unique ordering of the vertices, or as part of an assignment of direction to the complex.

To calculate the orientation of our suborbits we will need to construct the values of an orientation character on the stabilizer of each of our cells. Since for $n \leq 4$ we have that $\mathcal{W}$ is simplical, meaning all the cells are simplical, the orientation character can be calculated using the action of the stabilizer on the rays of a cell. To compute this action we first note that all of our cells are invariant under multiplication by -1. For convenience we let $\tilde{\sigma}$ denote the set of rays of the cell σ and will use a boldface font to symbolize the ray spanned by an element of a cell. Thus for the cell

$$\sigma_0 = \pm\{e_1, e_2, e_3\}$$

we see that

$$\tilde{\sigma_0} = \{\mathbf{e_1}, \mathbf{e_2}, \mathbf{e_3}\}.$$

Recall that

$$Stab(\sigma_0) = \left\langle \underbrace{\begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{h_1}, \underbrace{\begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}}_{h_2} \right\rangle.$$

To compute the orientation character ρ on a given matrix we note that h_1 acts on the ordered set $\tilde{\sigma_0}$ as the permutation $(\mathbf{e_1}, \mathbf{e_2})$; so $\rho(h_1) = -1$. Similarly h_2 acts as the permutation $(\mathbf{e_1}, \mathbf{e_2}, \mathbf{e_3})$ and we see that $\rho(h_2) = 1$. Given the list of elements of each stabilizer we then hand computed the orientation character for each element. We then crosschecked this list with standard facts about conjugacy classes to verify accuracy.

At this point we have assigned an orientation to each element of the stabilizer of a given cell. We now examine how this orientation affects which suborbits contribute to the cohomology.

Definition 2.24. An $Stab(\sigma)$-orbit of $\mathbb{P}^{n-1}(\mathbb{Z}/N\mathbb{Z})$ $\mathcal{O}$ is *non-orientable* if for some element $[a] = [a_1, \ldots, a_n] \in \mathcal{O}$, there is a $\gamma \in Stab(\sigma)$ with $\rho(\gamma) = -1$ such that $[a]\gamma = [a]$. Otherwise the orbit is *orientable*.

The following lemma reveals that non-orientable and orientable orbits are fairly easy to identify.

Lemma 2.25. *If an $Stab(\sigma)$-orbit of $\mathbb{P}^{n-1}(\mathbb{Z}/N\mathbb{Z})$ $\mathcal{O}$ is non-orientable then for every element $[b] \in \mathcal{O}$ there is $\gamma_{[b]} \in Stab(\sigma)$ such that $\rho(\gamma) = -1$ and $[b]\gamma = [b]$.*

Proof. Since $\mathcal{O}$ is non-orientable we can find an $[a]$ and a $\gamma \in Stab(\sigma)$ such that $\rho(\gamma) = -1$ and $[a]\gamma = [a]$. Then for any $[b] \in \mathcal{O}$ there is a $\gamma' \in Stab(\sigma)$ such that $[a]\gamma' = [b]$. If $\rho(\gamma') = 1$ then

$$[b]\gamma'^{-1}\gamma\gamma' = [a]\gamma\gamma' = [b]\gamma' = [b].$$

Since $\rho(\gamma'^{-1}\gamma\gamma') = \rho(\gamma) = -1$ we have proven the lemma. □

Thus the above lemma shows that an orbit is orientable if and only if every element of the orbit is only stabilized by orientation preserving elements of the stabilizer, which is the case if and only if any single element of the orbit is only stabilized by orientation preserving elements of the stabilizer. Hence for computation purposes we can determine if an orbit is orientable by examining a single element of the orbit.

As we will see in section 2.28, it will help to assign an orientation number of either $-1, 0,$ or 1 to every element of an orbit. If an orbit is non-orientable then every element will have orientation number 0. For an orientable orbit we will assign the smallest element of the orbit (under our ordering of $\mathbb{P}^{n-1}(\mathbb{Z}/N\mathbb{Z})$) the orientation number 1. Then other elements of the orbit $[b]$ will have an orientation number of 1 or -1 depending on the value of the orientation of γ where $[a]\gamma = [b]$.

Lemma 2.26. *The orientation number of* $[b]$ *in an orientable* $Stab(\sigma)$*-orbit of* $\mathbb{P}^{n-1}(\mathbb{Z}/N\mathbb{Z})$ *is well-defined.*

Proof. Suppose by way of contradiction that we have $\gamma, \gamma' \in Stab(\sigma)$ such that $[a]\gamma = [a]\gamma' = [b]$ where $\mathbf{a}$ is the smallest element of the orbit. Then $[a]\gamma\gamma'^{-1} = [a]$. Since the orbit is orientable we have $\rho(\gamma\gamma'^{-1}) = 1$. Hence $\rho(\gamma) = \rho(\gamma')$. □

The orientation number will be important when we calculate the boundary maps of our spectral sequence in section 2.8.

Thus for a given $[a] \in \mathbb{P}^{n-1}(\mathbb{Z}/N\mathbb{Z})$, we have attached some additional information. Attached to $[a]$ we have its primitive representation $(a_1, \ldots, a_n)$, an orientation (which is either plus or minus 1), and a matrix which takes the first element of the orbit containing $[a]$ to $[a]$. Appendix D illustrates this in great detail.

2.7 Orbit Implementation Details

Using the ordering mentioned above, we can calculate the orbits of $\mathbb{P}^{n-1}(\mathbb{Z}/N\mathbb{Z})$ under the action of the stabilizer of each of the corresponding cells. As part of our orbit information

we will also determine which orbits are orientable and for orientable orbits we will associate an orientation value to each class within the orbit.

To calculate the orbits, we create a boolean vector with $(d(N)-1)N^{n-1}+(d(N)-1)N^{n-2}+\cdots+(d(N)-1)N+1$ entries, where $d(N)$ is the number of divisors of N. While creating the vector we flag any non-minimal entries. In this way upon completion we will have a one-to-one correspondence with the classes of $\mathbb{P}^{n-1}(\mathbb{Z}/N\mathbb{Z})$ and unflagged entries. Then we run a program which takes the first unflagged entry and calculates its stabilizer within the stabilizer of the appropriate cell; if any element with negative orientation stabilizes the entry, the orbit is non-orientable. The program then calculates the orbit under the stabilizer of the cell and stores the appropriate orientation numbers for each element in a global vector of integers. If the orbit is orientable, the cell is stored. Otherwise the data is deleted. Regardless all the elements in the orbits are flagged. The program loops until every class of $\mathbb{P}^{n-1}(\mathbb{Z}/N\mathbb{Z})$ is in some orbit.

All the code needed for orbit calculations can be found in Appendix E.2.

2.8 Using the Orbit Information to Calculate Cohomology

We use the orientable orbits to calculate the cohomology $H^2(\Gamma_0(N);\mathbb{C})$ for $\Gamma_0(N)\subset \mathrm{SL}_3(\mathbb{Z})$ as done by Ash, Gunnells, and McConnell for $\Gamma_0(N)\subset \mathrm{SL}_4(\mathbb{Z})$ in [12], who follow the work of Brown [25]; this is done by using a spectral sequence.

We let $Stab(\sigma)_N = Stab(\sigma)\cap\Gamma_0(N)$. Then we let $\mathbb{C}_\sigma$ be $\mathbb{C}$ treated as a $Stab(\sigma)_N$-module where the action is given by multiplication by ± 1 depending on the value of the orientation character. Then the $E_1^{i,j}$ terms of our spectral sequence are

$$E_1^{i,j} = \oplus_{\sigma\in\mathcal{W}_i} H^j(\Gamma_\sigma;\mathbb{C}_\sigma),$$

where $\mathcal{W}_i$ is the set of $Stab(\sigma)$-orbits of $\mathbb{P}^{n-1}(\mathbb{Z}/N\mathbb{Z})$ (or equivalently cells corresponding to the action of $\Gamma_0(N)$) of each i dimensional cell of $\mathcal{W}$.

We recall two facts from Ash,Gunnells, and McConnell [12] that are applications of theorems found in Brown [25]:

- Since $\mathbb{C}$ has characteristic 0, all the terms vanish when $j \neq 0$. Thus the spectral sequence will degenerate at E_1.
- Any non-orientable orbit will have $H^0(\Gamma_o; \mathbb{C}_o) = 0$.

Equivalently we have the following theorem

Theorem 2.27. *The $E_1^{i,0}$ term of the equivariant cohomology spectral sequence for $H^i_\Gamma(\mathcal{W}; \mathbb{C})$ is a direct sum $\oplus_o \mathbb{C}_o$ where o runs through a set of i-dimensional cells in 1:1 correspondence with the orientable $Stab(\sigma)$-orbits of $\mathbb{P}^{n-1}(\mathbb{Z}/N\mathbb{Z})$, for all types σ of cells of dimension i.*

Thus, given the boundary maps of the above spectral sequence we can calculate the cohomology through the standard refinement of the kernel modulo the image.

In order to calculate the boundary map $d_1^{i,0} : E_1^{i,0} \to E_1^{i+1,0}$ we need to compute how the cells/orbits of dimension i glue together to make a cell/orbit of dimension $i+1$. The calculation of the boundary maps of the spectral sequence is very technical and we will refer to Brown for a full account [25] or to Ash, Gunnells, and McConnell [12] for an account specific to $\mathrm{SL}_n(\mathbb{Z})$. For our purposes

$$d_1^{i,0} : \oplus_{\tau \in W_i} H^0(Stab(\tau); \mathbb{C}_\tau) \to \oplus_{\sigma \in W_{i+1}} H^0(Stab(\sigma); \mathbb{C}_\sigma$$

will be a sum of terms, $d_{(\sigma,\tau')}$ where τ' is in the boundary of σ and we assume that τ' and τ are in the same Γ-orbit. To do this we will need to calculate the boundaries of a given orbit.

We did this by hand for each of the standard cells and Appendix A.7 contains the appropriate data. Our data also agrees with the following theorem of Ash, Gunnells, and McConnell.

Theorem 2.28. *Given a cell of σ of type T, meaning σ is in the $SL_3(\mathbb{Z}$-orbit of T, containing a cell of type T' in its boundary. Then by decomposing $\mathcal{O}$, the $Stab(T)$-orbit of $\mathbb{P}^2(\mathbb{Z}/N\mathbb{Z})$*

which corresponds to σ into its suborbits $\mathcal{O}_1, \ldots, \mathcal{O}_k$ under the group $Stab(T) \cap Stab(T')$, the $\gamma_j \in SL_3(\mathbb{Z})$ such that $\mathfrak{b}(\gamma_j) = [a_j] \in \mathcal{O}_j$ will be the translates of T' in the boundary of σ.

As a further check on the accuracy of our results we found that our results concurred with those presented in the Soulé cube [57].

For implementation purposes we hard coded the intersection of the stabilizers into the stabilizer data. The interested reader can find that data in Appendix A.6. Using the data from Theorem 2.28 we can calculate $d_{(\sigma,\tau')}$ as the composition $t_{\sigma\tau'}u_{\sigma\tau'}\nu_{\tau'}$ where $t_{\sigma\tau'}, u_{\sigma\tau'}, \nu_{\tau'}$ are defined below.

The map $t_{\sigma\tau'} : H^0(Stab(\sigma)_N \cap Stab(\tau')_N; \mathbb{C}_\sigma) \longrightarrow H^0(Stab(\sigma)_N; \mathbb{C}_\sigma)$ is the transfer map $\mathbb{C} \to \mathbb{C}$ which acts as multiplication by the integer $[Stab(\sigma) : Stab(\sigma) \cap Stab(\tau')]$. Since we can compute both of the finite groups above we can compute $t_{\sigma\tau'}$.

The map $u_{\sigma\tau'} : H^0(Stab(\tau')_N; \mathbb{C}_{\tau'}) \longrightarrow H^0(Stab(\sigma)_N \cap Stab(\tau')_N; \mathbb{C}_\sigma)$ is ± 1 depending on the orientation of τ' induced by the orientation of σ. We will write this induced orientation, which is either ± 1, as $[\sigma : \tau]$. To evaluate this number we use the following formula proposition derived from the work of [12] which reduces the problem to determining how the standard representatives of the $\mathrm{SL}_3(\mathbb{Z})$ orbits induce orientations in one another.

Proposition 2.29. *When $n = 3$ we have that*

$$[\sigma : \tau] = \mathcal{O}_T(\mathbf{b}(\gamma_0\hat{\gamma}))\mathcal{O}_T(\mathbf{b}(\gamma_0\hat{\gamma}\alpha)) \cdot [\sigma_T : (\tau_T)].$$

Hence Proposition 2.29 allows us to compute $[\sigma : \tau]$ for $n = 3$ from knowledge of how $[\sigma : \tau_2], [\sigma_0 : \tau_3]$, $[\tau_2 : \omega_2]$, and $[\tau_3 : \omega_2]$. We computed these numbers by hand and found that $[\tau_2 : \omega_2] = -1$ and all the other values are 1. Our computations agree with those of McConnell [45].

The map $\nu_{\tau'} : H^0(Stab(\tau)_N; \mathbb{C}_\tau) \longrightarrow H^0(Stab(\tau')_N; \mathbb{C}_{\tau'})$ will be the identity as shown in [12].

We then sum over all possible pairs (σ, τ') to get matrices which represent the boundary

action. To compute the cohomology we need to compute the kernel modulo the boundary. Upon row reducing these matrices we have a chain in the cell complex which represents a cohomology class. We are now left with determining how the Hecke operators should act on such a chain.

At this point the reader might gain some insight by viewing the example that occupies the entirety of chapter 6.

Chapter 3. The Sharbly Complex

The following chapter defines the Sharbly complex and examines some of its basic properties. The Sharbly complex is yet another complex which can be used to calculate the cohomology of $\mathrm{SL}_n(\mathbb{Z})$ and consequently can be used to calculate the cohomology of $\Gamma_0(N) \subset \mathrm{SL}_n(\mathbb{Z})$ via Shapiro's Lemma [61].

3.1 Definition

The following definition is the motivation for the Sharbly complex.

Definition 3.1. The Steinberg module $\mathrm{St}(n)$ is the $\mathbb{Z}\Gamma$-module $H^\nu(\Gamma; \mathbb{Z}\Gamma)$.

It is a theorem of Ash [4] that $\mathrm{St}(n)$ is isomorphic to the quotient of the $\mathbb{Z}\Gamma$ module of formal $\mathbb{Z}$-linear combinations of the elements $[v_1, \ldots, v_n]$, where $v_i \in \mathbb{Q}^n \backslash \{0\}$, quotiented by the submodule generated by the following relations:

i) If τ is a permutation on n letters and $\mathrm{sgn}(\tau)$ its sign as a permutation, then

$$[v_1, \ldots, v_n] = \mathrm{sgn}(\tau)[v_{\tau(1)}, \ldots, v_{\tau(n)}].$$

ii) If $q \in \mathbb{Q} \backslash \{0\}$ then

$$[qv_1, \ldots, v_n] = [v_1, \ldots, v_n].$$

iii) If the rank of the matrix $(v_1, \ldots, v_n)$ is less than n, then $[v_1, \ldots, v_n] = 0$.

iv) If $v_0, \ldots, v_n$ are nonzero points in $\mathbb{Q}^n$, then

$$\sum_i (-1)^i \sum_{i=1}^{n} (-1)^i [v_1, \ldots, \hat{v}_i, \ldots, v_n],$$

where $\hat{v}_i$ means omit that term.

The Steinberg module is an example of a dualizing module as defined by Brown [25]. As shown by Ash, Gunnells, and McConnell [16], $H^*(\Gamma; \mathbb{C})$ may be computed by computing the homology of a $\mathbb{Z}\Gamma$-free resolution of $St(n) \otimes \mathbb{Z}$. The Sharbly complex, defined below, is such a resolution.

The Sharbly complex was named by Ash in honor of Szczarba and Lee, two mathematicians who studied the cohomology of $\mathrm{SL}_n(\mathbb{Z})$ [42] and first used the Sharbly complex. The name is a phonetic abbreviation for the Szczarba Lee complex. Our definition is based on Ash [4].

Definition 3.2. The *Sharbly complex* is the chain complex $\{S_*, \partial\}$ defined as follows:

1) For $k \geq 0$, S_k is the module of formal $\mathbb{Z}$-linear combinations of basis elements $\mathbf{v} = [v_1, \ldots, v_{n+k}]$, where each $v_i \in \mathbb{Q}^n \backslash \{0\}$, modulo the relations:

   i) If τ is a permutation on (n_k) letters and $\mathrm{sgn}(\tau)$ its sign, then

   $$[v_1, \ldots, v_{n+k}] = \mathrm{sgn}(\tau)[v_{\tau(1)}, \ldots, v_{\tau(n+k)}].$$

   ii) If $q \in \mathbb{Q} \backslash \{0\}$ then

   $$[qv_1, \ldots, v_{n+k}] = [v_1, \ldots, v_{n+k}].$$

   iii) If the rank of the matrix $(v_1, \ldots, v_{n+k})$ is less than n, then $\mathbf{v} = 0$.

2) For $k \geq 1$ the boundary map $\partial : S_k \to S_{k-1}$ is

$$[v_1, \ldots, v_{n+k}] \longrightarrow \sum_{i=1}^{n+k} (-1)^i [v_1, \ldots, \hat{v_i}, \ldots, v_{n+k}],$$

where $\hat{v_i}$ means that term is omitted. For $k = 0$, the boundary map is the 0-map.

Elements of S_k are called k-sharblies; 0-sharblies are also known as modular symbols. A k-sharbly $\mathbf{v}$ can be considered as a collection of $n + k$ column vectors, subject to the above relations. We will refer to a collection of n of these vectors as a submodular symbol $\mathbf{u}$ of $\mathbf{v}$.

We note that S_* is a complex of $\mathbb{Z}\left[\mathrm{GL}_n(\mathbb{Q})\right]$-modules and hence $\mathbb{Z}\Gamma$-modules in the obvious way: an element of $\mathrm{GL}_n(\mathbb{Q})$ acts on a single k-sharbly as matrix multiplication on the left and the action extends linearly. It is worth noting that conditions i) and ii) imply that if $\mathbf{u}$ has any repeating columns, then $\mathbf{u} = 0$. The boundary of a k-sharbly can often include many $k-1$-sharblies with rank less than $k-1$, hence the boundary of a non-zero k-sharbly can include the zero sharbly.

Example 3.3. i) Let $\mathbf{v} = \begin{bmatrix} 2 & 2 & 3 \\ 4 & 5 & 6 \\ 10 & -1 & -2 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 5 & 6 \\ 5 & -1 & 2 \end{bmatrix}$.

However, $\mathbf{v'} = \begin{bmatrix} 2 & 4 & 10 \\ 2 & 5 & -1 \\ 3 & 6 & 2 \end{bmatrix} \neq \begin{bmatrix} 1 & 2 & 5 \\ 2 & 5 & -1 \\ 3 & 6 & 2 \end{bmatrix}$ since we are not allowed to modify the rows in any way.

ii) Let $\tau_{\mathbf{2}} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$. The reason for calling this 1-sharbly $\tau_{\mathbf{2}}$ will become apparent

later. We find that

$$\partial\tau_{\mathbf{2}} = -\begin{bmatrix}0&0&0\\1&0&1\\0&1&0\end{bmatrix} + \begin{bmatrix}1&0&1\\0&0&1\\0&1&0\end{bmatrix} - \underbrace{\begin{bmatrix}1&0&1\\0&1&1\\0&0&0\end{bmatrix}}_{\mathbf{0}} + \begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}.$$

From a computational perspective, the Sharbly complex mimics the utility of modular symbols in enabling the computations of the Hecke action on cohomology. We let $(S_k)_\Gamma$ be the module of Γ-coinvariants, or the quotient of S_k by the module generated by all elements of the form $\gamma \cdot \mathbf{u} - \mathbf{u}$ for $\gamma \in \Gamma$ and $\mathbf{u} \in S_k$.

We can use Borel-Serre duality [23] to establish an isomorphism between $H^k(\Gamma;\mathbb{Z})$ and $H_{\nu-k}((S_*)_\Gamma)$, where ν is the top dimension of the cohomology. Since $\{S_*, \delta\}$ is a $\mathbb{Z}\Gamma$-free resolution of $\mathrm{St}(n)$ we can use the homology of the Sharbly complex to calculate the needed cohomology. Explicit constructions of this isomorphism are difficult to come by, but justify the utility of the Sharbly complex for determining the action of the Hecke operations.

3.2 The Norm of a Sharbly

In order to use the Sharbly complex for our desired cohomology calculations we need to associate to each k-sharbly a norm; this norm will generalize the norm of a modular symbol which we recall below.

Definition 3.4. For a modular symbol

$$\mathbf{v} = [v_1, \ldots, v_n]$$

we define

$$||\mathbf{v}|| = |\det[u_1, \ldots, u_n]|$$

where $u = [u_1, \ldots, u_n]$ is in the class $\mathbf{v}$ and $u_i \in \mathbb{Z}^n \backslash \{0\}$ is primitive.

The definition begs the following observation:

Lemma 3.5. *For a modular symbol* $\mathbf{v}$ *there is a representative* u *of the class* $\mathbf{v}$*, such that* u *is integral with primitive columns. Moreover* u *is unique up to permutation and change of signs. Such a representative is called a lift of* $\mathbf{v}$*.*

Proof. Let $u = (u_1, \ldots, u_n)$ be a representation of the modular symbol $\mathbf{v}$. Clearing denominators of each column of u will make u integral. Dividing each column by its common divisor will make u primitive. Up to permutation and signs these actions are the only relations that determine the class $\mathbf{v}$; hence, the representation u is unique up to permutation and change of signs. □

We now return to k-sharblies. Recall that we can associate a set of $n+k$ primitive column vectors to the primitive representation of the k-sharbly $\mathbf{u}$. Then we call a modular symbol $\mathbf{v}$ a subsymbol of $\mathbf{u}$ if all the columns of the primitive representation of $\mathbf{u}$ appear as columns of the primitive representation of $\mathbf{v}$. For a nonzero k-sharbly $\mathbf{u}$ we define $Z(\mathbf{u})$ to be the set of all modular symbols that appear as a subsymbol of $\mathbf{u}$. Then

Definition 3.6.

$$||\mathbf{u}|| = \max \{||\mathbf{v}|| : \mathbf{v} \in Z(\mathbf{u})\}.$$

The last part of the following example shows why the norm of 0 is not defined above.

