# Enumeration of groups of prime-power order

Brett Edward Witty

August 2006

A thesis submitted for the degree of Doctor of Philosophy

of the Australian National University

# Declaration

The work in this thesis is my own except where otherwise stated.

Brett Witty

# Acknowledgements

First and foremost I would like to thank my supervisor, Professor Mike F. Newman for teaching me an unbelievable amount of mathematics, in its exploration, communication and understanding.

I would like to acknowledge the support, financial and otherwise, from the Mathematics Sciences Institute and the Australian National University. Special thanks to the administrative staff for being exceedingly helpful and supportive throughout my PhD.

Thanks to the mathematics graduate students, especially Bob, Daniel, Greg, Huy, Kathie, Mike and Norman, for making lunchtimes interesting and afternoon tea competitive.

Thanks to my "Brisbane crew": Danielle for supporting me, Duncan for having boundless interest in my work, and Joel for his camaraderie.

Thanks also to my housemates Rowan, Jon and Dima for support, cheese and board games.

My final thanks go to my family who might have not understood what I have worked on, but steadily followed my progress, and encouraged and supported me nonetheless.

# Abstract

Finite group theorists have been interested in counting groups of prime-power order, as a preliminary step to counting groups of any finite order and to assist in explicitly listing such groups. In 1960, G. Higman conjectured [21] that the number of groups of prime-power order $p^n$, for fixed $n$ and varying $p$, is a function of a particular form, called *polynomial on residue classes* (PORC). In this paper he proved that a certain class of groups of prime-power order satisfy this conjecture but did not furnish explicit results.

The aim of this thesis is to reinterpret Higman's proof to create and implement an algorithm to find explicit results in this known case. The first chapter of the thesis will outline some of the history behind the problem and introduce it in the context suggested by Higman. The subsequent chapters will describe the algorithm to solve this case. Each chapter will cover a major part of the algorithm and will describe the relevant theory and the algorithms involved.

We will describe some of the details regarding implementation, and list results obtained by this implementation.

# Contents

# Notation and terminology

## Notation

In this section we will outline notation that is used throughout this thesis, both in general and chapter-by-chapter. It will be in the form *notation, concept, page reference* (where applicable).

### Universal notation

The following notation follows the usual standards in mathematics and is universal throughout this thesis. As such we do not provide page numbers or references.

$\mathbb{Z}$, Ring of integers

$\mathbb{Q}$, Field of rational numbers

$\mathbb{C}$, Field of complex numbers

$K^*$, The multiplicative group of a field $K$

$\overline{K}$, The algebraic closure of a field $K$

$\mathbb{F}_p$, Finite field of order $p$, with $p$ prime

$\mathbb{F}_q$, Finite field of order $q$, where $q = p^d$

$\mathrm{GL}(n; K)$, $n$-by-$n$ invertible matrices over the field $K$

$\mathrm{GL}(n_1, \ldots, n_r\,; K)$, Direct sum of $\mathrm{GL}(n_i\,; K)$

$|\lambda|$, Sum of the parts of the partition $\lambda$

### Notation for Chapter 1

$f(n,p)$, Number of $p$-groups of order $p^n$

### Notation for Chapter 2

$f_2(n,p)$, For fixed $n$ and $p$, the number of exponent-$p$ class two groups of order $p^n$, pg 7

$g(r,s,p)$, For fixed $r,s,p$, the number of $r$-generator exponent-$p$ class two groups $G$ such that $[G,G]G^p$ has order $p^s$, pg 8

$f_2(n;p)$, For fixed $n$ and variable prime $p$, the number of exponent-$p$ class two groups of order $p^n$, pg 8

$g(r,s;p)$, For fixed $r$ and $s$ and variable prime $p$, number of $r$-generator exponent-$p$ class two groups $G$ such that $[G,G]G^p$ has order $p^s$, pg 8

$h(r,s;p)$, For fixed $r$ and $s$, and variable prime $p$, the number of orbits of subspace of codimension at most $s$ under the induced action of $\mathrm{GL}(r;p)$ on $(\mathbb{F}_p)^r \oplus (F_p)^r \wedge (\mathbb{F}_p)^r$, pg 13

$V \wedge W$, Exterior product of vector spaces $V$ and $W$, pg 131 of [34]

$C_2(g)$, Second compound matrix of $g$, pg 11

$\Gamma$, Polynomial representation

$\Gamma^{(s)}$, Induced (polynomial) representation of $\mathrm{GL}(r,s;\mathbb{F}_p)$ on subspaces of $V \oplus V \wedge V$ of codimension $s$, where $V = (\mathbb{F}_p)^r$, pg 14

$\chi(g)$, Number of points fixed by the action of $g$, pg 11

$\mathrm{vec}(A)$, The column vector obtained by stacking the columns of the matrix $A$ on top of each other, pg 14

$A \otimes B$, Kronecker product of two matrices, pg 14

### Notation for Chapter 3

$C(f)$, Companion matrix of the polynomial $f$, pg 20

$J_\lambda(\alpha)$, Jordan block of $\alpha$, pg 20

$\mathcal{T}_n$, The set of types of size $n$, pg 24

$(d, \lambda, l)$, Type-parameter with degree $d$, partition $\lambda$ and location $l$, pg 24

### Notation for Chapter 4

$\mathcal{A}$, Guiding member of a species, pg 39

$\mathcal{X}$, A countably infinite set of algebraically independent elements from $\mathbb{C}$, pg 39

$\mathcal{A}_\tau$, Guiding member of a type, pg 47

$T$, Master predegeneracy set, pg 41

$E_\tau$, Master degeneracy set, pg 45

$\chi(g_S)$, Number of fixed points of an element in a conjugacy class with degeneracy set $S$, pg 50

$\mathcal{H}$, Group representing the equivalence of degeneracy sets, pg 55

### Notation for Chapter 5

$d(S) = d(S, \tau; p)$, Number of specializations satisfying equations and inequalities given by degeneracy set $S$, pg 58

$c(S)$, Number of specializations satisfying just equations given by degeneracy set $S$, pg 59

$\Lambda$, The ring $\mathbb{Z}[\sigma]/\langle \sigma^d - 1 \rangle$, pg 60

$M_p$, The ring $\mathbb{Z}[\sigma]/\langle \sigma^d - 1, \sigma - p \rangle$, pg 61

$\mathrm{Tor}(X)$, Torsion submodule of $X$, pg 62

### Notation for Chapter 6



### Notation for Chapter 7



## Guide to pseudocode

For the important algorithms of this thesis, pseudocode for each will be provided. The actual code used in implementing this algorithm will not be included as it is approximately 120 pages in length. The algorithm was implemented in GAP 4.4.7, and the pseudocode will be presented in a similar fashion.

Functions are typeset as `Function`$(n)$. These are usually equivalent to those found in GAP (both in name and usage). The following is a list of the functions required in this thesis:

- `ClassTwo`$(p, d, s$ : parameters$)$ : The number of $d$-generator $p$-groups of $p$-class two of order $p^{d+s}$. The variable "parameters" specifies certain options. For our purposes, we will always consider it to be set to `Exponent := true`. The associated algorithm can be found in [15] and has only been implemented in MAGMA [7].

- `Combinations`$(S, n)$ : returns all subsets of $S$ of length $n$;

- `DivisorsInt`$(n)$ : returns the divisors of the integer $n$;
- `Partitions`$(n)$ : returns the list of all unordered, unrestricted integer partitions of $n$;
- `UnorderedTuples`$(S, n)$ : returns all unordered $n$-tuples of elements from a set $S$, with repetition allowed.

We make no assumptions about the practical efficiencies of these functions other than that they are "reasonable" implementations. Faster algorithms may exist for specialized uses for these functions.

# Chapter 1

# History

When studying finite groups, it would be desirable to be able to determine all groups of a given order. By "determine" we can mean several different things: if we find an explicit description for every group of that order, we shall call that *listing* the groups. If we already have such a list and just want the number of entries, we call that *counting* groups of a given order. Finally, if we want an implicit count of the groups of a given order *without using a list of such groups* we call this *enumerating* the groups. We allow enumeration to cover more general situations where the count is functional; that is, an enumeration of groups of order $n$ may be given as a function depending on $n$. We will endeavour to maintain this distinction between listing, counting and enumerating throughout this thesis. Note that this distinction is not always maintained in the literature.

It would be desirable to be able to have a listing or an enumeration of groups of arbitrary order, although both tasks are exceedingly difficult. One would expect that restricting the types of groups listed or enumerated would simplify this task. We will restrict our attention to groups of prime-power order, hereafter known as *p-groups*.

We will be most interested in the problem of enumerating $p$-groups and thus will not examine the history of listing groups, $p$-groups or otherwise. A detailed history on the determination of finite groups can be found in [43]. Nevertheless, we will refer to some results involved in the listing of $p$-groups as they motivate the enumeration

problem.

## 1.1 Enumeration of $p$-groups

Enumerating $p$-groups of a given order seems easier than listing them as in the latter situation we require an irredundant and complete list of explicit presentations of all isomorphism types of groups of that order. Nevertheless, the enumeration of $p$-groups is a difficult problem by itself. Most of the initial work in enumerating $p$-groups involved taking the list of $p$-groups of that order and counting the groups. If the lists were parametrised by the prime $p$, then from the results proving the correctness of these lists we could obtain as a corollary a counting function, which can be viewed as a form of enumeration. We will be more interested in the approaches where the result genuinely does not require a list of the groups of a given size, parametrised or not. Even then, enumeration can take two routes: asymptotic enumeration and exact enumeration.

Let $f(n,p)$ denote the number of isomorphism types of groups of order $p^n$. Asymptotic enumeration asks for the asymptotic behaviour of the function $f(n,p)$. The first asymptotic results for $p$-groups were found by G. Higman in 1960 [20]. Higman showed that

$$f(n,p) = p^{An^3},$$

where $A$ depends on $p$ and $n$, and $\frac{2}{27} - \epsilon_n \leq A \leq \frac{2}{15} + \epsilon_n$, where $\epsilon_n$ depends only on $n$ and tends to zero as $n$ tends to infinity. C. C. Sims improved this result [47], showing that $A = \frac{2}{27} + O(n^{-1/3})$. This result was later extended to arbitrary finite groups by various authors [11, 36, 38, 44]. A more comprehensive review on the development of asymptotic enumeration can be found in [45].

We will focus on the more ambitious task of finding an exact function for the number of groups of order $p^n$ for fixed $n$ but variable $p$. The asymptotic result of Higman's is the main result in the first of a series of papers [20, 21]. In the second paper, Higman provided the first results towards an exact enumeration function for the number of $p$-groups of a given order. Higman conjectured that the function $f(n,p)$ has

a particular form, and showed that if we restrict our attention to enumerating just a certain class of $p$-groups, then that function has the form conjectured. This form was dubbed "*Polynomial On Residue Classes*", or *PORC*.

To understand this form we should look at a few examples of enumeration functions, some which were available to Higman, and some which have only been recently discovered but have the form he conjectured. Fix $n$, but allow $p$ to vary. In this situation, $f(n,p)$ becomes a function of just $p$. The first few cases are listed below, with the first instances of the publication of these results.

| $n$ | $f(n,p)$ | Description | Publication |
|---|---|---|---|
| 1 | 1 | Cyclic group of order $p$ | Cayley [8] |
| 2 | 2 | Cyclic group, and an elementary abelian group | Cayley [8], Netto [37] |
| 3 | 5 | Three abelian groups and two non-abelian groups | Cole & Glover [10], Hölder [23], Young [51] |

None of these results depend on $p$. The case when $n = 4$ depends on $p$, but does not contain it as a variable:

$$f(4,p) = \begin{cases} 14, & \text{if } p = 2 \\ 15, & \text{if } p \geq 3 \end{cases}$$

The $n = 4$ case was first published independently by Hölder [23] and Young [51]. The first case in which $p$ appears as a variable can be found in $f(5,p)$ (first given by Bagnera [4] via a counting method, not strictly an enumeration method):

$$f(5,p) = \begin{cases} 51, & p = 2 \\ 67, & p = 3 \\ 61 + 2p + 2\gcd(p-1,3) + \gcd(p-1,4), & p \geq 5 \end{cases}$$

The last two examples regard the groups of order $p^6$ [19, 26, 40] and $p^7$ [42]:

$$f(6,p) = \begin{cases} 267 & p = 2 \\ 504 & p = 3 \\ 3p^2 + 39p + 344 + 24\gcd(p-1,3) & \\ \quad +11\gcd(p-1,4) + 2\gcd(p-1,5) & p \geq 5 \end{cases}$$

and

$$f(7,p) = \begin{cases} 2328 & p = 2 \\ 9310 & p = 3 \\ 34297 & p = 5 \\ 3p^5 + 12p^4 + 44p^3 + 170p^2 + 707p + 2455 & \\ \quad +(4p^2 + 44p + 291) \cdot \gcd(p-1,3) & \\ \quad +(p^2 + 19p + 135) \cdot \gcd(p-1,4) & p \geq 7 \\ \quad +(3p + 31) \cdot \gcd(p-1,5) + 4\gcd(p-1,7) & \\ \quad +5\gcd(p-1,8) + \gcd(p-1,9) & \end{cases}$$

These too are primarily counting methods as they determine $f(n,p)$ by examining the list of presentations.

The pattern that seems to form is that with only finitely many exceptions, $f(n,p)$ is a sum of functions of the form a polynomial in $p$ multiplied by $\gcd(p-1,k)$ where $k$ is some positive integer. Alternatively, we can recognize $f(n,p)$ as polynomial for each residue class modulo some positive integer $N$. For $f(6,p)$ this modulus would be $3 \cdot 4 \cdot 5 = 60$. Similarly for $f(7,p)$ this modulus would be 30240.

The form witnessed here is a stronger form of what Higman calls "*Polynomial On Residue Classes*" (*PORC*). Higman's PORC functions do not have to have any relation to the gcd function. The Higman definition of a PORC function is that the primes are split into residue classes modulo some positive integer $N$, and for each residue class modulo $N$ there is an associated polynomial in the prime $p$. The collection of such

polynomials is called a PORC function. Throughout this thesis, we will use a stronger form PORC functions, generalizing the gcd form above. Specifically,

**Definition** (PORC Function)**.** A function $f$ defined over the primes is *polynomial on residue classes* (*PORC*) if it is the sum of terms of the form:

$$a(p) \cdot b(p)$$

where $a(p)$ is a product of terms of the form $\gcd(p, c(p))$, $b(p)$ and $c(p)$ are polynomials in $p$ with rational coefficients, and the $k$ are positive integers.

Every example of $f(n, p)$ is PORC in this sense. We call the primes which do not follow the general gcd form *exceptional primes*. For example, the primes 2 and 3 are exceptional primes for the PORC function $f(6, p)$. In all our examples it is assumed that there are only finitely many exceptional primes.

## 1.2 Recent developments

Higman [21] conjectured that $f(n, p)$ is a PORC function (in his general sense) for all fixed $n$. If true, this would be remarkable; the current feeling is that the $p$-groups are a "wild" collection, yet if Higman's PORC conjecture is true, then they have an elegant regularity. Higman's paper [21] proved the PORC conjecture for a certain class of $p$-groups called (in modern terminology) exponent-$p$ class two groups. A $p$-group $G$ is of exponent-$p$ class two if $[G, G]G^p$. We will explore these groups in the next chapter.

Until recently, there were no further advancements on Higman's initial work. With the aid of computers, further results (such as the $p^7$ case [42]) were found to support the conjecture. Recently the work of du Sautoy and others (for example, [13, 14]) have attempted to prove Higman's PORC conjecture via algebraic geometry, in particular using zeta functions of groups. These developments are beyond the scope of this thesis, but nevertheless are important to note.

Not much work has been done on algorithmic approaches to the enumeration problem. One of the most important developments in the *listing* of $p$-groups of a fixed prime-

power order is the $p$-group generation algorithm of Newman [39] and O'Brien [41]. This algorithm can be used to *enumerate* exponent $p$-class two $p$-groups for fixed $p$ [15]. The goal of this thesis is to provide another algorithmic method of enumerating groups of exponent-$p$ class two, but for variable prime $p$. Higman's original paper [21] on the PORC conjecture forms the basis for the theory for this algorithm.

## 1.3 Thesis overview

The primary goal of this thesis is to demonstrate that there is a practical algorithm such that for fixed $n$, the algorithm can produce a function of $p$ enumerating all $p$-groups of order $p^n$ and exponent-$p$ class two. These resulting functions are PORC functions, as described on page 5.

Chapter 2 outlines the basic approach we take. It is based on the theory developed by Higman [21]. The theory divides into a few isolated parts and we will treat each part in Chapters 3 to 5. Each chapter will describe the associated theory and describe the algorithms required. Chapter 6 will detail issues and strategies regarding implementation of the algorithm. Chapter 7 will present results and further applications.

We must stress that Higman's original paper [21] is the basis for this work. In it he showed that the number of groups of order $p^n$ and exponent-$p$ class two, for fixed $n$ and variable $p$, is a PORC function in $p$. His paper did not provide explicit results. The task fulfilled by this thesis is to algorithmically generate these explicit results.

# Chapter 2

# Overview of the algorithm

Consider the lower exponent-$p$ central series of a group $G$ ([24], page 355)

$$G = P_0(G) \geq \cdots \geq P_{i-1}(G) \geq P_i(G) \geq \cdots$$

where $P_i(G) = [P_{i-1}(G), G]P_{i-1}(G)^p$ for $i \geq 1$. If there exists a least integer $c$ such that $P_c(G) = 1$ then $G$ has exponent-$p$ class $c$. We will be interested in groups of exponent-$p$ class two. Higman [20] calls such groups $\Phi$-class two groups since $P_1(G)$ is the Frattini subgroup which is often denoted by $\Phi$. We will use the more modern description.

Suppose we wanted to count exponent-$p$ class two groups of order $p^n$ for given $p$ and $n$. Denote the associated counting function by $f_2(n, p)$. There are several algorithms available to achieve this. One could list all groups of this order and determine how many of them have exponent-$p$ class two. This is a mammoth task, yet still a finite one. Another method involves subdividing the task based on the structure of $P_1(G)$. Suppose we have a finite, exponent-$p$ class two $p$-group $G$ whose subgroup $P_1(G)$ has index $p^r$ in $G$ and order $p^s$. We will say that such a group has *complexion*[1] $(r, s)$. For a fixed $r, s$ and $p$, the number of isomorphism types of $p$-groups of complexion $(r, s)$ is

[1] Higman [20] uses the notation $\Phi$-complexion.

denoted $g(r, s, p)$. It follows that

$$f_2(n, p) = \sum_{r=1}^{n-1} g(r, n - r, p). \tag{2.1}$$

O'Brien has implemented an effective algorithm in Magma [7] to compute $g(r, s, p)$ via the `ClassTwo` function. This function can calculate values of $g(r, s, p)$ for a significant range of values of $r, s$ and $p$ in reasonable time.

Can we do better? In 1960, Higman proved [21] that for a given $r$ and $s$, there is a PORC function in $p$ that calculates $g(r, s, p)$. To emphasis that this PORC function is variable in $p$, we denote it by $g(r, s; p)$ (note the semicolon). Moreover, by the appropriate substitution in Equation (2.1) there is a PORC function in $p$ that enumerates all exponent-$p$ class two groups. We denote it by $f_2(n; p)$. That is, to calculate $f_2(n; p)$ it suffices to calculate $g(r, s; p)$. We will focus on the latter task.

Higman does not provide explicit functions for either $f_2(n; p)$ or $g(r, s; p)$, nor does he suggest that they can be computed from his work. Given that these PORC functions exist, this suggests it would be profitable to try to find them. The goal of this thesis is to pursue this avenue of inquiry and show that there is an algorithm that finds the PORC functions $f_2(n; p)$ and $g(r, s; p)$ for fixed $n$, or $r$ and $s$, respectively. The underlying philosophy is that we can construct this algorithm by analysing the proof of Higman's result that inspired the task, interpreting these results in terms of calculations and filling in the gaps. As a result, the structure of the algorithm follows the series of reductions that Higman used to prove Theorem 1.1.1 in [21]. This chapter will detail this series of reductions and outline some of the theory. By the end of this chapter the calculation of $g(r, s; p)$ will be in a form suitable for algorithmic purposes. The body of this thesis will explain the relevant theory and provide algorithms for the different components of the main algorithm.

## 2.1 Reducing the problem

As previously mentioned, the main focus of this thesis is to provide an algorithm to find the PORC function in $p$ that gives the number $f_2(n;p)$ of exponent-$p$ class two groups of order $p^n$ for fixed $n$. We use a series of reductions to bring this problem to a state more amenable to algorithmic procedures. Many of these reductions are due to Higman [20, 21]. The first reduction is given by Equation (2.1); namely it suffices to find $g(r,s;p)$ for appropriate values of $r$ and $s$.

The rest of this chapter will detail three further major steps:

1. computing $g(r,s;p)$ is equivalent to counting orbits of subspaces of a specific vector space under a particular action;

2. the previous formulation is equivalent to a particular instance of orbit counting via the Cauchy-Frobenius theorem with a particular group and point set, where the group is a matrix group, the point set is a vector space and the action is the standard matrix action;

3. given the Cauchy-Frobenius approach of finding $g(r,s;p)$, we reduce the relevant function to a form that is both algorithmically amenable and has the property that our calculations keep $p$ variable.

We will take each in order.

The vector spaces we will be dealing with will be finite-dimensional. Vector spaces of dimension $d$ over a field $K$ are isomorphic to the canonical vector space $K^d$, where an element of $K^d$ is a column vector of length $d$. We will not usually distinguish an arbitrary $d$-dimensional vector space over $K$ from the column-vector representation of $K^d$.

Our base formulation is the following theorem from Higman (Theorem 2.2 of [20]).

**Theorem 2.1.** *Let $V_r$ denote the $r$-dimensional vector space $(\mathbb{F}_p)^r$, and let $V_r \wedge V_r$ be its exterior square. If $p > 2$ then the number $g(r,s;p)$ of isomorphism types of exponent-$p$ class two groups of complexion $(r,s)$ is equal to the number of orbits of subspaces of*

*codimension $s$ in the vector space $V_r \oplus V_r \wedge V_r$ under the induced action of the linear group acting on $V_r$.*

Our focus is therefore to enumerate the orbits of subspaces under this particular action. Note that this approach is not amenable for $p = 2$, so we consider 2 to be an exceptional prime for the PORC function $g(r, s; p)$. We can compute $g(r, s, 2)$ explicitly using the aforementioned `ClassTwo` function. Also note that Higman's theorem concerns $g(r, s, p)$ (that is, $p$ is fixed), but he later proves that the number of orbits of subspaces under this action is a PORC function in $p$, and so the result generalizes to $g(r, s; p)$.

The subsequent Cauchy-Frobenius approach to Theorem 2.1 will require us to know explicit details regarding the action of the associated group and vector space. To do this we will require a small amount of notation and theory. We will only be interested in vector spaces over finite fields $\mathbb{F}_p$ and their associated linear groups, but most of the theory can be generalized to arbitrary fields.

Let $V_r$ be an $r$-dimensional vector space over the finite field $\mathbb{F}_p$. We denote the full linear group on this vector space as $\mathrm{GL}(r; \mathbb{F}_p)$, or equivalently as $\mathrm{GL}(r; p)$. If we have a vector space $V$ of dimension $r$ and $W$ of dimension $s$, then we denote the linear group acting on $V \oplus W$ respecting the direct sum as $\mathrm{GL}(r, s; p)$.

To aid our explicit description, we will intentionally conflate a few concepts. We will solely deal with finite-dimensional vector spaces over $\mathbb{F}_p$. All $d$-dimensional vector spaces over $\mathbb{F}_p$ are isomorphic as vector spaces. We will identify them all with $(\mathbb{F}_p)^d$, considered as the length $d$ column vectors with entries in $\mathbb{F}_p$ (where the basis is considered to be the standard canonical basis $e_1, \ldots, e_d$ where $e_i$ is the vector with 1 in the $i$th row and zeroes elsewhere). Note that subspaces of the same dimension are *not* considered the same. Since our vectors are represented as column vectors, linear transformations are represented by matrices acting on the left. This also conflates the group of linear transformations of a vector space and the matrix group acting on column vectors. Again, we do this to simplify and standardise the explicit description of the vector spaces and groups we are dealing with.

Let $\mathrm{GL}(r;p)$ be the full linear group acting on $V_r$. The vector space $V_r \oplus V_r \wedge V_r$ can be seen to have dimension $R = \frac{1}{2}r(r+1)$. Recall that a (linear) representation of a group $G$ in $\mathrm{GL}(n;p)$ is a homomorphism $\Gamma : G \to \mathrm{GL}(n;p)$. The induced action of $\mathrm{GL}(r;p)$ on $V_r \oplus V_r \wedge V_r$ can be viewed as the image of a representation $\Gamma$ of $\mathrm{GL}(r;p)$ into $\mathrm{GL}(R;p)$. Denote this image by $\Gamma(\mathrm{GL}(r;p))$.

The image of a particular matrix $g$ in the representation of the induced action of $V_r \wedge V_r$ is often called the *second compound matrix of* $g$ [30, 34]. This is not to be confused with the *second induced matrix* which is the induced action on $V_r \otimes V_r$. Recall that a 2-rowed minor of a matrix $g$ is the determinant of a particular $2 \times 2$ submatrix. The second compound matrix of the matrix $g$, denoted $C_2(g)$, is the matrix whose entries are the 2-rowed minors of $g$ in the lexicographic ordering induced by the original rows and columns [30, 34]. It is worthwhile noting that this representation is actually a *polynomial representation* [18], that is, all the entries of $C_2(g)$ are polynomials in those of $g$. The natural action of a matrix is trivially a polynomial representation of that matrix, and so is the conjunction of the natural action of a matrix with the second compound matrix of the same matrix. That is, $\Gamma$ itself is a polynomial representation. This is a fact we will exploit later in Chapter 4.

Given this group, we would like to be able to count the number of orbits of subspaces of $V_r \oplus V_r \wedge V_r$ under this explicit induced action. We do not have many tools for investigating orbits of subspaces. One of the most useful tools in orbit enumeration is the Cauchy-Frobenius theorem (see pg 20, [24]).

**Theorem 2.2** (Cauchy-Frobenius theorem)**.** *Let the finite group $G$ act on a set of points $\Omega$. Then the number of orbits $\Omega$ under the action of $G$ is equal to*

$$\frac{1}{|G|}\sum_{g \in G} \chi(g),$$

*where $\chi(g)$ is the number of points in $\Omega$ fixed by $g$ (that is, $\chi(g) = |\{x \in \Omega \mid gx = x\}|$).*

This formulation counts orbits of points under the action of a finite group. We have a finite group $G = \mathrm{GL}(r;p)$ but we are acting on a set $\Omega$ of subspaces of $V_r \oplus V_r \wedge V_r$.

This choice of $\Omega$ is not helpful for our purposes. Therefore we will follow Higman's suggestion [21] to transform the problem of enumerating orbits of subspaces into an equivalent problem that enumerates orbits of elements under a related group. The elements here will be vectors of a particular vector space. The main advantage of this approach is that it provides a clear path to an explicit description as the group $G$ will be a matrix group, the set $\Omega$ will be a vector space and the action will be matrix multiplication.

Recall that $\mathrm{Hom}(V, W)$ is the collection of all linear transformations taking vectors in $V$ to vectors in $W$. Supposing $V$ is $r$-dimensional and $W$ is $s$-dimensional, and we represent both as vector spaces of column vectors then an element of $\mathrm{Hom}(V, W)$ can be represented as an $s \times r$ matrix. Note that in this representation $\mathrm{Hom}(V, W)$ can be considered an $(r \cdot s)$-dimensional vector space. Since the entries of $\mathrm{Hom}(V, W)$ are independent, this means that $\mathrm{Hom}(V, W)$ need not have row rank $r$. Furthermore, $V$ will be mapped to a subspace of $W$ of dimension at most $r$.

In our case, we will be interested in the set $\mathrm{Hom}(V_{R-s}, V_R)$. This maps the $(R-s)$-dimensional space $V_{R-s}$ into a subspace of dimension *at most* $R-s$ in the $R$-dimensional vector space $V_R$. We have an explicit action of $\Gamma(\mathrm{GL}(r;p))$ on $V_R$ so we want to map our $(R-s)$-dimensional subspaces into this space, so we can act appropriately. All vector spaces $V_{R-s}$ given in terms of their basis can be seen as the result of the action of an element from $\mathrm{GL}(R-s;p)$ on the canonical $(R-s)$-dimensional vector space $V_{R-s}$.

Reorganizing this information, we can see that the induced action of $\mathrm{GL}(r;p)$ on a $(R-s)$-dimensional subspace of $V_r \oplus V_r \wedge V_r$ can be represented as the action of $\Gamma(\mathrm{GL}(r;p))$ on the left of $\mathrm{Hom}(V_{R-s}, V_R)$, and $\mathrm{GL}(R-s;p)$ on the right of it, all acting on the canonical $(R-s)$-dimensional vector space $V_{R-s}$. The orbits of subspaces under the induced action of $\mathrm{GL}(r;p)$ on $V_r \oplus V_r \wedge V_r$ are the orbits of elements of $\mathrm{Hom}(V_{R-s}, V_R)$ under the left and right action of $\Gamma(\mathrm{GL}(r;p))$ and $\mathrm{GL}(R-s;p)$ respectively since $V_{R-s}$ is fixed. This two-sided action translates the problem of enumerating orbits of subspaces to enumerating orbits of elements. The group acting here is not

$\mathrm{GL}(r;p)$ but is isomorphic to $\mathrm{GL}(r, R-s;p)$.

There are two further things to note regarding this formulation. Firstly, because of our simplified representation of $\mathrm{Hom}(V_{R-s}, V_R)$ as $R \times (R-s)$ matrices of any rank, this translation only counts the number of orbits of subspaces of dimension *at most* $R-s$. Denote the resulting enumeration function by $h(r, R-s;p)$. It follows that

$$g(r,s;p) = h(r,R-s;p) - h(r,R-s-1;p) \tag{2.2}$$

since the number of orbits of subspaces of dimension at most $s$ but not of dimension at most $s-1$ is the number of orbits of subspaces of dimension exactly $s$. This is what we require.

Secondly, consider a $d$-dimensional subspace $V$ of the $D$-dimensional vector space $W$ over $K$. Let $g$ be an element of $\mathrm{GL}(D;K)$. Since $g$ is invertible, the image of $V$ under the action of $g$ is also a $d$-dimensional subspace of $W$. Consider a vector $v \in V$ and a vector $w \in W$ such that $v$ is orthogonal to $w$. Both vectors remain orthogonal under the action of $g$. Therefore, if $V'$ is a subspace orthogonal to $V$, then $gV$ is orthogonal to $gV'$. In particular, the orbit of $V$ under the action of $g$ induces an isomorphic orbit of the dual space $W \setminus V$. Therefore the number of orbits of $d$-dimensional subspaces is equal to the number of $(D-d)$-dimensional subspaces. This means that instead of considering orbits of subspaces of codimension $s$ in $V_r \oplus V_r \wedge V_r$, we can count orbits of subspaces of *dimension* $s$ in the same space. Specifically, since $h(r, R-s;p)$ counts the number of orbits of subspaces of codimension $s$,

$$h(r,R-s;p) = h(r,s;p).$$

We are free to choose either one in our calculations. Typically if $R-s$ is greater than $s$, we can reduce computation by utilizing the duality and work with the smaller group $\mathrm{GL}(r,s;p)$ instead of $\mathrm{GL}(r,R-s;p)$. For notational brevity, we will assume $s$ is smaller and use that.

Our counting theorem now expresses $g(r,s;p)$ in terms of the number of orbits

of matrices $\mathrm{Hom}(V_s, V_R)$ under the two-sided action of $\mathrm{GL}(r, s; p)$ under a particular representation. To simplify the analysis of the explicit matrix action, we would prefer the matrix action to be matrix acting on a column vector from the left. Therefore we require one last translation of this matrix action problem before it is suitable for explicit use with the Cauchy-Frobenius theorem.

We will be using the following lemma to convert the two-sided action of matrices acting on a third matrix into an equivalent left action of a matrix acting on a column vector.

Let $M$ be a matrix. Define $\mathrm{vec}(M)$ to be the column vector obtained by stacking the columns of $M$, one on top of another. Let $A$ be an $m \times n$ matrix and $B$ an $m' \times n'$ matrix. The *Kronecker product* of these two matrices, denoted $A \otimes B$, is the $mm' \times nn'$ matrix

$$\begin{pmatrix} A_{11}B & A_{12}B & \cdots & A_{1n}B \\ A_{21}B & A_{22}B & \cdots & A_{2n}B \\ \vdots & & \ddots & \\ A_{m1}B & A_{m2}B & \cdots & A_{mn}B \end{pmatrix}$$

**Lemma 2.3.** *Let $A$ be an $m \times n$ matrix, $B$ an $m' \times n'$ matrix and $X$ an $n \times m'$ matrix. Then*

$$\mathrm{vec}(AXB) = (B^T \otimes A) \cdot \mathrm{vec}(X)$$

*where $B^T$ is the transpose of $B$ and $A \otimes B$ denotes the Kronecker product of matrices $A$ and $B$.*

We omit the proof of this result as it amounts to manipulation of indices. The proof can be found in [28], page 410.

Since we are treating $\mathrm{Hom}(V_s, V_R)$ as an $(R \cdot s)$-dimensional vector space, we do not care if it is represented as a matrix or a column vector. Therefore we can use Lemma 2.3 to convert our two-sided action into a left action. Let $\Gamma^{(s)}$ be the homomorphism taking $\Gamma(\mathrm{GL}(r; p)) \times \mathrm{GL}(s; p)$ into $\mathrm{GL}(R \cdot s; p)$ via Lemma 2.3. This homomorphism is a representation, and furthermore, it is a polynomial representation. We will look

at the specifics of this representation in Chapter 4, but for now, we have converted Theorem 2.1 into the following theorem.

**Theorem 2.4.** *Let $R = \frac{1}{2}r(r+1)$, and let $V_{Rs}$ be the $(R \cdot s)$-dimensional vector space over $\mathbb{F}_p$. Let $G$ be the image of $\mathrm{GL}(r;p) \oplus \mathrm{GL}(s;p)$ under $\Gamma^{(s)} : (A, B) \to (B^T \otimes \Gamma(A))$, where $\Gamma$ is the representation of $\mathrm{GL}(r;p)$ under the induced action on $V_r \oplus V_r \wedge V_r$.*

*For $p > 2$, the number $g(r, s; p)$ of isomorphism types of exponent-$p$ class two groups of complexion $(r, s)$ is equal to $h(r, s; p) - h(r, s-1; p)$, where $h(r, s; p)$ is the number of orbits of elements of $V_{Rs}$ under the action of $G$ by usual matrix multiplication.*

## 2.2 Main formulation via Cauchy-Frobenius

Theorem 2.4 allows us to consider Theorem 2.1 in light of the Cauchy-Frobenius theorem where all the details of the action are explicit. That is we have a group $G$ which is the matrix representation of another group, and we have a set $\Omega = V_{Rs}$. We can treat these components abstractly and concentrate on computing

$$h(r, s; p) = (\#\text{ orbits of } \Omega \text{ under } G) = \frac{1}{|G|} \sum_{g \in G} \chi(g). \tag{2.3}$$

Corresponding to our original goal, we want the number of orbits to be a PORC function in $p$. Specifically, $p$ must be variable. This condition requires us to transform the Cauchy-Frobenius theorem into an another form that permits this. This transformation does not affect any of the reductions in the previous section, and also simplifies the computational workload.

The number of orbits of elements in this situation is a PORC function in $p$, as proved by Higman [21]. For this to be so, then the right-hand side of Equation (2.3) must be a PORC function in $p$. The order of $G = \mathrm{GL}(r, s; p)$ is a known polynomial in $p$, namely

$$|G| = |\mathrm{GL}(r, s; p)| = \left( \prod_{i=0}^{r-1} (p^r - p^i) \right) \cdot \left( \prod_{i=0}^{s-1} (p^s - p^i) \right). \tag{2.4}$$

Therefore our problem reduces to finding the PORC function that enumerates the total

number of fixed points under our particular action of $G$, for fixed $r$ and $s$.

Supposing we fixed $r$, $s$ and $p$, there is an obvious, naïve algorithm: loop through every element of $\mathrm{GL}(r, s; p)$ and compute the number of fixed points for each element. This is an impractical algorithm and cannot work if we let $p$ vary as the number of elements and (in general) the number of fixed points for a given element varies greatly with $p$. Nevertheless, this naïve algorithm is instructive in how we proceed from the general statement in Equation (3.1).

Firstly we notice that the naïve algorithm is unnecessarily redundant: the number of fixed points is constant over all elements of a given conjugacy class. In other words, it is a class function. If we exploit this, we obtain the following theorem.

**Theorem 2.5** (Modified Cauchy-Frobenius theorem)**.** *Under the same hypotheses as Theorem 2.2, the number of orbits of elements under the action of $G$ is*

$$\frac{1}{|G|} \sum_{c \in C} |c| \cdot \chi(g_c),$$

*where $C$ is the collection of conjugacy classes in $G$, $|c|$ is the number of elements in the conjugacy class $c$, and $\chi(g_c)$ is the number of elements fixed by the action of $g_c$, a representative element from $c$.*

There are fewer conjugacy classes in a group than elements, so the naïve algorithm becomes much more practical if we work over conjugacy classes.

Unfortunately, every part of this sum still depends on $p$: the set of conjugacy classes, the size of a particular conjugacy class, and the number of fixed points of a given element. The rest of this chapter briefly describes how we surmount these obstacles whilst allowing $p$ to vary.

The solution Higman proposes is to classify conjugacy classes by *types* (which we will discuss in detail in the next chapter). Types classify conjugacy classes in linear groups depending on how elements in the conjugacy class diagonalize, in a general sense. Specifically, types abstract the concept of a rational canonical form of a matrix, and this is done in such a way that the description is independent of the base field, and

hence, of $p$. As a result, we have a classification of conjugacy classes that is independent of $p$. Furthermore, the size of a conjugacy class of a given type and the number of fixed points of a given element in a conjugacy class of a given type are both polynomials in $p$. This means we can modify the formula in Theorem 2.5 to

$$\frac{1}{|G|}\sum_{\tau\in\mathcal{T}}\sum_{c\in\tau}|c|\cdot\chi(g_c),$$

where $\mathcal{T}$ is the set of all types of $\mathrm{GL}(r,s;p)$. It turns out that the size of a conjugacy class depends only on its type, so we may simplify the sum as

$$\frac{1}{|G|}\sum_{\tau\in\mathcal{T}}|c_\tau|\sum_{c\in\tau}\chi(g_c),$$

where $|c_\tau|$ is the size of any conjugacy class of type $\tau$.

The innermost sum is still problematic as we want to avoid depending on particular conjugacy classes (as they depend on the field, and hence, on $p$). To this end, we can derive from the type and the representation of $G$ a set called the *master degeneracy set* $E_\tau$ which describes all the possible structural information that a conjugacy class of a type has in relation to counting fixed points. A subset $S\subseteq E_\tau$, called a *degeneracy set*, of the master degeneracy set corresponds to a set of conjugacy classes. Furthermore, for conjugacy classes corresponding to the degeneract set $S$, the function $\chi(g_c)$ depends only on $S$, so we can amend the Cauchy-Frobenius formula further:

$$\frac{1}{|G|}\sum_{\tau\in\mathcal{T}}|c_\tau|\sum_{S\subseteq E_\tau}\sum_{c\in S}\chi(g_c),$$

where the innermost sum is interpreted as "the sum over all conjugacy classes $c$ with degeneracy set $S$, a subset of the master degeneracy set $E_\tau$". Now since $\chi(g_c)$ only depends on $S$ (and not on the particulars of conjugacy classes with structure given by the degeneracy set $S$), we rewrite this term as $\chi(g_S)$, and modify the Cauchy-Frobenius

formula accordingly:

$$\frac{1}{|G|}\sum_{\tau\in\mathcal{T}}|c_\tau|\sum_{S\subseteq E_\tau}\chi(g_S)\sum_{c\in S}1.$$

Higman proved [21] that the sum $\sum_{c\in S}1$ is a PORC function in $p$. We denote it by $d(S,\tau;p)$.

Our approach is now set: following the approach of using types as suggested by Higman, we calculate $h(r,s;p)$ (and hence $g(r,s;p)$ and $f_2(n;p)$) via

$$\frac{1}{|G|}\sum_{\tau\in\mathcal{T}}|c_\tau|\sum_{S\subseteq E_\tau}\chi(g_S)\cdot d(S,\tau;p). \tag{2.5}$$

The size of $G$ is known (Equation (2.4)), and we can easily calculate $g(r,s;p)$ from $h(r,s;p)$ (via Equation (2.2)), and $f_2(n;p)$ from $g(r,s;p)$ (via Equation (2.1)). The body of this thesis describes the relevant theory behind each of the following tasks and provides algorithms for them:

1. construct the set $\mathcal{T}$ of types for $\mathrm{GL}(r,s;p)$;
2. determine the size $|c_\tau|$ of a conjugacy class of type $\tau$;
3. construct the master degeneracy set $E_\tau$ of a type $\tau$;
4. determine the number $\chi(g_S)$ of fixed points for any element in any conjugacy class with degeneracy set $S$ and of type $\tau$;
5. and determine the number $d(S,\tau;p)$ of conjugacy classes in $\mathrm{GL}(r,s;p)$ of a given type $\tau$ and with degeneracy set $S$;

Following this is a discussion regarding implementation, and the results obtained from the implementation. Lastly, we discuss results, and modifications and extensions to the algorithm.

# Chapter 3

# Types

In this chapter we discuss types of conjugacy classes. Types classify conjugacy classes of matrices in a manner independent of the field over which the matrices are defined. Furthermore, types allow us to compute the size of a conjugacy class of a given type in $\mathrm{GL}(n;p)$ in a way depending only on the type. This chapter is devoted to detailing types and their properties, and presenting algorithms to list all types of a given size.

The term "type" seems to be first coined by J.A. Green in 1955 [17], but the concept appears long before this. R. Steinberg used a similar concept in 1951 [50], but applied it to characters of various representations. The work of D.E. Littlewood (for example, [30]) used the concept without naming it. An even earlier form can be found in L.E. Dickson's book on linear groups [12]. We will be using types in a form closest to that used by Higman [21], Green [17] and Macdonald [32, 33].

In a general sense, types are an abstraction of rational canonical forms of matrices over fields. To make this notion more precise, we will need to review some theory and introduce notation concerning conjugacy classes in $\mathrm{GL}(n;K)$, for any field $K$.

Let $f(x) \in K[x]$ be a monic polynomial $f(x) = x^d + \sum_{i=0}^{d-1} a_i x^i$. The *companion*

*matrix* of $f$ is the $d \times d$ matrix

$$C(f) = C_K(f) = \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 & -a_0 \\ 1 & 0 & \cdots & 0 & 0 & -a_1 \\ 0 & 1 & \cdots & 0 & 0 & -a_2 \\ & & \ddots & & & \vdots \\ 0 & 0 & \cdots & 1 & 0 & -a_{d-2} \\ 0 & 0 & \cdots & 0 & 1 & -a_{d-1} \end{pmatrix}.$$

Let $\lambda$ be a positive integer and $f(x)$ have degree $d$. Then $C^\lambda(f)$ is the block matrix

$$\begin{pmatrix} C(f) & I_d & & \\ & C(f) & I_d & \\ & & \ddots & I_d \\ & & & C(f) \end{pmatrix}$$

where $I_d$ is the $d \times d$ identity matrix, and there are $\lambda$ blocks along the diagonal. In particular, if $f(x) = (x - \alpha)$ then $C^\lambda(f)$ is called the *Jordan block* $J_\lambda(\alpha)$.

We usually assume that $f$ is irreducible in $K[x]$. Suppose $f(x)$ splits into linear factors in $\overline{K}[x]$ (where, $\overline{K}$ is the algebraic closure of $K$):

$$f(x) = (x - \alpha_1)(x - \alpha_2) \cdots (x - \alpha_d).$$

Then the companion matrix $C_{\overline{K}}(f)$ of $f$ in $\overline{K}$ is equivalent to the direct sum of matrices

$$\bigoplus_{i=1}^{d} (\alpha_i).$$

If $K = \mathbb{F}_p$ is a finite field, and $f$ is a monic, irreducible polynomial of degree $d$ over $\mathbb{F}_p$ then $f$ splits into linear factors in $\mathbb{F}_{p^d}$

$$f(x) = (x - \alpha)(x - \alpha^\sigma) \cdots (x - \alpha^{\sigma^{d-1}}),$$

where $\sigma$ is the Frobenius automorphism $\sigma : x \mapsto x^p$ of $\mathbb{F}_{p^d}$ over $\mathbb{F}_p$. The elements $\alpha, \alpha^\sigma, \ldots, \alpha^{\sigma^{d-1}}$ are called *Galois conjugates* (to help differentiate them from "conjugates" which we shall use primarily for group-conjugate matrices in $\mathrm{GL}(n;K)$). We will prefer to use $f$ and its companion matrix when $f$ is monic and irreducible, but we recognize that there is an equivalent form if we algebraically close the field. Later in this thesis, for particular contexts, will prefer the algebraically closed approach for ease of description.

The following result is well-known (see, for example, [25]).