Example 3.7. Part i) builds upon Example 3.3

i) Let $\mathbf{v} = \begin{bmatrix} 2 & 2 & 3 \\ 4 & 5 & 6 \\ 10 & -1 & -2 \end{bmatrix}$ then $||\mathbf{v}|| = \left\| \begin{bmatrix} 1 & 2 & 3 \\ 2 & 5 & 6 \\ 5 & -1 & 2 \end{bmatrix} \right\| = 13$. If we let $\mathbf{v}' = \begin{bmatrix} 2 & 4 & 10 \\ 2 & 5 & -1 \\ 3 & 6 & 2 \end{bmatrix}$ we have $||\mathbf{v}'|| = 26$.

ii) Let $\mathbf{u} = \begin{bmatrix} 2 & 3 & -3 \\ 1 & 2 & -4 \end{bmatrix}$. Then

$$||\mathbf{u}|| = \max\left\{ \left\| \begin{bmatrix} 2 & 3 \\ 1 & 2 \end{bmatrix} \right\|, \left\| \begin{bmatrix} 2 & -3 \\ 1 & -4 \end{bmatrix} \right\|, \left\| \begin{bmatrix} 3 & -3 \\ 2 & -4 \end{bmatrix} \right\| \right\} = \max\{1, 5, 6\} = 6.$$

iii) This example shows why the norm of 0 is particularly pesky. Let $\mathbf{v} = \mathbf{0} = \begin{bmatrix} 20 & 50 & 20 \\ -101 & 223 & -101 \end{bmatrix}$. Then the subsymbols $\begin{bmatrix} 20 & 50 \\ -101 & 223 \end{bmatrix}, \begin{bmatrix} 20 & 20 \\ -101 & -101 \end{bmatrix}, \begin{bmatrix} 50 & 20 \\ 223 & -101 \end{bmatrix}$ have norms $9510, 0,$ and 9510 respectively. For computational purposes we must check that none of the columns of a given k-sharbly repeat themselves.

We likewise define the norm of a 1-Sharbly chain

$$\xi = \sum n(\mathbf{u})\mathbf{u},$$

where almost all $n(\mathbf{u}) = 0$ as follows. Let the support of ξ be $\text{supp}(\xi) = \{\mathbf{u} : n(\mathbf{u}) \neq 0\}\backslash\{0\}$. We abuse notation and let $Z(\xi)$ be the set of all modular symbols that appear as a submodular symbol of any $\mathbf{u} \in \text{supp}(\xi)$.

Definition 3.8. With notation as above, define

$$||\xi|| = \max\{||\mathbf{v}|| : \mathbf{v} \in Z(\xi)\}.$$

This is well-defined on all nonzero chains. A modular symbol $\mathbf{v}$ is *unimodular* if $||\mathbf{v}|| = 1$. This definition extends to k-sharblies in the obvious way: a k-sharbly $\mathbf{u}$ is *unimodular* if

$||\mathbf{u}|| = 1$. For a nonzero 1-sharbly this is equivalent to all submodular symbols having norm at most 1, since to be nonzero at least one submodular symbol has positive norm.

We present another definition which, while defined using the norm of a sharbly, is only valid for $n \leq 4$.

Definition 3.9. A 0-sharbly for any n or a 1-sharbly for $n \leq 4$ is called *reduced* if it has norm 1.

While this definition can be made regardless of the value of n and extended to any k-sharbly there is an implicit assumption that the term *reduced* can be used in the reduction calculations described in chapter 6.

3.3 Implementation of the Sharbly Complex

Since k-sharblies correspond to equivalence classes of ordered sets of column vectors, we can identify them with matrices. However to represent the class for computations we find ourselves with the same problem we encountered for projective space. In order to do computations we need a canonical representative of each class or we must accept that multiple entries could correspond to the same class. Since sharbly reduction algorithms are computationally expensive, combining like terms in a canonical representative could shorten running time. As before we will want to define a lexicographical ordering on the Sharbly complex and use this ordering to determine a canonical representative of a given sharbly.

To define an ordering on the Sharbly complex we first use the natural ordering on column vectors. Namely,

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \leqq \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

if and only if $x_1 < y_1$ or $x_i = y_i$ for $i = 1, \ldots, j$ for some $j \leqq n$ maximal with respect to this property and if $j < n$ then $x_{j+1} < y_{j+1}$. We extend this ordering to k-sharblies by reordering

the columns in increasing order and requiring the first nonzero entry of each column to be positive. Finally, any nonzero k-sharbly is less than a nonzero $k+1$-sharbly. For two sharbly chains, we first order the sharblies in the individual chains and then extend our ordering lexicographically. That is to say that two chains are compared term-wise starting with the smallest element of each chain.

We claim the ordering on individual k-sharblies allows us to pick a natural representative of the class as follows.

Lemma 3.10. *A k-sharbly $\mathbf{v}$ has a unique minimal primitive integral representation, also called a lift of $\mathbf{v}$.*

Proof. We first rationalize and divide by common divisors to make a primitive representation, which will be unique up to permutation and sign change. By multiplying by -1 if necessary we can assume that the first nonzero entry in every column will be positive. We then order the columns and thus obtain a unique minimal primitive integral representation. □

For implementation purposes we will associate $\mathbf{v}$ with the corresponding minimal primitive integral representation. This correspondence allows us to work with sharblies as integral objects and thus maintain precision in calculations.

Chapter 4. The Hecke Operators

This chapter defines the Hecke operators associated to the arithmetic cohomology groups we are studying. We briefly recall the definition and fundamental facts about the Hecke operators for classical modular forms. Then we define the Hecke operators on the Sharbly complex.

4.1 The Hecke Operators for Classical Modular Forms

The author is heavily indebted to the introductory texts on modular forms by Kilford [39] and Diamond and Shurman [29] for his current understanding of the Hecke operators; most of the following definitions are based on a synthesis of the definitions given in the aforementioned texts.

Briefly recall that for a weight-k modular form f we have the weight-k slash operator

$$(f|[\gamma])(z) = f(cz+d)^{-k}\det(\gamma)^{k-1}f(\gamma z),$$

where $\gamma \in \mathrm{GL}_2(\mathbb{Q})^+$. We will use this operator to define the Hecke operators.

The motivation for the Hecke operators comes from the deep, plentiful, and important arithmetic information conveyed by the coefficients of the q-expansion of modular forms. Some of this information was famously conjectured, observed, or declared by Ramanujan and should be considered one of the most compelling reasons to study modular forms. In particular, the Hecke operators are one way to divine information about the q-expansion of a modular form. Since the Hecke operators are a family of commuting operators on the space of cusp forms, they can be simultaneously diagonalized to find an eigenbasis for the space of new forms.

For two congruence subgroups of $\mathrm{SL}_2(\mathbb{Z})$, say Γ_1, Γ_2, and $\alpha \in \mathrm{GL}_2^+(\mathbb{Q})$ we are interested in the double coset

$$\Gamma_1 \alpha \Gamma_2 = \{\gamma_1 \alpha \gamma_2 : \gamma_i \in \Gamma_i\}.$$

If Γ_1 and Γ_2 have the right properties then this coset will enable us to define Hecke operators for the pair Γ_1, Γ_2. The follow definition comes from Rhie and Whaples [48] and is exactly the right property.

Definition 4.1. Two subgroups A, B of a group G are *commensurable* if

$$[A : A \cap B] < \infty \text{ and } [B : A \cap B] < \infty.$$

Diamond and Shurman prove that if A, B are commensurable then the double coset AxB can be written as a finite number of left or right cosets. They also show that two congruence subgroups are commensurable. Hence any double coset of the form $\Gamma_0(N)\alpha\Gamma_0(N)$ can be written as a disjoint union of a finite number of right cosets of $\Gamma_0(N)$; equivalently, every double coset can be written as a disjoint union of a finite number of left cosets of $\Gamma_0(N)$.

For the double coset $\Gamma_0(P)\begin{pmatrix}1 & 0\\ 0 & p\end{pmatrix}\Gamma_0(P)$, the standard set (meaning most commonly used set) of right coset representatives for this double coset is

$$\Omega_p = \left\{ \begin{pmatrix} p^{e_1} & a \\ 0 & p^{e_2} \end{pmatrix} \right\}$$

where $e_i \in \{0, 1\}$ and $e_1 + e_2 = 1$. If $e_1 = 0$ then a is a nonnegative integer less than p, otherwise $a = 0$. We then define the Hecke operator T_p as

$$T_p \cdot f = \sum_{\omega \in \Omega_p} f|[\omega]_k.$$

Moreover, when γ has the form $\begin{pmatrix}1 & j\\ 0 & p\end{pmatrix}$, where $j < p$, there are explicit formulas for $f|[\gamma]$ using the Fourier q-expansion of $f(z) = \sum_{n=0}^{\infty} a_n q^n$. The following formula comes from Kilford [39]:

$$f|\left[\begin{pmatrix}1 & j\\ 0 & p\end{pmatrix}\right](z) = \sum_{\substack{n=0 \\ n\equiv 0 \pmod p}}^{\infty} a_{np} q^n$$

This simplifies to

$$(T_p f)(q) = \sum_{n=1}^{\infty} (a_{pn} + \chi(p) p^{k-1} a_{n/p}) q^n$$

where we follow the convention that $a_{n/p} = 0$ if $n/p \notin \mathbb{Z}$ and $\chi(p)$ is the identity character on $\mathbb{Z}/p\mathbb{Z}$ extended to act on $\mathbb{Z}$. A modular form f is an eigenform of T_p if $T_p f = a_p f$ for some

scalar a_p. It is known that the Hecke operators T_n and T_m commute when $\gcd(n, m) = 1$. Furthermore, the operators T_{p^r} is defined recursively as $T_{p^{r+1}} = T_p T_{p^r} - p^{2k-1} T_{p^{n-1}}$. For more details about these facts see Diamond and Shurman [29].

Example 4.2. Let Δ be the cusp form of weight 12 whose Fourier expansion is given by

$$\Delta = \sum_{n=1}^{\infty} \tau(n) q^n = q - 24q^2 + 252q^3 - 1472q^4 + 4830q^5 - 6048q^6 + \dots$$

We then have that $T_2 \cdot \Delta = -24\Delta, T_3 \cdot \Delta = 252\Delta$ and so forth. We see that the Hecke eigenvalues of Δ correspond to its Fourier coefficients.

Remark. It is a standard theorem that two eigenforms of weight k and level 1 which are eigenfunctions of the Hecke operators with the same eigenvalues are scalar multiples of each other (see Serre [53]). It would be interesting to examine this phenomena for the cohomology of congruence subgroups of $\mathrm{SL}_n(\mathbb{Z})$ for $n > 2$.

4.2 The Hecke Operators on the Sharbly Complex

Recall that implicit in the definition of the Sharbly complex was a positive integer n. We define functions $T_{p,k}$, where $0 \leq k \leq n$, called the Hecke operators that are analogous to the Hecke operators for modular forms described above. Thus for $n > 2$ we get multiple Hecke operators for each prime p. It should be noted that for any n we technically have $T_{p,0}$ acting as the identity operator; consequently, for $n = 2$ we have $T_{p,1} = T_p$ and we get the same Hecke operators as we defined in the previous section along with the new operator $T_{2,2}$.

In general one can define an action of the Hecke operators on the group cohomology $H^k(\Gamma, A)$ where Γ is a subgroup of some G and A has a G-module structure. This is a very fascinating line of study and it is an entertaining endeavor to demonstrate that the Hecke operators described below match the general definition as given by Rhie and Whaples

[48]. Such a general approach has not yet lent itself to computations. One of the major breakthroughs in the study of arithmetic cohomology and reciprocity is the use of the Sharbly complex to compute the Hecke action on $H^*(\Gamma, \mathbb{C})$ [33].

Let $\mathbf{v}$ be a k-sharbly. Then for a prime p and an integer $j < n$ we define the operator $T_{p,j}$ using the double coset decomposition as found in Krieg [40]:

$$T_{p,j} \cdot \mathbf{v} = \sum_{h \in \Omega_{p,j}} h \cdot \mathbf{v},$$

where $\Omega_{p,j}$ is the set of all matrices of the form $\begin{pmatrix} p^{e_1} & & a_{ij} \\ & \ddots & \\ & & p^{e_n} \end{pmatrix}$ where $e_i \in \{0, 1\}$ and $\sum_i e_i = j$, $a_{ij} = 0$ unless $e_i = 0$ and $e_j = 1$, in which case $0 < a_{ij} < p$. Equivalently $\Omega_{p,j}$ is the set of all $n \times n$ matrices in Hermite Normal Form (see section 5.5) whose diagonal is a permutation of $(\underbrace{1, \ldots, 1}_{n-j}, \underbrace{p \ldots, p}_{j})$. When $n = 2$ we recover the double coset decomposition for modular forms. The code which generates the set $\Omega_{p,j}$ can be found in Appendix E.3.

Example 4.3. i) For $n = 3$ we have $\Omega_{2,1} =$

$$\left\{ \begin{pmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & a & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & b \\ 0 & 1 & c \\ 0 & 0 & 2 \end{pmatrix} : 0 \leq a, b, c \leq 1 \right\}.$$

ii) For $n = 2$ we have

$$\Omega_{3,1} = \left\{ \begin{pmatrix} 3 & 0 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 0 & 3 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 0 & 3 \end{pmatrix}, \begin{pmatrix} 1 & 2 \\ 0 & 3 \end{pmatrix}. \right\}$$

Let u be a cohomology class in $H^k(\Gamma; \mathbb{Z})$. Then we determine the Hecke action on u by working with a $\xi \in S_k$ such that ξ is a cycle modulo Γ and ξ is the image of u under the

isomorphism $H^k(\Gamma;\mathbb{Z}) \cong H_{\nu-k}((S_*)_\Gamma)$. For $h \in \Omega_{p,j}$ we have that $\det(h) = p^j$. It is a result of basic linear algebra that the k-sharblies $h \cdot \mathbf{v}$ will probably have norm larger than $||\mathbf{v}||$. The following example demonstrates why the word "probably" is needed in the last sentence.

Example 4.4. Consider the modular symbol $\mathbf{v} = \begin{bmatrix} 5 & 2 \\ 2 & 1 \end{bmatrix}$. Then using the set Ω_2 from Example 4.3 we have

$$T_{2,1} \cdot \mathbf{v} =$$
$$\begin{pmatrix} 3 & 0 \\ 0 & 1 \end{pmatrix} \cdot \mathbf{v} + \begin{pmatrix} 1 & 0 \\ 0 & 3 \end{pmatrix} \cdot \mathbf{v} + \begin{pmatrix} 1 & 1 \\ 0 & 3 \end{pmatrix} \cdot \mathbf{v} + \begin{pmatrix} 1 & 2 \\ 0 & 3 \end{pmatrix} \cdot \mathbf{v} =$$
$$\begin{bmatrix} 15 & 6 \\ 2 & 1 \end{bmatrix} + \begin{bmatrix} 5 & 2 \\ 6 & 3 \end{bmatrix} + \begin{bmatrix} 7 & 3 \\ 6 & 3 \end{bmatrix} + \begin{bmatrix} 9 & 4 \\ 6 & 3 \end{bmatrix} =$$
$$\begin{bmatrix} 15 & 6 \\ 2 & 1 \end{bmatrix} + \begin{bmatrix} 5 & 2 \\ 6 & 3 \end{bmatrix} + \begin{bmatrix} 7 & 1 \\ 6 & 1 \end{bmatrix} + \begin{bmatrix} 3 & 4 \\ 2 & 3 \end{bmatrix}.$$

Despite the fact that $||\mathbf{v}|| = 1$, only the last two 0-sharblies in the sum are unimodular.

Since we are primarily interested in the Hecke operators on $H^2(\Gamma_0(N), \mathbb{C})$ for $\Gamma_0(N) \subset \mathrm{SL}_3(\mathbb{Z})$, the corresponding isomorphism associates cohomology classes with chains of reduced 1-sharblies. However, the Hecke action will generally take a single reduced 1-sharbly to a chain of 1-sharblies with higher norm; this causes complications since currently there are no known techniques which take a chain of generic 1-sharblies and return a cohomology class. Instead we must modify the 1-sharbly chain modulo the boundary action to get a homologous chain of reduced 1-sharblies; this very technical process occupies the majority of the next chapter.

Chapter 5. Sharbly Reduction Methods

The first few chapters discussed how to calculate the cohomology by storing the boundary maps in a matrix, introduced the Sharbly complex, and explained how the Hecke operators act on the Sharbly complex. In this chapter we discuss an explicit implementation of the aforementioned isomorphism between $H^k(\Gamma;\mathbb{Z})$ and $H_{\nu-k}((S_*)_\Gamma)$ given by Borel-Serre duality. Our implementation will follow that of Ash, Gunnells, and McConnell [12] and takes a chain in the cell complex to a chain of reduced 1-sharblies. The main focus of the chapter will be to explain how one then reduces the 1-sharbly chain outputted by the Hecke operators. The chapter concludes with a description of how to convert a chain of 1-sharblies back into a chain of cells.

Since the Hecke action will increase the norm of almost any input of reduced 1-sharblies corresponding to a cohomology class in the cell complex, we are left with the problem of rewriting the resulting 1-sharbly chain as a chain of reduced 1-sharblies modulo the boundary, thus finding a representation of the homology class consisting of only reduced 1-sharblies. Once we find such a representation we then convert the resulting 1-sharbly chain into the appropriate cohomology class in the cell complex.

The problem of reducing a 1-sharbly chain is analogous to the situation encountered with modular symbols. For chains of 0-sharblies there exist many reduction algorithms which take as input a 0-sharbly $\mathbf{v}$ and return a finite sum $\sum_i \mathbf{v_i}$ such that $||\mathbf{v_i}|| = 1$ and $\mathbf{v} = \sum \mathbf{v_i}$ modulo the boundary. The fastest such algorithm was conjectured by Gunnells [33] and proven by Doud and Ricks [30] for $n \leq 5$; this algorithm uses LLL reduction to find reduction candidates which are used to construct the $\mathbf{v_i}$.

Since the norm of a k-sharbly chain is defined over the norm of the submodular symbols, it is plausible that by reducing all of the submodular symbols one can lower the norm of the k-sharbly. For positive k, there are no known algorithms that take as input a nonunimodular k-sharbly and return a homologous sum of k-sharblies all with lower norm. However, in practice the techniques detailed by Gunnells [33] have been used to computationally lower

the norm. Since these techniques depend upon modular symbols, we will first review modular symbol reduction before explaining the insights made by Gunnells.

5.1 Modular Symbol Reduction

Algorithms to reduce modular symbols have been of interest since 1972 when modular symbols were defined by Yuri Manin [44]. Since that time there have been many modular symbol reduction algorithms and more information in this vein can be found in Stein's book [57]. One modular symbol reduction algorithm is specific to $n \leq 5$ and uses LLL reduction; since we are primarily interested in performing calculations for $n = 3$ we will use this method.

To reduce a modular symbol is to replace the symbol $\mathbf{v}$ with a sum of symbols $\mathbf{u}_i$ such that $\mathbf{v} = \sum_i \mathbf{u}_i \pmod{\partial S_1}$. To do this we need to find a candidate for $\mathbf{v}$ for all i.

Definition 5.1. A *candidate* for a modular symbol $\mathbf{v}$ where $\mathbf{v}$ has primitive representative $(v_1, \ldots, v_n)$ is a primitive vector u such that $\mathbf{v}(u, i) = [v_1, \ldots, v_{i-1}, u, v_{i+1}, \ldots, v_n]$ has strictly smaller norm than $\mathbf{v}$.

Theorem 5.2. *Every modular symbol* $\mathbf{v}$ *is either reduced or has a candidate. Consequently every modular symbol can be reduced modulo the boundary to a chain of reduced symbols.*

Proof. The proof that every modular symbol has a candidate originates from Minkowski and his geometry of numbers [46]. Barvinok provides a more modern proof and gives an implementation to find such candidates [22]. From now on we will assume that every modular symbol has a candidate.

Let $\mathbf{v}$ be a modular symbol with candidate u and primitive representation $(v_1, \ldots, v_n)$. Then consider the 1-sharbly $\mathbf{w}$ with primitive representation $(u, v_1, \ldots, v_n)$.

$$\partial\mathbf{w} = -[v_1, \ldots, v_n] + [u, v_2, \ldots, v_n] - [v_1, u, v_3, \ldots, v_n] + \cdots \pm [v_1, \ldots, v_{n-1}, v_n].$$

Hence $\mathbf{v}$ is homologous to $\mathbf{v}(u, 1) - \mathbf{v}(u, 2) + \cdots \pm \mathbf{v}(u, n)$, since they differ by a boundary.

Thus every homology class corresponding to a chain of nonreduced modular symbols has a representation consisting of modular symbols with strictly smaller norm. Iterating this process gives us the required result. □

It should be noted that for a given modular symbol there are potentially a large number of candidates. Geometrically candidates are integral points inside the parallelepiped created by the columns of the primitive representation of the modular symbol. As the size of the fundamental domain increases, so does the number of candidates.

Example 5.3. Consider the modular symbol $\mathbf{v} = \begin{bmatrix} 2 & 1 \\ 3 & 3 \end{bmatrix}$. $||\mathbf{v}|| = 3$. The following figure shows the parallelepiped created by the columns of the primitive representation of $\mathbf{v}$.

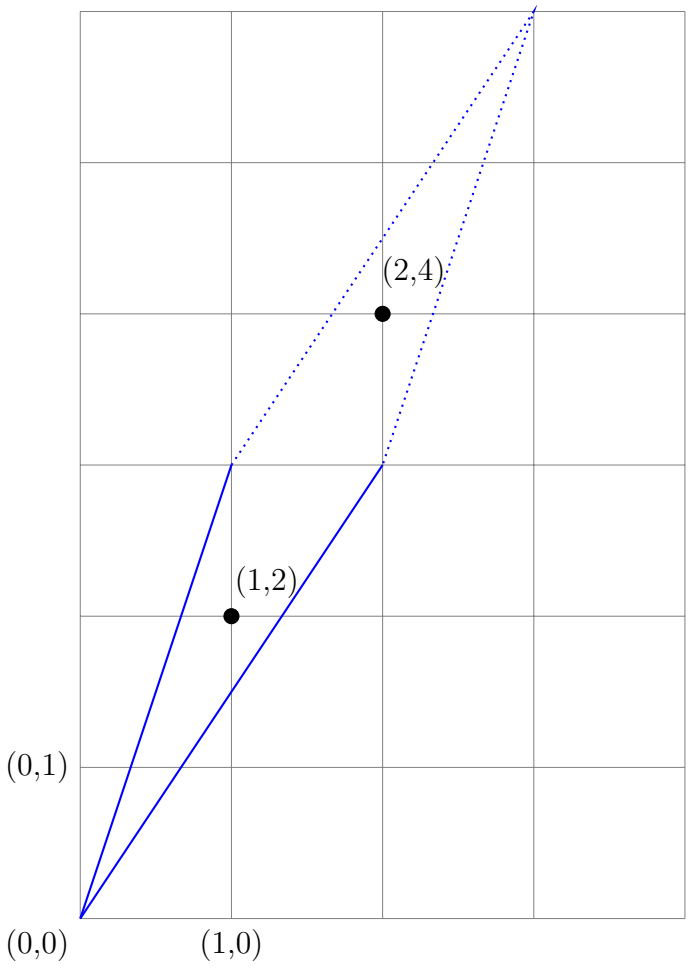

Figure 5.1: The parallelepiped formed by $(2,3)^T$ and $(1,3)^T$.

Note that there are two lattice points $(1,2)$ and $(2,4)$ in the parallelepiped formed by $\mathbf{v}$. The modular symbols $\mathbf{v}\left(1,(1,2)^T\right)$ and $\mathbf{v}\left(2,(1,2)^T\right)$ are both unimodular. Therefore $(1,2)^T$ is a candidate for $\mathbf{v}$. So $\mathbf{v} \equiv \mathbf{v}\left(1,(1,2)^T\right) + \mathbf{v}\left(2,(1,2)^T\right) \pmod{\partial S_1}$. We note that $(2,4)$ is not primitive and thus not a candidate.

It is often the case that a given modular symbol has many candidates. Choosing the best candidate u for a modular symbols $\mathbf{v}$, meaning the candidate which lowers the maximum norm of the resulting $\mathbf{v}(i,u)$, will speed up calculations. Unfortunately there are no known algorithms to calculate the best candidate, but there are algorithms to calculate better candidates. Van Geemen, van der Kallen, Top, and Verberkmoes [59] developed an approach which relies on modular arithmetic that finds candidates such that the resulting symbols have norm less than half the norm of the input symbol. Their approach works for any n and is the fastest reduction algorithm with known bounds. However, there is an approach proven by Doud and Ricks [30] which is experimentally more efficient; this approach utilizes an LLL-reduced basis for the lattice formed by the columns of a primitive representation of the modular symbol.

5.2 LLL Reduction

LLL reduction [43] discovered by Arjen Lenstra, Hendrik Lenstra, and László Lovász is one type of lattice reduction algorithm. In general lattice reduction algorithms seek to produce a *good* basis for a given lattice from a generic basis, where good has some inherent utility depending on the needed calculations. Recall that a lattice is the integral span of a collection of linearly independent vectors in Euclidean space. For our purposes we will also assume that we are dealing with a full lattice, meaning the rank of the vectors is equal to the dimension of the ambient space. A given lattice has infinitely many bases corresponding to its orbit under the action of $\mathrm{GL}_n(\mathbb{Z})$ on $\mathbb{M}_n(\mathbb{R})$. LLL reduction takes as input a basis and returns a

basis consisting of vectors that are nearly orthogonal. We expound on this definition below.

Let $\{b_1, \ldots, b_n\}$ be a basis of a lattice. Then define by induction:

$$b_i^* = b_i - \sum_{j=1}^{i-1} \mu_{i,j} b_j^* \quad \text{and} \quad \mu_{i,j} = b_i \cdot b_j^* / (b_j^* \cdot b_j^*).$$

Definition 5.4. A basis $\{b_1, \ldots, b_n\}$ is said to be LLL reduced if $\{b_1^*, \ldots, b_n^*\}$ is the basis described above and the following conditions hold:

1) $|\mu_{i,j}| \leq \frac{1}{2}$.

2) $|b_i^* + \mu_{i,i-1} b_{i-1}^*|^2 \geq \frac{3}{4} |b_{i-1}^*|^2$.

Lenstra, Lenstra, and Lovász [43] proved that every lattice has at least one LLL reduced basis and gave an algorithm to determine an LLL reduced basis for any matrix. A given lattice may contain many LLL reduced bases. However, each of these bases can be used to find a reduction candidate for a modular symbol as seen in the following theorem of Doud and Ricks [30]:

Theorem 5.5. *Let* $\mathbf{v}$ *be a modular symbol with* $n \leq 5$ *such that the primitive representation* $(v_1, \ldots, v_n)$ *of* $\mathbf{v}$ *is LLL reduced. Then one of the standard basis vectors* e_i *will be a candidate for* $\mathbf{v}$.

Let u be a candidate for $\mathbf{v}$. Then for $\gamma \in \mathrm{GL}_n(\mathbb{Z})$, γu will be a candidate for $\gamma \mathbf{v}$, since

$$\det(\gamma(v_1, \ldots, v_{i-1}, u, v_{i+1}, \ldots, v_n)) = \det(\gamma) \det(v_1, \ldots, v_{i-1}, u, v_{i+1}, \ldots, v_n).$$

Thus if $\gamma(v_1, \ldots, v_n)$ is LLL reduced with candidate e_j, then $\gamma^{-1} e_j$ will be a candidate for $(v_1, \ldots, v_n)$. Given a non-LLL reduced primitive representation of $\mathbf{v}$ we can use an LLL reduction algorithm to produce a matrix $\gamma \in \mathrm{GL}_n(\mathbb{Z})$ such that the primitive representation of $\gamma \mathbf{v}$ will be reduced. Then we can search the standard basis vectors to find a candidate for $\gamma \mathbf{v}$ and multiply by γ^{-1} to find a candidate for $\mathbf{v}$.

For implementation purposes we wrote an integral LLL reduction algorithm, meaning the code only used integer arithmetic, based on pseudo-code given by Cohen [26]. The code can be found in Appendix E.4. We also have a code that finds a candidate for a modular symbol $\mathbf{v}$ by choosing the best candidate from $\{\gamma^{-1}e_1, \gamma^{-1}e_2, \gamma^{-1}e_3\}$ where $\gamma\mathbf{v}$ has an LLL reduced primitive representation. The candidate selection code can be found in Appendix E.5

5.3 Converting Cohomology to Sharblies

The data previously computed in chapter 2, allows us to construct a 1-sharbly cycle ξ corresponding to the W-cocycle u. Recall that a W-cocycle corresponds to a collection of orbits with coefficients. In order to construct ξ we will choose a representative of each orbit (which is really just a 3-tuple $[a]$) and use this to construct a matrix γ, such that the bottom row of γ is in the equivalence class of the 3-tuple $[a]$. The following lemma explains how this is done.

Lemma 5.6. *For a projective 3-tuple $[a] \in \mathbb{P}^2(\mathbb{Z}/N\mathbb{Z})$ there is a matrix $M([a]) \in SL_3(\mathbb{Z})$ such that the bottom row of $M([a])$ is equal to the minimal representation of $[a]$.*

Proof. Let (a_1, a_2, a_3) be the minimal representation of $[a]$. If any of the a_i are 1, then the matrix can be taken to be one of

$$\begin{pmatrix} 0 & 0 & 1 \\ 0 & -1 & 0 \\ 1 & * & * \end{pmatrix}, \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ * & 1 & * \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ * & * & 1 \end{pmatrix},$$

where the $*$'s are the other entries of the minimal representative of $[a]$.

Otherwise we will use a constructive argument modified from Schenkman [49] to produce the desired matrix. Because $[a] \in \mathbb{P}^2(\mathbb{Z}/N\mathbb{Z})$, we have that $\gcd(a_1, a_2, a_3, N) = 1$. Since one of the $a_i | N$ we see that $\gcd(a_1, a_2, a_3) = 1$. Let $\gcd(a_1, a_2) = d$, then there exists integers x, y such that $\frac{a_1}{d}x + \frac{a_2}{d}y = 1$. We know $\gcd(d, a_3) = 1$ since $(\gcd(a_1, a_2, a_3) = 1$. Hence there

exist integers w, z such that $dw + a_3 z = 1$. So the matrix

$$M = \begin{pmatrix} -y & x & 0 \\ -z\frac{a_1}{d} & -z\frac{a_2}{d} & w \\ a_1 & a_2 & a_3 \end{pmatrix}$$

will satisfy

$$\det(M) = a_3(+\frac{a_2}{d}yz + \frac{a_1}{d}xz) + w(a_1x + a_2y) = dw + a_3(\frac{a_1}{d}xz + \frac{a_2}{d}yz) = dw + a_3z = 1.$$

□

The above argument can be directly implemented and extended by induction to any n. For $[a] \in \mathbb{P}^2(\mathbb{Z}/N)$ we will write $M([a])$ for the corresponding matrix constructed.

For each cell type in our cohomology class we associate the cell with its set of rays as in 2.6. Furthermore we identify the standard $\mathrm{SL}_3(\mathbb{Z})$-orbit representatives with 1-sharblies by taking the columns of the 1-sharbly to be the rays of the cell. Since we can scale any column in a 1-sharbly, this correspondence is well-defined. For congruence subgroups of $\mathrm{SL}_3(\mathbb{Z})$ we have only two cell types of dimension 2 in the Voronoi polyhedra and they correspond to the 1-sharblies

$$\tau_{\mathbf{2}} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \tau_{\mathbf{3}} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}.$$

The top dimensional cell corresponds to the 0-sharbly $\sigma_{\mathbf{0}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$.

Let $\xi_{\mathcal{W}}$ be a cycle in $H^2(\Gamma_0(N), \mathcal{C})$, we can write $\xi_{\mathcal{W}}$ as a chain of orbits,

$$\xi_{\mathcal{W}} = \sum_{\substack{i \\ \text{type } \tau_2}} \eta_{\mathcal{O}_i} \mathcal{O}_i + \sum_{\substack{j \\ \text{type } \tau_3}} \eta_{\mathcal{O}_j} \mathcal{O}_j.$$

Then the 1-sharbly chain

$$\xi = \sum_i \eta_{\mathcal{O}_i} M([a]) \cdot \tau_2 + \sum_j \eta_{\mathcal{O}_j} M([a]) \cdot \tau_3$$

will be the image of ξ under the isomorphism of $H^2(\Gamma_0(N), \mathcal{C})$ and $H_1(S, \mathcal{C})$.

The Hecke operator $T_{p,j}$ acts on this chain linearly as described in section 4.2. We can act on the chain to get a larger chain and then reduce the corresponding cells. But, the reduction technique will utilize the fact that ξ corresponds to a cohomology class of $H^{\nu-1}(\Gamma_0(N), \mathbb{C})$. Thus, we cannot choose lifts of $\partial M([a])$ independently of one another. Instead we will need to choose $\Gamma_0(N)$-equivariant lifts as done by Ash, Gunnells, and McConnell [12].

5.4 Choosing Appropriate Lifts

Since we our primarily interested in explicit computations we will need to choose integral representations for our 1-sharbly chain.

As we described in chapter 2, all of the cells corresponding to 1-sharblies in ξ consist entirely of orientable orbits. However, it is often the case that the boundary of an orientable cell will contain nonorientable orbits. Recall from section 2.6, that a $Stab(\sigma)$-orbit is non-orientable if and only if there is $\gamma \in Stab(\sigma)$, such that γ stabilizes an element of the orbit, but the value of the orientation character at γ is -1. Equivalently, every element of the orbit is stabilized by a matrix with negative orientation. For a 1-sharbly $\mathbf{u}$ in the support of ξ containing a boundary element corresponding to a non-orientable cell $[a]$, we replace

$$\eta_{\mathbf{u}} \mathbf{u} \cdot \tau_i \to \frac{\eta_{\mathbf{u}}}{2} \mathbf{u} \cdot \tau_i + \gamma \frac{\eta_{\mathbf{u}}}{2} \mathbf{u} \cdot \tau_i$$

where γ stabilizes $[a]$ and has orientation number -1. We do this for each non-orientable $[a]$ in boundary of η, possibly replacing the original 1-sharbly with a sum of four 1-sharblies. By so doing, we have not changed the homology class, but have made it computationally very clear that the non-orientable cells will cancel each other out in the boundary. We are now left with the task of choosing $\Gamma_0(N)$-equivariant lifts for the orientable 0-sharblies in the boundary of ξ.

For each 1-sharbly $\mathbf{u}$ in our 1-sharbly chain ξ, we need to choose a set of lifts that are γ-equivariant, in the following way: if $\mathbf{u}, \mathbf{u}'$ are two 1-sharblies in our chain, with $\mathbf{v} \in \partial\mathbf{u}$ and $\mathbf{v}' \in \partial\mathbf{u}'$ such that $\mathbf{v} = \gamma \cdot \mathbf{v}'$ for some $\gamma \in \Gamma_0(N)$, then we want $\mathbf{v}(\gamma) = \gamma L(\mathbf{v}')$; here $L(\mathbf{v})$ denotes the lift of $\mathbf{v}$.

Ultimately we want to implement our lifts locally, meaning without the need to cross reference the lifts we have already chosen. For a given $\mathbf{v}$ we compute an arbitrary lift $L(\mathbf{v})$. Then we find the orbit corresponding to the bottom row of $\mathbf{v}(\gamma)$. Let $[a]$ to be the smallest element in the orbit. Recall that attached to $[a]$ we have its primitive representation (a_1, a_2, a_3), an orientation (which is positive), and a matrix which takes the first element of the orbit to $[a]$. Since $[a]$ is the first element of its orbit we can relate $[a]$ to $(a_1, a_2, a_3, \pm 1, I)$ and $\mathfrak{b}(\mathbf{v}(\gamma)) = (b_1, b_2, b_3, \pm 1, \gamma_{\mathbf{b}})$, where $[a]\gamma_{\mathbf{b}} = [b_1, b_2, b_3]$. We replace $L(\mathbf{v})$ with $\gamma_{\mathbf{b}} L(\mathbf{v})$.

Before examining Gunnells' techniques to reduce 1-sharblies, we first make a small note about how our choice of candidates for a modular symbol will respect the $\Gamma_0(N)$ equivariant condition. Otherwise we'd be forced to reexamine the lifts after reducing the 1-sharby; since a given 1-sharbly will be replaced with a larger number of 1-sharblies, this is not a viable approach.

5.5 Hermite Normal Form

Since we need our calculations to be $\Gamma_0(N)$-equivariant we must make sure that whenever we choose a candidate for a 0-sharbly it respects the action of $\Gamma_0(N)$. That is to say that if two 0-sharblies are in the same $\Gamma_0(N)$-orbit, $\mathbf{v} = \gamma\mathbf{v}'$ for $\gamma \in \Gamma_0(N)$, then we want their respective

reduction candidates w, w' to satisfy $w = \gamma w'$. Clearly $\gamma w'$ will be a candidate for $\mathbf{v}$; but, one may have selected an alternative candidate for $\mathbf{v}$. To avoid such potential blunders, we first calculate a unique $\mathrm{GL}_n(\mathbb{Z})$-orbit representative ν for the matrix represented by the 0-sharbly $\mathbf{v}$, say $\nu\gamma'' = \mathbf{v}$. Then if w'' is the candidate selected by our algorithm for **nu** we choose $w''\gamma''$ as our candidate for $\mathbf{v}$. The problem is identifying a canonical $\mathrm{GL}_n(\mathbb{Z})$-orbit representative for each 0-sharbly; our solution is to use Hermite Normal Form.

Hermite Normal Form is a generalization of row echelon form; it corresponds to reducing a matrix using only row (resp. column) operations that preserve the absolute value of the determinant. Most authors, such as Cohen define HNF using column operations, meaning the $\mathrm{GL}_n(\mathbb{Z})$ action happens on the right. We will however, follow the definition in Stein [57], with the appropriate modifications due to typographical errors in the original. An invertible integral matrix A is in HNF if it satisfies the following:

- $a_{ij} = 0$, for $i < j$,
- $a_{ii} \gneqq 0$,
- $a_{ji} < a_{ii}$ for all $j < i$ unless $a_{ii} = 0$.

Lemma 5.7. *Every invertible integral matrix has a unique HNF.*

The proof for the above statement and a beautiful generalization can be found in Newman's book *Integral Matrices* [47]. Our code which returns the HNF form corresponding to a matrix can be found in Appendix E.6.

Remark. We note that the problem of choosing $\Gamma_0(N)$ equivariant lifts does not occur for 0-sharblies. This is because the boundary of every 0-sharbly is automatically trivial. Moreover, two 0-sharblies chosen to have primitive columns are equal iff one can be permuted into the other (up to sign change). Consider the case $n = 3$. Since the stabilizer of σ_0 has 24 elements the $Stab(\sigma_0)$-orbit of $[a, b, c]$ contains all the tuples equivalent to $[a, b, c]$ under permutation (up to sign change). Hence two 0-sharblies with primitive lifts will be in the same $\Gamma_0(N)$-orbit if and only if their corresponding bottom rows are in the same $Stab(\sigma_0)$-orbit.

5.6 Sharbly Reduction Methods: Introduction

As mentioned we need a reduction method for 1-sharblies. To do this we want to package together the potential candidates of all the submodular symbols of a 1-sharbly in such a way that the submodular symbols with highest norms cancel each other out. In [33], Gunnells presents his approach and provides experimental evidence wherein it replaces a 1-sharbly with a homologous chain of 1-sharblies all of which have smaller norm. We will summarize his computational technique while discussing the specific implementation for $n = 3$. It should be noted that these techniques have only been implemented computationally and are not known to terminate and thus do not give an actual algorithm.

For a 1-sharbly $[v_1, \ldots, v_n, v_{n+1}]$, rearranging the vectors if necessary, we will assume that the j modular symbols $[\hat{v_1}, v_2, \ldots, v_{n+1}], [v_1, \hat{v_2}, \ldots, v_{n+1}], \ldots, [v_1, v_2, \ldots, \hat{v_j}, \ldots, v_{n+1}]$ are not reduced and the other $n - j$ modular symbols are reduced. Let $w_1, \ldots, w_j$ be the candidates for the above modular symbols (where w_i is the candidate for the modular symbol $[\hat{v_1}, v_2, \ldots, v_{n+1}]$).

In general the method consists of finding a 2-sharbly chain η such that the boundary of η will cancel with the original terms and leave behind sharblies with smaller norm. As stated before, the techniques for packaging together the candidates and the remaining columns of the sharbly do not provably terminate or always produce a chain of 1-sharblies with lower norm. However, these methods have successfully reduced 1-sharblies in practice and thus allowed the computation of the Hecke eigenvalues.

For example, suppose that $n = 2$ and we have the symbol $\mathbf{v} = [v_1, v_2, v_3]$ such that none of the submodular symbols $[v_2, v_3], [v_1, v_3], [v_1, v_2]$ are reduced. Then we have three candidates w_1, w_2, w_3. We form the 2-sharbly chain $\eta = [v_1, v_3, v_2, w_1] + [v_1, w_2, v_3, w_1] + [v_1, w_3, w_2, w_1] + [v_1, v_2, w_3, w_1]$.

Then

$$[v_1, v_2, v_3] + \partial\eta =$$

$$\begin{aligned}
&[v_1, v_2, v_3] && - [v_3, v_2, w_1] + [v_1, v_2, w_1] - [v_1, v_3, w_1] + [v_1, v_3, v_2] \\
& && - [w_2, v_3, w_1] + [v_1, v_3, w_1] - [v_1, w_2, w_1] + [v_1, w_2, v_3] \\
& && - [w_3, w_2, w_1] + [v_1, w_2, w_1] - [v_1, w_3, w_1] + [v_1, w_3, w_2] \\
& && - [v_2, w_3, w_1] + [v_1, w_3, w_1] - [v_1, v_2, w_1] + [v_1, v_2, w_3].
\end{aligned}$$

We can cancel as seen in the following

$$[v_1, v_2, v_3] + \partial\eta =$$

$$\begin{aligned}
&[v_1, v_2, v_3] && - [v_3, v_2, w_1] + [v_1, v_2, w_1] - [v_1, v_3, w_1] + [v_1, v_3, v_2] \\
& && - [w_2, v_3, w_1] + [v_1, v_3, w_1] - [v_1, w_2, w_1] + [v_1, w_2, v_3] \\
& && - [w_3, w_2, w_1] + [v_1, w_2, w_1] - [v_1, w_3, w_1] + [v_1, w_3, w_2] \\
& && - [v_2, w_3, w_1] + [v_1, w_3, w_1] - [v_1, v_2, w_1] + [v_1, v_2, w_3].
\end{aligned}$$

Thus we have that $[v_1, v_2, v_3] + \partial\eta =$

$$-[v_3, v_2, w_1] - [w_2, v_3, w_1] + [v_1, w_2, v_3] - [w_3, w_2, w_1] + [v_1, w_3, w_2] - [v_2, w_3, w_1] + [v_1, v_2, w_3].$$

If we are using $\Gamma_0(N)$-equivariant lifts then the elements of the form $[v_i, v_j, w_k]$ where i, j, k are distinct will vanish in $\partial_\Gamma\eta$, as shown by Gunnells in the appendix of [57]. Thus we can replace $[v_1, v_2, v_3]$ with four 1-sharblies:

$$[v_1, v_2, v_3] \to -[w_2, v_3, w_1] - [w_2, w_2, w_1] + [v_1, w_3, w_2] - [v_2, w_3, w_1].$$

Remark. The above equation highlights the main difficulty with reducing 1-sharblies. In some sense Gunnells' approach is extremely natural, but it replaces the 1-sharbly $[v_1, v_2, v_3]$

with four 1-sharblies that are composed of the columns v_i and the candidates for $\partial[v_1, v_2, v_3]$. However, each candidate w_i is chosen independent of the other two candidates; there is no reason the symbol $[w_2, v_3, w_1]$ should have a norm smaller than $[v_1, v_2, v_3]$ and there are examples when it doesn't have a smaller norm. But, it practice the repeated use of the above replacement will eventually find an expression for the homology class of $[v_1, v_2, v_3]$ as a sum of reduced 1-sharblies.

One might be wondering how we knew to work with the 2-sharbly chain η. In some sense, our choice of η is a happy coincidence or a lucky choice from the universe. In another sense η is the most logical choice for a 2-sharbly whose boundary will reduce $[v_1, v_2, v_3]$. We know that $\partial\eta$ needs to cancel out $[v_1, v_2, v_3]$, thus somewhere in η we will want $[v_1, v_3, v_2, w]$ where w is some other vector. But, $\partial[v_1, v_2, v_3, w] = [v_1, v_3, v_2] - [v_1, v_2, w] + [v_1, v_3, w] - [v_1, v_2, w]$. In general a non-reduced 1-sharbly will not contain any reduced modular symbols in its boundary. Thus $[v_i, v_j, w]$ might have a higher norm unless w is the reduction point for v_i, v_j as a modular symbol. It is unlikely that w is a reduction for all of the v_i, v_j, but we can choose w to be the reducing point of $[v_2, v_3]$. Then the other terms are selected so as to cancel with the remaining boundary elements and hopefully produce sharblies with smaller norms.