**Theorem 3.1.** *Let $A \in \mathrm{GL}(n;K)$. The matrix $A$ is conjugate to a block matrix $B$ of the form $C^{\lambda_1}(f_1) \oplus \cdots \oplus C^{\lambda_r}(f_r)$, where the $f_i$ are monic irreducible polynomials over $K$ and the $\lambda_i$ are positive integers. The matrix $B$ is unique up to the order of the blocks $C^{\lambda_i}(f_i)$.*

The polynomials $f_i$ are the *invariant factors* of $A$. If we embed $A$ in $\mathrm{GL}(n;\overline{K})$ the invariant factors in this context are called the *elementary divisors* of $A$. Alternatively, if we split each $f_i$ over $\overline{K}$ into linear factors $g_{i,j}$ then the elementary divisors are $g_{i,j}$ with associated positive integer $\lambda_i$. The elementary divisors $g_{i,j}$ correspond to Jordan blocks in the matrix $B$. In the literature there are many different definitions for the terms "invariant factors" and "elementary divisors". The above definitions will be standard for this thesis.

The matrix $B$ in Theorem 3.1 corresponding to the invariant factors of $A$ over $K$ is called the *rational canonical form* of $A$. If all invariant factors are linear, then $B$ is the *Jordan canonical form* of $A$ where the block matrices are the corresponding Jordan blocks. Of course, the rational canonical form differs from the Jordan canonical form only in the case when one of the irreducible polynomials $f_i$ has degree greater than one over $K$. We will be concerned with finite fields, where there are irreducible polynomials of every degree, and so we will customarily use the rational canonical form of matrices over these fields.

Theorem 3.1 states that all matrices in $\mathrm{GL}(n;K)$ have a rational canonical form. A simple corollary is that all matrices in a specific conjugacy class in $\mathrm{GL}(n;K)$ have

the same rational canonical form. The invariant factors specify the rational canonical form, and so each conjugacy class is classified by the invariant factors of the associated rational canonical form. If we algebraically close the field, we find the same result holds for elementary divisors. Generalizing this result to considering invariant factors to be different only if their degree is different allows us to classify conjugacy classes into *types*. The next section will elaborate on this generalization. This will be followed by discussion on how to algorithmically generate all types, and the number of types of a given size as well as the size of a conjugacy class of a given type. The chapter will close with a short discussion of pursuing this sort of classification but using elementary divisors instead, called the *species classification*. Species play a small but important role in the following chapters.

## 3.1 Classification into types

There are three equivalent descriptions of types: the first two determine when two matrices (or conjugacy classes) are of the same type, and the last description presents types as combinatorial objects. We will describe all three approaches as we will be using them in various contexts later on, depending on which is the most comfortable to use. We will primarily use the combinatorial approach as it is easier to use.

Let $A$ be a matrix in $\mathrm{GL}(n;K)$, and furthermore, suppose we know its invariant factors $\{f_i\}$. Let $\rho_A(f)$ be a integer-partition-valued function taking an irreducible polynomial $f$ to the partition $\lambda = (\lambda_1, \ldots, \lambda_k)$ where $\lambda_i$ is the positive integer associated to some instance of the invariant factor $f$. Let $\tau(A)$ be the complete set of pairs $(f, \lambda)$ where $f$ is irreducible and $\lambda = \rho_A(f)$ is nonzero. We repeat this for $B$ in $\mathrm{GL}(n;K)$, yielding a set $\tau(B)$ of pairs $(g, \mu)$. We say then that the matrices $A$ and $B$ are of the same *type* if there is an isomorphism between $\tau(A)$ and $\tau(B)$ such that the pair $(f, \lambda)$ is isomorphic to $(g, \mu)$ if and only if the degree of $f$ is the same as the degree of $g$, and $\lambda = \mu$. We can also say that $A$ and $B$ are *type-equivalent*.

Note that this description depends only on the degrees of the polynomials, not on the polynomials themselves. We can therefore extend the definition for $A$ in $\mathrm{GL}(n;K)$

and $B$ in $\mathrm{GL}(n; L)$ where $K$ and $L$ are fields that do not have to be isomorphic. Thus the notion of types is independent from the underlying fields. Also note that the polynomials $f$ in the pairs in $\tau(A)$ (and similarly, $g$ in pairs of $\tau(B)$) must be distinct otherwise they should be combined as, say, $(f, \lambda + \lambda')$.

This first description gives a good intuition regarding the invariant factors of a matrix and the corresponding type for this matrix. However in computations it may be more useful to represent the associated companion matrices as companion matrices for the roots of the polynomials in invariant factors rather than the polynomials themselves (as per the discussion on page 21). Note that this does not derive the type-equivalence via elementary divisors since the roots of a polynomial will be grouped together, whereas the elementary divisor approach (examined in Section 3.5) does not make this identification. This grouping is fundamental to the description.

The basic idea in the second description is that a pair $(f, \lambda)$ for a matrix $A$ is replaced with the pairs $(\alpha_1, \lambda), \ldots, (\alpha_d, \lambda)$, where $\alpha_1, \ldots, \alpha_d$ are the roots of $f$ in $\overline{K}$. These roots are conjugate under the Galois action of $K[\alpha_1, \ldots, \alpha_d]$ over $K$. Thus the set $\tau(A)$ is replaced by a possibly larger set $\tau'(A) = \{(\alpha_i, \lambda_i)\}$ (and similarly for $\tau(B)$, replaced by $\tau'(B) = \{(\beta_i, \mu_i)\}$). The correspondence between $\tau(A)$ and $\tau(B)$ extends to a correspondence between $\tau'(A)$ and $\tau'(B)$ such that with a possible reordering of the pairs in $\tau'(B)$, we have $(\alpha_i, \lambda)$ is isomorphic to $(\beta_i, \mu)$ if and only if $\lambda = \mu$ and for all $j$ such that $\alpha_j$ is Galois conjugate over $K$ to $\alpha_i$, then $\beta_j$ must be Galois conjugate over $L$ to $\beta_i$. In short, two matrices are of the same type if there is a correspondence between their elementary divisors accommodating both Galois conjugacy and the correspondence for their relevant invariant factors.

Higman [21] uses this second description, but has a later modification to considering the correspondence between sets of Galois conjugates of the same size. Thus a set of conjugates $\alpha_1, \ldots, \alpha_d$ is represented by a set $y$, which has the appropriate partition $\lambda$ associated to it. Therefore the correspondence is between pairs $(y, \lambda)$ and $(y', \mu)$ such that the partitions must be the same and the set $y$ must have the same cardinality as the set $y'$.

These two descriptions allow us to determine when two matrices are of the same type. Note that the important data was the degree of the polynomial (or equivalently, the number of Galois conjugates) and the associated partition. To describe types as combinatorial objects in themselves, we ignore all the technical details involved in the fields and concentrate on the core data as suggested in the previous paragraphs.

A *type* of size $n$ is a multiset $\tau$ of pairs $(d, \lambda)$, where $d$ is a positive integer and $\lambda$ is a nonzero integer partition, such that

$$\sum_{(d,\lambda)\in\tau} d \cdot |\lambda| = n. \tag{3.1}$$

We call a pair $(d, \lambda)$ in $\tau$ a *type-parameter* of that type. The types of $\mathrm{GL}(n; K)$ are all the types of size $n$. We denote the set of all types of size $n$ by $\mathcal{T}_n$.

The first description of type-equivalence is equivalent to this description by a one-to-one mapping of $(f, \lambda)$ to a type-parameter $(d, \lambda)$ such that $f$ has degree $d$. Two identical type-parameters given by $(d, \lambda)$ represent two distinct pairs $(f, \lambda)$ and $(f', \lambda)$, where $f$ and $f'$ are distinct polynomials of degree $d$ over $K$. This distinction is strict: the type $\{(d, \lambda), (d, \lambda')\}$ is *not* equivalent to $\{(d, \lambda + \lambda')\}$.

We prefer the description of types as combinatorial objects as it is cleaner and rightly de-emphasises the role of the underlying field. Types are independent of any field $K$ so we needn't discuss the field at all.

As an example, $\mathcal{T}_2$ is the set of types:

$$\tau_1 = \{(1, (1)), (1, (1))\},$$
$$\tau_2 = \{(1, (1, 1))\},$$
$$\tau_3 = \{(1, (2))\},$$
$$\tau_4 = \{(2, (1))\}.$$

These four types have the following respective representations $R_1, \ldots, R_4$ in terms of

matrices in $\mathrm{GL}(n;p)$:

$$R_1 = \begin{pmatrix} \alpha & \\ & \beta \end{pmatrix} \qquad R_2 = \begin{pmatrix} \alpha & \\ & \alpha \end{pmatrix}$$
$$R_3 = \begin{pmatrix} \alpha & 1 \\ & \alpha \end{pmatrix} \qquad R_4 = \begin{pmatrix} \gamma & \\ & \gamma^p \end{pmatrix}$$

where $\alpha$ and $\beta$ are nonzero elements in $\mathbb{F}_p$ with $\alpha \neq \beta$, and $\gamma$ is an element of $\mathbb{F}_{p^2}$ but not $\mathbb{F}_p$. This list is the same as the one provided by Green [17] (although the indexing is different because of the way we order partitions). Note that $R_4$ can be alternatively given by the companion matrix of an irreducible quadratic over $\mathbb{F}_p$, which is essentially the difference between the first and second descriptions of type-equivalence. In explicit matrix representations of types, we will prefer the split form because we have more utility using Galois conjugates and it saves us from introducing more finite field elements (and restrictions on them) than we require.

We can extend types for $\mathrm{GL}(n_1, \ldots, n_r; K)$ by adding a third number to each type-parameter called the *location*. A type in $\mathrm{GL}(n_1, \ldots, n_r; K)$ is then given by triples $(d, \lambda, l)$ where $l$ is an index indicating that $\mathrm{GL}(n_l; K)$ has type with type-parameter $(d, \lambda)$. Two type-parameters $(d, \lambda, l)$ and $(d, \lambda, l')$ need not represent two distinct invariant factors with polynomials of degree $d$. In other words, type-parameters only define distinct invariant factors within their location.

Another important point is although the type of a matrix $A$ in $\mathrm{GL}(n; K)$ depends on the underlying field $K$, the types themselves are independent of any field. For a particular field $K$, it may happen that there are *no* matrices of a certain type. For example, in $\mathrm{GL}(4, \mathbb{F}_2)$ there are no elements of type $\{(2,1),(2,1)\}$ as this requires two distinct irreducible quadratics over $\mathbb{F}_2$, but only one such polynomial exists. These exceptions pose no problem for us; the calculations we perform take care of them. In the specific example above, the number of matrices in $\mathrm{GL}(4; \mathbb{F}_p)$ of this type is

$$\frac{1}{4}p^8 - p^7 + p^6 + \frac{1}{2}p^5 - \frac{5}{4}p^4 + \frac{1}{2}p^3 = \frac{1}{4}(p-2)\cdot(p-1)^3 \cdot p^3 \cdot (p+1).$$

This polynomial has a factor of $(p-2)$, so for $p = 2$ there are no elements of that type. All types of a given size are considered in our calculations, although some may not contribute to the answer if we specialize the field.

## 3.2 Listing types

As mentioned at the end of Chapter 2, we will require a list of the types of $\mathrm{GL}(r, s; K)$. This section details algorithms for this task. We generate types in $(d, \lambda, l)$ form. Of course, we can just generate the types in $(d, \lambda)$ form for $\mathrm{GL}(r; K)$ and $\mathrm{GL}(s; K)$ individually and append the location afterwards. Thus it suffices to find the types for $\mathrm{GL}(n; K)$, that is, the types of size $n$.

The general idea is to generate types in the $(d, \lambda)$ form focussing on Equation (3.1):

$$\sum_{(d,\lambda)\in\tau} d \cdot |\lambda| = n.$$

Each type-parameter $(d, \lambda)$ contributes a "weight" of $d \cdot |\lambda|$ to the final sum. These weights are strictly positive integers that sum to $n$, and conversely, the weights partition $n$. Thus to list types of size $n$, we first partition $n$ into weights, and for each weight, we replace it with an appropriate type-parameter $(d, \lambda)$ of that weight. We break the algorithm into two simple parts: the first finds all possible type-parameters of a given weight, and the second partitions $n$ and substitutes each part with a type-parameter of the appropriate weight, thus creating a type.

The first algorithm is straightforward: given a weight $k$, we list all appropriate degrees $d$, and for each such choice, list every appropriate choice of partition $\lambda$. This order of choice is much simpler than the reverse. The possible $d$ are just the divisors of $k$, and the partitions are the integer partitions of $k/d$. It is clear that the type-parameters generated are unique and complete since we consider all unique possibilities for $d$ and $\lambda$ required for Equation (3.1).

Recall that pseudocode follows GAP conventions. Specifically, the function `DivisorsInt`$(k)$ lists all divisors of an integer $k$ and the function `Partitions`$(k)$ lists

all partitions of size $k$. These functions can be found in GAP.

**Algorithm 3.1** Generate all type-parameters of weight $k$

**Require:** Positive integer $k$

1: *paramlist* := [ ]
2: **for** $d$ in `DivisorsInt`($k$) **do**
3: **for** $\lambda$ in `Partitions`($k/d$) **do**
4: Add $(d, \lambda)$ to *paramlist*.
5: **od**
6: **od**
7: **return** List of possible type-parameters *paramlist*

Given the algorithm to list all possible type-parameters of a particular size, we now list all types of size $n$. Firstly we list all partitions of $n$. Let $\Lambda = (\lambda_1, \ldots, \lambda_t)$ be such a partition. For ease of presentation we write $\Lambda$ in terms of multiplicities of parts (that is, $\Lambda = (1^{m_1}, 2^{m_2}, \ldots)$. Then for each nonzero $m_i$, we find unordered $m_i$-tuples of type-parameters of weight $i$. That is, we allow repetition of type-parameters of a given weight, but order is irrelevant.

The GAP command `Partitions`($k$) gives partitions in the standard list form $\Lambda = (\lambda_1, \ldots, \lambda_t)$. In the algorithm that follows, we assume we can collect like parts as in the multiplicity form of partitions described in the previous paragraph. We denote the size of a part $\lambda$ in this collected form by $|\lambda|$, and its multiplicity by $\#\lambda$. This is simple to achieve in code if we replace each element $\Lambda$ in `Partitions`($k$) by `Collected`($\Lambda$). This modification is not essential to the algorithm, it just makes the presentation simpler.

**Algorithm 3.2** Generate all types of size $n$

**Require:** Positive integer $n$

1: Let $\mathcal{T}_n$ := [ ]
2: **for** $\Lambda$ in `Partitions`($n$) **do**
3: Let *paramlist* := [ ]
4: **for** $\lambda$ in $\Lambda$ **do**
5: *param* := `UnorderedTuples`(`TypeParameters`($|\lambda|$), $\#\lambda$)
6: Append *param* to *paramlist*
7: **od**
8: Add cartesian product of elements of *paramlist* to $\mathcal{T}_n$
9: **od**
10: **return** The set of types $\mathcal{T}_n$

The function `UnorderedTuples`($S, k$) takes a set $S$ and lists all unordered $k$-tuples

of elements in $S$. The function `TypeParameters`$(k)$ lists all type-parameters of size $k$, which is the function detailed in Algorithm 3.1.

## 3.3 Number of types

Now that we have an algorithm to list all types, we want to know how many types there are of a given size. The algorithms given run quickly for small inputs (which is what we are primarily interested in). Therefore we can count types just by listing them. However, we can derive a formula for this result which can give us an idea of the rough computational workload. We will first present a generating function for the number of types, and from that, derive a recursive function for the number of types of any size.

Suppose we have a sequence of integers indexed by the non-negative integers. We can encode this sequence as a function, say $a(n)$. We say that the *generating function* for this sequence (or equivalently, for $a(n)$) is the formal power series

$$A(x) = \sum_{n \geq 0} a(n) x^n.$$

We maintain the convention that the sequence is given by a function with a lowercase label whose variable is $n$, and the corresponding generating function is given by the appropriate uppercase label with a variable $x$. We also require that $a(0) = 1$ for all sequences.

An important sequence of numbers is the number of (unrestricted) integer partitions. We denote the number of partitions of an integer $n$ by $p(n)$. This has a generating function

$$P(x) = \sum_{n \geq 0} p(n) x^n.$$

It is well-known [3] that

$$P(x) = \prod_{n \geq 1} (1 - x^n)^{-1}.$$

The following theorem provides us with a generating function for the number of

types.

**Theorem 3.2.** *Let $t(n)$ be the number of types of size $n$. The generating function for $t(n)$ is*

$$T(x) = \sum_{n\geq 0} t(n)x^n = \prod_{n\geq 1} P(x^n)^{p(n)},$$

*where $p(n)$ is the number of (unrestricted) integer partitions of size $n$, and $P(x)$ is its generating function.*

Steinberg [50] and Green [17] both provide this result without proof[1]. We will not give a proof of this result, since it is just manipulation of generating functions according to the description of types given by Equation (3.1) and the intuition behind the two algorithms in the previous section.

The result most useful for our purposes is the following corollary.

**Corollary 3.2.1.** *The number of types satisfies the following recursion equation*

$$t(0) = 1,$$
$$t(n) = \frac{1}{n}\sum_{i=0}^{n-1} g(n-i)\cdot t(i) \qquad \text{for } n \geq 1$$

*where*

$$g(n) = \sum_{d|n} p(d)\cdot d\cdot \sigma(n/d),$$

*where $\sigma(k)$ is the sum of the divisors of $k$.*

*Proof.* This proof follows a similar proof providing a recursion equation for the number of partitions $p(n)$ (see [3], page 323).

Let

$$\begin{aligned} T(x) &= \prod_{n\geq 1} P(x^n)^{p(n)} = \prod_{n\geq 1}\prod_{i\geq 1} \frac{1}{(1-x^{ni})^{p(n)}} \\ &= \sum_{n\geq 0} t(n)x^n. \end{aligned}$$

[1] Steinberg conjectures the result, and Green states that "it is not hard to show".

Take the logarithmic derivative (the logarithm and then the derivative, see [2], page 98) of

$$T(x) = \prod_{n\geq 1}\prod_{i\geq 1}\frac{1}{(1-x^{ni})^{p(n)}}$$

and then multiply both sides by $x$, yielding

$$\frac{T'(x)}{T(x)} = \sum_{n\geq 1}\sum_{i\geq 1} i\cdot n\cdot p(n)\frac{ix^{ni}}{1-x^{ni}}.$$

Substituting $T(x) = \sum_{n\geq 0} t(n)x^n$ and the standard power series expansion of $(1-x)^{-1}$ in the previous equation, and rearranging we get

$$\sum_{n\geq 1} n\cdot t(n)\cdot x^n = \left(\sum_{n\geq 1}\sum_{i\geq 1}\sum_{j\geq 1} i\cdot n\cdot p(n)x^{ijn}\right)\left(\sum_{n\geq 0} t(n)x^n\right).$$

If we compare the coefficients of $x^k$ on both sides we see that

$$k\cdot t(k) = \sum_{i=0}^{k-1} g(k-i)\cdot t(i),$$

where

$$g(n) = \sum_{d|n} p(d)\cdot d\cdot \sum_{j|n/d} j. \qquad (*)$$

Since the internal sum in Equation $(*)$ is just $\sigma(n/d)$, we obtain the required result. □

We can use this recursive formula to compute the first few values of $t(n)$ which are listed in the table below. This is identical to a list provided by Green [17] although we include three extra terms. A list is available on-line [1] which contains many more terms than in Table 3.1. Calculating further values of $t(n)$ is not difficult providing one has access to previously computed values of $t(n)$ and $g(n)$ (so as to reduce the amount of recursion one needs to do).

For completeness it would be nice to obtain asymptotics for $t(n)$, perhaps to apply to run-time estimates of the main algorithm of this thesis. This turns out to be very difficult. Analogous results found for $p(n)$ are difficult to translate to $t(n)$. However,

| $n$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $t(n)$ | 1 | 4 | 8 | 22 | 42 | 103 | 199 | 441 | 859 | 1784 |

Table 3.1: Number of types $t(n)$ for $n = 1, \ldots, 10$.

as mentioned before, we really only need types for small values of $n$; calculations later in the algorithm significantly dominate the running time. Nevertheless, it is easy to see that $t(n)$ is far greater than $p(n)$, and $p(n)$ is asymptotically exponential (see [2], page 97, or [3], page 316).

## 3.4 Size of a conjugacy class in a type

Types provide us with another useful result: the size of a conjugacy class of a given type defined over $\mathbb{F}_p$ is a polynomial in $p$ that depends only on the type. In particular:

**Theorem 3.3.** *Let $\tau$ be a type in $\mathrm{GL}(r, s; p)$ with type parameters given by $(d_i, \lambda_i, l_i)$ for $i = 1, \ldots, w$. For a partition $\lambda$ denote its conjugate partition by $\lambda' = (\lambda'_1, \ldots, \lambda'_t)$ and let*

$$a_\lambda(p) = p^{|\lambda|+2n_\lambda} \prod_{j=1}^{t} \phi_{\lambda'_i - \lambda'_{i+1}}(p^{-1})$$

*where $n_\lambda = \sum_{i=1}^{t} \binom{\lambda'_i}{2}$, and $\lambda'_{t+1} = 0$, and*

$$\phi_r(p) = \prod_{i=1}^{r} (1 - p^i)$$

*for $r \geq 1$ and $\phi_0(p) = 1$. Then the size $|c_\tau|$ of a conjugacy class in $\tau$ defined over $\mathbb{F}_p$ is*

$$|c_\tau| = |\mathrm{GL}(r, s; p)| \cdot \prod_{i=1}^{w} a_{\lambda_i}(p^{d_i})^{-1}.$$

A proof of this result can be found on page 181 of [32], and a sketch for an alternative proof is in [17].

We know $|\mathrm{GL}(r, s; p)|$ by Equation (2.4) on page 15, so computing $|c_\tau|$ is straightforward. Note that the location property of the type-parameters do not play a role

in $|c_\tau|$. In GAP the function $a_\lambda(p)$ can be computed with the undocumented function `SizePolynomialUnipotentClassGL`$(\lambda)$.

We will compute $|c_\tau|$ for all four types in in $\mathrm{GL}(2;p)$. We know that $|\mathrm{GL}(2;p)| = (p^2-1)\cdot(p^2-p)$. Let $\tau_1 = \{(1,(1)),(1,(1))\}$. Since

$$\begin{aligned} a_{(1)}(p) &= p^{1+2\cdot 0}\phi_{1-0}(p^{-1}) \\ &= p\cdot(1-p^{-1}) \\ &= (p-1), \end{aligned}$$

it follows that $|c_{\tau_1}| = |\mathrm{GL}(2;p)|/(p-1)^2 = p^2+p$.

Now consider $\tau_2 = \{(1,(1,1))\}$. Then

$$\begin{aligned} a_{(1,1)}(p) &= p^{2+2\cdot 1}\phi_{2-0}(p^{-1}) \\ &= p^4\cdot(1-p^{-1})\cdot(1-p^{-2}) \\ &= p\cdot(p-1)\cdot(p^2-1). \end{aligned}$$

Then $|c_{\tau_2}| = |\mathrm{GL}(2;p)|/(p\cdot(p-1)\cdot(p^2-1)) = 1$. Note that the representative matrix for both $\tau_1$ and $\tau_2$ are diagonal, but they have different conjugacy class sizes solely because the diagonal entries are different or identical respectively.