5.7 Sharbly Reduction Methods: Geometric

The geometric motivation for Gunnells' 1-sharbly reduction method comes from considering a 2-sharbly as a simplex in $n + 1$ dimensions. Begin with the standard n-simplex labeled with the columns of the 1-sharbly. Then add vertices for any needed candidates and add edges from w_i to w_j, and v_j for $i \neq j$. Then the 2-sharbly η comes from decomposing the resulting polytope into $n+1$ dimensional simplices. In order to decompose the polytope into simplices, one needs to add an edge between of the w_i and the corresponding v_i. In general which w_i doesn't matter. In practice one can test all of the w_i and see which decomposition yields a chain with the smallest norm. For our purposes we will always add an edge from w_1

to v_1.

Equivalently, one can construct the needed polytope for a 1-sharbly $\mathbf{v}$ by taking the $(n-j)$th iterated cone over the j-orthoplex. The j-orthoplex is dual to the j-hypercube. Then when one labels the graph, place the j candidates w_i and the corresponding columns v_i as the vertices of the j-orthoplex. The other $n-j$ columns of the 1-sharbly are used to label the remaining vertices. From a geometric perspective this approach allows us to see the decomposition.

Remark. To find the simplicial decomposition the $n-j$th iterated cone over the j-orthoplex, we need only decompose the j-orthoplex into $j+1$-dimensional simplices and then cone over each simplex to obtain a higher dimension simplex; this observation significantly simplified the geometry needed to understand Gunnells' reduction technique.

Example 5.8. Reconsider the case $n=2$ where $\mathbf{v}=[v_1,v_2,v_3]$ and none of the submodular are reduced. Then we form a triangle and add 3 vertices labeled w_1,w_2,w_3 for the three candidates of the submodular symbols of $\mathbf{v}$.

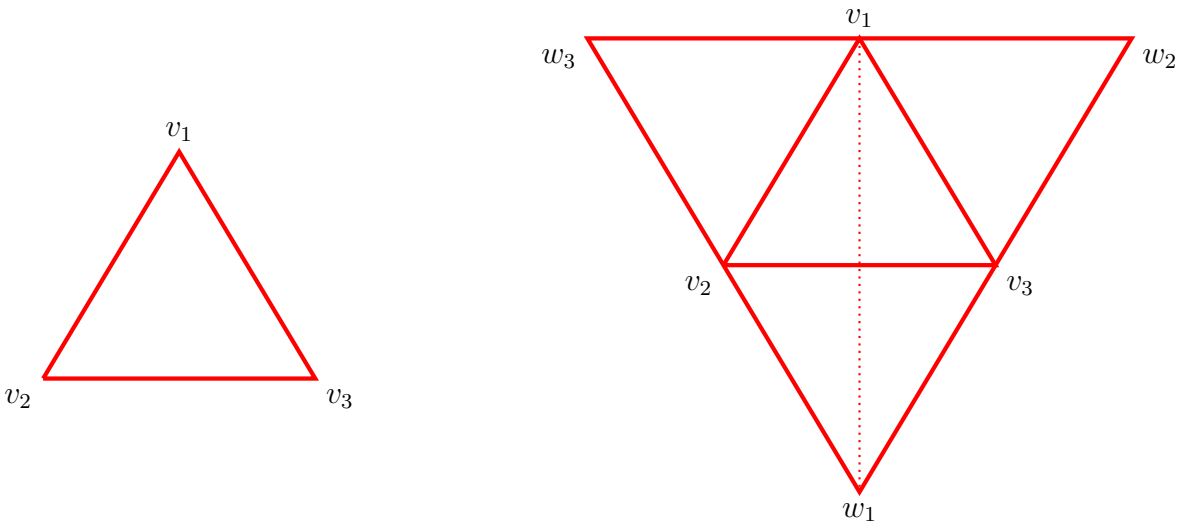

Figure 5.2: Determing η when $n=2$

The right-hand side of figure 5.2, is equivalent to the octahedron as seen in figure 5.3. Consequently we have the decomposition given in the previous section by cutting the octahedron along the planes containing w_1,v_1,w_i,v_i for $i=2$ and 3.

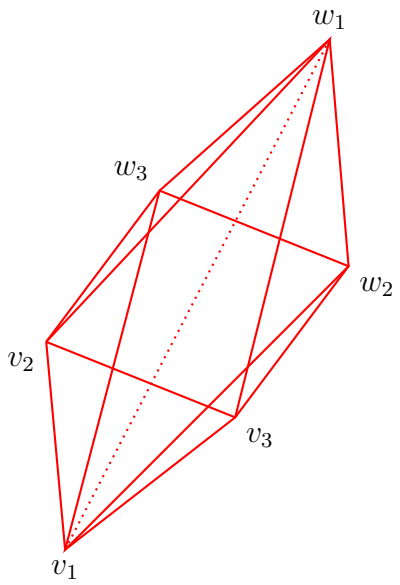

Figure 5.3: Decomposing the octahedron

We modify the construction if $\partial\mathbf{v}$ contains reduced 0-sharblies. As before we will assume that $\mathbf{v} = [v_1, v_2, v_3, v_4]$ is ordered so the first j submodular symbols are non reduced. The next few subsection gives the geometric shapes for each of the possible values of $j < 4$.

5.7.1 Completely Unimodular, $j = 0$. If all the 0-sharblies in the boundary of our 1-sharbly are unimodular, then the 1-sharbly is already reduced.

5.7.2 One nonunimodular Boundary Symbol, $j = 1$. If one of the 0-sharblies is nonunimodular say $[\mathbf{v_1}, \mathbf{v_2}, \mathbf{v_3}]$ then we only need a $\mathbf{w_1}$. Then $\eta = [\mathbf{v_1}, \mathbf{v_2}, \mathbf{v_3}, \mathbf{v_4}, \mathbf{w_1}]$. Geometrically we have the construction seen in figure 5.4.

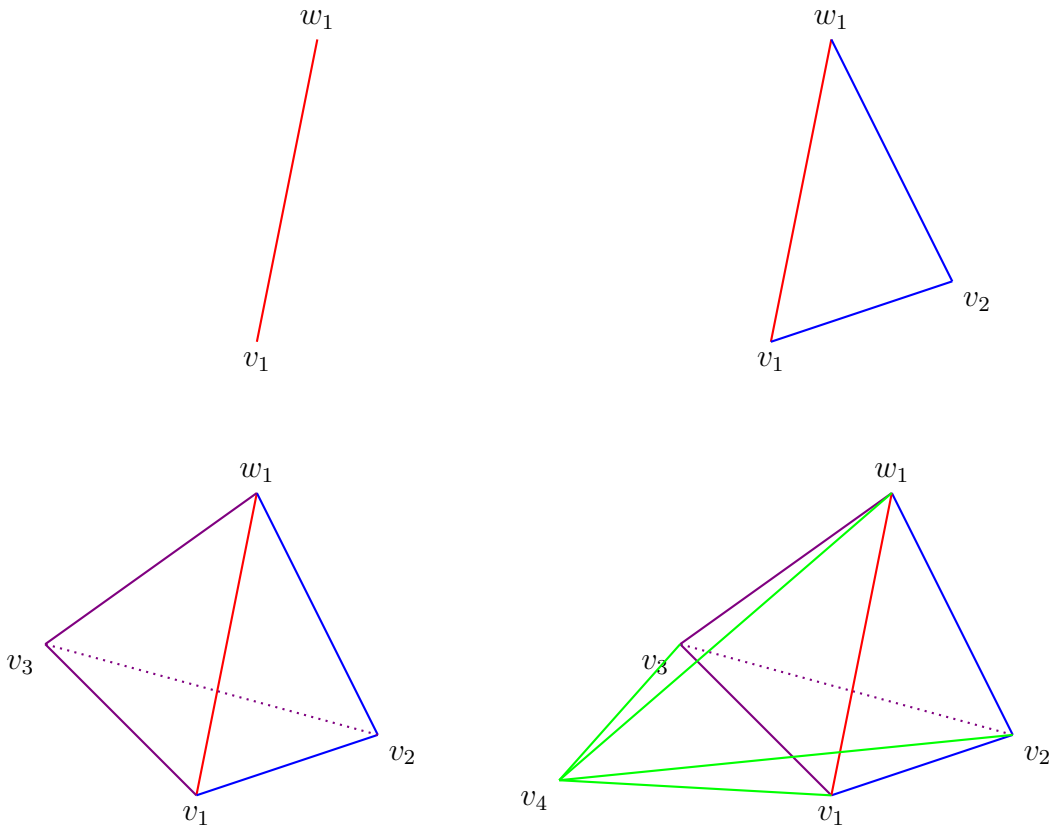

Figure 5.4: Constructing the 3-iterated cone over the 1 polytope.

5.7.3 Two nonunimodular Boundary Symbols, $j = 2$. We construct the 2-iterated cone over the 2-orthoplex in figure 5.5.

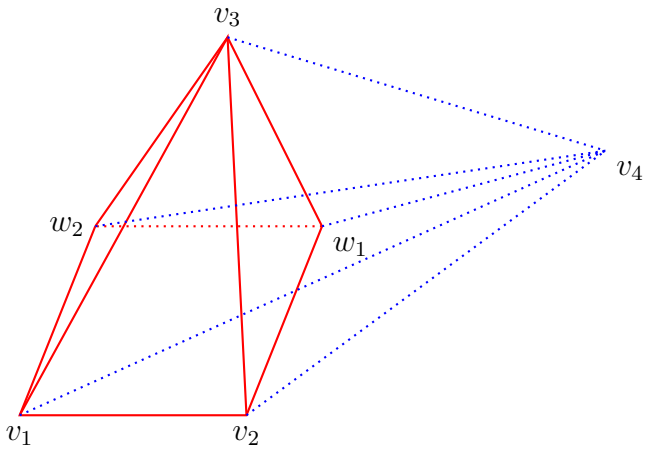

Figure 5.5: The 2-iterated cone over the 2 polytope.

This decomposes into 2 simplices giving us $\eta = [v_1, v_3, v_2, w_2] + [v_1, v_2, w_3, w_3]$.

5.7.4 Three nonunimodular Boundary Symbols, $j = 3$. We cone over the 3 orthoplex as seen in figure 5.6

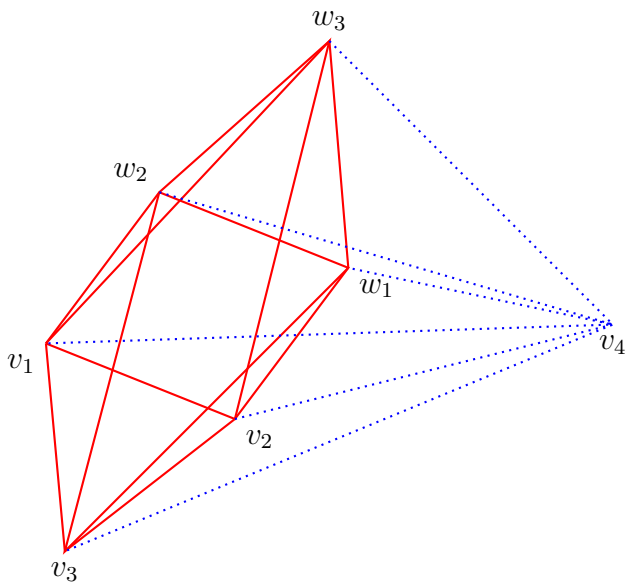

Figure 5.6: The 1-iterated cone over the 3-orthoplex.

.

Now we notice that in general such a geometric problem can be daunting, but we note that we already decomposed the octahedron (the 3-orthoplex) into simplices as seen in figure 5.3. Coning over the simplices with the same point (in this case v_4) preserves the simplicial decomposition. Thus we have

$$\eta = [v_1, v_3, v_2, w_1, v_4] + [v_1, w_2, v_3, w_1, v_4] + [v_1, w_3, w_2, w_1, v_4] + [v_1, v_2, w_3, w_1, v_4].$$

5.7.5 Four nonunimodular Boundary Symbols, $j = 4$. In this case we have the 4-orthoplex or 16-cell as our base figure. Instead of decomposing figure 5.7 directly we will use the following proposition to determine η.

Proposition 5.9. *The n-orthoplex labeled with $v_1, \ldots, v_n, w_1, \ldots, w_n$ such that v_i and w_i are*

not connected has simplicial decomposition equal

$$\sum_{u_i \in \{v_i, w_i\}} \pm(-1)^{k(u_2,\ldots,u_n)}[v_1, u_2, \ldots, u_{n-1}, w_1], \tag{5.1}$$

where $k(u_2, \ldots, u_n)$ *is the number of* u_i *that are equal to* w_i*.*

Proof. All of the facets of an n-orthoplex so labeled have the form $[u_1, \ldots, u_n]$ with $u_i \in \{v_i, w_i\}$. Adding the point w_1 or v_1 to each of this facets will create a simplex $[v_1, u_2, \ldots, u_n, w_1]$ and we claim these simplices give a simplicial decomposition of our n-orthoplex with the added edge between v_1 and w_1. To see this we note that any interior facet, that is a facet containing v_1, w_1 is in two distinct simplices corresponding to adding either v_1 or w_1 to the facet. Hence given a fixed started simplex with say positive, the orientations of its neighboring simplices will be negative, the orientations of their neighboring simplices will be positive, their neighbors will have negative orientation, and so forth. Thus, the boundary of our extended facet construction is the boundary of our n-orthoplex and we have a simplicial decomposition. The ± 1 in (5.1) refers to the two choices of orientation such a decomposition gives us. □

For our purposes we want $[v_1, \ldots, v_n, w_1]$ to have whatever orientation cancels out the 1-sharbly $[v_1, \ldots, v_n]$. Hence for $n = 4$ we have

$$\begin{aligned}\eta =&[v_1, v_2, v_3, v_4, w_1] - [v_1, v_2, v_3, w_4, w_1] - [v_1, v_2, w_3, v_4, w_1] - [v_1, w_2, v_3, v_4, w_1] \\ &+ [v_1, v_2, w_3, w_4, w_1] + [v_1, w_2, v_3, w_4, w_1] + [v_1, w_2, w_3, v_4, w_1] - [v_1, w_2, w_3, w_4, w_1].\end{aligned}$$

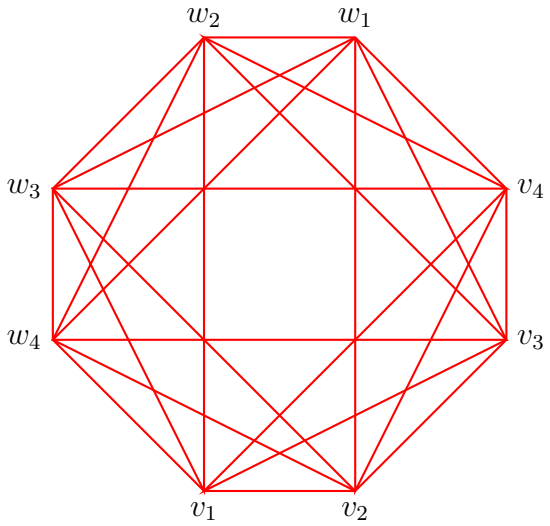

Figure 5.7: The 4-orthoplex or 16-cell.

Chapter 6. Example When $N = 11$

This chapter contains examples of the data need to calculated the cohomology of $\Gamma_0(11)$ as well as the calculation of the Hecke eigenvalues.

6.1 Orbits of the Stabilizers

This section explores the needed calculations when we are looking to commute the cohomology of $\Gamma_0(N)$ for $N = 11$. We start by determining the orbits of $\mathbb{P}^2(\mathbb{Z}/11\mathbb{Z})$ under $Stab(\sigma_0), Stab(\tau_2), Stab(\tau_3)$ and $Stab(\omega_2)$. We only store the orientable orbits, although we keep a vector of the orientation of each element of $\mathbb{P}^2(\mathbb{Z}/11/\mathbb{Z})$ under each of the cells; this information will be useful when we calculate the boundary maps. These orbits are stored in their entirety in Appendix D. Because there are 18 $Stab(\omega_2)$-orbits of $\mathbb{P}^2(\mathbb{Z}/11\mathbb{Z})$ we do not record that information here.

Orbits	${}_{\sigma_0}\mathcal{O}_1$	${}_{\sigma_0}\mathcal{O}_2$	${}_{\sigma_0}\mathcal{O}_3$	${}_{\sigma_0}\mathcal{O}_4$
Size of Orbit	12	12	24	24
Orbit Repres.	(1,0,2)	(1,0,3)	(1,2,3)	(1,2,4)

Table 6.1: Size and Representatives of $Stab(\sigma_0)$-orbits of $\mathbb{P}^2(\mathbb{Z}/11/\mathbb{Z})$

Orbits	${}_{\tau_2}\mathcal{O}_1$	${}_{\tau_2}\mathcal{O}_2$	${}_{\tau_2}\mathcal{O}_3$	${}_{\tau_2}\mathcal{O}_4$	${}_{\tau_2}\mathcal{O}_5$	${}_{\tau_2}\mathcal{O}_6$
Size of Orbit	6	12	12	12	12	12
Orbit Repres.	(1,2,0)	(1,2,1)	(1,2,2)	(1,2,3)	(1,2,4)	(1,2,5)

Table 6.2: Size and Representatives of $Stab(\tau_2)$-orbits of $\mathbb{P}^2(\mathbb{Z}/11/\mathbb{Z})$

Orbits	${}_{\tau_3}\mathcal{O}_1$	${}_{\tau_3}\mathcal{O}_2$	${}_{\tau_3}\mathcal{O}_3$	${}_{\tau_3}\mathcal{O}_4$
Size of Orbit	24	24	12	12
Orbit Repres.	(1,0,2)	(1,2,3)	(1,2,9)	(1,3,8)

Table 6.3: Size and Representatives of $Stab(\tau_3)$-orbits of $\mathbb{P}^2(\mathbb{Z}/11/\mathbb{Z})$

6.2 Making the Boundary Matrix

Next we decompose the $Stab(\sigma_0)$-orbits into their suborbits under the intersection of $Stab(\sigma_0) \cap Stab(\tau_2)$ and under the intersection $Stab(\sigma_0) \cap Stab(\tau_3)$. We also decompose the $Stab(\tau_2)$ and $Stab(\tau_3)$-orbits into their suborbits under the intersection of $Stab(\tau_2) \cap Stab(\omega_2)$ and $Stab(\tau_3) \cap Stab(\omega_2)$ respectively. A copy of the suborbits can be found in Appendix D.5.

We can now calculate the entries of the boundary matrices as described in section 2.28. The matrix produced will be a 4×10 matrix, where the 4 rows are the 4 $Stab(\sigma_0)$-orbits and the 10 columns correspond to the 6 $Stab(\tau_2)$-orbits and the 4 $Stab(\tau_3)$-orbits; we will combine these two types of orbits and refer to them as the 10 $Stab(\tau)$-orbits. Then the matrix entry a_{ij} will correspond to the boundary to the $Stab(\tau)$-orbits that appear in a suborbit of $Stab(\sigma_0)$.

Using the maps from 2.28 we find that the boundary map $d_1^{2,0} : E_2^{i,0} \to E_1^{3,0}$ is given by the matrix

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 \\ -2 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 \\ 0 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Appendix A. Stabilizers of Cells

A.1 Notation

We denote a cell by a lowercase Greek letter, either σ, τ, or ω, with a subscript. A cell can be though of as a set of vectors, which is invariant under multiplication by -1. If we wish to restrict to the rays spanned by these vectors (and thus identify each vector with its coset under the action of -1), we write $\tilde{\sigma}$. All of the cells all contain the standard unit vectors e_1, e_2, e_3 and their negatives. Some will also contain the elements $t_2 = (1,1,0)^T$, $t_3 = (1,1,1)^T$ and their negatives. For convenience we will use a boldface font to symbolize the ray spanned by an element. For example, $\mathbf{e_1}$ is the x-axis.