Consider $\tau_3 = \{(1,(2))\}$. Then

$$\begin{aligned} a_{(2)}(p) &= p^{2+2\cdot 0}\phi_{1-1}(p^{-1})\cdot\phi_{1-0}(p^{-1}) \\ &= p^2\cdot 1\cdot(1-p^{-1}) \\ &= p\cdot(p-1). \end{aligned}$$

In this case, $|c_{\tau_3}| = |\mathrm{GL}(2;p)|/(p^2-p) = p^2-1$.

Finally consider $\tau_4 = \{(2,(1)\}$. We showed above that

$$a_{(1)}(p) = (p-1),$$

so $a_{(1)}(p^2) = (p^2 - 1)$ and therefore $|c_{\tau_4}| = |\mathrm{GL}(2;p)|/(p^2-1) = (p^2 - p)$.

## 3.5 Species

Although our primary method of classifying conjugacy classes in $\mathrm{GL}(n;\mathbb{F}_p)$ is via types, there is another classification we will use. This classification is into *species* and is based on the elementary divisors of matrices in the algebraic closure of the underlying field. Note that species are not intentionally related to Joyal's theory of combinatorial species [5, 27]. They can be considered in this light, but this is of no benefit to us. Species, like types, are independent of the base field.

The elementary divisors of a matrix $A$ in $\mathrm{GL}(n;K)$ can be described by an integer-partition-valued function $\Delta_A(\alpha)$ for nonzero $\alpha$ in the algebraic closure of $K$. The value of this function is $\lambda = (\lambda_1, \ldots, \lambda_k)$ if the elementary divisors of $A$ corresponding to $(x-\alpha)$ have corresponding positive integers $\lambda_1, \ldots, \lambda_k$. We can define conjugacy classes to be equivalent by using the gross structure indicated by their elementary divisors, or similarly, by $\Delta$. Define the summation of two integer partitions as their concatenation (with appropriate reordering of parts). Since there are only finitely many elementary divisors of a matrix in $\mathrm{GL}(n;K)$, and each elementary divisor can only contribute a finite part, we can form the sum

$$\sum_{\alpha \in \overline{K}} \Delta_A(\alpha).$$

Let $A$ and $B$ be any two matrices in $\mathrm{GL}(n;K)$. We say they are of the same *species* if

$$\sum_{\alpha \in \overline{K}} \Delta_A(\alpha) = \sum_{\alpha \in \overline{K}} \Delta_B(\alpha).$$

Clearly if $A$ is of the same species as $B$, then any conjugate of $A$ is also of the same species as $B$. Therefore species classify conjugacy classes in $\mathrm{GL}(n;K)$. We can extend the definition of species to compare matrices with different base fields, so that

$A \in \mathrm{GL}(n;K)$ and $B \in \mathrm{GL}(n;L)$ are of the same species if

$$\sum_{\alpha\in\overline{K}} \Delta_A(\alpha) = \sum_{\alpha\in\overline{L}} \Delta_B(\alpha).$$

Therefore, a species is just an integer partition $\Lambda$ and a matrix $A$ is of that species if its sum of $\Delta_A(\alpha)$ over $\alpha$ in the appropriate field is $\Lambda$.

Species also can classify types. For a type $\tau$ given as pairs $(d,\lambda)$, the species of $\tau$ is the partition given by the sum

$$\bigcup_{(d,\lambda)\in\tau} \bigcup_{i=1}^{d} \{\lambda\}.$$

That is, each type-parameter $(d,\lambda)$ contributes $d$ copies of $\lambda$ to the concatenation-sum.

We can enumerate and list species of $\mathrm{GL}(n;K)$ easily, since they are just partitions of $n$. We can also extend species for $\mathrm{GL}(n_1,\ldots,n_k;K)$ by taking ordered $k$-tuples of partitions of the appropriate integers.

We do not use species for classifying conjugacy classes as it is too weak. Species do not allow us to calculate the size of conjugacy classes of that species as two types can be of the same species but have different conjugacy class sizes. For example, types $\tau = \{(1,(1,1))\}$ and $\tau' = \{(2,(1))\}$ both are in the species $(1,1)$ but as we calculated in the previous section, $|c_\tau| = p^2 + p$, whereas $|c_{\tau'}| = 1$. We will have use for species: the elementary divisors of $A$ corresponding to $(x-1)$ indicate the number of fixed points of $A$, which we will require. This will be explained in Chapter 4.

# Chapter 4

# Degeneracy sets

As indicated in Chapter 2, we need a way to distinguish conjugacy classes of matrices of a given type. This needs to be independent of the field the matrices are defined over, and is specific to finding the number of fixed points under a particular action. In our case, this action is the image of a matrix under the representation $\Gamma^{(s)}$ as described in Chapter 2.

This chapter introduces the concept of a *degeneracy set*, which is an invariant assigned to every conjugacy class. It was developed by Higman for this task. Conjugacy classes with the same degeneracy set have the same number of fixed points per element. As such, we can classify conjugacy classes by their degeneracy set. This will be a classification of conjugacy classes, not a characterization; conjugacy classes with the same number of fixed points per element need not have the same degeneracy set.

Every degeneracy set is a subset of the *master degeneracy set*. The master degeneracy set contains all the information relevant to counting fixed points under a given representation for conjugacy classes of a given type. Most importantly, the master degeneracy set is independent of the field the matrices are defined over.

This chapter is devoted to constructing and describing degeneracy sets, and computing the number of fixed points for an element in a conjugacy class with a given degeneracy set. Throughout this chapter we will be primarily concerned with the representation $\Gamma^{(s)}$, but our methods will be applicable for a wide range of

representations, and our examples may use simpler representations.

The previous chapter showed how to calculate the size of a conjugacy class of a given type. This chapter shows how to calculate the number of fixed points per element in a conjugacy class of given type and degeneracy set. The next chapter shows how to calculate the number of conjugacy classes with a given degeneracy set. Combining these three results gives us the number of fixed points for elements in a given type and by summing over all types, we use the results in Chapter 2 to obtain the number of $p$-groups of exponent-$p$-class two.

To understand how we will approach the problem of counting fixed points, consider the following example. For the moment, we do not concern ourselves with being field-free, and fix a finite field $\mathbb{F}_p$. Consider the group $\mathrm{GL}(2;\mathbb{F}_p)$ and the type $\tau = \{(1,(1,1))\}$. A representative element of a conjugacy class of this type is

$$A = \begin{pmatrix} \alpha & \\ & \alpha \end{pmatrix}, \tag{4.1}$$

where $\alpha$ is a nonzero element of $\mathbb{F}_p$. Each conjugacy class of this type corresponds to a unique choice of $\alpha$. This means there are exactly $p-1$ conjugacy classes of this type in $\mathrm{GL}(2;\mathbb{F}_p)$.

Consider the induced (polynomial) representation $\Gamma$ of $\mathrm{GL}(2;\mathbb{F}_p)$ on $V \oplus V \wedge V$ where $V = (\mathbb{F}_p)^2$. The vector space $V \oplus V \wedge V$ has dimension three, and therefore is isomorphic to $(\mathbb{F}_p)^3$. The image of $A$ under $\Gamma$, acting on $(\mathbb{F}_p)^3$, can be given explicitly as

$$\Gamma(A) = \begin{pmatrix} \alpha & & \\ & \alpha & \\ & & \alpha^2 \end{pmatrix}. \tag{4.2}$$

The number of fixed points of $\Gamma(A)$ depends on the value of $\alpha$. If $\alpha = 1$ then there are $p^3$ fixed points. If $\alpha^2 = 1$ but $\alpha \neq 1$, then there are $p$ fixed points. If $\alpha^2 \neq 1$, then there is only one fixed point, the zero vector.

The number of fixed points of $\Gamma(A)$ was easy to find as $\Gamma(A)$ was in diagonal form.

Counting fixed points for matrices in Jordan canonical form is just as straightforward. A Jordan block

$$J_n(\alpha) = \begin{pmatrix} \alpha & 1 & & \\ & \alpha & 1 & \\ & & \ddots & 1 \\ & & & \alpha \end{pmatrix}$$

fixes a one-dimensional subspace if and only if $\alpha = 1$. Otherwise there is only one fixed point. In other words, if $J_n(\alpha)$ is defined over $\mathbb{F}_p$, the number of fixed points of the matrix $J_n(\alpha)$ is

$$\#\text{ fixed points} = \begin{cases} p & \text{if } \alpha = 1, \\ 1 & \text{if } \alpha \neq 1. \end{cases} \tag{4.3}$$

Suppose for an arbitrary matrix $A$ in $\mathrm{GL}(2;\mathbb{F}_p)$ we can reduce $\Gamma(A)$ to its Jordan canonical form, and the Jordan blocks are given as $J_{n_i}(\alpha_i)$. The elements $\alpha_i$ will be in terms of the entries in $A$ since $\Gamma$ is a polynomial representation. Then it is clear that we can count the number of fixed points of $\Gamma(A)$ by counting the number of elements $\alpha_i$ that are unity. If the number of $\alpha_i$ equal to one is $f$, then there are $p^f$ fixed points of $\Gamma(A)$.

Following such an approach whilst remaining independent of the field $\mathbb{F}_p$ has two difficulties: some matrices $\Gamma(A)$ cannot be reduced to Jordan canonical form over $\mathbb{F}_p$, and we need to be able to determine which $\alpha_i$ are equal to 1 in a manner independent of the prime $p$. These difficulties can be overcome by an approach proposed by Higman:

1. For conjugacy classes of a given species, abstract the Jordan canonical form of matrices in those classes defined over an appropriate field. This abstract Jordan canonical form is called the *guiding member of a species*, denoted $\mathcal{A}$.

2. Consider the Jordan canonical form of $\Gamma(\mathcal{A})$. This is an abstraction of the Jordan canonical form of $\Gamma(A)$ for all matrices $A$ of the same species. We can obtain a set containing information about the Jordan blocks of $\Gamma(\mathcal{A})$ in terms of the Jordan blocks of $\mathcal{A}$. This set is called the *master predegeneracy set* and depends only on

the species and the representation.

3. The information in the master predegeneracy set can be brought back into the context of types, whilst remaining independent of the field the matrices are defined over. The master predegeneracy set translated into the context of a particular type is called the *master degeneracy set*. It depends on the type.

4. Subsets of the master degeneracy set, called *degeneracy sets*, correspond to a set of conjugacy classes of a given type but do not depend on the field. Conjugacy classes defined over $\mathbb{F}_p$ with the same degeneracy set have the same number of fixed points. We can determine this number as a function of $p$ using just the degeneracy set.

In short, we can count the number of fixed points for a matrix $\Gamma(A)$ over all finite fields by abstracting $A$ first to the guiding member, then consider the representation of the guiding member and then bring the fixed point information back down via types using degeneracy sets. This approach allows us to count fixed points whilst keeping $p$ variable. Species and types enable this generality by being sufficiently independent of the field $A$ is defined over.

The term "degeneracy" was proposed by Higman [21], and it is used to mean that the associated conjugacy class is limited or special, in light of the fixed point calculation. That is, the degeneracy set represents the restrictions on a conjugacy class so that it has the prescribed number of fixed points. These restrictions are similar to those in Equation (4.3). We must stress this is a classification of conjugacy classes, not a characterization. Each conjugacy class of a given type has an associated degeneracy set. Every matrix defined over $\mathbb{F}_p$ in a conjugacy class with a given degeneracy set has the same number of fixed points. This number depends only on $p$ and the degeneracy set.

The rest of the chapter is as follows. First we will discuss the guiding member of a species and how to construct the master predegeneracy set. Then we will show how to construct the master degeneracy set from the master predegeneracy set. Following

this we show how to count the number of fixed points for a conjugacy class of a given degeneracy set. We finish with a discussion of properties of degeneracy sets which form a basis for the next chapter which deals with counting conjugacy classes with a given degeneracy set.

## 4.1 The guiding member of a species

This section describes the first step in Higman's approach: abstracting matrices of a given species via the guiding member of that species. The theory in this section is due to Higman. The terminology and presentation is inspired by Higman's paper, although certain details are original.

Recall that the species of a matrix is independent of the field that the matrix is defined over. Matrices of the same type have the same species. The guiding member of a species represents an abstraction of the Jordan canonical form for all matrices of that species (defined over the appropriate splitting field), and in particular, all matrices of a given type of that species. This abstraction frees us from algebraic considerations particular to the field the matrices are defined over. The guiding member is defined over a particular field that is "trouble-free" in an algebraic sense. The field we choose is the complex numbers $\mathbb{C}$. The field $\mathbb{C}$ has characteristic zero so we do not have any unwanted coincidences like $2x = 0$ which occurs in a field with even characteristic. Matrices defined over $\mathbb{C}$ always have a Jordan canonical form over that field. The field $\mathbb{C}$ also provides us with a countably infinite set $\mathcal{X}$ of elements $x_1, x_2, \ldots$ and their inverses, where the elements of $\mathcal{X}$ are algebraically independent over $\mathbb{Q}$. We want to use these elements as variables so we do not want any algebraic relations between them. We could have equivalently used the field $\mathbb{Q}(x_1, x_2, \ldots, )$ over countably many variables, but it is much more convenient to choose algebraically independent elements from $\mathbb{C}$ and just treat them like variables, which is what we will do. We assume the elements of $\mathcal{X}$ have a fixed order.

A species is given by a partition, which we will denote by $\lambda = (\lambda_1, \ldots, \lambda_v)$. The

guiding member of a species given by $\lambda$ is the matrix

$$\mathcal{A} = \bigoplus_{i=1}^{v} J_{\lambda_i}(x_i),$$

where $x_i \in \mathcal{X}$. Note that for a fixed species, we only require the first $v$ elements from $\mathcal{X}$. We made $\mathcal{X}$ countably infinite so that we had a single, convenient context for all species. For matrices in $\mathrm{GL}(r, s; \mathbb{F}_p)$, we know that $v$ is at most $r + s$. Typically for $\mathrm{GL}(r, s; \mathbb{F}_p)$ we will consider the guiding member in $\mathrm{GL}(r; \mathbb{F}_p)$ and $\mathrm{GL}(s; \mathbb{F}_p)$ separately, though the $x_i$ will be chosen from $x_1, \ldots, x_{r+s}$.

As an example, let $\lambda = (2, 2)$. The guiding member of the species given by $\lambda$ is

$$\mathcal{A} = \begin{pmatrix} x_1 & 1 & & \\ & x_1 & & \\ & & x_2 & 1 \\ & & & x_2 \end{pmatrix}.$$

Consider all the types of size 4 with species $\lambda$:

$$\tau_1 = \{(1, (2)), (1, (2))\}, \quad \tau_2 = \{(1, (2, 2))\}, \quad \tau_3 = \{(2, (2))\}.$$

These types have representative matrices $A_1, A_2$, and $A_3$ respectively:

$$A_1 = \begin{pmatrix} \beta_1 & 1 & & \\ & \beta_1 & & \\ & & \beta_2 & 1 \\ & & & \beta_2 \end{pmatrix}, \; A_2 = \begin{pmatrix} \beta_3 & 1 & & \\ & \beta_3 & & \\ & & \beta_3 & 1 \\ & & & \beta_3 \end{pmatrix}, \; A_3 = \begin{pmatrix} \beta_4 & 1 & & \\ & \beta_4 & & \\ & & \beta_4^\sigma & 1 \\ & & & \beta_4^\sigma \end{pmatrix},$$

where $\beta_1, \beta_2, \beta_3 \in \mathbb{F}_p$, and $\beta_4, \beta_4^\sigma \in \mathbb{F}_{p^2}$. The matrices $A_1$, $A_2$ and $A_3$ are in Jordan canonical form. We can see how $\mathcal{A}$ represents all three matrices by ensuring that each Jordan block is distinct, even if in a particular type, two Jordan blocks are identical. Identical blocks, or blocks that are related by Galois conjugation (as in $A_3$) is a restriction from the type, not the species. These restrictions are included later when

we specialize to a particular type.

Note that Higman defines the guiding member of a species to be the matrix

$$\bigoplus_{i=1}^{v} x_i J_{\lambda_i}(1).$$

This matrix is conjugate to $\mathcal{A}$ over $\mathbb{C}$. He chooses this form to make the proof of Lemma 3.1 in [21] easier. For our approach, there is no advantage in adopting this form.

## 4.2 Master predegeneracy set

Having abstracted the Jordan canonical form via the guiding member $\mathcal{A}$, we now want to consider the Jordan canonical form of $\Gamma(\mathcal{A})$. In this section we assume the representation $\Gamma$ is an arbitrary polynomial representation, although one made up of components geared towards the representation we are most interested in: $\Gamma^{(s)}$ from Chapter 3. The part of the main algorithm corresponding to this section is the only part that depends on the representation $\Gamma^{(s)}$. That is, the algorithm is modular with respect to the representation; if we want to use a different representation, we only need to change one part of the main algorithm.

The Jordan canonical form of a matrix specifies its elementary divisors and vice-versa. We will examine the Jordan canonical form of $\Gamma(\mathcal{A})$ by considering the elementary divisors of $\Gamma(\mathcal{A})$ in terms of the elementary divisors of $\mathcal{A}$. For the species given by $\lambda = (\lambda_1, \ldots, \lambda_v)$, the elementary divisors of $\mathcal{A}$ are $(x - x_i)^{\lambda_i}$, for $i = 1, \ldots, v$.

All of the representations we consider will be polynomial representations. As such the entries of $\Gamma(\mathcal{A})$ are polynomials in terms of those of $\mathcal{A}$. Specifically, every nonzero entry of $\Gamma(\mathcal{A})$ will be a sum of products of the form

$$x_1^{t_1} \cdots x_v^{t_v}$$

for nonnegative integers $t_1, \ldots, t_v$. The Jordan canonical form of $\Gamma(\mathcal{A})$ also has this

property. Equivalently, the elementary divisors of $\Gamma(\mathcal{A})$ are of the form $(x-\alpha)^m$ for some positive integer $m$ and $\alpha = x_1^{t_1} \cdots x_v^{t_v}$ for some choice of $t_1, \ldots, t_v$. Later we will want to allow some $t_i$ to be negative, so we accommodate this generality now.

We will find the elementary divisors of $\Gamma(\mathcal{A})$ by a "*calculus of elementary divisors*". This calculus transforms the set of elementary divisors of $\mathcal{A}$ to those of $\Gamma(\mathcal{A})$ via transformations related to how $\Gamma$ takes $\mathcal{A}$ to its image. This is done via five rules, based on the following results obtained from Marcus' books on multilinear algebra [34, 35]. Throughout these results, $A$ and $B$ are matrices over $\mathbb{C}$ whose elementary divisors are of the form $(x-\alpha)^m$ where $\alpha \neq 0$. We use positive integers $m$ and $n$, but stress that they do not relate to $m$ and $n$ used elsewhere in this thesis.

**Theorem 4.1** (Elementary divisors of transposed matrices)**.** *The set of elementary divisors of a matrix $A^T$ is identical to the set of elementary divisors of $A$.*

**Theorem 4.2** (Elementary divisors of direct sums)**.** *The set of elementary divisors of $A \oplus B$ is the union of the set of elementary divisors of $A$ with the set of elementary divisors of $B$.*

**Theorem 4.3** (Elementary divisors of tensor products)**.** *Suppose $(x-\alpha)^m$ is an elementary divisor of $A$ and $(x-\beta)^n$ is an elementary divisor of $B$. They produce a set $(x-\alpha)^m \otimes (x-\beta)^n$ of elementary divisors of $A \otimes B$*

$$(x-\alpha)^m \otimes (x-\beta)^n := \left\{ (x-\alpha\beta)^{m+n-(2t-1)} \mid \text{ where } t = 1, \ldots, \min(m,n) \right\}.$$

*The set of all elementary divisors of $A \otimes B$ is the union of sets $(x-\alpha)^m \otimes (x-\beta)^n$ for all choices of elementary divisors $(x-\alpha)^m$ from $A$ and $(x-\beta)^n$ from $B$.*

**Theorem 4.4** (Elementary divisors of compound matrices)**.** *Suppose $A = A_1 \oplus A_2$, and $A$ acts on the corresponding vector space $U = V \oplus W$. Let $C_m(A)$ be the induced action of $A$ on $\bigwedge_{i=1}^m U$, that is, the $m$th compound matrix of $A$. Then the elementary divisors of $C_m(A)$ are the same as those of the linear map*

$$\bigoplus_{i=0}^{m} C_i(A_1) \otimes C_{m-i}(A_2).$$

*The matrices $C_i(A_1)$ and $C_{m-i}(A_2)$ are regarded as linear maps on $\bigwedge^i V$ and $\bigwedge^{m-i} W$ respectively, for all $i = 0, \ldots, m$, where $\bigwedge^0 V$ and $\bigwedge^0 W$ are both isomorphic to the trivial vector space.*

For the last theorem, suppose we have natural numbers $n \geq m$. Let $p(r, m, n)$ be the number of integer partitions of $r$ into not more than $m$ parts, each part of size no greater than $n - m$. We define $p(0, m, n) = 1$ for all $m$ and $n$. Let $c(r, m, n) = p(r, m, n) - p(r - 1, m, n)$ for $r \geq 1$, and $c(0, m, n) = 1$.

**Theorem 4.5** (Elementary divisors of the compound matrix of a single block)**.** *Let $A$ be the Jordan block $J_n(\alpha)$, with $\alpha$ nonzero. Then for $m > 0$, the elementary divisors of $C_m(A)$ are*

$$(x - \alpha^m)^{m(n-m)-(2r-1)},$$

*repeated $c(r, m, n)$ times, where $r = 0, \ldots, m(n - m)$. For $m = 0$, we say that $C_m(A)$ has no elementary divisors.*

With these five results, we can form a calculus of elementary divisors for our purposes. Our calculus transforms sets of elementary divisors via the above results. For brevity we represent an elementary divisor $(x - \alpha)^m$ by a pair $(\alpha, m)$. Let $X$ and $Y$ be sets of pairs representing elementary divisors. We have four operations on sets of elementary divisors: two unary operators $(\cdot)^T$ and $C_m(\cdot)$, and two binary operators $\oplus$ and $\otimes$. These operators have the following rules, corresponding to the previous five results.

**Theorem 4.6.** *Let $X$ and $Y$ be sets of pairs $(\alpha, m)$, each representing an elementary divisor of a particular matrix, where $\alpha \in \mathbb{C}$ and $m$ is a positive integer. We define a calculus of elementary divisors using the operators $(\cdot)^T$, $\oplus$, $\otimes$ and $C_m(\cdot)$ satisfying:*

*Rule 1.* $X^T = X$.

*Rule 2.* $X \oplus Y = X \cup Y$.

*Rule 3.* $X \otimes Y = \{(\alpha\beta, m + n - (2t - 1)) \mid (\alpha, m) \in X, (\beta, n) \in Y, 1 \leq t \leq \min(m, n)\}$.

*Rule 4.* *Suppose $X = X' \cup X''$, with $X'$ and $X''$ both nonempty. Then*

$$C_m(X) = \bigoplus_{i=0}^{m} C_i(X') \otimes C_{m-i}(X'').$$

*Rule 5.* *Suppose $X = \{(\alpha, n)\}$ and $c(r, m, n)$ is defined as before. Then if $m > 0$, $C_m(X)$ contains $c(r, m, n)$ copies of $(\alpha^m, m(n-m) - (2r-1))$, for $r = 0, \ldots, m(n-m)$. If $m = 0$ then $C_m(X) = \emptyset$.*

By recursively applying these rules, we can calculate the elementary divisors of images of matrices under various kinds of representations. Note that $\oplus$ and $\otimes$ are commutative, and $C_m(X)$ in Rule 4 does not depend on how the set $X$ is partitioned into $X'$ and $X''$. To compute $C_m(X)$ in this case, we take $X'$ to be a singleton set (and thus Rule 5 applies) and recursively apply Rule 4 to $X \setminus X'$.

Applying these rules is a nuisance to do by hand, but it is particularly easy using a computer. Since the operations require very little calculation, the associated algorithm is fast.

We now show how to use this calculus of elementary divisors for the particular representation $\Gamma^{(s)}$. Let $A \in \mathrm{GL}(r; K)$ have a set of elementary divisors given by $X$ in $(\alpha, m)$ pair form, and let $B \in \mathrm{GL}(s; K)$ have a set of elementary divisors given by $Y$ in the same form. In terms of the calculus of elementary divisors, the elementary divisors of $\Gamma^{(s)}(A, B)$ are given by

$$Y^T \otimes (X \oplus C_2(X)). \tag{4.4}$$

Of course $Y = Y^T$ so we can simplify this in the obvious way.

Suppose we know the types of $A$ and $B$. From each type we can determine the species of $A$ and $B$, and from that, the guiding members of those species. From the guiding member of the species of $A$, we obtain a matrix

$$\mathcal{A} = \bigoplus_{i=1}^{v_1} J_{\lambda_i}(x_i),$$

where the $\lambda_i$ are the parts of the partition given by the species of $A$ and $v_1$ is sum of

$d_i \cdot \mu_i$ over all type-parameters $(d_i, \mu_i, 1)$. We obtain a similar matrix $\mathcal{A}'$ from $B$,

$$\mathcal{A}' = \bigoplus_{i=v_1+1}^{v_2} J_{\lambda'_i}(x_i),$$

where $\lambda'_i$ are the parts of the partition given by the species of $B$ and $v_2$ is the sum of $d_i \cdot \mu_i$ over all type-parameters $(d_i, \mu_i, 2)$, plus $v_1$ (to keep the indices of the $x_i$ in $\mathcal{A}'$ from colliding with those in $\mathcal{A}$). Therefore our sets $X$ and $Y$ of elementary divisors (represented as pairs) are

$$X = \{(x_i, \lambda_i) \mid i = 1, 2, \dots\}, \quad \text{and} \quad Y = \{(x'_i, \lambda'_i) \mid i = 1, 2, \dots\}.$$

Given these two sets and the representation given by (4.4), we can apply the rules in Theorem 4.6 to obtain the elementary divisors of $\Gamma^{(s)}(A, B)$. The elementary divisors of $\Gamma^{(s)}(A, B)$ are pairs of the form $(x_1^{t_1} \dots x_v^{t_v}, m)$ where $v = r + s$. From the set of products $x_1^{t_1} \dots x_v^{t_v}$ we obtain a set of tuples $(t_1, \dots, t_v)$. This set of tuples is called the *master predegeneracy set* and we denote it by $T$. We choose this tuple form as we will eventually map the elements $x_i$ to finite field elements and the exponents will be the only information we need to consider.

Note that in the process of creating the tuples $(t_1, \dots, t_v)$ in $T$, we discard the associated integers $m$ from the pair representing elementary divisors. This information is irrelevant to counting fixed points as we only need to know that it is positive, which we already assume (since it corresponds to an elementary divisor $(x - x_1^{t_1} \dots x_v^{t_v})^m$.)

## 4.3 Master degeneracy set

The master predegeneracy set contains all the information regarding the Jordan canonical form of $\Gamma(\mathcal{A})$. In other words, this information is in the context of species. Other results in this thesis rely on the context of types. For example, the size of a conjugacy class of a given type is a specific polynomial depending only on the type. A similar result does not exist for species. Therefore we would like to take the information

in the master predegeneracy set and bring it back down into the context of types. To do this we need to undo some of the generalization that took Jordan blocks of a representative element of a type to Jordan blocks in the guiding member of the species. On page 40 there are examples of the generalization process (three types are shown to have the same guiding member). That example may be helpful in visualizing what is going on in the context of the Jordan canonical form.

To determine the species given by a partition $\lambda$ of a type $\tau$, we concatenate the partitions of the type-parameters, taken with multiplicity given by its degree. For example, the type $\{(1,(1,2)),(2,(3))\}$ has species $(1,2,3,3)$. Reversing this process, and thus taking a species to a particular type, collects parts in $\lambda$ into sub-partitions and collects a selection of these sub-partitions into a single type-parameter. In the example before, we first collect $(1,2,3,3)$ into sub-partitions $(1,2)$, $(3)$, and $(3)$. The last two sub-partitions are collected to form a type-parameter of degree 2, and the first sub-partition forms a type-parameter of degree 1. We can visualize this as

$$\lambda = (1,2,3,3) = (\underbrace{1,2}_{(1,(1,2))}\ ,\ \underbrace{3,3}_{(2,(3))}).$$

Recall that via the guiding member of a species, a part $\lambda_i$ in the partition $\lambda$ has a corresponding element $x_i$. The process collecting parts in $\lambda$ can therefore be considered as collecting the elements $x_i$. The two stages of collection can be interpreted as:

1. Collecting parts $\lambda_i$ into a sub-partition identifies the elements $x_i$ for the appropriate indices $i$;

2. Collecting sub-partitions into type-parameters collects elements $x_i$ in a way analogous to identifying finite field elements as Galois conjugates.

This last stage means that specializing from a species to a type $\tau$, we have an induced automorphism $\sigma_\tau$ that permutes the elements $x_i$ according to how these elements collect according to the type.

For example, take again the species given by $\lambda = (1,2,3,3)$ and the type

$\tau = \{(1,(1,2)),(2,(3))\}$. The guiding member of the species specifies elements $x_1$, $x_2$, $x_3$ and $x_4$ corresponding to parts 1, 2, 3 and 3 respectively. The process that forms the type-parameter $(1,(1,2))$ identifies $x_1$ and $x_2$. The process that forms the type-parameter $(2,(3))$ collects the elements $x_3$ and $x_4$, but then specifies that $x_4 = (x_3)^{\sigma_\tau}$ via the "Galois conjugation" analogy.

The process taking a species to a particular type $\tau$ allows us to write a term $x_1^{t_1} \dots x_v^{t_v}$ in terms of elements $x_i \in \mathcal{A}$ and the automorphism $\sigma_\tau$. If in the types context we are permitted to renumber the indices of $x_i$ from $x_1$ to $x_w$ for convenience, a term

$$x_1^{t_1} \cdots x_v^{t_v}$$

is rewritten in the types context as a term

$$x_1^{u_1} \cdots x_w^{u_w},$$

where the exponents $u_i$ are polynomials in $\sigma$ with integer coefficients. That is, a term

$$x_i^{a_0+\sigma a_1+\cdots+\sigma^d a_d} \qquad \text{is equivalent to} \qquad (x_i)^{a_0} \cdot (x_i^{\sigma})^{a_1} \cdots (x_i^{\sigma^d})^{a_d}.$$

In the types context, there are $w$ variables $x_1, \dots, x_w$ — precisely the number of type-parameters in $\tau$. Furthermore, the variable $x_i$ in the types context is associated to the type-parameter $(d_i, \mu_i, l_i)$. That is, we keep $\tau$ ordered and associate the $i$th type-parameter to the $i$th element $x_i$ of $\mathfrak{X}$.

The translation from the species context to the context of a particular type $\tau$ restates the master predegeneracy set $T$ as a set called the *master degeneracy set*, denoted $E_\tau$. It takes a tuple $(t_1, \dots, t_v)$ to $(u_1, \dots, u_w)$. The master degeneracy set $E_\tau$ describes the Jordan canonical form of $\Gamma(\mathcal{A}_\tau)$ where $\mathcal{A}_\tau$ is the analogue of the guiding member of a species, but for a type $\tau$. In other words, $\mathcal{A}$ translates to $\mathcal{A}_\tau$ according to how the species specializes to the type $\tau$.

## 4.4 Number of fixed points

The master degeneracy set $E_\tau$ gives us the general structure of the Jordan canonical form of $\Gamma(A)$ for a matrix $A$ of a given type. We will use this structure to choose conjugacy classes defined over finite fields such that the elements in these conjugacy classes all have the same number of fixed points.

Let $x_1, \ldots, x_w$ be the variables used in the master degeneracy set for the type $\tau$. Each variable represents a type-parameter in $\tau$. If $d$ is the least common multiple of the degrees $d_i$ of the type-parameters $(d_i, \mu_i, l_i) \in \tau$, then there is an associated finite field $\mathbb{F}_q$ for $q = p^d$. A matrix of type $\tau$ read over $\mathbb{F}_q$ has a Jordan canonical form. We can specify a conjugacy class in $\tau$ read over $\mathbb{F}_q$ by what is called a *specialization*. A specialization is a map $\phi$ taking $x_1, \ldots, x_w$ and their $\sigma_\tau$ conjugates to $\mathbb{F}_q^*$, the multiplicative group of $\mathbb{F}_q$, with the following conditions:

- $\phi(x_i) \in \mathbb{F}_{q_i}^*$ for $q_i = p^{d_i}$, but note that $\phi(x_i)$ might be in a proper subfield of $\mathbb{F}_{q_i}$;
- $\sigma \cdot \phi(x_i) = \phi(x_i^{\sigma_\tau})$.

In other words, a specialization maps the variables $x_i$ to finite field elements, weakly respecting the conditions from the type $\tau$: the choice of $\phi(x_i)$ do not have to be distinct and $\phi(x_i)$ need not have exactly $d_i$ Galois conjugates. This gives us the freedom to use specializations without worrying about the technicalities required by the type. We impose the restrictions from the type in other parts of the algorithm. This is described in the next section.

Suppose $\phi$ is a specialization taking $x_i$ to $\beta_i$ for $i = 1, \ldots, w$. We can form the image of a matrix under $\phi$ by mapping each of its entries via $\phi$. The image of $\mathcal{A}_\tau$ under a specialization $\phi$ gives a representative element of a conjugacy class of type $\tau$. The representative element of the conjugacy class given by the specialization is formed by replacing every Jordan block $J_k(x_i)$ in $\mathcal{A}_\tau$ with $J_k(\beta_i)$.

Similarly, the image of $\Gamma(\mathcal{A}_\tau)$ under a specialization $\phi$ yields a matrix $\Gamma(A)$ where $A$ is a representative element of a particular conjugacy class. Abstracting conjugacy

classes through species, then specializing back down through types permits this coherency of $\Gamma(\mathcal{A}_\tau)$ to $\Gamma(A)$ via a specialization. This is the value of Higman's approach.

A conjugacy class of a certain type is given by a specialization. The question we now ask is: how many fixed points does an element of a particular conjugacy class, under the image of a representation $\Gamma$, have? Recall that in Equation (4.3), a Jordan block $J_n(\alpha)$ in a matrix has $p$ fixed points over $\mathbb{F}_p$ if and only if $\alpha = 1$, and has 1 fixed point otherwise. Therefore, to compute the number of fixed points of a matrix $\Gamma(A)$, we need to know its Jordan blocks. If $A$ is the specialization of $\mathcal{A}_\tau$ via the map $\phi(x_i) = \beta_i$, then the Jordan blocks of $\Gamma(A)$ are $J_k(\beta)$ for some integer $k$ and

$$\beta = \beta_1^{u_1} \cdots \beta_w^{u_w}$$

where $(u_1, \ldots, u_w)$ is an element of the master degeneracy set $E_\tau$. Thus a particular Jordan block $J_k(\beta)$ has $p$ fixed points if

$$\beta_1^{u_1} \cdots \beta_w^{u_w} = 1. \tag{4.5}$$

This depends on the choice of $\beta_i$, and hence the specialization $\phi$. A specialization will satisfy a subset of the equations of the form in Equation (4.5). Let $S$ be the set of tuples $(u_1, \ldots, u_w)$ for which Equation (4.5) holds for that particular specialization. This will be some (possibly empty) subset of $E_\tau$. We call such a set the *degeneracy set* of the specialization $\phi$.

Two different specializations can satisfy the same set of equations, and thus have the same degeneracy set. Since these equations correspond to Jordan blocks in $\Gamma(A)$, specializations with the same degeneracy set correspond to conjugacy classes such that their elements have the same number of fixed points under the image of $\Gamma$. We say that a conjugacy class given by a specialization with degeneracy set $S$ has "degeneracy" given by $S$. Therefore the conjugacy classes with degeneracy $S$ all have the same number of fixed points under $\Gamma$, and thus we have our classification of conjugacy classes of a given type.

From this discussion it follows that the number of fixed points of an element in a conjugacy class, defined over $\mathbb{F}_p$, under the image of $\Gamma$, depends only on the type of the conjugacy class and the degeneracy set $S$. The remainder of this section is devoted to showing how to calculate it.

A degeneracy set $S$ specifies a set of equations in the form of Equation (4.5). By definition, these are the only equations specified by elements in $E_\tau$ that hold. Let $\beta(u) = \beta_1^{u_1} \cdots \beta_w^{u_w}$ for $u = (u_1, \ldots, u_w) \in E_\tau$ and some specialization $\phi : x_i \mapsto \beta_i$. Let $A$ be the matrix formed by the specialization $\phi$ applied to $\mathcal{A}_\tau$. For $u \in S$, there is at least one Jordan block in $A$ corresponding to $J_k(\beta(u))$. Let $f_S$ be the total number of Jordan blocks $J_k(\beta(u))$ in $A$ for some positive integer $k$ and some $u \in S$. Let $\chi(A)$ be the number of fixed points of $\Gamma(A)$, for $A$ in a conjugacy class with degeneracy set $S$. Then if $A$ is defined over $\mathbb{F}_p$

$$\chi(A) = p^{f_S}.$$

This does not depend on the specialization, so we pick an element $g_S$ that represents all conjugacy classes of degeneracy $S$. Therefore we want to calculate

$$\chi(g_S) = p^{f_S}$$

and thus calculating $f_S$ suffices to find $\chi(A)$ for all conjugacy classes with degeneracy $S$, regardless of the specialization taking $\mathcal{A}_\tau$ to $A$.

We now want to compute $f_S$ for a given degeneracy set $S$. Note that $f_S$ is zero if $S$ is empty. Otherwise, $f_S$ is positive. Furthermore, $f_S$ is at least as large as $S$ since there must be a Jordan block for each equation given by $S$. The only complication is that a single equation $\beta(u) = 1$ may have multiple Jordan blocks. For example, let our species be given by $\lambda = (1, 1)$ and let our type be $\tau = \{(1, (1, 1))\}$. The master predegeneracy set specifies terms of the form $x_1^{t_1} x_2^{t_2}$. The master degeneracy set specifies terms of the form $x_1^{u_1}$ since $x_1$ and $x_2$ are identified as $x_1$ by the species-to-type translation, and $u_1 = t_1 + t_2$. For simplicity's sake, let the representation $\Gamma$ be the natural representation. In this case the master predegeneracy set is $T = \{(1, 0), (0, 1)\}$ and the master degen-

eracy set $E_\tau = \{(1)\}$. The single element in $E_\tau$ actually corresponds to two Jordan blocks. Therefore if $S = E_\tau$, then $f_S = 2$, and so a conjugacy class of degeneracy $S$ has $p^2$ fixed points per element.

Recall that the master degeneracy set is the translation of the master predegeneracy set to the types context. Therefore a degeneracy set $S$ is the translation of some subset $S'$ of the master predegeneracy set. The size of $S'$ is $f_S$ as an element of the master predegeneracy set corresponds to exactly one Jordan block. Therefore during the translation process we keep track of the map $\psi : (t_1, \ldots, t_v) \mapsto (u_1, \ldots, u_w)$, and then the preimage $\psi^{-1}(S)$ of $S$ under $\psi$ is the set $S'$, and so $f_S = |\psi^{-1}(S)|$.

## 4.5 Base equalities and inequalities

Degeneracy sets contain information relevant to counting fixed points for a particular type and representation. A conjugacy class with a given degeneracy $S$ is the result of a specialization satisfying equations specified by $S$. However, specializations for a particular conjugacy class are only restricted by the degeneracy, and may not respect the restrictions imposed by the type. For example, a specialization $\phi$ may choose the same value for elements $\beta_i$ and $\beta_j$ that are supposed to correspond to different type parameters, and hence be distinct. Such restrictions are irrelevant to calculating the number of fixed points for an element in a conjugacy class of degeneracy $S$. However, it is relevant to the number of conjugacy classes with degeneracy $S$ (the calculation of which is the focus of Chapter 5). For the purpose of counting conjugacy classes with degeneracy $S$, we create another set in the same form as $S$ but it encapsulates the restrictions specified by the type. This set is called the *base type inequalities* for reasons soon to be apparent.

Recall that if $u \in S$ then any specialization with that degeneracy must satisfy the equation $\beta(u) = 1$. Conversely, if $u \in E_\tau$ but $u \notin S$, then a specialization must not satisfy $\beta(u) = 1$. That is, $\beta(u) \neq 1$ in this case. Therefore a degeneracy set specifies a set of equations and inequalities a specialization with that degeneracy set must obey. The base type inequalities is a set of tuples $u'$ that are *never* contained in a degeneracy

set, and thus always specify that $\beta(u') \neq 1$. As mentioned before, these elements $u'$ correspond to restrictions imposed by the type.

As an example of these inequalities, consider a small example. Suppose $\beta_i$ and $\beta_j$ are finite field elements that must be distinct as they correspond to distinct type-parameters. This is simply stated as $\beta_i \neq \beta_j$. Equations specified by the elements in the master degeneracy set are always of the form $\beta(u) = \beta_1^{u_1} \cdots \beta_w^{u_w} = 1$, and inequalities as $\beta_1^{u_1} \cdots \beta_w^{u_w} \neq 1$. We can rewrite our inequality $\beta_i \neq \beta_j$ as $\beta_i \beta_j^{-1} \neq 1$. We write this in tuple form $(u_1, \ldots, u_w)$ as

$$(0, \ldots, 0, 1, 0, \ldots, 0, -1, 0, \ldots, 0),$$

where 1 is in the $i$th position and $-1$ in the $j$th position. This is enforced as an inequality by not including it in any degeneracy set.

Specializations yielding conjugacy classes of a given type must satisfy the restrictions given by the base type inequalities. We could also impose *equalities*, by constructing a set (called the *base type equalities*) and requiring that every degeneracy set $S$ contain this set. We will not require this extension, although the algorithm can easily accommodate it. This extension may be useful for, say, representations of $\mathrm{GL}(r, s; K)$ into $\mathrm{GL}(m; K)$ modulo some normal subgroup. This is beyond the scope of this thesis. We could conceivably also use it to reduce the number of types we work with so that, for example, instead of considering the distinct types $\{(1, (1, 1))\}$ and $\{(1, (1)), (1, (1))\}$, we can just take the latter and impose equalities in the degeneracy sets to obtain the former type. However, this method interferes with calculating the size of a conjugacy class of that type, and also substantially increases computation time beyond what we'd expect to save by considering the two types separately. So we do not consider this approach. Nevertheless, the algorithm we present has the capability to include base type equalities if a need arose.

### 4.5.1 Constructing the base type inequalities

In Lemma 3.3 of [21], Higman outlines the different inequalities we will require. Suppose we have type-parameters $y_1, \ldots, y_w$, where $y_i$ represents the triple $(d_i, \lambda_i, l_i)$ as explained in Chapter 3. The generic specialization on this type chooses a value of $\beta_i$ for the type-parameter $y_i$. The first requirement is that

$$\beta_i = \beta_i^{\sigma^d} \tag{4.6}$$

where $d = d_i$, but

$$\beta_i \neq \beta_i^{\sigma^b} \tag{4.7}$$

for any proper divisor $b$ of $d_i$. This is interpreted as $\beta_i$ must have degree *exactly* $d_i$. Also, if type-parameters $y_i$ and $y_j$ are in the same location (that is, $l_i = l_j$ but $i \neq j$) then elements $\beta_i$ must not be conjugate to $\beta_j$ (as they are supposed to be from distinct sets of Galois conjugates). Therefore for each pair $(i, j)$ with $i \neq j$ and $l_i = l_j$,

$$\beta_i \neq \beta_j^{\sigma^b}, \qquad \text{for } b = 0, 1, \ldots, d_j - 1. \tag{4.8}$$

The set of base type inequalities are simply those from (4.7) and (4.8), written in tuple form. The equalities in Equation (4.6) could possibly be incorporated in the base type equalities, but in practice we implicitly include these equalities by requiring for each $i = 1, \ldots, w$, the polynomial $u_i$ in $\sigma$ from

$$\beta_1^{u_1} \cdots \beta_w^{u_w} = 1$$

has degree at most $d_i - 1$. In the implementation of the algorithm, the polynomial $u = a_0 + a_1\sigma + \cdots + a_{d_i-1}\sigma^{d_i-1}$ is stored as a fixed-length vector of integers $(a_0, \ldots, a_{d_i-1})$. Galois conjugation $\sigma$ acts on this vector by cyclically permuting the entries in the appropriate manner. This is how we implement the equalities obtained from Equation (4.6).

## 4.6 Symmetries of degeneracy sets

Given a master degeneracy set $E_\tau$ of size $N$, there will be $2^N$ possible degeneracy sets. From a computational point of view, this is usually too many; the size of $E_\tau$ does not have to be too large for the collection of all degeneracy sets to be unmanageable. However, due to the particular framework we have at the moment, we can take advantage of certain symmetries in the degeneracy sets to alleviate this problem. Take for example the type $\tau = \{(1,(1)),(1,(1))\}$ and the natural representation so that $E_\tau = \{(1,0),(0,1)\}$. The element $(1,0)$ corresponds to the equation $\beta_1 = 1$, and the element $(0,1)$ to the equation $\beta_2 = 1$. The two associated type-parameters $y_1$ and $y_2$ are equivalent so a degeneracy set $S = \{(1,0)\}$ is equivalent to the degeneracy set $S' = \{(0,1)\}$. The type-parameters relate to the rational canonical form which is supposed to be independent of the ordering of the blocks, or equivalently, of the ordering of the type-parameters. The fixed point information will be the same. Specialization with degeneracy set $S$ gives precisely the same conjugacy class as a specialization with degeneracy $S'$, mapping the elements $x_i$ to $\beta_i$ in an equivalent manner. Therefore to obtain the set of degeneracy sets yielding distinct sets of conjugacy classes, we formulate an equivalence of degeneracy sets and pick one representative from each equivalence class.