A.2 Stabilizer of σ_0

Recall that

$$\sigma_0 = \{\pm e_1, \pm e_2 \pm e_3\}.$$

$$Stab(\sigma_0) = \left\langle \underbrace{\begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{h_1}, \underbrace{\begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}}_{h_2} \right\rangle.$$

We see that $|Stab(\sigma_0)| = 24$. We now list the 24 elements along with the value of the permutation character.

$$s_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \rho(s_1) = 1, \qquad s_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{pmatrix}, \rho(s_2) = -1,$$

$$s_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix}, \rho(s_3) = 1, \qquad s_4 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix}, \rho(s_4) = -1,$$

$$s_5 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{pmatrix}, \rho(s_5) = -1, \qquad s_6 = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}, \rho(s_6) = 1,$$

$$s_7 = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \rho(s_7) = -1, \qquad s_8 = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & -1 \\ -1 & 0 & 0 \end{pmatrix}, \rho(s_8) = 1,$$

$$s_9 = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}, \rho(s_9) = 1, \qquad s_{10} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{pmatrix}, \rho(s_{10}) = -1,$$

$$s_{11} = \begin{pmatrix} 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{pmatrix}, \rho(s_{11}) = 1, \qquad s_{12} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & -1 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \rho(s_{12}) = -1,$$

$$s_{13} = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix}, \rho(s_{13}) = 1, \qquad s_{14} = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}, \rho(s_{14}) = -1,$$

$$s_{15} = \begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \rho(s_{15}) = 1, \qquad s_{16} = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & -1 & 0 \end{pmatrix}, \rho(s_{16}) = -1,$$

$$s_{17} = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \rho(s_{17}) = -1, \qquad s_{18} = \begin{pmatrix} 0 & -1 & 0 \\ 0 & 0 & 1 \\ -1 & 0 & 0 \end{pmatrix}, \rho(s_{18}) = 1,$$

$$s_{19} = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & -1 \end{pmatrix}, \rho(s_{19}) = -1, \qquad s_{20} = \begin{pmatrix} 0 & -1 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & 0 \end{pmatrix}, \rho(s_{20}) = 1,$$

$$s_{21} = \begin{pmatrix} 0 & 0 & -1 \\ 1 & 0 & 0 \\ 0 & -1 & 0 \end{pmatrix}, \rho(s_{21}) = 1, \qquad s_{22} = \begin{pmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \rho(s_{22}) = -1,$$

$$s_{23} = \begin{pmatrix} 0 & 0 & -1 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}, \rho(s_{23}) = 1, \qquad s_{24} = \begin{pmatrix} 0 & 0 & -1 \\ 0 & -1 & 0 \\ -1 & 0 & 0 \end{pmatrix}, \rho(s_{24}) = -1.$$

A.3 Stabilizer of τ_2

Recall that $\tau_2 = \{\pm e_1, \pm e_2, \pm e_3, \pm(t_2)\}$. Then the stabilizer of τ_2 consists of the following matrices:

$$t_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \rho(s_1) = 1, \qquad t_2 = \begin{pmatrix} 1 & -1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix}, \rho(s_2) = -1,$$

$$t_3 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{pmatrix}, \rho(t_3) = -1, \qquad t_4 = \begin{pmatrix} 0 & -1 & 0 \\ 1 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \rho(t_4) = 1,$$

$$t_5 = \begin{pmatrix} 1 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \rho(t_5) = 1, \qquad t_6 = \begin{pmatrix} 1 & 0 & 0 \\ 1 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \rho(t_6) = -1,$$

$$t_7 = \begin{pmatrix} -1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix}, \rho(t_7) = -1, \qquad t_8 = \begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \rho(t_8) = 1,$$

$$t_9 = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \rho(t_9) = 1, \qquad t_{10} = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & -1 \end{pmatrix}, \rho(t_{10}) = -1,$$

$$t_{11} = \begin{pmatrix} -1 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \rho(t_{11}) = 1, \qquad t_{12} = \begin{pmatrix} -1 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix}, \rho(t_{12}) = -1,$$

A.4 Stabilizer of τ_3

Given the set $\tau_3 = \{\pm e_1, \pm e_2, \pm e_3, \pm(e_1+e_2+e_3)\}$, we find the stabilizer $Stab(\tau_3)$. We claim $St(\tau_3)$ has order 24 and is generated by the following matrices:

$$Stab(\tau_3) = \left\langle \begin{pmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & -1 \end{pmatrix}, \begin{pmatrix} 0 & 0 & -1 \\ 0 & -1 & 0 \\ -1 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 1 & -1 & 0 \\ 1 & 0 & -1 \\ 1 & 0 & 0 \end{pmatrix} \right\rangle.$$

$$r_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \rho(r_1) = 1, \qquad r_2 = \begin{pmatrix} 1 & 0 & -1 \\ 0 & 0 & -1 \\ 0 & 1 & -1 \end{pmatrix}, \rho(r_2) = 1,$$

$$r_3 = \begin{pmatrix} 1 & -1 & 0 \\ 0 & -1 & 1 \\ 0 & -1 & 0 \end{pmatrix}, \rho(r_3) = 1, \qquad r_4 = \begin{pmatrix} 0 & 1 & -1 \\ 1 & 0 & -1 \\ 0 & 0 & -1 \end{pmatrix}, \rho(r_4) = 1,$$

$$r_5 = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}, \rho(r_5) = 1, \qquad r_6 = \begin{pmatrix} 0 & -1 & 0 \\ 1 & -1 & 0 \\ 0 & -1 & 1 \end{pmatrix}, \rho(r_6) = 1,$$

$$r_7 = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}, \rho(r_7) = 1, \qquad r_8 = \begin{pmatrix} 0 & 0 & -1 \\ 0 & 1 & -1 \\ 1 & 0 & -1 \end{pmatrix}, \rho(r_8) = 1,$$

$$r_9 = \begin{pmatrix} 0 & -1 & 1 \\ 0 & -1 & 0 \\ 1 & -1 & 0 \end{pmatrix}, \rho(r_9) = 1, \qquad r_{10} = \begin{pmatrix} 1 & -1 & 0 \\ 1 & 0 & -1 \\ 1 & 0 & 0 \end{pmatrix}, \rho(r_{10}) = -1,$$

$$r_{11} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & -1 & 0 \\ 1 & 0 & -1 \end{pmatrix}, \rho(r_{11}) = -1, \qquad r_{12} = \begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & 0 \\ 1 & -1 & 0 \end{pmatrix}, \rho(r_{12}) = -1,$$

$$r_{13} = \begin{pmatrix} -1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & -1 \end{pmatrix}, \rho(r_{13}) = -1, \qquad r_{14} = \begin{pmatrix} -1 & 0 & 1 \\ 0 & -1 & 1 \\ 0 & 0 & 1 \end{pmatrix}, \rho(r_{14}) = -1,$$

$$r_{15} = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & -1 & 0 \end{pmatrix}, \rho(r_{15}) = -1, \qquad r_{16} = \begin{pmatrix} 0 & 1 & -1 \\ -1 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix}, \rho(r_{16}) = -1,$$

$$r_{17} = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & -1 \end{pmatrix}, \rho(r_{17}) = -1, \qquad r_{18} = \begin{pmatrix} 0 & 0 & 1 \\ -1 & 0 & 1 \\ 0 & -1 & 1 \end{pmatrix}, \rho(r_{18}) = -1,$$

$$r_{19} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & -1 \\ -1 & 1 & 0 \end{pmatrix}, \rho(r_{19}) = -1, \qquad r_{20} = \begin{pmatrix} 0 & -1 & 1 \\ 0 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix}, \rho(r_{20}) = -1,$$

$$r_{21} = \begin{pmatrix} 0 & 0 & -1 \\ 0 & -1 & 0 \\ -1 & 0 & 0 \end{pmatrix}, \rho(r_{21}) = -1, \qquad r_{22} = \begin{pmatrix} -1 & 1 & 0 \\ -1 & 0 & 0 \\ -1 & 0 & 1 \end{pmatrix}, \rho(r_{22}) = 1,$$

$$r_{23} = \begin{pmatrix} -1 & 0 & 1 \\ -1 & 1 & 0 \\ -1 & 0 & 0 \end{pmatrix}, \rho(s_{23}) = 1, \qquad r_{24} = \begin{pmatrix} -1 & 0 & 0 \\ -1 & 0 & 1 \\ -1 & 1 & 0 \end{pmatrix}, \rho(s_{24}) = 1.$$

A.5 Stabilizer of ω_2

Recall that

$$\omega_2 = \{\pm e_1, \pm e_2, \pm e_3, \pm t_2, \pm t_3\}.$$

$$Stab(\omega_2) = \left\langle \begin{pmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & -1 \end{pmatrix}, \begin{pmatrix} -1 & 0 & 0 \\ -1 & 0 & 1 \\ -1 & 1 & 0 \end{pmatrix} \right\rangle$$

$$w_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \rho(w_1) = 1, \qquad w_2 = \begin{pmatrix} 0 & 1 & -1 \\ 1 & 0 & -1 \\ 0 & 0 & -1 \end{pmatrix}, \rho(w_2) = 1,$$

$$w_3 = \begin{pmatrix} 0 & -1 & 1 \\ 0 & -1 & 0 \\ 1 & -1 & 0 \end{pmatrix}, \rho(w_3) = 1, \qquad w_4 = \begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & 0 \\ 1 & -1 & 0 \end{pmatrix}, \rho(w_4) = -1,$$

$$w_5 = \begin{pmatrix} -1 & 0 & 1 \\ 0 & -1 & 1 \\ 0 & 0 & 1 \end{pmatrix}, \rho(w_5) = -1, \qquad w_6 = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & -1 \end{pmatrix}, \rho(w_6) = -1,$$

$$w_7 = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & -1 \\ -1 & 1 & 0 \end{pmatrix}, \rho(w_7) = -1, \qquad w_8 = \begin{pmatrix} -1 & 0 & 0 \\ -1 & 0 & 1 \\ -1 & 1 & 0 \end{pmatrix}, \rho(w_8) = 1,$$

A.6 Intersection Data

A.7 Boundaries of the Standard Cells

To determine the boundary of a cell σ we view σ as a collection of $2k$ vectors corresponding to the minimal vectors of some quadratic form. Then from Voronoi's work [60] the boundary of σ will be all cells τ which correspond to collections of $2(k+1)$ vectors containing the original $2k$ vectors. Since we have explicit cells to work with, we will use the labeling from chapter 2 for $n = 3$. Thus $\sigma_0 = \pm\{e_1, e_2, e_3\}$, $\tau_2 = \pm\{e_1, e_2, e_3, t_2\}$, $\tau_3 = \pm\{e_1, e_2, e_3, t_3\}$, and $\omega_2 = \pm\{e_1, e_2, e_3, t_2, t_3\}$.

A.7.1 Boundary of σ_0. Then we have that the boundary of σ_0 contains 10 cells, 6 of type τ_2 and 2 of type τ_3. The type τ_2 cells in the boundary of σ_0 are

$$\tau_2, \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \tau_2, \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \tau_2, \begin{pmatrix} 0 & 0 & -1 \\ 1 & 0 & 0 \\ 0 & -1 & 0 \end{pmatrix} \tau_2, \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \tau_2, \begin{pmatrix} 0 & -1 & 0 \\ 0 & 0 & 1 \\ -1 & 0 & 0 \end{pmatrix} \tau_2.$$

The type τ_3 cells in the boundary of σ_0 are

$$\tau_3, \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \tau_3, \begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tau_3, \begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \tau 3.$$

A.7.2 Boundary of τ_2. The boundary of τ_2 contains 6 cells all of type ω_2, they are

$$\omega_2, \begin{pmatrix} 0 & 1 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \omega_2, \begin{pmatrix} 0 & 1 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \omega_2, \begin{pmatrix} 1 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \omega_2, \begin{pmatrix} 1 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{pmatrix} \omega_2, \begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \omega_2.$$

A.7.3 Boundary of τ_3. The boundary of τ_3 contains 3 cells all of type ω_2, they are

$$\omega_2, \begin{pmatrix} -1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & -1 & 0 \end{pmatrix} \omega_2, \begin{pmatrix} 0 & -1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \omega_2.$$

Appendix B. Algorithms for $\mathbb{P}^2(\mathbb{Z}/N\mathbb{Z})$

The following is a collection of code for C++, used to construct and work with elements of $\mathbb{P}^2(\mathbb{Z}/N\mathbb{Z})$.

B.0.4 Comments and Explanations.

```
/* Creates the class of Projective Three_tuples (P^2(Z/NZ)),
     the ordering and conversion to integers,
     along with the action of a matrix on a three tuple (three_tuple is a row)

%%%%Depends on Mat_Class%%%% a class of 3 by 3 matrices stored as 9 integers
     class Three_tuple -- a class of threetuples, can construct, print, outfile,
          reduce to a standard representations, and reduction for nonprime inputs

     We order reduced three_tuples lexiographically.
     Create Ordering This include
                                        1) (a,b,c) to i;
                                        2) i to (a,b,c);
                                        3) reducetuple;
               For nonprime level, we must first determine a list of "good" positions
```

```
                We will simply fill all positions starting with a divisor of n,
                then mark those entries that are already occupied.
                This happens in the prologue

     int abc_to_i -- returns the i in the lexiographical ordering,
assumes the input is already reduces
     Three_tuple i_to_abc(int i) -- gives the abc corresponding to a given i;
     void Three_tuple::Reduce_tuple() -- reduces a tuple into a fixed representative,
which is unique for prime (n)-level
     bool Three_tuple::isReduced() -- tells if a tuple is reduced;

     Three_tuple Mat_act_on_Tuple(Mat X, Three_tuple T) -- returns T*X
*/

class Three_tuple // a class of three tuples
{
public:
     Three_tuple(int one, int two, int three);
     Three_tuple();
     int one,two,three;
     void PrintThree_tuple() const;
     void Reduce_tuple();
     void OutThree_tuple() const;
     bool isEqual(Three_tuple T);
     bool isReduced();
};
```

```
Three_tuple::Three_tuple(int new_one, int new_two, int new_three)
{
     one=new_one; two=new_two; three=new_three;
}

Three_tuple::Three_tuple(){;}

bool Three_tuple::isEqual(Three_tuple T)
{
     return ((one==T.one)&&(two==T.two)&&(three==T.three));
}
void Three_tuple::PrintThree_tuple() const
{
     cout << "[" << one << ", " << two << ", " << three <<"]\n";
     //out_file << "[" << one << ", " << two << ", " << three <<"]";//"n";
}

void Three_tuple::OutThree_tuple() const
{
     //cout << "[" << one << ", " << two << ", " << three <<"]\n";
     out_file << "[" << one << ", " << two << ", " << three <<"]";//"n";
}

int abc_to_i(int d,int e, int f)
{
     int a,b,c;
     Three_tuple T (d,e,f);
```

```
    T.Reduce_tuple();
    a=T.one;
    b=T.two;
    c=T.three;
    if((a==0)*(b==0)*(c==1))
    {
        return (k)*n*n+k*n+1;
    }
    if((a==0)*(b!=0))
    {
        for(int i=0;i<k;i++) //replace with divisors_of_n.size()
        {
            if(divisors_of_n[i]==b)
            {
                return k*n*n+(i)*n+c+1;
            }
        }
    }
    for(int i=0;i<k;i++)
    {
        if(divisors_of_n[i]==a)
        {
            return i*n*n+b*n+c+1;
        }
    }
}
```

```
Three_tuple i_to_abc(int i)
{
     int a,b,c;
     if(i==k*n*n+k*n+1)
     {
          Three_tuple M(0,0,1);
          return M;
     }
     if(i>k*n*n)
     {
          int j=i-k*n*n;
          if((j%n)!=0)
          {
               b=divisors_of_n[j/n];
          }
          else
          {
               b=divisors_of_n[j/n - 1];
          }
          c=modn(j-1);
          Three_tuple M(0,b,c);
          return M;
     }
     if(i%(n*n)!=0)
     {
          a=divisors_of_n[i/(n*n)];
     }
```

```
     else
     {
          a=divisors_of_n[i/(n*n)-1];
          b=n-1;
          c=n-1;
          Three_tuple M(a,b,c);
          return M;
     }
     int j=i%(n*n);
     if(j%n!=0)
     {
          b=modn(j/n);
     }
     else
     {
          b=modn(j/n-1);
     }
     c=modn(i-1);
     Three_tuple M(a,b,c);
     return M;
}

//Need to check if tuples with a divisor in the first entry/ second are reduced
//(i.e. reduce to a fixed representative).
//assumes that the tuple is already in reduced (a,b,c) form meaning
//a|n or a==0 and b|n or a==b==0, c==1;
```

```
void Three_tuple::Reduce_tuple()
{
     if(gcddd(one,two,three,n)!=1)
     {
          one=0;two=0;three=0;
          return;
     }
     one=modn(one);two=modn(two);three=modn(three);
     if(gcd(one,n)==1)
     {
          int one_inverse=modn_inverse[modn(one)];
          two=modn(two*one_inverse);
          three=modn(three*one_inverse);
          one=1;
          return;
     }
     if(one==0)
     {
          if(gcd(two,n)==1)
          {
               int two_inverse=modn_inverse[modn(two)];
               three=modn(three*two_inverse);
               two=1;
               one=0;
               return;
          }
          if(two==0)
```

```
          {
               three=gcd(three,n);
               return;
          }
          //assumes that one is 0 and two is a 0-divisor
          int two_inverse=modn_inverse[modn(two)];
          two=modn(two*two_inverse);
          three=modn(three*two_inverse);
          int index=identify_divisors(two);
          int min=n*n*k+n*k+1;
          //cout << index << "\t" << min;
          for(int i=0;i<elem_stabilizer[index].size();i++)
          {
               //cout << "elem_stabilizer[index][i]*one = "
<< modn(elem_stabilizer[index][i]*one) << "\n";
               //cout << "elem_stabilizer[index][i]*two = "
<< modn(elem_stabilizer[index][i]*two) << "\n";
               //cout << "elem_stabilizer[index][i]*three = "
<< modn(elem_stabilizer[index][i]*three) << "\n";


               int minimal=abc_to_i(0,modn(elem_stabilizer[index][i]*two),
modn(elem_stabilizer[index]                    [i]*three));
               //cout <<"minimal = " << minimal <<"\n";
               if(minimal<=min)
               {
                    min=minimal;
```

```
                    // cout << "new min = " << min << "\n";
               }
          }
          Three_tuple T=i_to_abc(min);
          //T.PrintThree_tuple(); cout <<"\n";
          one=T.one; two=T.two; three=T.three;
          return;
     }
     int one_inverse = modn_inverse[modn(one)];
     one=modn(one*one_inverse); two=modn(two*one_inverse);
three=modn(three*one_inverse);
     int index=identify_divisors(one);
     //cout << index <<"\n";
     int min=k*n*n+k*n+1;
     //cout << "min = " << min  << "\n";
     for(int i=0;i<elem_stabilizer[index].size();i++)
     {
          //cout << "elem_stabilizer[index][i]*one = "
<< modn(elem_stabilizer[index][i]*one) << "\n";
          //cout << "elem_stabilizer[index][i]*two = "
<< modn(elem_stabilizer[index][i]*two) << "\n";
          //cout << "elem_stabilizer[index][i]*three = "
<< modn(elem_stabilizer[index][i]*three) << "\n";

          int minimal=abc_to_i(modn(elem_stabilizer[index][i]*one),
modn(elem_stabilizer[index][i]*two), modn(elem_stabilizer[index][i]*three));
          //cout << "minimal = " << minimal << "\n";
```

```
          if(minimal<=min)
          {
               min=minimal;
          // cout << "min = " << min << "\n";
          }
     }
          Three_tuple T=i_to_abc(min);
          //T.PrintThree_tuple();
          //cout <<"\n";
          one=T.one; two=T.two; three=T.three;
          return;
}

Three_tuple Mat_act_on_Tuple(Mat X, Three_tuple T)
//as a row times a three by three matrix.
{
     int dum1=X.a*T.one+X.d*T.two+X.g*T.three;
     int dum2=X.b*T.one+X.e*T.two+X.h*T.three;
     int dum3=X.c*T.one+X.f*T.two+X.i*T.three;
     Three_tuple TT (dum1,dum2,dum3);
     return TT;
}

bool Three_tuple::isReduced()
// needs to determine if the entry is really the smallest in its row
 // checks that the entry is reduced.
{
```

```
    Three_tuple TT (one,two,three);
    TT.Reduce_tuple();
    return isEqual(TT);
}
```

Appendix C. Computing the Kernel Modulo the Boundary

Let V_1, V_2, V_3 be vector spaces of dimensions n_1, n_2, n_3 respectively. If we have the maps $T_A : V_1 \to V_2$ and $T_B : V_2 \to V_3$ s.t. $T_B \circ T_A = 0$, we can examine the cohomology of the sequence at V_2. To do this we need to calculate the kernel of T_B mod the image of T_A. One way to do this would be to calculate a basis for the image of T_A and extend it to a basis for the kernel of T_B. If one wanted to use this approach, the algorithms given by Cohen [26] should work.

Without loss of generality let $\{v_1, \ldots, v_k, v_{k+1} \ldots v_m \ldots v_n\}$ be a basis for V_2 where the $\{v_1, \ldots, v_k\}$ is a basis for the image of T_A and $\{v_1, \ldots, v_m\}$ is a basis for the kernel of T_B.

Then let $A = [T_A]$ with respect to this basis for V_2 and a convenient basis for V_1, we can assume that the leftmost k columns of A are

$$\begin{pmatrix} 1 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ \vdots & & & \\ 0 & 0 & \ldots & 1 \\ 0 & 0 & \ldots & 0 \\ \vdots & & & \\ 0 & 0 & \ldots & 0 \end{pmatrix}.$$

Let $B = [T_B]$ with respect to the above basis for V_2 and a convenient basis for V_3, we can assume that the leftmost m columns of B are all zero.

If we form the matrix $B' = \begin{pmatrix} B \\ A^t \end{pmatrix}$ and row reduce to find the null space, the first k columns will no longer be contained in the kernel. However, the rest of the reduced row echelon form of B will not change. Hence the rank-nullity theorem shows that the kernel of B' is the kernel of B minus the image of A.

Since the kernel T_B and the image of T_A are basis independent, the kernel of $B' = \begin{pmatrix} [T_B] \\ [T_A]^t \end{pmatrix}$ for any choice of basis of V_1, V_2, V_3 will be the kernel of T_B mod the image of T_A.

Over $\mathbb{R}$ the $(\mathrm{im}(A))^\perp = \ker(A^t)$ calculating the intersection of $\ker(B)$ with $(\mathrm{im}(A))^\perp$ reduces to calculating the intersection of two kernels; which is done by row reducing the matrix composed of the vertical concatenation of the two matrices.

Appendix D. Data for the Example $N = 11$

The below orbits are presented in the form $[a, b, c] \pm 1$ where ± 1 is the orientation number of the element.

D.1 $Stab(\sigma_0)$-orbits of $\mathbb{P}^2(\mathbb{Z}/11\mathbb{Z})$

${}_{\sigma_0}\mathcal{O}_1 =$

{

```
[1, 0, 2] 1
[1, 9, 0] -1
[1, 0, 9] 1
[1, 2, 0] -1
[0, 1, 9] -1
[1, 6, 0] 1
[0, 1, 2] -1
```

[1, 5, 0] 1

[0, 1, 6] 1

[1, 0, 5] -1

[0, 1, 5] 1

[1, 0, 6] -1

}

${}_{\sigma_0}\mathcal{O}_2 =$

{

[1, 0, 3] 1

[1, 8, 0] -1

[1, 0, 8] 1

[1, 3, 0] -1

[0, 1, 8] -1

[1, 4, 0] 1

[0, 1, 3] -1

[1, 7, 0] 1

[0, 1, 4] 1

[1, 0, 7] -1

[0, 1, 7] 1

[1, 0, 4] -1

}

${}_{\sigma_0}\mathcal{O}_3 =$

{

[1, 2, 3] 1

[1, 8, 2] -1

[1, 9, 8] 1

[1, 3, 9] -1

[1, 6, 4] -1

[1, 4, 8] 1

[1, 5, 4] -1

[1, 7, 8] 1

[1, 7, 6] 1

[1, 3, 7] -1

[1, 7, 5] 1

[1, 3, 4] -1

[1, 9, 3] 1

[1, 8, 9] -1

[1, 2, 8] 1

[1, 3, 2] -1

[1, 5, 7] -1

[1, 4, 3] 1

[1, 6, 7] -1

[1, 7, 3] 1

[1, 4, 5] 1

[1, 8, 7] -1

[1, 4, 6] 1

[1, 8, 4] -1

}

$_{\sigma_0}\mathcal{O}_4 =$

{

[1, 2, 4] 1

[1, 7, 2] -1

[1, 9, 7] 1

[1, 4, 9] -1

[1, 6, 9] -1

[1, 3, 6] 1

[1, 5, 9] -1

[1, 8, 6] 1

[1, 2, 6] 1

[1, 5, 8] -1

[1, 2, 5] 1

[1, 5, 3] -1

[1, 9, 4] 1

[1, 7, 9] -1

[1, 2, 7] 1

[1, 4, 2] -1

[1, 5, 2] -1

[1, 3, 5] 1

[1, 6, 2] -1

[1, 8, 5] 1

[1, 9, 5] 1

[1, 6, 8] -1

[1, 9, 6] 1

[1, 6, 3] -1

}

D.2 $Stab(\tau_2)$-orbits of $\mathbb{P}^2(\mathbb{Z}/11\mathbb{Z})$

${}_{\tau_2}\mathcal{O}_1 =$

{

[1, 2, 0] 1

[1, 8, 0] -1

[1, 6, 0] -1

[1, 4, 0] 1

[1, 7, 0] 1

[1, 3, 0] -1

}

$_{\tau_2}\mathcal{O}_2 =$

{

[1, 2, 1] 1

[1, 8, 10] -1

[1, 6, 5] -1

[1, 4, 6] 1

[1, 7, 4] 1

[1, 3, 7] -1

[1, 8, 1] -1

[1, 2, 10] 1

[1, 4, 5] 1

[1, 6, 6] -1

[1, 7, 7] 1

[1, 3, 4] -1

}

$_{\tau_2}\mathcal{O}_3 =$

{

[1, 2, 2] 1

[1, 8, 9] -1

[1, 6, 10] -1

[1, 4, 1] 1

[1, 7, 8] 1

[1, 3, 3] -1

[1, 8, 2] -1

[1, 2, 9] 1

[1, 4, 10] 1

[1, 6, 1] -1

[1, 7, 3] 1

[1, 3, 8] -1

}

${}_{\tau_2}\mathcal{O}_4 =$

{

[1, 2, 3] 1

[1, 8, 8] -1

[1, 6, 4] -1

[1, 4, 7] 1

[1, 7, 1] 1

[1, 3, 10] -1

[1, 8, 3] -1

[1, 2, 8] 1

[1, 4, 4] 1

[1, 6, 7] -1

[1, 7, 10] 1

[1, 3, 1] -1

}

${}_{\tau_2}\mathcal{O}_5 =$

{

[1, 2, 4] 1

[1, 8, 7] -1

[1, 6, 9] -1

[1, 4, 2] 1

[1, 7, 5] 1

[1, 3, 6] -1

[1, 8, 4] -1

[1, 2, 7] 1

[1, 4, 9] 1

[1, 6, 2] -1

[1, 7, 6] 1

[1, 3, 5] -1

}

$_{\tau_2}\mathcal{O}_6 =$

{

[1, 2, 5] 1

[1, 8, 6] -1

[1, 6, 3] -1

[1, 4, 8] 1

[1, 7, 9] 1

[1, 3, 2] -1

[1, 8, 5] -1

[1, 2, 6] 1

[1, 4, 3] 1

[1, 6, 8] -1

[1, 7, 2] 1

[1, 3, 9] -1

}

D.3 $Stab(\tau_3)$-orbits of $\mathbb{P}^2(\mathbb{Z}/11\mathbb{Z})$

${}_{\tau_3}\mathcal{O}_1 =$

}

[1, 0, 2] 1

[1, 2, 8] 1

[1, 8, 0] 1

[0, 1, 8] 1

[0, 1, 6] 1

[0, 1, 3] 1

[1, 6, 0] 1

[1, 0, 4] 1

[1, 4, 6] 1

[1, 7, 0] -1

[1, 0, 3] -1

[1, 3, 7] -1

[1, 8, 2] -1

[1, 0, 8] -1

[1, 2, 0] -1

[0, 1, 7] -1

[0, 1, 2] -1

[0, 1, 4] -1

[1, 4, 0] -1

[1, 6, 4] -1

[1, 0, 6] -1

[1, 7, 3] 1

[1, 0, 7] 1

[1, 3, 0] 1

}

$_{\tau_3}\mathcal{O}_2 =$

}

[1, 2, 3] 1

[1, 3, 5] 1

[1, 5, 2] 1

[1, 6, 8] 1

[1, 7, 6] 1

[1, 8, 7] 1

[1, 4, 8] 1

[1, 8, 9] 1

[1, 9, 4] 1

[1, 9, 7] -1

[1, 7, 5] -1

[1, 5, 9] -1

[1, 5, 3] -1

[1, 2, 5] -1

[1, 3, 2] -1

[1, 8, 6] -1

[1, 6, 7] -1

[1, 7, 8] -1

[1, 9, 8] -1

[1, 4, 9] -1

[1, 8, 4] -1

[1, 9, 5] 1

[1, 7, 9] 1

[1, 5, 7] 1

}

${}_{\tau_3}\mathcal{O}_3 =$

}

[1, 2, 9] 1

[1, 9, 10] 1

[1, 10, 2] 1

[1, 6, 5] 1

[1, 10, 6] 1

[1, 5, 10] 1

[1, 10, 9] -1

[1, 9, 2] -1

[1, 2, 10] -1

[1, 5, 6] -1

[1, 6, 10] -1

[1, 10, 5] -1

}

${}_{\tau_3}\mathcal{O}_4 =$

}

[1, 3, 8] 1

[1, 8, 10] 1

[1, 10, 3] 1

[1, 4, 7] 1

[1, 10, 4] 1

[1, 7, 10] 1

[1, 10, 8] -1

[1, 8, 3] -1

[1, 3, 10] -1

[1, 7, 4] -1

[1, 4, 10] -1

[1, 10, 7] -1

}

D.4 $Stab(\omega_2)$-orbits of $\mathbb{P}^2(\mathbb{Z}/11\mathbb{Z})$

${}_{\omega_2}\mathcal{O}_1 =$

}

[1, 0, 0] 1

[0, 1, 10] 1

[1, 0, 10] -1

[0, 1, 0] -1

}

${}_{\omega_2}\mathcal{O}_2 =$

}

[1, 0, 1] 1

[0, 1, 9] 1

[1, 9, 1] 1

[1, 5, 5] -1

[1, 0, 9] -1

[0, 1, 1] -1

[1, 9, 0] -1

[1, 5, 0] 1

}

${}_{\omega_2}\mathcal{O}_3 =$

}

```
[1, 0, 2] 1
[0, 1, 8] 1
[1, 4, 6] 1
[1, 3, 7] -1
[1, 0, 8] -1
[0, 1, 2] -1
[1, 4, 0] -1
[1, 3, 0] 1
```

}

${}_{\omega_2}\mathcal{O}_4 =$

}

```
[1, 0, 3] 1
[0, 1, 7] 1
[1, 6, 4] 1
[1, 2, 8] -1
[1, 0, 7] -1
[0, 1, 3] -1
[1, 6, 0] -1
[1, 2, 0] 1
```

}

${}_{\omega_2}\mathcal{O}_5 =$

}

```
[1, 0, 4] 1
```

[0, 1, 6] 1

[1, 7, 3] 1

[1, 8, 2] -1

[1, 0, 6] -1

[0, 1, 4] -1

[1, 7, 0] -1

[1, 8, 0] 1

}

$_{\omega_2}\mathcal{O}_6 =$

}

[1, 2, 1] 1

[1, 6, 9] 1

[1, 7, 1] 1

[1, 8, 8] -1

[1, 2, 7] -1

[1, 6, 6] -1

[1, 7, 2] -1

[1, 8, 5] 1

}

$_{\omega_2}\mathcal{O}_7 =$

}

[1, 2, 2] 1

[1, 6, 3] 1

[1, 3, 6] 1

[1, 4, 2] -1

[1, 2, 6] -1

[1, 6, 1] -1

[1, 3, 1] -1

[1, 4, 4] 1

}

$_{\omega_2}\mathcal{O}_8 =$

}

[1, 2, 3] 1

[1, 6, 8] 1

[1, 9, 4] 1

[1, 5, 9] -1

[1, 2, 5] -1

[1, 6, 7] -1

[1, 9, 8] -1

[1, 5, 7] 1

}

$_{\omega_2}\mathcal{O}_9 =$

}

[1, 2, 9] 1

[1, 6, 5] 1

[1, 2, 10] -1

[1, 6, 10] -1

}

$_{\omega_2}\mathcal{O}_{10} =$

}

[1, 3, 2] 1

[1, 4, 9] 1

[1, 8, 6] 1

[1, 7, 9] -1

[1, 3, 5] -1

[1, 4, 8] -1

[1, 8, 7] -1

[1, 7, 5] 1

}

$_{\omega_2}\mathcal{O}_{11} =$

}

[1, 3, 3] 1

[1, 4, 5] 1

[1, 5, 4] 1

[1, 9, 3] -1

[1, 3, 4] -1

[1, 4, 1] -1

[1, 5, 1] -1

[1, 9, 9] 1

}

$_{\omega_2}\mathcal{O}_{12} =$

}

[1, 3, 8] 1

[1, 4, 7] 1

[1, 3, 10] -1

[1, 4, 10] -1

}

$_{\omega_2}\mathcal{O}_{13} =$

}

```
[1, 5, 2] 1
[1, 9, 5] 1
[1, 7, 6] 1
[1, 8, 4] -1
[1, 5, 3] -1
[1, 9, 7] -1
[1, 7, 8] -1
[1, 8, 9] 1
```

}

$_{\omega_2}\mathcal{O}_{14} =$

}

```
[1, 5, 6] 1
[1, 9, 2] 1
[1, 5, 10] -1
[1, 9, 10] -1
```

}

$_{\omega_2}\mathcal{O}_{15} =$

}

```
[1, 7, 4] 1
[1, 8, 3] 1
[1, 7, 10] -1
[1, 8, 10] -1
```

}

$_{\omega_2}\mathcal{O}_{16} =$

}

```
[1, 10, 1] 1
[1, 10, 10] -1
```

}

$_{\omega_2}\mathcal{O}_{17} =$

}

```
[1, 10, 2] 1
[1, 10, 6] 1
[1, 10, 5] -1
[1, 10, 9] -1
```

}

$_{\omega_2}\mathcal{O}_{18} =$

}

```
[1, 10, 3] 1
[1, 10, 4] 1
[1, 10, 7] -1
[1, 10, 8] -1
```

}

D.5 Calculating Suborbits Under the Intersection

The orbits are presented in the order given for the orbits of $Stab(\sigma_0)$. The data reads as $[a, b, c] \pm 1o$ where ± 1 is the orientation number of the element under $Stab(\sigma)$ and o is the orientation number under $Stab(\tau)$ where σ is the main cell and τ is the boundary cell. Line spaces separate the suborbits and two line spaces separated the orbits.

D.5.1 Decomposition of the $Stab(\sigma_0)$-orbits under $Stab(\sigma_0) \cap Stab(\tau_2)$ and $Stab(\sigma_0) \cap Stab(\tau_3)$.

Decomposition of the $Stab(\sigma_0)$-orbits under $Stab(\sigma_0) \cap Stab(\tau_2)$

```
[1, 0, 2] 1 0
[0, 1, 9] -1 0
[1, 0, 9] 1 0
[0, 1, 2] -1 0

[1, 9, 0] -1 0
[1, 5, 0] 1 0

[1, 2, 0] -1 1
[1, 6, 0] 1 -1

[0, 1, 6] 1 0
[1, 0, 5] -1 0
[0, 1, 5] 1 0
[1, 0, 6] -1 0


[1, 0, 3] 1 0
[0, 1, 8] -1 0
[1, 0, 8] 1 0
[0, 1, 3] -1 0

[1, 8, 0] -1 -1
[1, 7, 0] 1 1
```

```
[1, 3, 0] -1 -1
[1, 4, 0] 1 1

[0, 1, 4] 1 0
[1, 0, 7] -1 0
[0, 1, 7] 1 0
[1, 0, 4] -1 0


[1, 2, 3] 1 1
[1, 6, 4] -1 -1
[1, 2, 8] 1 1
[1, 6, 7] -1 -1

[1, 8, 2] -1 -1
[1, 7, 8] 1 1
[1, 8, 9] -1 -1
[1, 7, 3] 1 1

[1, 9, 8] 1 0
[1, 5, 4] -1 0
[1, 9, 3] 1 0
[1, 5, 7] -1 0

[1, 3, 9] -1 -1
[1, 4, 8] 1 1
[1, 3, 2] -1 -1
```

[1, 4, 3] 1 1

[1, 7, 6] 1 1

[1, 8, 7] -1 -1

[1, 7, 5] 1 1

[1, 8, 4] -1 -1

[1, 3, 7] -1 -1

[1, 4, 5] 1 1

[1, 3, 4] -1 -1

[1, 4, 6] 1 1

[1, 2, 4] 1 1

[1, 6, 9] -1 -1

[1, 2, 7] 1 1

[1, 6, 2] -1 -1

[1, 7, 2] -1 1

[1, 8, 6] 1 -1

[1, 7, 9] -1 1

[1, 8, 5] 1 -1

[1, 9, 7] 1 0

[1, 5, 9] -1 0

[1, 9, 4] 1 0

[1, 5, 2] -1 0

```
[1, 4, 9] -1 1
[1, 3, 6] 1 -1
[1, 4, 2] -1 1
[1, 3, 5] 1 -1

[1, 2, 6] 1 1
[1, 6, 8] -1 -1
[1, 2, 5] 1 1
[1, 6, 3] -1 -1

[1, 5, 8] -1 0
[1, 9, 5] 1 0
[1, 5, 3] -1 0
[1, 9, 6] 1 0
```

Decomposition of the $Stab(\sigma_0)$-orbits under $Stab(\sigma_0) \cap Stab(\tau_3)$.

```
[1, 0, 2] 1 1
[1, 6, 0] 1 1
[0, 1, 6] 1 1
[1, 2, 0] -1 -1
[0, 1, 2] -1 -1
[1, 0, 6] -1 -1

[1, 9, 0] -1 0
[0, 1, 9] -1 0
[1, 0, 5] -1 0
[1, 0, 9] 1 0
```

[1, 5, 0] 1 0

[0, 1, 5] 1 0

[1, 0, 3] 1 -1

[1, 4, 0] 1 -1

[0, 1, 4] 1 -1

[1, 3, 0] -1 1

[0, 1, 3] -1 1

[1, 0, 4] -1 1

[1, 8, 0] -1 1

[0, 1, 8] -1 1

[1, 0, 7] -1 1

[1, 0, 8] 1 -1

[1, 7, 0] 1 -1

[0, 1, 7] 1 -1

[1, 2, 3] 1 1

[1, 4, 8] 1 1

[1, 7, 6] 1 1

[1, 3, 2] -1 -1

[1, 6, 7] -1 -1

[1, 8, 4] -1 -1

[1, 8, 2] -1 -1

[1, 6, 4] -1 -1

[1, 3, 7] -1 -1

[1, 2, 8] 1 1

[1, 7, 3] 1 1

[1, 4, 6] 1 1

[1, 9, 8] 1 -1

[1, 7, 8] 1 -1

[1, 7, 5] 1 -1

[1, 8, 9] -1 1

[1, 5, 7] -1 1

[1, 8, 7] -1 1

[1, 3, 9] -1 0

[1, 5, 4] -1 0

[1, 3, 4] -1 0

[1, 9, 3] 1 0

[1, 4, 3] 1 0

[1, 4, 5] 1 0

[1, 2, 4] 1 0

[1, 3, 6] 1 0

[1, 2, 6] 1 0

[1, 4, 2] -1 0

[1, 6, 2] -1 0

[1, 6, 3] -1 0

```
[1, 7, 2] -1 0

[1, 6, 9] -1 0

[1, 5, 8] -1 0

[1, 2, 7] 1 0

[1, 8, 5] 1 0

[1, 9, 6] 1 0


[1, 9, 7] 1 -1

[1, 8, 6] 1 -1

[1, 2, 5] 1 -1

[1, 7, 9] -1 1

[1, 5, 2] -1 1

[1, 6, 8] -1 1


[1, 4, 9] -1 -1

[1, 5, 9] -1 -1

[1, 5, 3] -1 -1

[1, 9, 4] 1 1

[1, 3, 5] 1 1

[1, 9, 5] 1 1
```

D.5.2 Decomposition of the $Stab(\tau_2)$ and $Stab(\tau_3)$-orbits under $Stab(\tau_2) \cap Stab(\omega_2)$ and $Stab(\tau_3) \cap Stab(\omega_3)$.

Decomposition of the $Stab(\tau_2)$-orbits under $Stab(\tau_2) \cap Stab(\omega_2)$.

```
Suborbits of Tau_2 under Omega_2

[1, 2, 0] 1 1

[1, 6, 0] -1 -1
```

```
[1, 8, 0] -1 1
[1, 7, 0] 1 -1

[1, 4, 0] 1 -1
[1, 3, 0] -1 1



[1, 2, 1] 1 1
[1, 6, 6] -1 -1

[1, 8, 10] -1 -1
[1, 7, 4] 1 1

[1, 6, 5] -1 1
[1, 2, 10] 1 -1

[1, 4, 6] 1 1
[1, 3, 7] -1 -1

[1, 8, 1] -1 0
[1, 7, 7] 1 0

[1, 4, 5] 1 1
[1, 3, 4] -1 -1



[1, 2, 2] 1 1
```

[1, 6, 1] -1 -1

[1, 8, 9] -1 1

[1, 7, 8] 1 -1

[1, 6, 10] -1 -1

[1, 2, 9] 1 1

[1, 4, 1] 1 -1

[1, 3, 3] -1 1

[1, 8, 2] -1 -1

[1, 7, 3] 1 1

[1, 4, 10] 1 -1

[1, 3, 8] -1 1

[1, 2, 3] 1 1

[1, 6, 7] -1 -1

[1, 8, 8] -1 -1

[1, 7, 1] 1 1

[1, 6, 4] -1 1

[1, 2, 8] 1 -1

[1, 4, 7] 1 1

[1, 3, 10] -1 -1

[1, 8, 3] -1 1

[1, 7, 10] 1 -1

[1, 4, 4] 1 1

[1, 3, 1] -1 -1

[1, 2, 4] 1 0

[1, 6, 2] -1 0

[1, 8, 7] -1 -1

[1, 7, 5] 1 1

[1, 6, 9] -1 1

[1, 2, 7] 1 -1

[1, 4, 2] 1 -1

[1, 3, 6] -1 1

[1, 8, 4] -1 -1

[1, 7, 6] 1 1

[1, 4, 9] 1 1

[1, 3, 5] -1 -1

```
[1, 2, 5] 1 -1
[1, 6, 8] -1 1

[1, 8, 6] -1 1
[1, 7, 9] 1 -1

[1, 6, 3] -1 1
[1, 2, 6] 1 -1

[1, 4, 8] 1 -1
[1, 3, 2] -1 1

[1, 8, 5] -1 1
[1, 7, 2] 1 -1

[1, 4, 3] 1 0
[1, 3, 9] -1 0
```

Decomposition of the $Stab(\tau_3)$-orbits under $Stab(\tau_3) \cap Stab(\omega_2)$.

```
Suborbits of Tau_3 under Omega_2
[1, 0, 2] 1 1
[0, 1, 8] 1 1
[1, 4, 6] 1 1
[1, 3, 7] -1 -1
[1, 0, 8] -1 -1
[0, 1, 2] -1 -1
```

```
[1, 4, 0] -1 -1
[1, 3, 0] 1 1

[1, 2, 8] 1 -1
[1, 6, 0] 1 -1
[1, 0, 7] 1 -1
[0, 1, 7] -1 1
[1, 2, 0] -1 1
[1, 6, 4] -1 1
[1, 0, 3] -1 1
[0, 1, 3] 1 -1

[1, 8, 0] 1 1
[1, 7, 3] 1 1
[0, 1, 6] 1 1
[1, 0, 6] -1 -1
[1, 8, 2] -1 -1
[1, 7, 0] -1 -1
[0, 1, 4] -1 -1
[1, 0, 4] 1 1


[1, 2, 3] 1 1
[1, 6, 8] 1 1
[1, 9, 4] 1 1
[1, 5, 9] -1 -1
[1, 2, 5] -1 -1
```

[1, 6, 7] -1 -1

[1, 9, 8] -1 -1

[1, 5, 7] 1 1

[1, 3, 5] 1 -1

[1, 4, 8] 1 -1

[1, 7, 9] 1 -1

[1, 8, 6] -1 1

[1, 3, 2] -1 1

[1, 4, 9] -1 1

[1, 7, 5] -1 1

[1, 8, 7] 1 -1

[1, 5, 2] 1 1

[1, 9, 5] 1 1

[1, 7, 6] 1 1

[1, 8, 4] -1 -1

[1, 5, 3] -1 -1

[1, 9, 7] -1 -1

[1, 7, 8] -1 -1

[1, 8, 9] 1 1

[1, 2, 9] 1 1

[1, 6, 5] 1 1

[1, 2, 10] -1 -1

[1, 6, 10] -1 -1

[1, 9, 10] 1 -1

[1, 5, 10] 1 -1

[1, 5, 6] -1 1

[1, 9, 2] -1 1

[1, 10, 2] 1 1

[1, 10, 6] 1 1

[1, 10, 5] -1 -1

[1, 10, 9] -1 -1

[1, 3, 8] 1 1

[1, 4, 7] 1 1

[1, 3, 10] -1 -1

[1, 4, 10] -1 -1

[1, 8, 10] 1 -1

[1, 7, 10] 1 -1

[1, 7, 4] -1 1

[1, 8, 3] -1 1

[1, 10, 3] 1 1

[1, 10, 4] 1 1

[1, 10, 7] -1 -1

[1, 10, 8] -1 -1

Appendix E. Computer Code

E.1 Stabilizer Algorithm

```
#include<iostream>
#include<vector>
#include<string>
#include<fstream>
#include<cstdlib>

using namespace std;

ofstream out_file;

int det(vector <vector<int> > mat)
{
      return mat[0][0]*mat[1][1]*mat[2][2]+mat[1][0]*mat[2][1]*mat[0][2]
+mat[2][0]*mat[0][1]*mat[1][2]
      -mat[2][0]*mat[1][1]*mat[0][2]-mat[1][0]*mat[0][1]*mat[2][2]
-mat[0][0]*mat[2][1]*mat[1][2];
}

bool instabilizer (vector<int> vec, vector <vector<int> > mat)
{
      for (int i=0; i<mat.size(); i++)
      {
            if (vec==mat[i])
            {
```

```
                return true;
            }
        }
        return false;
}

vector<int> matrixmul (vector <vector<int> > mat, vector<int> vec)
{
        vector<int> ans;
        for (int i=0; i<mat.size(); i++)
        {
            int tally=0;
            for (int j=0; j<mat.size(); j++)
            {
                tally+=mat[j][i]*vec[j];
            }
            ans.push_back(tally);
        }
        return ans;
}

int main()
{
        string location;
        string name;
        while(true){
        cout << "Tell Me The Name\n\n";
```

```
	cin >> name;
	cin >> location;
	out_file.open(location);
	out_file << "/* We compute and store the stabilizer of "
<< name <<" acting on row. \n\n";
	vector< vector <int > > possible;

	/* This is where we input the cells */
	vector<int> entry;
	entry.push_back(1);entry.push_back(0);entry.push_back(0);
	possible.push_back(entry);
	entry[0]=0;
	entry[1]=1;
	possible.push_back(entry);
	entry[1]=0;entry[2]=1;
	possible.push_back(entry);
	entry[0]=1;entry[1]=1;entry[2]=0;
	possible.push_back(entry);
	entry[2]=1;
	possible.push_back(entry);


	int size =possible.size();
	for (int i=0;i<size;i++)
	{
		for (int j=0; j<3; j++)
		{
```

```
            entry[j]=-1*possible[i][j];
      }
      possible.push_back(entry);
}

for (int i=0; i< possible.size(); i++)
{
      for (int j=0; j<3; j++)
      {
            out_file << possible[i][j] << ", " ;
      }
      out_file << endl <<endl;
}

out_file << "*/\n\n";

system("pause");

vector<vector<vector <int> > > stabilizer;

vector<vector<int> > mat;
mat.push_back(possible[0]);
mat.push_back(possible[1]);
mat.push_back(possible[2]);

size=possible.size();
```

```
	for (int c1=0; c1<size; c1++)
	{
		vector<int> vec1=possible[c1];
		mat[0]=vec1;
		for (int c2=0;c2<size; c2++)
		{
			vector<int> vec2=possible[c2];
			mat[1]=vec2;
			for (int c3=0; c3<size; c3++)
			{
				vector<int> vec3=possible[c3];
				mat[2]=vec3;

				if (det(mat)==1)
				{
					bool isin = true;
					for (int i=0; i<possible.size(); i++)
					{
						isin= isin &&
instabilizer(matrixmul(mat,possible[i]),possible);
					}
					if (isin)
					{
						stabilizer.push_back(mat);
					}
				}
			}
```

```
            }
      }

      for (int i=0; i<stabilizer.size(); i++)
      {
            out_file << "Mat M"<< name <<"_"<< i+1 << " (";
            for (int j=0; j<3; j++)
            {
                  for (int k=0; k<3; k++)
                  {
                        out_file << stabilizer[i][k][j];
                        if(k!=2)
                        {out_file << ", ";}
                  }
                  if(j!=2)
                  {out_file << " ,";}
            }
            out_file << ");";
            out_file << "stab_" << name
<<".push_back(M"<<name<<"_"<< i+1 <<");";
            out_file << endl;
      }

      //out_file << "The order of the Stabilizer is "
<< stabilizer.size() << endl;

      system("pause");
```

```
      return 0;
}}
```