There are three steps to defining our equivalence of degeneracy sets. Firstly, we create an equivalence of type-parameters. We can then create an equivalence of elements from the master degeneracy set based on the equivalence of type-parameters. This equivalence of elements in the master degeneracy set, combined with the Galois action $\sigma$, induces the required equivalence of degeneracy sets. The details follow.

Suppose we have type-parameters $y_1, \ldots, y_w$. Two type-parameters $y_i$ and $y_j$ are equivalent if and only if their degrees, partitions and locations are equal. That is, if $y_i = (d_i, \lambda_i, l_i)$ and $y_j = (d_j, \lambda_j, l_j)$ then $y_i \sim y_j$ if and only if $d_i = d_j$, $\lambda_i = \lambda_j$ and $l_i = l_j$. This is easily shown to be an equivalence relation. This equivalence induces a group action that permutes the indices on the $y_i$ such that each orbit corresponds to a set of mutually equivalent type-parameters. Denote the corresponding permutation

group by $\mathcal{G}$.

The equivalence amongst type-parameters induces an equivalence of elements from $E_\tau$. Let $u = (u_1, \ldots, u_w)$ be an element of $E_\tau$ corresponding to an equation

$$\beta_1^{u_1} \cdots \beta_w^{u_w} = 1.$$

If $\rho$ is an element of $\mathcal{G}$, it acts on the type-parameters by $\rho \cdot \beta_i = \beta_{(i)\rho}$. In other words, it permutes the indices. This induces an action of $\rho$ on $u$:

$$\rho \cdot u = \rho \cdot (u_1, \ldots, u_w) = (u_{(1)\rho}, \ldots, u_{(w)\rho}).$$

This permutes the elements of the vector $u$. Two elements $u$ and $u'$ from $E_\tau$ are equivalent if there is an element $\rho \in \mathcal{G}$ such that $\rho \cdot u = u'$. The Galois action $\sigma$ also induces an equivalence on elements of $E_\tau$. Recall that $u_i$ is a polynomial in $\sigma$ with integral coefficients with maximum degree $d_i - 1$, which can be represented as a vector $(a_0, \ldots, a_{d_i-1})$. The Galois action $\sigma$ acts on this vector by cyclically permuting the indices. That is,

$$\sigma \cdot (a_0, \ldots, a_{d_i-1}) = (a_{d_i-1}, a_0, \ldots, a_{d_i-2}).$$

This action on such polynomials induces an action on tuples of them:

$$\sigma \cdot (u_1, \ldots, u_w) = (\sigma u_1, \ldots, \sigma u_w).$$

This is an action on elements of $E_\tau$, and so two elements $u$ and $u'$ from $E_\tau$ are equivalent if $\sigma \cdot u = u'$. Since

$$\beta_1^{u_1} \cdots \beta_w^{u_w} = 1,$$

then it follows that

$$(\beta_1^{u_1} \cdots \beta_w^{u_w})^\sigma = \beta_1^{\sigma u_1} \cdots \beta_w^{\sigma u_w} = 1.$$

We encapsulate both equivalences via the orbits of a group action. The group acting will be denoted $\mathcal{H}$, and is the direct product of the group of permutations $\mathcal{G}$

and the group corresponding to the Galois action $\sigma$. The group $\mathcal{H}$ acts on elements on $E_\tau$. This induces an action on subsets of $E_\tau$, namely the degeneracy sets. That is, degeneracy sets $S$ and $S'$ are equivalent if there exists a $h \in \mathcal{H}$ such that $h \cdot S = S'$, where $h \cdot S = \{h \cdot s \mid s \in S\}$. The group $\mathcal{H}$ depends only on the type as $\mathcal{G}$ depends on the equivalence of type-parameters, and the Galois action $\sigma$ generates a cyclic group such that $\sigma^d = 1$, where $d$ is the lowest common multiple of the degrees of the type-parameters.

Note that the action of $\mathcal{H}$ preserves the size of the degeneracy set. This action partitions the collection of degeneracy sets into orbits. Each orbit corresponds to a set of conjugacy classes with equivalent degeneracy sets such that a specialization corresponds to the same conjugacy class. We only want to consider each conjugacy class once, so we need only consider a representative of each orbit. We will call representatives of these orbits *representative degeneracy sets*.

In practice the group $\mathcal{H}$ is easy to construct, as is the group action on degeneracy sets. Representative degeneracy sets can be found using a straightforward orbit calculation which is available in GAP by using the command

$$\texttt{Orbits}(\mathcal{H}, E_\tau, \texttt{OnDegeneracySets})$$

where $\mathcal{H}$ and $E_\tau$ are the appropriate GAP objects and `OnDegeneracySets` implements the action of $\mathcal{H}$ on degeneracy sets. This gives us a list of orbits and we choose one representative degeneracy set from each orbit. However, the collection of all degeneracy sets is large which makes such calculations very expensive. In Chapter 6 we discuss a method that determines representative degeneracy sets faster than the standard orbit calculation and requires much less memory. For now we assume we can obtain the representative degeneracy sets with no assumptions on the efficiency.

# Chapter 5

# Number of conjugacy classes with a given degeneracy set

We turn now to our final task: determine the number of conjugacy classes in $\mathrm{GL}(r, s; p)$ of a given type $\tau$ and degeneracy $S$. A conjugacy class with degeneracy $S$ is given by a specialization taking $x_1, \ldots, x_w$ in the types context to finite field elements $\beta_1, \ldots, \beta_w$, satisfying the restrictions dictated by $S$. Therefore our task is to count specializations satisfying the restrictions for a given degeneracy set $S$.

We must remember that in the current framework we do not distinguish two distinct specializations that yield the same conjugacy class. For example, if we have two identical type-parameters $y$ and $y'$ (both equal to $(d, \lambda, l)$), and a specialization chooses a value of $\beta$ for $y$ and a value of $\beta'$ for $y'$, then there is an equivalent specialization where the choices are swapped. This yields precisely the same conjugacy class. This is a continuation of the phenomena of equivalent degeneracy sets as described in Section 4.6. The root cause of this problem is in the arbitrary order we impose on our type-parameters, and the order of terms $x_i$ inside tuples in the master predegeneracy set and master degeneracy set which follow from the type-parameter ordering. It is computationally convenient to impose this order and we can easily accommodate it by defining an equivalence of specializations.

Simply stated, two specializations are equivalent if they produce the same conjugacy

class. This can also be described using the group $\mathcal{H}$ which defines the equivalence classes of degeneracy sets, as per the previous chapter. Recall that an element $h \in \mathcal{H}$ can be uniquely decomposed into a permutation $\pi$ acting on indices and a power of the Galois action $\sigma^j$. The element $h$ acts on elements of the degeneracy set, but can also act on finite field elements, and furthermore, on specializations into finite fields. Suppose we have a specialization $\phi$ taking $x_i$ to $\beta_i$ for $i = 1, \ldots, w$. Then the image of $\beta_i$ under the action of $h$ is $\beta_{(i)\pi}^{\sigma^j}$. This action is extended to the specialization $\phi$ by defining

$$h \cdot \phi(x_i) = \beta_{(i)\pi}^{\sigma^j} \qquad \text{where } h = (\pi, \sigma^j) \in \mathcal{H}.$$

A specialization $\phi$ with degeneracy set $S$ is equivalent to a specialization $\phi'$ with degeneracy set $S'$ if there is an element $h \in \mathcal{H}$ such that $h \cdot S = S'$ and for all $i = 1, \ldots, w$, the image $h \cdot \phi(x_i) = \phi'(x_i)$. This is easily shown to be an equivalence class.

Equivalent specializations correspond to the same conjugacy class. To count the correct number of distinct specializations we will first choose a representative degeneracy set $S$, and count all specializations with that degeneracy set. Two specializations $\phi$ and $\phi'$ both with degeneracy set $S$ are equivalent if $h \cdot S = S$ and for all $i = 1, \ldots, w$, the image $h \cdot \phi(x_i) = \phi'(x_i)$. Conversely, the number of equivalent specializations with degeneracy set $S$ is the size of the stabilizer $\mathcal{H}_0 \leq \mathcal{H}$ of the degeneracy set $S$. Therefore the number of distinct specializations with degeneracy set $S$ is the total number of specializations with degeneracy set $S$ divided by the order of $\mathcal{H}_0$.

Let $p$ be variable and $d$ be the least common multiple of the degrees of the type-parameters. Our focus now is to compute the total number of specializations with degeneracy set $S$ mapping into $\mathbb{F}_{p^d}$ for a given type $\tau$. One of the major results in Higman [21] is that this number is given by a PORC function in $p$. This function is denoted $d(S, \tau; p)$. Throughout this chapter we assume $\tau$ is fixed and assume the function $d(S, \tau; p)$ is always in terms of a variable prime $p$, so for brevity we denote it as $d(S)$. This matches Higman's notation.

Recall that a degeneracy set $S$ and the type $\tau$ specify a list of equations and inequal-

ities that the image of all specializations counted by $d(S)$ must obey. In a situation where we want to count objects specified by a list of properties that must hold and a list of properties that must not hold, a common method is the Principle of Inclusion-Exclusion. This is a well-known sieve method. A thorough treatment can be found in Volume I, Chapter 2 of [49]. For our situation the Principle of Inclusion-Exclusion states that

$$d(S) = d(S, \tau; p) = \frac{1}{|\mathcal{H}_0|} \sum_{T \supseteq S} (-1)^{|T-S|} c(T), \tag{5.1}$$

where the sum is over all degeneracy sets $T$ containing $S$, and $c(T)$ is the number of specializations satisfying just the equations specified by the degeneracy set $T$. Higman [21] proved that $c(T)$ is a PORC function in $p$, and thus so is $d(S)$. The rest of this chapter will focus on computing the function $c(T)$ for all degeneracy sets $T$. Note that the factor of $|\mathcal{H}_0|^{-1}$ is present to deal with equivalent specializations producing the same conjugacy class so that $d(S)$ is correctly interpreted to be the number of conjugacy classes in $\mathrm{GL}(r, s; p)$ with degeneracy set $S$. This factor is not a component of the Principle of Inclusion-Exclusion.

Before we show how we compute $c(S)$, we need to take note of some issues regarding PORC functions and the Principle of Inclusion-Exclusion. Higman's proof states that both $c(S)$ and $d(S)$ are PORC functions in the sense that Higman defines it: there is an integer $N$ such that for each residue class of $p$ modulo $N$, there is a polynomial in $p$ associated to this residue class, and the collection of these polynomials is called a PORC function. As we will see, the explicit calculations indicate that we have a stronger form of PORC functions. The function $c(S)$ is given as a product of terms $a(p) \cdot b(p)$ where $a(p)$ is a product of terms of the form $\gcd(k, f(p))$ where $k$ is a positive integer, and $f(p)$ and $b(p)$ are polynomials in $p$ with integer coefficients. That is, the integers $k$ inside the gcds specify $N$ in Higman's form of PORC functions, and the product of the gcd functions themselves are constant for all primes in the same residue class. Thus for each residue class of $p$ modulo $N$ there is an associated polynomial, and so we can see our form is captured by Higman's definition of a PORC function. In

this chapter, "PORC function" will always refer to the stronger form. The term $a(p)$ will be sometimes called the *periodic* or gcd part of $c(S)$, and $b(p)$ will be called the *polynomial part*.

Also note that if we have a degeneracy set $S$, then using Equation (5.1) to calculate $d(S)$ requires us to consider $2^N$ sets, where $N = |E_\tau| - |S|$. If $N$ is large, then the inclusion-exclusion calculation will be computationally expensive. We will provide a more efficient manner of calculating inclusion-exclusion in Chapter 6, but for now we assume we calculate $d(S)$ in the form dictated by Equation (5.1).

## 5.1 Calculating $c(S)$

Higman proved in his second paper [21] that $c(S)$ is a PORC function. His approach is to restate the problem of counting specializations satisfying certain equalities as a problem in homological algebra. The methods described in this chapter are original, but are strongly influenced by Higman's proof. That is, we will use the same approach using homological algebra but will provide explicit calculations for each component of Higman's proof.

Our goal is to count the number $c(S)$ of specializations into $\mathbb{F}_{p^d}$ restricted by equations implied by the degeneracy set $S$. Recall that a specialization $\phi$ is a homomorphism taking elements $x_1, \ldots, x_w$ to $\mathbb{F}_{p^d}$, where

$$\phi(\sigma_\tau \cdot x_i) = \sigma \cdot \phi(x_i). \tag{5.2}$$

That is, it respects the Galois action on both $\mathbb{C}$ and $\mathbb{F}_{p^d}$.

To effectively calculate with the two components $x_1, \ldots, x_w$ and $\mathbb{F}_{p^d}$, we need a single context for them. The context proposed by Higman is that of a $\Lambda$-module, where $\Lambda$ is the ring

$$\Lambda = \frac{\mathbb{Z}[\sigma]}{(\sigma^d - 1)},$$

and $d$ is the least common multiple of the degrees of the type-parameters. Although it

is an abuse of notation, we use the symbol $\sigma$ here as it will play the role of $\sigma$ over $\mathbb{F}_{p^d}$ *and* $\sigma_\tau$ over $x_1, \ldots, x_w$ when we bring both into the context of $\Lambda$-modules. For the rest of this chapter, $\sigma$ refers to the indeterminate in $\Lambda$.

It is easy to bring $\mathbb{F}_{p^d}$ into this context. The equations restricting the specializations into $\mathbb{F}_{p^d}$ are all of the form

$$\beta_1^{u_1} \cdots \beta_w^{u_w} = 1.$$

Here $\beta_1, \ldots, \beta_w$ are elements in the multiplicative group of $\mathbb{F}_{p^d}$ and we only require them in this multiplicative role. The multiplicative group of $\mathbb{F}_{p^d}$ is cyclic of order $p^d - 1$, so $\beta_1, \ldots, \beta_w$ are elements of this cyclic group. The Galois action $\sigma$ acts as an automorphism of this group, and this automorphism has order $d$. Using this information, we represent the multiplicative group of $\mathbb{F}_{p^d}$ as the *additive* group of the parametrised ring $M_p$

$$M_p := \frac{\mathbb{Z}[\sigma]}{(\sigma^d - 1, \sigma - p)}.$$

The additive group of $M_p$ is a $\Lambda$-module where the action of an element from $\Lambda$ is just multiplication within $\mathbb{Z}[\sigma]$ with the appropriate reduction. The additive group of $M_p$ is cyclic of order $p^d - 1$, which is isomorphic to the multiplicative group of $\mathbb{F}_{p^d}$. The action of $\sigma$ from $\Lambda$ is the same as the Galois action $\sigma$ of $\mathbb{F}_{p^d}$ over $\mathbb{F}_p$.

We turn now to the elements $x_1, \ldots, x_w$. These elements have "conjugates" given by the automorphism $\sigma_\tau$. For a type $\tau$ given by type-parameters $(d_i, \lambda_i, l_i)$, we have a set $x_i^{\sigma^j}$ for $i = 1, \ldots, w$ and $j = 0, \ldots, d_i - 1$. Recall that an element $(u_1, \ldots, u_w) \in S$ represents an element

$$x_1^{u_1} \cdots x_w^{u_w} \in \mathbb{C}.$$

We only need the elements $x_1, \ldots, x_w$ and their $\sigma_\tau$-conjugates in their multiplicative role in $\mathbb{C}$. Recall that there are no restrictions on these elements. Then we can think of the elements $x_i^{\sigma^j}$ as generators of a finitely-generated free abelian group $F$. To emphasise the abelian group context, we write additively, so

$$x_1^{u_1} \cdots x_w^{u_w} \qquad \text{is written as} \qquad u_1 x_1 + \cdots + u_w x_w.$$

The group $F$ is a $\Lambda$-module where the action of $\mathbb{Z}$ is just the standard action on abelian groups, and $\sigma \in \Lambda$ acts as $\sigma_\tau$, taking $(\sigma^j x_i)$ to $(\sigma^{j+1} x_i)$ (where we reduce using $\sigma^{d_i} x_i = x_i$ if need be). We impose the restrictions in the degeneracy set $S$ by imposing the relations

$$u_1 x_1 + \cdots + u_w x_w = 0 \qquad \text{for } (u_1, \ldots, u_w) \in S.$$

That is, we form the group $X$ which is $F$ modulo the group generated by the words $u_1 x_1 + \cdots + u_w x_w$ for $(u_1, \ldots, u_w) \in S$. The group $X$ is a finitely-generated $\Lambda$-module, where $\sigma \in \Lambda$ acts as $\sigma_\tau$ on generators $(\sigma^j x_i)$ for $i = 1, \ldots, w$ and $j = 0, \ldots, d_i - 1$.

Now we can complete the last step in our restatement in terms of homological algebra: a specialization with degeneracy set $S$ is a $\Lambda$-module homomorphism $\phi : X \to M_p$. The restrictions on the choices of $\beta_1, \ldots, \beta_w \in \mathbb{F}_{p^d}$ are imposed in $X$ instead of $\mathbb{F}_{p^d}$ and are brought across by the homomorphism. The number of specializations with degeneracy set $S$ is

$$c(S) = |\operatorname{Hom}_\Lambda(X, M_p)|,$$

and we wish to find $|\operatorname{Hom}_\Lambda(X, M_p)|$ as a function of $p$. An important component of Higman's proof [21] is that $|\operatorname{Hom}_\Lambda(X, M_p)|$ factorises in a useful way:

$$|\operatorname{Hom}_\Lambda(X, M_p)| = |\operatorname{Hom}_\Lambda(\operatorname{Tor}(X), M_p)| \cdot |\operatorname{Hom}_\Lambda(X/\operatorname{Tor}(X), M_p)|, \tag{5.3}$$

where $\operatorname{Tor}(X)$ is the *torsion submodule* of $X$, namely the set of elements $x \in X$ for which there exists a nonzero integer $m$ such that $m \cdot x = 0$. We call the quotient module $X/\operatorname{Tor}(X)$ the *torsionfree quotient module*. The first term in the right-hand side of Equation (5.3) provides the periodic (gcd) part of our PORC function, and the second term provides the polynomial part. We are able to calculate these two parts independently, and indeed, they require different algorithms. The remainder of this chapter is devoted to describing them. The algorithm to compute $|\operatorname{Hom}_\Lambda(\operatorname{Tor}(X), M_p)|$ will be called the *torsion calculation*, and the algorithm to compute $|\operatorname{Hom}_\Lambda(X/\operatorname{Tor}(X), M_p)|$

will be called the *torsionfree calculation*.

Since we primarily deal with $\Lambda$-module homomorphisms we will often just write $\mathrm{Hom}_\Lambda(A, B)$ as $\mathrm{Hom}(A, B)$, and explicitly indicate when we are working with any other kind of module homomorphism.

## 5.2 Torsion calculation

In his paper [21], Higman proves that if $p \equiv q$ modulo $m$, then

$$|\operatorname{Hom}_\Lambda(\mathrm{Tor}(X), M_p)| = |\operatorname{Hom}_\Lambda(\mathrm{Tor}(X), M_q)|.$$

In other words, $|\operatorname{Hom}(\mathrm{Tor}(X), M_p)|$ is periodic with period $m$. This suffices for Higman's purposes but we need to find this function explicitly.

An element of $X$ is in $\mathrm{Tor}(X)$ solely because of the $\mathbb{Z}$-module action on $X$. The structure of finitely-generated $\mathbb{Z}$-modules is well-known, and we assume the reader is familiar with the relevant theory, especially with the theory of normal forms (Hermite and Smith normal forms) over the integers. We will use [24] as our primary reference for this theory; it is contained in Section 9.2 of [24]. Chapters 8 and 10 of Sims [48] also contain details on this theory.

Given a finite presentation of $X$, we can determine the $\mathbb{Z}$-module structure of $X$, and in particular, of $\mathrm{Tor}(X)$. Every torsion $\mathbb{Z}$-module decomposes into a direct sum of finitely many *finite* cyclic $\mathbb{Z}$-modules. As a $\mathbb{Z}$-module,

$$\mathrm{Tor}(X) \cong \bigoplus_i \mathbb{Z}_{s_i}$$

where there are finitely many summands $\mathbb{Z}_{s_i}$, which are the additive groups $\mathbb{Z}$ modulo $s_i$.

We want to know the $\Lambda$-module structure of $\mathrm{Tor}(X)$. This is essentially the $\mathbb{Z}$-module structure except that the $\mathbb{Z}$-module generators may be redundant under the action of $\sigma \in \Lambda$. For example, suppose $X$ is generated by two elements $x_1$ and $(\sigma x_1)$, and has the

single $\Lambda$-module relation $2 \cdot x_1 = 0$ from the degeneracy set $S$. Of course this includes the equivalent $\Lambda$-module relation $2 \cdot (\sigma x_1) = 0$. If we consider $X$ as a $\Lambda$-module, then $\sigma x_1$ is generated by $x_1$ via the action of $\sigma \in \Lambda$. Therefore $X$ is a cyclic $\Lambda$-module, and its generator has torsion 2. As a $\mathbb{Z}$-module, the generators $x_1$ and $\sigma x_1$ are independent, and $X$ is isomorphic to the $\mathbb{Z}$-module $\mathbb{Z}_2 \oplus \mathbb{Z}_2$. The difference is important because any homomorphism from $X$ to $M_p$ depends only on the image of generators of $X$. The number of $\Lambda$-module homomorphisms in this case is $\gcd(2, p^2 - 1)$ but the number of $\mathbb{Z}$-module homomorphisms is $\gcd(2, p^2 - 1)^2$. Nevertheless, the structure of $\mathrm{Tor}(X)$ as a $\mathbb{Z}$-module can tell us the structure of $\mathrm{Tor}(X)$ as a $\Lambda$-module if we correctly incorporate the action of $\sigma \in \Lambda$. This is the main idea of the torsion calculation: treat $X$ as a $\mathbb{Z}$-module for calculating the main data relevant to torsion, and modify this information in light of the action of $\sigma \in \Lambda$.

Our first step is to determine an appropriate counting formula for the number of $\Lambda$-module homomorphisms from $\mathrm{Tor}(X)$ to $M_p$, assuming we know generators of $\mathrm{Tor}(X)$ and certain additional information. Recall that as a $\mathbb{Z}$-module,

$$\mathrm{Tor}(X) \cong \bigoplus_{i=1}^{k} \mathbb{Z}_{s_i}$$

where the summands $\mathbb{Z}_{s_i}$ are the additive groups $\mathbb{Z}$ modulo $s_i$. In other words, $\mathrm{Tor}(X)$ is generated by $k$ elements $g_1, \ldots g_k$ with $s_i \cdot g_i = 0$ for $i = 1, \ldots, k$, and $g_i$ are independent under the $\mathbb{Z}$-action.