E.2 Calculating Orbits

E.2.1 Prologue.h.

```
/*This file contains useful information regarding the input, output,
and other common number theory functions;*/
/*
      in_file--file to read from
      out_file--file to print to;
      int n -- desired level;
      int k -- phi(n);

      vector<int> units -- vector of units
      vector<vector<int> > elem_stabilizer --
                                    a vector of vectors, the ith vector
                                 corresponds to the ith zero divisor
      vector<int> divisors_of_n -- vector of the divisors of n

      void make_divisors() -- fills the divisors of n vector

      vector<int> modn_inverse--vector of inverses modulo n;
      int gcd(int x, int y), int gcdd(int x, int y, int z),
      int gcddd --obvious
      int modn(int a) --reduces a modulo n and
returns a minimal nonzero output
```

```
      int modred(int p,int a) -- reduced a modulo p and returns a
   minimal nonzero output, p is prime
      int modn_inversef(int j) -- determines the "inverse" of j modulo n
      void make_inverse_vector() --
creates the vector of inverses modn_inverse
 also creates the vector of units
      vector<int> stab_of_a(int a) -- creates the stabilizer of a in Z/nZ;
      void make_element_stabilizers() -- creates the stabilizers

      int inversemodp(int i, int p) -- determines the inverse of i
 modulo p; p is prime
      int primechar -- the primecharacteristic we are
working over for convenience
      vector<int> primeinverses -- vector of inverses of
said prime characteristic

*/
char str[10], str2[10];
int print_flag;
ifstream in_file;               //file to read from;
ofstream out_file;             //file to print to;
int n;                              //n for gamma knot n
int k;                              //k=phi(n)
int prime_ell;                   //prime_ell for the Hecke_Operator

vector<int> units;
vector<vector<int> > elem_stabilizer;
```

```
vector<int> divisors_of_n;


void make_divisors()
{
      for(int i=1;i<=n/2;i++)
      {
            if((n/i)*i==n)
            {
                  divisors_of_n.push_back(i);
            }
      }
      k=divisors_of_n.size();
}


int identify_divisors(int x)
{
      for(int i=0;i<k;i++)
      {
            //cout << divisors_of_n[i] << " ";
            if(x==divisors_of_n[i])
            {
                  //cout << "\n\n\n";
                  return i;
            }
      }
      //cout << "\n" << x << "\n";
      //cout <<"\n\n\terror in identify divisors\n\n";
```

```
      return 1;
}

vector<int> modn_inverse;
//we are defining the "inverse"
//of a zero-divisor to be a minimalizer

int gcd(int x, int y)
{
      int t;
      while(y!=0)
      {
            t=y;
            y=x%y;
            x=t;
      }
      return abs(x);
}

int gcdd(int x, int y, int z)
{
      return gcd(gcd(x,y),z);
}

int gcddd(int x, int y, int z, int w)
{
      return gcd(gcdd(x,y,z),w);
```

```
}

int modn(int a)
{
	int c= a%n;
	if(c>=0)
	{
		return c;
	}
	return c+n;
}

int modred(int p, int a)
{
	int c= a%p;
	if(c>=0)
	{
		if(c<p)
		{
			return c;
		}
		return 0;
	}
	return c+p;
}

int modn_inversef(int j)
```

```
{
	if(gcd(j,n)==1)
	{
		for(int i=1;i<n;i++)
		{
			if(i*j%n==1)
			{
				return i;
			}
		}
	}
	else //the goal is to minimize j by a unit action,
	{ //so replace j with uj, uj minimal, uj=(j,n)
		int z=gcd(j,n);
		for(int i=1;i<n;i++)
		{
			if(gcd(i,n)==1)
			{
				if(modn(i*j)==z)
				{
					return i;
				}
			}
		}
	}
}
```

```
void make_inverse_vector()
{
	modn_inverse.push_back(0);
	modn_inverse.push_back(1);
	units.push_back(1);
	for(int i=2;i<n;i++)
	{
		if(gcd(i,n)==1)
		{
			units.push_back(i);
		}
		modn_inverse.push_back(modn_inversef(i));
	}
}

vector<int> stab_of_a(int a)
{
	vector<int> stab;
	for (int i=0; i<units.size();i++)
	{
		if(modn(units[i]*a)==a)
		{
			stab.push_back(units[i]);
		}
	}
	return stab;
}
```

```
void make_element_stabilizers()
{
	for(int i=0;i<divisors_of_n.size();i++)
	{
		elem_stabilizer.push_back(stab_of_a(divisors_of_n[i]));
	}
}

int inversemodp(int i,int p)
{
	int ans=1;
	while(((ans*i)%p) != 1)
	{
		ans = ((ans*i)%p);
	}
	return ans;
}

int primechar;
vector<int> primeinverses;

int max(int a, int b)
{
	if(a<b)
	{
		return b;
```

```
	}
	return a;
}
```

E.2.2 Generate_Orbits.h.

```
/*
	vector<int> calculate_orbits_thrust_generic(int position,
		vector<OMat> stabilizer, Three_tuple T) --
takes as input a tuple  and calculates the orbits.
Uses has_been_included
%%%%Needs the machinary and constructions from Preliminary_Calc%%%%
*/

vector<int> calculate_orbits_thrust_generic(int position,
vector<OMat> stabilizer, Three_tuple T)
//assumes that T is reduced;
{	//cout <<"\n\n";
	int locator=abc_to_i(T.one,T.two,T.three);
	bool orientated=true;
	vector<int> int_representation_of_orbit;
	vector<Bit_numb> orbit;
	/*if(isgood(abc_to_i(T.one,T.two,T.three))==false)
	{
		cout << "Buzz Buzz\n";
	}*/
	for(int i=0;i<stabilizer.size();i++)
	{
		Three_tuple TT=Mat_act_on_Tuple(stabilizer[i].M,T);
```

```
	TT.Reduce_tuple();
	if(TT.isEqual(T))
	{
		orientated=((orientated)&&(stabilizer[i].orient));
	}
	int j=abc_to_i(TT.one,TT.two,TT.three);
	if(isgood(j)!=true)
	{
		cout << "Bleeping Error\n";
	}
	if(has_been_included[j]==false)
	{
		Bit_numb J (j,stabilizer[i].orient,stabilizer[i].M);
		has_been_included[j]=true;
		orbit.push_back(J);
		int_representation_of_orbit.push_back(J.num);
		if(position==0)
		{//cout << j << "\t" << i << "\t";
			orbit_stabilizing_element[j]=i;
			//cout << orbit_stabilizing_element[j];
			//system("pause");
		}
	}
}

if(orientated==true) // 1 means orientatable
{
```

```
            for(int ii=0;ii<orbit.size();ii++)
            {
                  orbit_first_element[position][orbit[ii].num]=locator;
                  if(orbit[ii].orientation==true)
                  {
                        orbit_orientations[position][orbit[ii].num]=1;
                  }
                  else
                  {
                        orbit_orientations[position][orbit[ii].num]=-1;
                  }
            }
      }
      return int_representation_of_orbit;
}

//combined with the thrust function stores the orbits
vector< vector<int> > calculate_orbits_generic(int position)
{
      has_been_included=has_been_dummy_false; //sets it to false
      vector< vector<int> > orbit_list;
      vector<OMat> stabilizer= stabs[position];
      for(int i=1;i<=k*n*n+k*n+1;i++)
      {
            /*if(isgood(i)==false)
            {
            has_been_included[i]=true;
```

```
            }*/
            if(has_been_included[i]==false)
            {
                  vector<int> orbs =
calculate_orbits_thrust_generic(position, stabilizer, i_to_abc(i));
                  if(orbit_orientations[position][orbs[0]]!=0)
                  {
                        orbit_list.push_back(orbs);
                  }
            }
      }
      has_been_dummy_true=has_been_included;
      return orbit_list;
}

//sets orientable orbits in each dimension
//and creates a uniform row thereof.

void generate_orbits(int print_flag)
{
      orbits_of_generic.push_back(calculate_orbits_generic(0));
      orbits_of_generic.push_back(calculate_orbits_generic(1));
      orbits_of_generic.push_back(calculate_orbits_generic(2));
      orbits_of_generic.push_back(calculate_orbits_generic(3));

      if (print_flag!=0)
      {
```

```
            out_file << "Orbits of Sigma_0\n";
            for(int i=0;i<orbits_of_generic[0].size();i++)
            {
                  for(int ii=0;ii<orbits_of_generic[0][i].size();ii++)
                  {
                  i_to_abc(orbits_of_generic[0][i][ii]).OutThree_tuple();
                        out_file <<"\t"
<< orbit_orientations[0][orbits_of_generic[0][i][ii]] << "\t"
<< orbit_stabilizing_element[orbits_of_generic[0][i][ii]] << "\n";
                  }
                  out_file <<"\n";
            }

            out_file << "Orbits of Tau_2\n";
            for(int i=0;i<orbits_of_generic[1].size();i++)
            {
                  for(int ii=0;ii<orbits_of_generic[1][i].size();ii++)
                  {
                  i_to_abc(orbits_of_generic[1][i][ii]).OutThree_tuple();
                        out_file <<"\t"
<< orbit_orientations[1][orbits_of_generic[1][i][ii]] << "\n";
                  }
                  out_file <<"\n";
            }

            out_file << "Orbits of Tau_3\n";
            for(int i=0;i<orbits_of_generic[2].size();i++)
```

```
            {
                  for(int ii=0;ii<orbits_of_generic[2][i].size();ii++)
                  {
                  i_to_abc(orbits_of_generic[2][i][ii]).OutThree_tuple();
                        out_file <<"\t"
<< orbit_orientations[2][orbits_of_generic[2][i][ii]] << "\n";
                  }
                  out_file <<"\n";
            }

            out_file << "Orbits of Omega_2\n";
            for(int i=0;i<orbits_of_generic[3].size();i++)
            {
                  for(int ii=0;ii<orbits_of_generic[3][i].size();ii++)
                  {
                  i_to_abc(orbits_of_generic[3][i][ii]).OutThree_tuple();
                        out_file <<"\t"
<< orbit_orientations[3][orbits_of_generic[3][i][ii]] << "\n";
                  }
                  out_file <<"\n";
            }
      }
}

void find_suborbits(int position, int lowerposition) // finds the suborbits
{
      int intersection_position=find_intersection_position
```

```
(position,lowerposition);
      for(int i=0;i<orbits_of_generic[position].size();i++)
      {
            has_been_included=has_been_dummy_true;

            for(int j=0;j<orbits_of_generic[position][i].size();j++)
// turns the orbit off
            {
            has_been_included[orbits_of_generic[position][i][j]]=false;
            }

            vector<vector<int> > orbits_list;

            for(int iii=0;iii<orbits_of_generic[position][i].size();iii++)
            {
            if(has_been_included[orbits_of_generic[position][i][iii]]==false)
            {
            vector<int> A;
            for(int jj=0;jj<intersections[intersection_position].size();jj++)
            {
            Three_tuple TT=
Mat_act_on_Tuple(intersections[intersection_position]
[jj],i_to_abc(orbits_of_generic[position][i][iii]));
                  TT.Reduce_tuple();

                  //TT.PrintThree_tuple(); out_file << "\t"
//<< abc_to_i(TT.one,TT.two,TT.three)
```

```
<< "Coming from " << i << " times" << jj << "\n";
                    int marker=abc_to_i(TT.one,TT.two,TT.three);
                    if(has_been_included[marker]==false)
                    {
                    for(int jj=0;jj<orbits_of_generic[position][i].size();jj++)
                    {
                    if(marker==orbits_of_generic[position][i][jj])
                    {
                    A.push_back(orbits_of_generic[position][i][jj]);
                    }
                    }
                    has_been_included[marker]=true;
                    }
                    }
                          orbits_list.push_back(A);
                    }
              }
              orbits_of_generic_osn[intersection_position].
push_back(orbits_list);//
      }
}

void generate_suborbits(int print_flag)
{
      find_suborbits(0,1);
      find_suborbits(0,2);
      find_suborbits(1,3);
```

```
	find_suborbits(2,3);


	if(print_flag==1)
	{
		for(int i=0;i<4;i++)
		{
			if(i==0){out_file <<"Suborbits of Sigma_0 under Tau_2\n";}
			if(i==1){out_file <<"Suborbits of Sigma_0 under Tau_3\n";}
			if(i==2){out_file <<"Suborbits of Tau_2 under Omega_2\n";}
			if(i==3){out_file <<"Suborbits of Tau_3 under Omega_2\n";}
			for(int ii=0;ii<orbits_of_generic_osn[i].size();ii++)
			{
			for(int iii=0;iii<orbits_of_generic_osn[i][ii].size();iii++)
			{
			for(int j=0;j<orbits_of_generic_osn[i][ii][iii].size();j++)
			{
		i_to_abc(orbits_of_generic_osn[i][ii][iii][j]).OutThree_tuple();
		if(i==0){out_file <<"\t"
<< orbit_orientations[0][orbits_of_generic_osn[i][ii][iii][j]]
<<"\t" << orbit_orientations[1][orbits_of_generic_osn[i][ii][iii][j]]
<< "\n";}
				if(i==1){out_file <<"\t"
<< orbit_orientations[0][orbits_of_generic_osn[i][ii][iii][j]]
<<"\t" << orbit_orientations[2][orbits_of_generic_osn[i][ii][iii][j]]
<< "\n";}
				if(i==2){out_file <<"\t"
<< orbit_orientations[1][orbits_of_generic_osn[i][ii][iii][j]]
```

```
<<"\t" << orbit_orientations[3][orbits_of_generic_osn[i][ii][iii][j]]
<< "\n";}
                        if(i==3){out_file <<"\t"
<< orbit_orientations[2][orbits_of_generic_osn[i][ii][iii][j]]
<<"\t" << orbit_orientations[3][orbits_of_generic_osn[i][ii][iii][j]]
<< "\n";}
                              }
                              out_file <<"\n";
                        }
                        out_file <<"\n";
                  }
                  out_file<<"\n";
            }
      }
}
```

E.3 Constructing Hecke Operators

```
* Makes the Hecke_Matrices and their action on a chain of matrices.
      vector<Mat> Hecke_Matrices -- stores the Hecke_Matrices;
      void make_Hecke_Matrices(int prime) --
makes the set of matrices T_p,1;
      vector<Mat> Hecke_Action(vector<Mat> Chain)
-- multiples the Chain by the Hecke_Matrices
*/

vector<Mat> Hecke_Matrices;
```

```
void make_Hecke_Matrices (int prime) //right now assumes T_p,1
{
	vector<Mat> Hecke_Representatives;
	for(int i=0;i<prime;i++)
	{
		Mat M (1,0,0,0,1,0,0,0,prime);
		Mat N (1,0,0,0,prime,0,0,0,1);
		M.f=i;
		N.b=i;
		for(int j=0;j<prime;j++)
		{
			M.c=j;
			Hecke_Representatives.push_back(M);
		}
		Hecke_Representatives.push_back(N);
	}
	Mat M (prime,0,0,0,1,0,0,0,1);
	Hecke_Representatives.push_back(M);
	/*for(int i=0;i<Hecke_Representatives.size(); i++)
	{
		Hecke_Representatives[i].Print(); cout <<"\n\n";
	}
	cout << Hecke_Representatives.size();*/
	Hecke_Matrices=Hecke_Representatives;
}
```

E.4 Integer LLL Reduction Algorithm

```
/* The machinary to run LLL on the column space of a 3 by 3 matrix
*/

Mat Gram(Mat A)//computes the Gram matrix A^t A
{                          //assumes the basis is given as columns
      return product(transposeMat(A),A);
}

      // we are reusing k in a bad way since $k=\sigma(n)$. Oh Well.
      int k_max;
      long long int d[4];
      long long int lam[4][4];
      Mat H;
      Mat B;

void step1(), step2(), step3(), step4(), REDI(int k, int L), SWAPI(int k);

int EucDivQuo(int a, int b) //returns q where $a=qb+r$ for nonnegative r,
{
      if(b==0)
      {
            return 0;
      }
      if(b<0)
      {
            b=-b;
```

```
            a=-a;
      }
      int r= modred(b,a); //r is positive
      //cout << " a & " << abs(b) << " = " << r <<"\n\n";
      int q=(a-r)/b;
      if(q*b+r!=a)
      {
            cout << "Quotient Error \n";
      }
      return (a-r)/b; //should be q
}

void Print_LLL()
{
      cout << "B=";
      B.Print(); cout <<"\n\n";
      cout << "H=";
      H.Print(); cout <<"\n\n";
      cout << "lam_11= " << lam[1][1] << " lam_21= " << lam[2][1]
<< " lam_31= " << lam[3][1] << " lam_32= " << lam[3][2] << "\n";
      cout << "d0 = " << d[0] << " d1 = " << d[1] << " d2 = "
<< d[2] << " d3 = " << d[3] <<"\n";
      cout << "k= " << k << " kmax= " << k_max << "\n\n";
}

void CreateVariable(Mat L)
{
```

```
	B=L;
	d[0]=1;d[1]=1;d[2]=1;d[3]=1;
	lam[1][1]=1; lam[1][2]=1; lam[1][3]=1;
	lam[2][1]=1; lam[2][2]=1; lam[3][3]=1;
	step1();
	return;
}

void step1()
{
	k=2;
	k_max=1;
	d[0]=1;
	d[1]=B.dotprod(1,1);
	Mat Id (1,0,0,0,1,0,0,0,1);
	H=Id;
	if(print_flag==1)
	{
		cout << "After Step 1();\n";
		Print_LLL();
		system("pause");
	}
	step2();
	return;
}

void step2()
```

```
{
	if(k>k_max)
	{
		k_max=k;
		long long int u;
		for(int j=1;j<=k;j++)
		{
			u=B.dotprod(k,j);
			if(j-1>0)
			{
				for(int i=1;i<=j-1;i++)
				{
					//cout << d[i-1] << "\n\n";
					u=(d[i]*u-lam[k][i]*lam[j][i])/(d[i-1]);
				}
			}
			if(j<k)
			{
				lam[k][j]=u;
			}
			if(j==k)
			{
				d[k]=u;
			}
		}
	}
	if(print_flag==1)
```

```
	{
		cout << "After Step 2 \n";
		Print_LLL();
		system("pause");
	}
	step3();
	return;
}

void step3()
{
	REDI(k,k-1);
	if(print_flag==1)
	{
		cout << "After REDI(" << k<<"," <<k-1 <<")\n";
		Print_LLL();
		system("pause");
	}
	if(4*d[k]*d[k-2]<3*d[k-1]*d[k-1]-4*lam[k][k-1]*lam[k][k-1])
	{
		SWAPI(k);
		if(print_flag==1)
		{
			cout << "After SWAPI("<<k <<")\n";
			Print_LLL();
			system("pause");
		}
```

```
      k=max(2,k-1);


      step3();
}
else
{
      for(int i=1; i<=k-2; i++)
      {
            int m=k-2+1-i;
            REDI(k,m);
            if(print_flag==1)
            {
                  cout << "After REDI("<<k<<","<<m<<")\n";
                  Print_LLL();
                  system("pause");
            }
      }
}
k++;
if(print_flag==1)
{
      cout << "After Step 3\n";
      Print_LLL();
      system("pause");
}
step4();
return;
```

```
}


void step4()
{
      if(k<=3)
      {
            step2();
      }
      //Print_LLL();
      return;
}


void REDI(int k, int L)
{
      if(2*abs(lam[k][L])>d[L])
      {
            int q=EucDivQuo(2*lam[k][L]+d[L],2*d[L]);
            H.Columnadd(k,H,L,-q);
            B.Columnadd(k,B,L,-q);
            lam[k][L]=lam[k][L]-q*d[L];
            for(int i=1; i<=L-1; i++)
            {
                  lam[k][i]=lam[k][i]-q*lam[L][i];
            }
      }
}
```

```
void SWAPI(int k)
{
      H.swapcolumns(k,k-1);
      B.swapcolumns(k,k-1);
      if(k>2)
      {
            for(int j=1;j<=k-2;j++)
            {
                  long long int v=lam[k][j];
                  lam[k][j]=lam[k-1][j];
                  lam[k-1][j]=v;
            }
      }
      long long int lamb=lam[k][k-1];
      long long int Be=(d[k-2]*d[k]+lamb*lamb)/d[k-1];
      for(int i=k+1; i<=k_max; i++)
      {
            long long int t=lam[i][k];
            lam[i][k]=((d[k]*lam[i][k-1]-lamb*t)/d[k-1]);
            lam[i][k-1]=(Be*t+lamb*lam[i][k])/d[k];
      }
      d[k-1]=Be;
}

float inner_product(float a, float b, float c)
{
      return a*a+b*b+c*c;
```

```
}

bool test_LLL(Mat B)
{
      float mu_11=(B.a*B.a+B.d*B.d+B.g*B.g*1.)/(B.a*B.a+B.d*B.d+B.g*B.g*1.);
      //cout << "mu_11= " << mu_11 << "\n";
      float mu_21=(B.a*B.b+B.d*B.e+B.g*B.h*1.)/(B.a*B.a+B.d*B.d+B.g*B.g*1.);
      //cout << "mu_21= " << mu_21 << "\n";
      float mu_31=(B.a*B.c+B.d*B.f+B.g*B.i*1.)/(B.a*B.a+B.d*B.d+B.g*B.g*1.);
      //cout << "mu_31= " << mu_31 << "\n";

      //second column
      float bp,ep,hp; bp=B.b-mu_21*B.a; ep=B.e-mu_21*B.d; hp=B.h-mu_21*B.g;
      float mu_22=(bp*B.b+ep*B.e+hp*B.h*1.)/(1.*bp*bp+ep*ep+hp*hp);
      //cout << "mu_22= " << mu_22 << "\n";
      float mu_32=(bp*B.c+ep*B.f+hp*B.i*1.)/(1.*bp*bp+ep*ep+hp*hp);
      //cout << "mu_32= " << mu_32 << "\n";

      float cp,fp,ip; cp=B.c-mu_31*B.a-mu_32*bp;
                              fp=B.f-mu_31*B.d-mu_32*ep;
                              ip=B.i-mu_31*B.g-mu_32*hp;
      bool flag;
      flag=((abs(mu_21<=.5))&&((abs(mu_32<=.5))&&(abs(mu_31<=.5))));
//conditions on the mu;
      //cout << flag <<"\n\n";
      if(flag)
      {
```

```
            if(abs((inner_product(bp,ep,hp))>=
                  abs(((.75-mu_21*mu_21)*inner_product(B.a,B.d,B.g))))&&
                  (abs(inner_product(cp,fp,ip))>=
                  abs(((.75-mu_32*mu_32)*inner_product(bp,ep,hp)))))
            {
                  return true;
            }
            //conditions on the columns
      }
      return false;
}

Mat LLL_Reduce_A_column (Mat A) //takes A and outputs H where $AH$ is LLL
{
      CreateVariable(A);
      return H;
}

void LLL_Reduce_A_row (Mat A) //takes A and outputs H
{                        //where the rows of $HA$ are LLL reduced
      Mat B=A;
      B.Transpose();
      CreateVariable(B);
      H.Transpose();
      return; //H;
}
```

E.5 Candidate Selection Code

```
int test_Candidate(int i, Mat A)
{
      int det_A=abs(A.Determinant());
      if(i==1)
      {//test e1
            Mat B1=A;
            B1.a=1;
            B1.d=0;
            B1.g=0;
            int b1_det=abs(B1.Determinant());
            //cout <<"b1_det= " <<b1_det <<"\n";
            if(b1_det<det_A)
            {
                  Mat B2=A;
                  B2.b=1;
                  B2.e=0;
                  B2.h=0;
                  int b2_det=abs(B2.Determinant());
                  //cout <<"b2_det= " <<b2_det <<"\n";
                  if(b2_det<det_A)
                  {
                        Mat B3=A;
                        B3.c=1;
                        B3.f=0;
                        B3.i=0;
                        int b3_det=abs(B3.Determinant());
```

```
                        //cout <<"b2_det= " <<b3_det <<"\n";
                        if(b3_det<det_A)
//e1 is a candidate; but, maybe not the best
                        {
                              return max(max(b1_det,b2_det),b3_det);
                        }
                  }
            }
            return det_A;
      }
      if(i==2)
      {//test e2
            Mat B1=A;
            B1.a=0;
            B1.d=1;
            B1.g=0;
            int b1_det=abs(B1.Determinant());
            //cout <<"b1_det= " <<b1_det <<"\n";
            if(b1_det<det_A)
            {
                  Mat B2=A;
                  B2.b=0;
                  B2.e=1;
                  B2.h=0;
                  int b2_det=abs(B2.Determinant());
                  //cout <<"b2_det= " <<b2_det <<"\n";
                  if(b2_det<det_A)
```

```
                {
                        Mat B3=A;
                        B3.c=0;
                        B3.f=1;
                        B3.i=0;
                        int b3_det=abs(B3.Determinant());
                        //cout <<"b3_det= " <<b3_det <<"\n";
                        if(b3_det<det_A)
//e2 is a candidate; but, maybe not the best
                        {
                              return max(max(b1_det,b2_det),b3_det);
                        }
                }
          }
          return det_A;
      }
      if(i==3)
      {//test e3
            Mat B1=A;
            B1.a=0;
            B1.d=0;
            B1.g=1;
            int b1_det=abs(B1.Determinant());
            //cout <<"b1_det= " <<b1_det <<"\n";
            if(b1_det<det_A)
            {
                  Mat B2=A;
```

```
                B2.b=0;
                B2.e=0;
                B2.h=1;
                int b2_det=abs(B2.Determinant());
                //cout <<"b2_det= " <<b2_det <<"\n";
                if(b2_det<det_A)
                {
                      Mat B3=A;
                      B3.c=0;
                      B3.f=0;
                      B3.i=1;
                      int b3_det=abs(B3.Determinant());
                      //cout <<"b3_det= " <<b3_det <<"\n";
                      if(b3_det<det_A)
//e3 is a candidate; but, maybe not the best
                      {
                            return max(max(b1_det,b2_det),b3_det);
                      }
                }
            }
            return det_A;
      }
      cout <<"Error, invalid input in CandidateTesting\n\n";
      return 0;
}

int min(int a, int b)
```

```
{
      if(a<b)
      {
            return a;
      }
      return b;
}

int min_position(int a, int b, int c)
{
      a=abs(a);
      b=abs(b);
      c=abs(c);
      int minimum=min(min(a,b),c);
      if(minimum==a)
      {
            return 1;
      }
      if(minimum==b)
      {
            return 2;
       }
      return 3;
}

int FindCandidate(Mat K) //assumes K is LLL-reduced
{
```

```
      return min_position
(test_Candidate(1,K),test_Candidate(2,K),test_Candidate(3,K));
}
```

E.6 HNF Reduction Algorithm

```
Mat swaprows (Mat A, int row1, int row2)
{
      long long int ap,bp,cp,dp,ep,fp,gp,hp,ip;
      if (row1==0 && row2==1)
      {ap=A.d;bp=A.e;cp=A.f;dp=A.a;ep=A.b;fp=A.c;gp=A.g;hp=A.h;ip=A.i;}
      if (row1==0 && row2==2)
      {ap=A.g;bp=A.h;cp=A.i;dp=A.d;ep=A.e;fp=A.f;gp=A.a;hp=A.b;ip=A.c;}
      if (row1==1 && row2==2)
      {ap=A.a;bp=A.b;cp=A.c;dp=A.g;ep=A.h;fp=A.i;gp=A.d;hp=A.e;ip=A.f;}
      if (row1==row2)
      {ap=A.a;bp=A.b;cp=A.c;dp=A.d;ep=A.e;fp=A.f;gp=A.g;hp=A.h;ip=A.i;}
      Mat C (ap,bp,cp,dp,ep,fp,gp,hp,ip);
      return C;
}


Mat HNFprelim (Mat X)
{
      Mat A=X;
      if(A.a==0)
      {
            A=swaprows(A,0,2);
      }
```

```
	if(A.a==0)
	{
		A=swaprows(A,0,1);
	}
	return A;
}

Mat HNFstageonea (Mat X)
//creates a matrix of the form (a,b,c,d,e,f,0,h,i) from (a,b,c,d,e,f,g,h,i)
{
	Mat A=X;
	while(A.g!=0)
	{
		if(A.a<0)
		{A=multrowbynegone(A,0);}
		if(A.g<0)
		{A=multrowbynegone(A,1);}
		if(A.a<=A.g)
		{ int q; q=A.g/A.a; A.g-=q*A.a; A.h-=q*A.b; A.i-=q*A.c;}
		else{A=swaprows(A,0,2);}
	}
	return A;
}

Mat HNFstageoneb (Mat X)
//creates a matrix of the form (a,b,c,0,e,f,0,h,i) from (a,b,c,d,e,f,0,h,i)
{
```

```
	Mat A=X;
	while(A.d!=0)
	{
		if(A.a<0)
		{A=multrowbynegone(A,0);}
		if(A.d<0)
		{A=multrowbynegone(A,1);}
		if(A.a<=A.d)
		{ int q; q=A.d/A.a; A.d-=q*A.a; A.e-=q*A.b; A.f-=q*A.c;}
		else{A=swaprows(A, 0, 1);}
	}
	return A;
}

Mat HNFstagetwo (Mat X)
//creates a matrix of the form (a,b,c,0,e,f,0,0,i) from (a,b,c,0,e,f,0,h,i)
{
	Mat A=X;
	while(A.h!=0)
	{
		if(A.e<0)
		{A=multrowbynegone(A,1);}
		if(A.h<0)
		{A=multrowbynegone(A,2);}
		if(A.e<=A.h)
		{ int q; q=A.h/A.e; A.h-=q*A.e; A.i-=q*A.f;}
		else{A=swaprows(A,1,2);};
```

```
	}
	return A;
}

Mat HNFstagethree (Mat X)
//HNF of a matrix of the form (a,b,c,0,e,f,0,0,i)
//with all nonnegative entries
{
	Mat A=X;
	//A.Print(); cout <<endl << "Check1" << endl;
	if(A.a<0)
	{A=multrowbynegone(A,0);}
	if(A.e<0)
	{A=multrowbynegone(A,1);}
	if(A.i<0)
	{A=multrowbynegone(A,2);}

	if(A.i!=0)
	{
		while(A.f<0)
		{A.f+=A.i;}
		while(A.f>=A.i)
		{A.f-=A.i;}
	}

	if(A.e!=0)
	{
```

```
            while(A.b<0)
            {A.b+=A.e;A.c+=A.f;}
            while(A.b>=A.e)
            {A.b-=A.e;A.c-=A.f;}
      }
      if(A.i!=0)
      {
            while(A.c<0)
            {A.c+=A.i;}
            while(A.c>=A.i)
            {A.c-=A.i;}
      }
      //A.Print(); cout <<endl << "Check2" << endl;
      return A;
}

Mat HNF(Mat X)
{
      Mat A=X;
      //cout << "Prelim" << endl;
      A=HNFprelim(A);
      //A.Print(); cout << endl << endl << "StaeoneA\n";
      A=HNFstageonea(A);
      //A.Print(); cout << endl <<endl <<"StageOneB\n";
      A=HNFstageoneb(A);
      //A.Print(); cout << endl <<endl <<"StageTwo\n";
      A=HNFstagetwo(A);
```

```
	//A.Print(); cout << endl <<endl <<"StageThree\n";
	A=HNFstagethree(A);
	return A;
}

Mat HNFfindU(Mat X, Mat A) // Finds U where UX=A, A is HNF
{
	Mat B=X.Inverse();
	int k=-X.Determinant();
	Mat U = product(A,B);
	U.a/=k;U.b/=k;U.c/=k;U.d/=k;U.e/=k;U.f/=k;U.g/=k;U.h/=k;U.i/=k;
	return U;
}
```

Bibliography

[1] Allison, G., Ash, A., and Conrad, E. Galois representations, Hecke operators, and the mod-p cohomology of $\mathrm{GL}(3, \mathbf{Z})$ with twisted coefficients. *Experiment. Math. 7*, 4 (1998), 361–390.

[2] Ash, A. Cohomology of congruence subgroups $\mathrm{SL}(n, \mathbb{Z})$. *Math. Ann. 249*, 1 (1980), 55–73.

[3] Ash, A. Galois representations attached to mod p cohomology of $\mathrm{GL}(n, \mathbf{Z})$. *Duke Math. J. 65*, 2 (1992), 235–255.

[4] Ash, A. Unstable cohomology of $\mathrm{SL}(n, \mathcal{O})$. *J. Algebra 167*, 2 (1994), 330–342.

[5] Ash, A. Monomial Galois representations and Hecke eigenclasses in the mod-p cohomology of $\mathrm{GL}((p-1), \mathbf{Z})$. *Math. Ann. 315*, 2 (1999), 263–280.

[6] Ash, A. Direct sums of mod p characters of $\mathrm{GAL}(\overline{\mathbb{Q}}/\mathbb{Q})$ and the homology of $GL(n, \mathbb{Z})$. *Comm. Algebra 41*, 5 (2013), 1751–1775.

[7] Ash, A., and Doud, D. Relaxation of the strict parity for reducible galois representations attached to the cohomology of $\mathrm{GL}(3, \mathbb{Z})$. in review.

[8] Ash, A., and Doud, D. Highly reducible galois representations and the homology of $\mathrm{GL}_n(\mathbb{Z}$. *Proc. AMS* (2014), to appear.

[9] Ash, A., and Doud, D. Reducible Galois representations and the homology of $\mathrm{GL}(3, \mathbb{Z})$. *Int. Math. Res. Not. IMRN*, 5 (2014), 1379–1408.

[10] Ash, A., Doud, D., and Pollack, D. Galois representations with conjectural connections to arithmetic cohomology. *Duke Math. J. 112*, 3 (2002), 521–579.

[11] Ash, A., Grayson, D., and Green, P. Computations of cuspidal cohomology of congruence subgroups of $\mathrm{SL}(3, \mathbf{Z})$. *J. Number Theory 19*, 3 (1984), 412–436.

[12] Ash, A., Gunnells, P. E., and McConnell, M. Cohomology of congruence subgroups of $\mathrm{SL}_4(\mathbb{Z})$. *J. Number Theory 94*, 1 (2002), 181–212.

[13] Ash, A., Gunnells, P. E., and McConnell, M. Cohomology of congruence subgroups of $\mathrm{SL}(4, \mathbb{Z})$. II. *J. Number Theory 128*, 8 (2008), 2263–2274.

[14] Ash, A., Gunnells, P. E., and McConnell, M. Cohomology of congruence subgroups of $\mathrm{SL}_4(\mathbb{Z})$. III. *Math. Comp. 79*, 271 (2010), 1811–1831.

[15] Ash, A., Gunnells, P. E., and McConnell, M. Torsion in the cohomology of congruence subgroups of $\mathrm{SL}(4, \mathbb{Z})$ and Galois representations. *J. Algebra 325* (2011), 404–415.

[16] Ash, A., Gunnells, P. E., and McConnell, M. Resolutions of the Steinberg module for $GL(n)$. *J. Algebra 349* (2012), 380–390.

[17] Ash, A., and McConnell, M. Experimental indications of three-dimensional Galois representations from the cohomology of $\mathrm{SL}(3, \mathbf{Z})$. *Experiment. Math. 1*, 3 (1992), 209–223.

[18] Ash, A., Pollack, D., and Sinnott, W. A_6-extensions of $\mathbb{Q}$ and the mod p cohomology of $\mathrm{GL}_3(\mathbb{Z})$. *J. Number Theory 115*, 1 (2005), 176–196.

[19] Ash, A., Pollack, D., and Soares, D. $\mathrm{SL}_3(\mathbb{F}_2)$-extensions of $\mathbb{Q}$ and arithmetic cohomology modulo 2. *Experiment. Math. 13*, 3 (2004), 298–307.

[20] Ash, A., and Sinnott, W. An analogue of Serre's conjecture for Galois representations and Hecke eigenclasses in the mod p cohomology of $\mathrm{GL}(n, \mathbf{Z})$. *Duke Math. J. 105*, 1 (2000), 1–24.

[21] Barnes, E. S. The perfect and extreme senary forms. *Canad. J. Math. 9* (1957), 235–242.

[22] Barvinok, A. I. A polynomial time algorithm for counting integral points in polyhedra when the dimension is fixed. *Math. Oper. Res. 19*, 4 (1994), 769–779.

[23] Borel, A., and Serre, J.-P. Cohomologie d'immeubles et de groupes S-arithmétiques. *Topology 15*, 3 (1976), 211–232.

[24] Bosma, W., Cannon, J., and Playoust, C. The Magma algebra system. I. The user language. *J. Symbolic Comput. 24*, 3-4 (1997), 235–265. Computational algebra and number theory (London, 1993).

[25] Brown, K. S. *Cohomology of groups*, vol. 87 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1994. Corrected reprint of the 1982 original.

[26] Cohen, H. *A course in computational algebraic number theory*, vol. 138 of *Graduate Texts in Mathematics*. Springer-Verlag, Berlin, 1993.

[27] Cremona, J. E. Hyperbolic tessellations, modular symbols, and elliptic curves over complex quadratic fields. *Compositio Math. 51*, 3 (1984), 275–324.

[28] Dembélé, L. Explicit computations of Hilbert modular forms on $\mathbb{Q}(\sqrt{5})$. *Experiment. Math. 14*, 4 (2005), 457–466.

[29] Diamond, F., and Shurman, J. *A first course in modular forms*, vol. 228 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 2005.

[30] Doud, D., and Ricks, R. LLL reduction and a conjecture of Gunnells. *Proc. Amer. Math. Soc. 138*, 2 (2010), 409–415.

[31] Fuchs, U. Different realizations of the upper half plane $\mathbb{H}$. Available at `http://www.math.ethz.ch/education/bachelor/seminars/ws0607/modular-forms/Different_realisations_of_the_upper_half_plane.pdf`, accessed June 2014.

[32] Gelfand, I. M., Gindikin, S. G., and Graev, M. I. *Selected topics in integral geometry*, vol. 220 of *Translations of Mathematical Monographs*. American Mathematical Society, Providence, RI, 2003. Translated from the 2000 Russian original by A. Shtern.

[33] Gunnells, P. E. Computing Hecke eigenvalues below the cohomological dimension. *Experiment. Math. 9*, 3 (2000), 351–367.

[34] Gunnells, P. E., Hajir, F., and Yasaki, D. Modular forms and elliptic curves over the field of fifth roots of unity. *Exp. Math. 22*, 2 (2013), 203–216. With an appendix by Mark Watkins.

[35] Hatcher, A. *Algebraic topology*. Cambridge University Press, Cambridge, 2002.

[36] Khare, C. Serre's modularity conjecture: the level one case. *Duke Math. J. 134*, 3 (2006), 557–589.

[37] Khare, C., and Wintenberger, J.-P. Serre's modularity conjecture. I. *Invent. Math. 178*, 3 (2009), 485–504.

[38] Khare, C., and Wintenberger, J.-P. Serre's modularity conjecture. II. *Invent. Math. 178*, 3 (2009), 505–586.

[39] Kilford, L. J. P. *Modular forms*. Imperial College Press, London, 2008. A classical and computational introduction.

[40] Krieg, A. Hecke algebras. *Mem. Amer. Math. Soc. 87*, 435 (1990), x+158.

[41] Lam, T. Y. *A first course in noncommutative rings*, second ed., vol. 131 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 2001.

[42] Lee, R., and Szczarba, R. H. On the homology and cohomology of congruence subgroups. *Invent. Math. 33*, 1 (1976), 15–53.

[43] Lenstra, A. K., Lenstra, Jr., H. W., and Lovász, L. Factoring polynomials with rational coefficients. *Math. Ann. 261*, 4 (1982), 515–534.

[44] Manin, J. I. Parabolic points and zeta functions of modular curves. *Izv. Akad. Nauk SSSR Ser. Mat. 36* (1972), 19–66.

[45] McConnell, M. Classical projective geometry and arithmetic groups. *Math. Ann. 290*, 3 (1991), 441–462.

[46] Minkowski, H. *Geometrie der Zahlen*. Bibliotheca Mathematica Teubneriana, Band 40. Johnson Reprint Corp., New York-London, 1968.

[47] Newman, M. *Integral matrices*. Academic Press, New York-London, 1972. Pure and Applied Mathematics, Vol. 45.

[48] Rhie, Y. H., and Whaples, G. Hecke operators in cohomology of groups. *J. Math. Soc. Japan 22* (1970), 431–442.

[49] Schenkman, E. The basis theorem for finitely generated abelian groups. *Amer. Math. Monthly 67* (1960), 770–771.

[50] Scholze, P. Perfectoid spaces. *Publ. Math. Inst. Hautes Études Sci. 116* (2012), 245–313.

[51] Schürmann, A. Enumerating perfect forms. In *Quadratic forms—algebra, arithmetic, and geometry*, vol. 493 of *Contemp. Math.* Amer. Math. Soc., Providence, RI, 2009, pp. 359–377.

[52] Schwermer, J. Holomorphy of Eisenstein series at special points and cohomology of arithmetic subgroups of $\mathrm{SL}_n(\mathbf{Q})$. *J. Reine Angew. Math. 364* (1986), 193–220.

[53] Serre, J.-P. *A course in arithmetic*. Springer-Verlag, New York-Heidelberg, 1973. Translated from the French, Graduate Texts in Mathematics, No. 7.

[54] Serre, J.-P. Sur les représentations modulaires de degré 2 de $\mathrm{Gal}(\overline{\mathbf{Q}}/\mathbf{Q})$. *Duke Math. J. 54*, 1 (1987), 179–230.

[55] Sikirić, M. D., Schürmann, A., and Vallentin, F. Classification of eight-dimensional perfect forms. *Electron. Res. Announc. Amer. Math. Soc. 13* (2007), 21–32 (electronic).

[56] Soulé, C. Cohomologie de $SL_3(Z)$. *C. R. Acad. Sci. Paris Sér. A-B 280*, 5 (1975), Ai, A251–A254.

[57] Stein, W. *Modular forms, a computational approach*, vol. 79 of *Graduate Studies in Mathematics*. American Mathematical Society, Providence, RI, 2007. With an appendix by Paul E. Gunnells.

[58] The PARI Group. *PARI/GP version* `2.7.0`. Bordeaux, 2014. available from `http://pari.math.u-bordeaux.fr/`.

[59] van Geemen, B., van der Kallen, W., Top, J., and Verberkmoes, A. Hecke eigenforms in the cohomology of congruence subgroups of $\mathrm{SL}(3, \mathbf{Z})$. *Experiment. Math. 6*, 2 (1997), 163–174.

[60] Voronoi, G. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. *J. Reine Angew. Math. 133* (1908), 97–178.

[61] Weibel, C. A. *An introduction to homological algebra*, vol. 38 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, Cambridge, 1994.