By a well-known result in homological algebra (see, for example, [22], or [31], page 192)

$$|\operatorname{Hom}_\Lambda(\bigoplus_i \mathbb{Z}_{s_i}, M_p)| = \prod_i |\operatorname{Hom}_\Lambda(\mathbb{Z}_{s_i}, M_p)|.$$

Therefore it suffices to find $|\operatorname{Hom}_\Lambda(\mathbb{Z}_{s_i}, M_p)|$ for $\Lambda$-module $\mathbb{Z}_{s_i}$. Note that $M_p$ is isomorphic (as a $\mathbb{Z}$-module) to $\mathbb{Z}_{p^d-1}$. It is also well-known [16] that

$$|\operatorname{Hom}_\mathbb{Z}(\mathbb{Z}_m, \mathbb{Z}_n)| = \gcd(m, n), \tag{5.4}$$

where $\mathrm{Hom}_{\mathbb{Z}}(\mathbb{Z}_m, \mathbb{Z}_n)$ is the group of $\mathbb{Z}$-module homomorphisms, that is, the group-homomorphisms from $\mathbb{Z}_m$ to $\mathbb{Z}_n$.

We can utilize the proof of this last result to prove a similar one for $\Lambda$-modules.

**Theorem 5.1.** *Let $A$ be a finite, cyclic $\Lambda$-module generated by $g$, and suppose there is a least nonzero $m$ such that $m \cdot g = 0$ and a polynomial $f(\sigma)$ of least degree such that $f(\sigma)$ divides $\sigma^d - 1$, and $f(\sigma) \cdot g = 0$. Then the number of $\Lambda$-module homomorphisms from $A$ to $M_p$ is*

$$|\operatorname{Hom}_\Lambda(A, M_p)| = \gcd(m, f(p)).$$

*Proof.* Let $\psi \in \operatorname{Hom}_\Lambda(A, M_p)$. Since both $A$ and $M_p$ are cyclic $\Lambda$-modules, it is necessary and sufficient to determine $\psi$ by specifying $\psi(g)$. Note that since $m \cdot g = 0$ and $f(\sigma) \cdot g = 0$, then $m \cdot \psi(g) = 0$ and $f(\sigma) \cdot \psi(g) = f(p) \cdot \psi(g) = 0$. That is, $\psi(g)$ has order dividing $\gcd(m, f(p))$. Suppose the order of $\psi(g)$ is $k$ and note that $k$ divides $p^d - 1$ since $f(p)$ does. Then $\psi(g)$ generates a submodule $B$ of order $k$. Since $M_p$ is cyclic, $B$ is unique. The choice of $\psi(g)$ was one of $\varphi(k)$ generators of $B$, where $\varphi(k)$ is the Euler phi function. Therefore the total number of choices for $\psi(g)$ is

$$\sum_{k \mid \gcd(m, f(p))} \varphi(k).$$

It is well-known in number theory that this sum is equal to $\gcd(m, f(p))$ (see [3], page 26). Therefore the number of $\Lambda$-module homomorphisms from $A$ to $M_p$ is $\gcd(m, f(p))$. □

Combining the above results we see that

$$|\operatorname{Hom}_\Lambda(\mathrm{Tor}(X), M_p)| = \prod_i |\operatorname{Hom}_\Lambda(A_i, M_p)| = \prod_i \gcd(m_i, f_i(p)), \tag{5.5}$$

where $A_i$ is a cyclic torsion $\Lambda$-submodule of $\mathrm{Tor}(X)$ generated by $g_i$, and $m_i$ is the least positive integer such that $m_i \cdot g_i = 0$ and $f_i(\sigma)$ is the nonconstant polynomial of least degree such that $f_i(\sigma) \cdot g_i = 0$ and $f(\sigma) \mid \sigma^d - 1$. It follows that once we decompose $\mathrm{Tor}(X)$ into cyclic submodules, and determine properties of generators of

such submodules, then this completes the torsion calculation. The rest of this section is devoted to finding generators $g_i$ and their associated $m_i$ and $f_i(\sigma)$.

There are four main steps to finding this information:

1. Turn the $\Lambda$-module presentation of $X$ into a $\mathbb{Z}$-module presentation;
2. Find generators of the $\mathbb{Z}$-module $\mathrm{Tor}(X)$ and their associated torsion via a normal form algorithm;
3. Find an irredundant list of $\Lambda$-module generators $g_i$ from the $\mathbb{Z}$-module generators;
4. Determine the polynomials $f_i(\sigma)$ for each generator $g_i$.

As mentioned before, the general approach is to work in the context of finite abelian groups (that is, $\mathbb{Z}$-modules) and modify the results to accommodate the $\Lambda$-module structure.

### 5.2.1 Changing presentations

Treating $X$ as a $\mathbb{Z}$-module for the purpose of calculating torsion requires turning the $\Lambda$-module presentation for $X$ into a $\mathbb{Z}$-module presentation. We do this by closing the $\Lambda$-module relations under the action of $\sigma$.

Take the example we used before: $X$ is generated by $x_1$, where $\sigma \cdot x_1 = (\sigma x_1)$, with the single $\Lambda$-module relation $2x_1 = 0$. There is an equivalent $\Lambda$-module relation $2(\sigma x_1) = 0$. The second relation is in the closure of the first under $\sigma \in \Lambda$. To interpret $X$ as a $\mathbb{Z}$-module we "forget" that the generators and the relations are related by the action of $\sigma \in \Lambda$.

To translate the $\Lambda$-module relations into $\mathbb{Z}$-module relations, we follow the general idea of acting on the relations by $\sigma \in \Lambda$ and reducing the relations under the action of $\mathbb{Z}$. We keep repeating this process until the action of $\sigma \in \Lambda$ on every relation is contained in the $\mathbb{Z}$-span of the current set of relations. This follows a similar procedure to ZXMODULE on page 459 of Sims' book [48].

In our case, we have a set of free generators $(\sigma^j x_i)$ generating $F$, and a set of relations on $F$ that define the group $X$. Our algorithm reduces this presentation to

another set of generators and relations on $F$, and these generators form a $\mathbb{Z}$-basis for the same module $X$. Our procedure is similar to Sims' ZXMODULE, the difference being that $\sigma \in \Lambda$ has a prescribed relation $\sigma^d - 1 = 0$ which we use.

A relation from $S$ is of the form

$$\sum_{i=1}^{w} \sum_{j=0}^{d_i} u_{ij}(\sigma^j x_i) = 0.$$

This can be represented by a row-vector of length equal to the sum of the degrees $d_i$ for type-parameters $(d_i, \lambda_i, l_i)$. The relations in $S$ therefore can be represented by a matrix $U$ where the rows are the vectors representing each relation from $S$.

If we write the relations from $S$ in matrix form, then the action of $\sigma \in \Lambda$ can be represented by a matrix $\mathcal{M}_\tau$ acting on the right. The matrix $\mathcal{M}_\tau$ just permutes the entries of a row-vector in the same way $\sigma \in \Lambda$ permutes the generators $\sigma^j x_i$. This matrix depends only on the type, which we emphasise by using the index $\tau$.

The procedure to write the $\Lambda$-module presentation of $X$ as a $\mathbb{Z}$-module presentation is as follows:

1. Write the equations from the degeneracy set $S$ in terms of the generators $(\sigma^j x_i)$ store them in a matrix $U$;

2. Act on this set by the matrix $\mathcal{M}_\tau$, and append this set of relations to our matrix $U$ as additional rows;

3. Find a minimal $\mathbb{Z}$-basis of the row-space of $U$ via the Hermite normal form of $U$;

4. Repeat until $U$ is invariant under the steps 1-3.

For algorithmic purposes we consider the Hermite normal form of a matrix to be equivalent to the same form with any number of rows of zeroes. That is, we remove rows of zeroes throughout this procedure.

We will call the resulting matrix the $\sigma$-Hermite normal form of $U$ (or $\sigma\mathrm{HNF}(U)$) as it $\sigma$-closes the Hermite normal form.

**Algorithm 5.1** $\sigma$-close relations on $\Lambda$-module $X$

**Require:** Relations of $X$ in a matrix $U$, and a matrix $\mathcal{M}_\tau$ representing the $\sigma$-action on the relation vectors.

1: **repeat**
2: Set $A := U$
3: Set $d > 0$ such that $\mathcal{M}_\tau^d = 1$.
4: **for** $1 \leq i \leq d-1$ **do**
5: Append to $U$ the rows of the matrix $U \cdot M^i$.
6: **od**
7: Set $U :=$ `HermiteNormalForm`$(A)$.
8: Remove all rows of zeroes from $U$.
9: **until** $U = A$.
10: **return** $\sigma$-closed $\mathbb{Z}$-module relations $U$.

### 5.2.2 Determining $\mathbb{Z}$-module generators of $\mathrm{Tor}(X)$

Having expressed $\mathrm{Tor}(X)$ as a $\mathbb{Z}$-module, we now use established results to find the decomposition of $\mathrm{Tor}(X)$ into finite cyclic $\mathbb{Z}$-submodules. For this we use the Smith normal form (see Section 9.2.4 of [24], or Section 8.3 of [48]). A $m \times n$ matrix $A$ defined over $\mathbb{Z}$ is in *Smith normal form* if the following conditions hold:

1. $A_{ij} = 0$ if $i \neq j$;
2. $A_{ii} = d_i \geq 0$ for $i = 1, \ldots, \min(m, n)$;
3. $d_i \mid d_{i+1}$ for $i = 1, \ldots, \min(m, n) - 1$.

A Smith normal form algorithm is an algorithm that takes an arbitrary matrix $A$ defined over the integers, performs integer row and column operations on $A$ and returns a matrix in Smith normal form that is equivalent to $A$. That is, it finds invertible matrices $R$ and $C$ such that $RAC$ is a matrix in Smith normal form. There are many different implementations for this procedure each designed to handle certain performance problems. We do not require any particular implementation; the one provided in GAP is sufficient.

An $m \times n$ matrix in Smith normal form, where rows of zeroes are removed, can be given by the list of nonzero diagonal entries $\{d_1, \ldots, d_r\}$. The $\mathbb{Z}$-module associated to this Smith normal form is

$$\mathbb{Z}_{d_1} \oplus \cdots \oplus \mathbb{Z}_{d_r} \oplus \mathbb{Z}^{n-r}.$$

Therefore the torsion submodule of this matrix is just

$$\mathbb{Z}_{d_1} \oplus \cdots \oplus \mathbb{Z}_{d_r}.$$

The standard treatment of Smith normal forms requires that $d_i \mid d_{i+1}$ so that this decomposition is unique. We won't insist on this requirement as we will be undoing this work anyway. As an example, suppose we reduce our $\mathbb{Z}$-module relations for $\mathrm{Tor}(X)$ via the Smith normal form, obtaining a list of diagonal entries $\{1, 2, 6\}$. As a $\Lambda$-module, this could correspond to a situation where we have three $\Lambda$-module generators $g_1, g_2, g_3$ where $1 \cdot g_1 = 0$, $2 \cdot g_2 = 0$ and $6 \cdot g_3 = 0$. Alternatively, the relations could be $2 \cdot g_1 = 0$, $2 \cdot g_2 = 0$ and $3 \cdot g_3 = 0$. Moreover, the $\Lambda$-module could be generated by $g_1, \sigma g_1$ and $g_2$ where the $\Lambda$-module relations are $2 \cdot g_1 = 0$ and $3 \cdot g_2 = 0$. These three situations are quite different in light of our counting theorem given by Equation (5.5). In general, a single list $\{d_1, \ldots, d_r\}$ could correspond to several different $\Lambda$-modules. The next section shows how to differentiate them.

In preparation for this, we need to ensure that the algorithm that computes the Smith normal form of our matrix also provides us with the matrices $R$ and $C$ (such that $RAC$ is in Smith normal form). We actually only require their inverses for reasons explained in the next section. The matrix $R^{-1}$ will be called the *row transformation matrix* and $C^{-1}$ will be called the *column transformation matrix*. These matrices reverse the integer row- and column-transformations respectively, taking the matrix back to its Smith normal form.

To complete this stage of the algorithm we need to find the Smith normal form of the $\sigma$-Hermite normal form of the matrix $U$ created from the degeneracy set $S$. From this we obtain a matrix $U'$, as well as the row and column transformation matrices $R$ and $C$. This describes the decomposition of $\mathrm{Tor}(X)$ as a $\mathbb{Z}$-module and how we obtained it.

### 5.2.3 Determining $\Lambda$-module generators of $\mathrm{Tor}(X)$

At this point we have a matrix $U'$ which is the Smith normal form of the $\sigma$HNF of the matrix $U$, and the matrices $R$ and $C$. These matrices satisfy the equation $RUC = U'$, or equivalently $U = R^{-1}U'C^{-1}$. The matrix $U'$ is in Smith normal form, so it looks like

$$\begin{pmatrix} d_1 & 0 & \dots & \\ 0 & d_2 & 0 & \\ & & \ddots & 0 \\ & & 0 & d_r \end{pmatrix}$$

The matrix $U'$ gives a presentation of $\mathrm{Tor}(X)$ in terms of $\mathbb{Z}$-module generators $g_i'$ and relations given by the matrix $U'$. From the rows of $U'$ we know that $g_i'$ has torsion $d_i$. The generators $g_i'$ form a $\mathbb{Z}$-basis for $\mathrm{Tor}(X)$ in the presentation given by $U'$. Suppose that the presentation given by $\sigma\mathrm{HNF}(U)$ has generators $g_i$. If we determine what elements in the original basis $g_i$ correspond to $g_i'$ (by using the matrices $R^{-1}$ and $C^{-1}$), then we have a $\mathbb{Z}$-basis for $\mathrm{Tor}(X)$ in terms of these original generators. We know how to act on generators $g_i$ by $\sigma \in \Lambda$ because they are in terms of the original basis. That is, $\sigma$ acts on the elements $g_i$, expressed as vectors, by the matrix $M$. We can then determine which $g_i'$ are equivalent under the action of $\Lambda$ via the $\sigma$-Hermite normal form. The set of nonequivalent $g_i'$ are a minimal $\Lambda$-basis for $\mathrm{Tor}(X)$. Since we know their torsion from the presentation $U'$ and can determine the polynomials $f_i(\sigma)$ such that $f_i(\sigma) \cdot g_i = 0$, we have all we need to determine the number of $\Lambda$-module homomorphisms from $\mathrm{Tor}(X)$ to $M_p$. We take this general approach.

This method hinges on the fact that a row-vector can represent a relation

$$\sum_{i=1}^{w} \sum_{j=0}^{d_i} u_{ij}(\sigma^j x_i) = 0,$$

in a certain presentation but also represent an element in terms of the same presentation (in this case, this element is just the left-hand side of the above equation). Thus the

row-vector

$$(0, \ldots, 0, 1, 0, \ldots, 0),$$

where 1 is in the $i$th position and there are zeroes elsewhere, corresponds to the $i$th generator in the particular presentation we are using. Invertible matrix transformations acting on row-vectors restate an element given in terms of one presentation in terms of another presentation.

In particular, we can represent the generator $g'_i$ in the presentation given by $U'$ by a matrix $Q_i$ where the $i$th element on the diagonal is 1 and zeroes elsewhere. The matrix $P_i := R^{-1}G_iC^{-1}$ specifies $g'_i$ in terms of the original generators $g_i$. We can now determine whether the generator $g'_j$ is in the span of another generator $g'_i$ by determining if the relations of $g'_j$ (that is, the matrix $P_j$) is in the $\Lambda$-module span of those of $g'_i$ (that is, the matrix $P_i$). See page 74 of [9] for a proof of this. To compute this, we append the rows of $P_j$ to $P_i$ and compute the $\sigma$HNF of the resulting matrix. If this matrix is equal to the $\sigma$HNF of $P_i$, then $g'_j$ is in the $\Lambda$-module span of $g'_i$.

The next question is which sets of generators do we choose? Suppose generator $g'_i$ has torsion $d_i$, and $g'_j$ has torsion $d_j$, where $i < j$. Then by the Smith normal form, $d_i \mid d_j$. Therefore it is possible that $g'_i$ is generated $g'_j$ since its torsion divides the torsion of $g'_j$. If we used a normal form algorithm that didn't insist on the uniqueness of the decomposition of $\mathbb{Z}$-modules, then this would reduce the search. We can use a similar reduction, using the polynomials $f_i(\sigma)$ which have yet to be computed. Once we do, we only need to check pairs $(i, j)$ such that $f_i(\sigma) \mid f_j(\sigma)$. For $i < j$, if $g'_i$ and $g'_j$ are such that $f_i(\sigma) \nmid f_j(\sigma)$ then this would imply that the gcd of $f_i(\sigma)$ and $f_j(\sigma)$ was the real minimal polynomial for $g'_i$. But then $1 \cdot g'_i = 0$, which is absurd. Therefore this is a necessary condition for two pairs to be redundant. The actual test is done by the $\sigma$-Hermite normal form calculation.

The process to find $f_i(\sigma)$ is straightforward. If we had the polynomial $f_i(\sigma)$ then we would show that it is the minimal polynomial for $g'_i$ via matrices by substituting the matrix $\mathcal{M}_\tau$ (signifying the action of $\sigma \in \Lambda$) into the polynomial $f_i$, and then act $f_i(\mathcal{M}_\tau)$ on $P_i$, the matrix representing $g'_i$ in terms of the original generators of $X$. If

the resulting matrix is zero, then $f_i(\sigma) \cdot g_i' = 0$

Since $(\sigma^d - 1) \cdot u = 0$ for all $u$ in a $\Lambda$-module, the minimal polynomial $f_i(\sigma)$ must divide $\sigma^d - 1$. Therefore we can find $f_i(\sigma)$ by creating the list of divisors $\delta(\sigma)$ of $\sigma^d - 1$, sorting them in order of increasing degree, and in order, test each divisor if it satisfies $\delta(\mathcal{M}_\tau) \cdot P_i = 0$. The first $\delta(\sigma)$ to satisfy this equation is the minimal polynomial $f_i(\sigma)$.

We can speed this process up a little. The irreducible polynomials dividing $\sigma^d - 1$ are called the *cyclotomic polynomials* (over $\mathbb{Q}$). The cyclotomic polynomials are ordered. The $i$th cyclotomic polynomial is denoted $\Phi_i(\sigma)$. It is well-known (see Theorem 2.45 on page 64 of [29]) that

$$\sigma^d - 1 = \prod_{i|d} \Phi_i(\sigma). \tag{5.6}$$

Since the cyclotomic polynomials are irreducible, it follows that any divisor of $\sigma^d - 1$ is a product of cyclotomic polynomials $\Phi_i(\sigma)$ where $i$ divides $d$, and any such cyclotomic polynomial $\Phi_i(\sigma)$ occurs at most once. There are algorithms for computing $\Phi_i(\sigma)$ (in GAP this is done using `CyclotomicPolynomial(Rationals, i)`). We can precompute these polynomials, and since $d$ is fixed for a type, we can precompute all possible $f_i(\sigma)$.

While this does not lead to a massive speedup, it minimises the time that the program needs to compute this data (which is often dominated by the computer determining which indeterminate is involved in a given polynomial). This optimization is included in the implementation. One of the main reasons for this detour is that we will require cyclotomic polynomials in the next section, and it is convenient to define them now.

Combining all the above procedures, given the presentation of $X$ via the degeneracy set we can determine a list of generators $g_i'$ that are independent over $\Lambda$ and we know $d_i$ and $f_i(\sigma)$ for each generator so that

$$|\operatorname{Hom}(\operatorname{Tor}(X), M_p)| = \prod_i \gcd(d_i, f_i(p)).$$

This completes the torsion calculation.

## 5.3 Torsionfree calculation

The torsionfree calculation involves finding

$$|\operatorname{Hom}_\Lambda(X/\operatorname{Tor}(X), M_p)|$$

as a polynomial in $p$. The $\Lambda$-module $X/\operatorname{Tor}(X)$ is torsionfree, that is, for all nonzero elements $u$ in $X/\operatorname{Tor}(X)$, the only integer $m$ such that $mu = 0$ is $m = 0$.

For convenience, we denote $X/\operatorname{Tor}(X)$ as $X^*$. Since $X^*$ is torsionfree, it is naturally embedded in the tensor product $X^* \otimes \mathbb{Q}$. Similarly, $\Lambda$ is naturally embedded in $\Lambda \otimes \mathbb{Q}$. Thus it does no harm to reformulate our problem in terms of these two tensor products. In practice, we redefine $\Lambda$ as

$$\frac{\mathbb{Q}[\sigma]}{\langle \sigma^d - 1 \rangle}$$

and consider the relations of $X$ to have coefficients in $\mathbb{Q}$ rather than $\mathbb{Z}$. Doing this simplifies our approach.

We will use the two following results from Higman, which correspond to Lemma 2.1.7 and 2.1.8 in [21].

**Theorem 5.2.** *If $X^*$ is a finitely-generated torsionfree $\Lambda$-module then*

$$|\operatorname{Hom}_\Lambda(X^*, M_p)| = |b(p)|,$$

*where $b(x)$ is a polynomial such that every irreducible factor of $b(x)$ divides $x^d - 1$.*

That is, $b(x)$ is a product of cyclotomic polynomials $\Phi_i(x)$ where $i$ divides $d$.

**Theorem 5.3.** *Suppose we have a $\Lambda$-module $A = \mathbb{Q}[\sigma]/\langle g(\sigma) \rangle$, where the polynomial $g(x)$ divides $x^d - 1$. Then*

$$|\operatorname{Hom}_\Lambda(A, M_p)| = |g(p)|.$$

These two results suggest that if we can decompose $X^*$ into modules of the form $A = \mathbb{Q}[\sigma]/\langle g(\sigma) \rangle$ where the polynomials $g(\sigma)$ are cyclotomic polynomials, then we can easily compute $|\operatorname{Hom}(X^*, M_p)|$. Our approach does just that.

Consider the ring $\Lambda = \mathbb{Q}[\sigma]/\langle\sigma^d - 1\rangle$. Every $\Lambda$-module is a quotient of $\Lambda^k$ for some $k$. Now consider the principal ideals generated by the cyclotomic polynomials $\Phi_i(\sigma)$ for $i \mid d$. Since the cyclotomic polynomials are irreducible, the principal ideals $(\Phi_i(\sigma))$ and $(\Phi_j(\sigma))$ are coprime for $i \neq j$. By the Chinese remainder theorem [6] and Equation (5.6) we know that

$$\Lambda = \frac{\mathbb{Q}[\sigma]}{\langle\sigma^d - 1\rangle} \cong \bigoplus_{i|d} \frac{\mathbb{Q}[\sigma]}{\langle\Phi_i(\sigma)\rangle}. \tag{5.7}$$

So $X^* = \Lambda^w/\langle S\rangle$ where $\langle S\rangle$ is the ideal of elements given by the degeneracy set $S$. The above decomposition says that we can decompose the ideal $\langle S\rangle$ into a direct product of ideals over $\mathbb{Q}[\sigma]/\langle\Phi_i(\sigma)\rangle$ for $i \mid d$.

Take for example $d = 2$ so that $\Lambda = \mathbb{Q}[\sigma]/\langle\sigma^2 - 1\rangle$. The polynomial $\sigma^2 - 1$ factorises into cyclotomic polynomials $\Phi_1(\sigma) = (\sigma - 1)$ and $\Phi_2(\sigma) = (\sigma + 1)$. Thus

$$\frac{\mathbb{Q}[\sigma]}{\langle\sigma^2 - 1\rangle} \cong \frac{\mathbb{Q}[\sigma]}{\langle\sigma - 1\rangle} \oplus \frac{\mathbb{Q}[\sigma]}{\langle\sigma + 1\rangle}$$

We can frame this for single elements in terms of $\Lambda$:

$$a + b\sigma + \langle\sigma^2 - 1\rangle \equiv (A + \langle\sigma - 1\rangle) + (B + \langle\sigma + 1\rangle)$$

for $a, b, A, B \in \mathbb{Q}$. Note that the element on the left-hand side is one annihilated by $\sigma^2 - 1$, the element $A + \langle\sigma - 1\rangle$ is an element annihilated by $\sigma - 1$ and therefore is equivalent to $A(\sigma + 1)$. Similarly, $B + \langle\sigma + 1\rangle$ is annihilated by $\sigma + 1$ and therefore is equivalent to $B(\sigma - 1)$. We can therefore restate the equivalence as

$$a + b\sigma \equiv A(\sigma + 1) + B(\sigma - 1).$$

This gives us a relation between $a$ and $b$, and $A$ and $B$.

Now assume that $X^*$ is generated (in terms of $\Lambda$) by a single element $x_1$ and that it has relation $x_1 + \sigma x_1 = (1 + \sigma)x_1 = 0$. We want to decompose the ideal $\langle 1 + \sigma\rangle \in \Lambda$ into

ideals related to the cyclotomic polynomials. That is, we want to determine rationals $A$ and $B$ such that

$$1 + \sigma \equiv A(\sigma + 1) + B(\sigma - 1).$$

This is trivially solved by $A = 1$ and $B = 0$. Therefore

$$\begin{aligned} X^* = \frac{\Lambda}{\langle 1 + \sigma \rangle} &\cong \frac{\mathbb{Q}[\sigma]}{\langle 1, \sigma - 1 \rangle} \oplus \frac{\mathbb{Q}[\sigma]}{\langle 0, \sigma + 1 \rangle} \\ &\cong \frac{\mathbb{Q}[\sigma]}{\langle \sigma + 1 \rangle}. \end{aligned}$$

Then by Theorem 5.3 the number of homomorphisms from $X^*$ to $M_p$ is $p + 1$. This approach is how our algorithm for the torsionfree part works for a single generator, single relation case. We will build up to the most general case (and the general algorithm) in steps.

Note that in the previous example, the two-dimensional module $\Lambda$ decomposed into two one-dimensional submodules. In general, the module $\mathbb{Q}[\sigma]/\langle \sigma^d - 1 \rangle$ does not always decompose into one-dimensional submodules, but into modules of dimension $\phi(i)$ for $i \mid d$ and where $\phi(i)$ is the Euler phi function. This is because $\Phi_i(\sigma)$ has degree $\phi(i)$ (see [29], page 64), and therefore $\mathbb{Q}[\sigma]/\langle \Phi_i(\sigma) \rangle$ has dimension $\phi(i)$. For example, suppose $\Lambda = \mathbb{Q}[\sigma]/\langle \sigma^3 - 1 \rangle$. An element in $\Lambda$ has decomposition

$$a + b\sigma + c\sigma^2 + \langle \sigma^3 - 1 \rangle \equiv (A + \langle \sigma - 1 \rangle) + (B + C\sigma + \langle \sigma^2 + \sigma + 1 \rangle).$$

Thus the three-dimensional space $\Lambda$ decomposes into a one-dimensional space and a two-dimensional space. In the two-dimensional space, an element $B + C\sigma + \langle \sigma^2 + \sigma + 1 \rangle$ produces another element under the action of $\sigma$. We must be careful though — in this space, $\sigma$ has the relation $\sigma^2 + \sigma + 1 = 0$, not just $\sigma^3 - 1 = 0$. Therefore

$$\begin{aligned} \sigma \cdot (B + C\sigma + \langle \sigma^2 + \sigma + 1 \rangle) &= B\sigma + C\sigma^2 + \langle \sigma^2 + \sigma + 1 \rangle \\ &= B\sigma + C(-\sigma - 1) + \langle \sigma^2 + \sigma + 1 \rangle \\ &= -C + (B - C)\sigma + \langle \sigma^2 + \sigma + 1 \rangle. \end{aligned}$$

The number of homomorphisms from the two-dimensional subspace to $M_p$ depends on whether $B + C\sigma$ and $\sigma \cdot (B + C\sigma)$ generate this two-dimensional subspace. If they do, then $\langle B + C\sigma, \sigma \cdot (B + C\sigma), \sigma^2 + \sigma + 1\rangle = \langle 1\rangle$ as an ideal in $\Lambda$ and there will be only one homomorphism corresponding to this submodule. In general, suppose we have a component corresponding to $\Phi_i(\sigma)$. An element in the corresponding submodule defines a vector $V$ of length $\phi(i)$. The action of $\sigma$ particular to this submodule defines $\phi(i) - 1$ other vectors $\sigma \cdot V$, $\sigma^2 \cdot V$, .... Let $M$ be the matrix whose rows are given by the vectors $V$, $\sigma \cdot V$, .... The *nullity* of $M$ is the number of columns of $M$ minus the rank of $M$. The relation given by $V$ contributes $\Phi_i(p)^k$ to $|\operatorname{Hom}(X^*, M_p)|$ if the nullity divided by $\phi(i)$ is $k$. We divide by $\phi(i)$ because computing the rank is done over $\mathbb{Q}$, not $\mathbb{Q}[\sigma]/\langle\Phi_i(\sigma)\rangle$ (a $\phi(i)$-dimensional vector space over $\mathbb{Q}$ corresponds to a 1-dimensional vector space over $\mathbb{Q}[\sigma]/\langle\Phi_i(\sigma)\rangle$).

Now let $X^*$ be generated by two elements $x_1$ and $x_2$, where $\sigma$ acts trivially (that is, $x_1$ and $x_2$ correspond to type-parameters of degree 1). Therefore $X^*$ is a quotient $\Lambda$-module of $\Lambda^2$. The relations on $x_1$ and $x_2$ are of the form $ax_1 + bx_2 = 0$ for some $a, b \in \mathbb{Z}$. Suppose $X^*$ has a single relation. It can be one of four kinds:

- $a = b = 0$. Then $X^* \cong \mathbb{Q}[\sigma]/\langle\sigma - 1\rangle \oplus \mathbb{Q}[\sigma]/\langle\sigma - 1\rangle$;
- $a = 0$ and $b \neq 0$. Then $X^* \cong \mathbb{Q}[\sigma]/\langle\sigma - 1\rangle$;
- $a \neq 0$ and $b = 0$. Then $X^* \cong \mathbb{Q}[\sigma]/\langle\sigma - 1\rangle$;
- $a \neq 0$, and $b \neq 0$. Then $X^* \cong \mathbb{Q}[\sigma]/\langle\sigma - 1\rangle$ which is some diagonal product of the two submodules isomorphic to $\mathbb{Q}[\sigma]/\langle\sigma - 1\rangle$.

These are all variants on the nullity approach: for each relation $ax_1 + bx_2 = 0$ form the vector $(a, b)$, and compute the nullity of this as a matrix. The nullity, denoted nullity$(A)$, of this matrix indicates that $|\operatorname{Hom}(X^*, M_p)| = (p-1)^{\text{nullity}(A)}$. If $X^*$ has more than one relation, then the matrix we calculate nullity for has as many rows as there are relations.

We now consider the most general case, where $X^*$ has $w$ generators of possibly different degrees. Then $X^*$ is a quotient $\Lambda$-module of $\Lambda^w$. The module $\Lambda$ decomposes

into a direct sum

$$\Lambda \cong \bigoplus_{i|d} \frac{\mathbb{Q}[\sigma]}{\langle \Phi_i(\sigma) \rangle}.$$

Note that submodules $\mathbb{Q}[\sigma]/\langle \Phi_i(\sigma) \rangle$ and $\mathbb{Q}[\sigma]/\langle \Phi_j(\sigma) \rangle$ intersect trivially for $i \neq j$ as $\Phi_i(\sigma)$ and $\Phi_j(\sigma)$ are irreducible and coprime. This applies for the same modules but across different copies of $\Lambda$. Therefore the parts of relations corresponding to the module $\mathbb{Q}[\sigma]/\langle \Phi_i(\sigma) \rangle$ have nothing to do with the parts of relations corresponding to the module $\mathbb{Q}[\sigma]/\langle \Phi_j(\sigma) \rangle$ as far as the torsionfree calculation is concerned. Therefore we consider each cyclotomic polynomial separately.

The procedure for our torsionfree calculation is as follows:

- For each cyclotomic $\Phi_i(\sigma)$ dividing $\sigma^d - 1$:
    1. Take each generator $x_j$ corresponding to a type-parameter of degree divisible by $i$, and for each relation on $X^*$, extract the part of the coefficient of $x_j$ corresponding to the submodule $\mathbb{Q}[\sigma]/\langle \Phi_i(\sigma) \rangle$ and store it in a vector;
    2. For each such generator in a single relation, form a row vector $V$ by concatenating all the vectors;
    3. Form the matrix $M$ where the rows are the vectors $V$ from the previous step;
    4. Compute the nullity $N_i$ of $M$;
    5. Then $|\operatorname{Hom}(X^*, M_p)|$ has a factor $\Phi_i(p)^{N_i/\phi(i)}$.
- The product of factors $\Phi_i(p)^{N_i}$ gives $|\operatorname{Hom}(X^*, M_p)|$, thus completing the torsionfree calculation.

The only remaining part of this process is how to algorithmically split a coefficient of $x_j$ in a relation into parts corresponding to the different cyclotomic polynomials. For this we develop the concept of a *splitter matrix*. By Equation (5.7), an element in $\Lambda$ decomposes as

$$\left( \sum_{j=0}^{d-1} a_j \sigma^j \right) + \langle \sigma^d - 1 \rangle \equiv \sum_{i|d} \left( \sum_{j=0}^{\phi(i)-1} A_j \sigma^j \right) + \langle \Phi_i(\sigma) \rangle.$$

Each term in the sum over $i$ in the right-hand side is equivalent to a term annihilated by the appropriate $\Phi_i(\sigma)$, and so this is equivalently stated as:

$$\left(\sum_{j=0}^{d-1} a_j \sigma^j\right) \equiv \sum_{i|d} \left(\sum_{j=0}^{\phi(i)-1} A_{ij}\sigma^j\right) \cdot \frac{\sigma^d - 1}{\Phi_i(\sigma)}. \tag{5.8}$$

For a given $i$, we want to know how the vector $(a_1, \ldots, a_{d-1})$ corresponds to the vector $(A_{i0}, \ldots, A_{i\phi(i)-1})$ given the above equation. By the Chinese remainder theorem this correspondence is unique. We can consider the correspondence over all $i$ (for a single coefficient of a generator in a relation) to take

$$a = (a_1, \ldots, a_{d-1}) \longrightarrow A = (A_{10}, \ldots, A_{ij}, \ldots, A_{d\phi(d)-1}).$$

These are both row-vectors of length $d-1$ (the latter vector because $\sum_{i|d} \phi(i) = d$.) If we consider them as column-vectors instead, then we can represent this transformation by a matrix $Q$:

$$Q \cdot a = A.$$

The matrix $Q$ is called the *splitter matrix* as it "splits" the vector $a$ into the vectors $(A_{i1}, \ldots, A_{i\phi(i)-1})$. To construct $Q$, consider the correspondence from $A$ to $a$ given by expanding out Equation (5.8) and comparing like terms. That is, $a_0$ is the sum of the constant terms in the right-hand side of Equation (5.8), $a_1$ is the sum of the linear terms, and so on. This correspondence is itself a matrix, where the first row is the coefficients of $(\sigma^d - 1)/\Phi_1(\sigma)$ in the standard order, the next row is the coefficients of $(\sigma^d - 1)/\Phi_i(\sigma)$ for $i$ the first nontrivial divisor of $d$. This is followed by $\phi(i) - 1$ many rows corresponding to the coefficients of $\sigma^j \cdot (\sigma^d - 1)/\Phi_i(\sigma)$, and so on. This matrix has an inverse because the Chinese remainder theorem assures the isomorphism in Equation (5.7). The inverse of this matrix is the *splitter matrix*. For a given $d$, the splitter matrix is a $d \times d$ matrix, and depends only on $d$. Therefore for our torsionfree calculation we precompute the splitter matrices of size $d_i$ for each $d_i$ in the type-parameters $(d_i, \lambda_i, l_i)$ of the given type.

The entire torsionfree calculation is as follows:

1. Precompute splitter matrices of size $d_i$ for all type-parameters $(d_i, \lambda_i, l_i)$;

2. For each relation

$$u_1 x_1 + \cdots + u_w x_w = 0,$$

and each polynomial coefficient $u_i$ for generator $x_i$, create the vector of coefficients of $u_i$ and split it by multiplication by the appropriate splitter matrix;

3. Determine $d$, the least common multiple of the $d_i$ in the type;

4. For each cyclotomic polynomial $\Phi_i(\sigma)$ for $i$ dividing $d$:

   (a) For each relation, collect the coefficients corresponding to $\Phi_i(\sigma)$ into a row-vector and add it to the matrix $M$;

   (b) Close these rows of $M$ by the action of $\sigma$ satisfying $\Phi_i(\sigma) = 0$ (the matrix multiplication of direct sums of copies of the companion matrix of $\Phi_i(\sigma)$ achieves this);

   (c) Compute the nullity $N_i$ (number of columns minus the rank) of the $\sigma$-closed matrix $M$;

   (d) Return $\Phi_i(\sigma)^{N_i/\phi(i)}$;

5. Return the product of powers of cyclotomics from the previous step.

This completes the computation of $c(S)$ for a given degeneracy set $S$, and thus by inclusion-exclusion, $d(S)$.

# Chapter 6

# Implementation

This chapter is devoted to aspects regarding implementation of the algorithm. If implemented naïvely, the algorithm will be slow and will not take full advantage of its redundant nature. A few main optimizations will be discussed, followed by a few smaller implementation details.

The algorithm presented in this thesis has been implemented in GAP 4.4.7. There is no compelling reason for this choice; it could be implemented just as easily in Magma, or even in more traditional programming languages such as C++ with some extra work. We will be focussing primarily on implementing algorithms to do particular tasks rather than the structure of the data involved in the algorithm.

The two main algorithms that benefit from optimization are the algorithm to generate the representative degeneracy sets, and the algorithm to compute inclusion-exclusion, given that the functions $c(S)$ have been precomputed. This is due to the fact that if implemented naïvely they operate on a combinatorial explosion of sets, which quickly puts a strain on memory and computational speed.

By their very nature, the degeneracy sets we deal with have a high degree of redundancy due to the symmetry discussed in Section 4.6. This symmetry can be exploited to reduce calculations to a more manageable set. Although the algorithm cannot avoid a certain amount of combinatorial explosion (since we must examine every possible conjugacy class structure provided), the methods outlined in this chapter go a long

way towards making this as manageable and practical as possible.

## 6.1 Generating representative degeneracy sets

In Section 4.3 we encountered the master degeneracy set $E_\tau$ which represented the set of all equations that we would deal with when examining fixed points for a particular type and representation. A degeneracy set $S$ was a particular choice of equations from this set, represented by making $S$ a subset of $E_\tau$. We need to examine every conjugacy class in this type, which meant looking at every degeneracy set. On the face of it, if $E_\tau$ is a set of $N$ elements, then we need to work with $2^N$ degeneracy sets.

However, in Section 4.6 we noted that there is a high degree of redundancy in our sets, due to the equivalence of type-parameters. The equivalence of type-parameters led to an equivalence of degeneracy sets.This equivalence can be viewed as a group action on degeneracy sets where the group involved is denoted $\mathcal{H}$. The action of $\mathcal{H}$ induces orbits of degeneracy sets and we choose one representative from each orbit to form our set of *representative degeneracy sets*. The PORC function $c(S)$ is the same for a equivalent degeneracy sets. Furthermore, the number of fixed points is the same for equivalent degeneracy sets. Therefore for each orbit representing a set of equivalent degeneracy sets we need only compute one instance of $c(S)$ and $\chi(g_S)$. Therefore, if we can find this set of representative degeneracy sets, then we can reduce the computational workload significantly.

We can easily construct the group $\mathcal{H}$, as detailed in Section 4.6. By Cayley's theorem, we can rewrite the action of $\mathcal{H}$ on the master degeneracy set $E_\tau$ as a permutation group acting on $\{1, \ldots, |E_\tau|\}$. This is achieved in GAP via the `Action` command. The permutation representation is much more concise, and it simplifies and speeds up group action calculations. This step is not essential, but it helps. We now represent a degeneracy set $S$ as a set of integers from $\{1, \ldots, |E_\tau|\}$. In an explicit form, $S$ would be a set of tuples of polynomials with integer coefficients, which is stored as a list of lists of lists of integers (where the deepest list is the coefficient list of the appropriate polynomial). In this permutation representation, a degeneracy set is just a list of integers which is

undoubtedly easier to use and more memory efficient. Conversion back to the explicit representation is easy: if the integer $i$ is in $S$, replace it with the $i$th element of the master degeneracy set $E_\tau$.

We now have a more efficient representation of our data, so now we move onto the algorithm to find the representatives of orbits of degeneracy sets under the action of $\mathcal{H}$. We still have a search space of $2^N$ elements. The naïve algorithm would compute the orbits of this domain and select representatives from each. Even with the more efficient representation, we would still need to firstly generate all $2^N$ elements, and then perform an orbit calculation on this large set. This is extremely impractical.

Our method approaches the problem in the opposite direction: instead of being a deductive algorithm (finding the representatives from the domain), it is a generative algorithm (we *create* the representatives and nothing else). In general this is a form of combinatorial generation. One of the earliest approaches to this sort of problem can be found in a paper of R. C. Read [46].

The guiding principle is that finding orbits of a small set is reasonably cheap, and that partitioning a large set into smaller ones and performing orbit calculations only on the smaller domains is cheaper than orbit calculation for the entire set. For instance, we will consider finding the orbits of elements in the master degeneracy set a "cheap" calculation. Knowing these orbits instantly gives us representatives of degeneracy sets of size one. In general, to find the sets of size two, we just need to consider combinations of two elements from the same orbit, or two elements from different orbits. This extends to degeneracy sets of any size if we choose our representatives properly.

The major complicating factor is that a given constraint ("two elements from orbit one, one element from orbit two, ...") can have multiple solutions that are actually in different orbits under the induced action of $\mathcal{H}$. For example, let $E_\tau = \{1, 2, 3, 4\}$, and suppose $\mathcal{H}$ is a group of order 4 acting on the elements of $E_\tau$ such that there is only one orbit of elements. We want to consider the orbits of subsets under the induced action. If $\mathcal{H}$ is the cyclic group of order 4 generated by $(1, 2, 3, 4)$, then we get the following

representative subsets:

$$\begin{aligned} \text{Size one:} &\quad \{1\} \\ \text{Size two:} &\quad \{1,2\} \text{ and } \{1,3\} \\ \text{Size three:} &\quad \{1,2,3\} \end{aligned}$$

with the obvious representatives of size zero and four (the empty set and the set $E_\tau$, respectively). However, if $\mathcal{H}$ was the Klein four-group generated by (12)(34) and (13)(24), then we get the following representative subsets:

$$\begin{aligned} \text{Size one:} &\quad \{1\} \\ \text{Size two:} &\quad \{1,2\}, \{1,3\} \text{ and } \{1,4\} \\ \text{Size three:} &\quad \{1,2,3\} \end{aligned}$$

again with the obvious representatives of size zero and four. Notice that the representatives of size one are the same, but in the sets of size two, there are more representatives in the latter example.

This difference can be viewed as a "failure of multiple transitivity". If $\mathcal{H}$ acting on $E_\tau$ was $|E_\tau|$-transitive, then we could easily generate the representative subsets of a given size by considering all constraints of the form "$c_i$ elements from orbit $i$", as there would be only one representative subset per constraint. Unfortunately our groups $\mathcal{H}$ are rarely like this, so we need to efficiently account for this difference. The method we use resembles a stabilizer chain algorithm.

A set of constraints of choosing elements from orbits of $E_\tau$ will be called a *configuration*. The size of a configuration is the size of the set that satisfies the constraints. So a configuration "two elements from orbit one, and one from orbit two" has size three. If we order and index the orbits of $E_\tau$, a configuration is a tuple $c = (c_1, \ldots, c_t)$ where $c_i$ is a non-negative integer indicating we take that many elements

from orbit $i$. The size of a configuration $c$ is thus

$$|c| = \sum c_i.$$

Our algorithm will first require us to generate all configurations, and then for each configuration, determine the representative sets of that configuration. The first point to note is that a configuration $c$ of size $n$ defines a dual configuration $c'$ of size $|E_\tau| - n$. In other words, any representative $S$ of configuration $c$ provides the complement $E_\tau \backslash S$, which can also be taken as a representative. Therefore we need only consider the configurations of size $N = 1, \ldots, \lceil \frac{1}{2}|E_\tau| \rceil$ and for each representative calculated, we add in its complement. This is a significant reduction of work. The representatives from configurations of sizes 0 and 1 (and thus, $|E_\tau|$ and $|E_\tau| - 1$) are already known: the empty set is the only representative of size 0, and the representatives from the orbits on $E_\tau$ (which we have already calculated) are the representatives of size 1. This further reduction is minor but helpful nonetheless.

Suppose $E_\tau$ has $t$ orbits under the action of $\mathcal{H}$, and denote the orbits as $\omega_1, \ldots, \omega_t$. A configuration will therefore be a $t$-tuple of nonnegative integers $(c_1, \ldots, c_t)$ such that $c_i \leq |\omega_t|$ for all $1 \leq i \leq t$. That is, the highest number of distinct elements you can take from an orbit is the number of elements in that orbit. One way to view a configuration of size $n$ is as a unordered tuple of choices from $\{1, \ldots, t\}$ where $i$ can be repeated at most $|\omega_i|$ times. In GAP we can find the set of configurations of size $n$ by the function `UnorderedTuples`$([1..t], n)$ and filtering appropriately.

Now we need to compute the representative sets for a given configuration. We will use a recursive algorithm. Let `Reps`$(c, G)$ be a function with $c$ a configuration and $G$ a finite (permutation) group acting on the set $E_\tau = \{1, \ldots, |E_\tau|\}$ (since we interpret $E_\tau$ via the permutation representation.) We will assume the function has access to $\Omega$, the set of orbits of $E_\tau$ under $\mathcal{H}$. The function `Reps`$(c, G)$ outputs a set of representative sets of configuration $c$ under the action of $G$. We begin our recursion by setting $G = \mathcal{H}$.

`Reps`$(c, G)$ begins by finding the first nonzero entry $c_i$ of $c$. If the configuration

has no nonzero entry, then $\texttt{Reps}(c, G)$ returns the empty set. Otherwise, the function generates all subsets of $\omega_i$ of size $c_i$, and calculates the orbits of these subsets under the induced action of $G$. From these orbits, it chooses a representative for each, and the set of all such representatives is stored as $R$. Form a new configuration $c'$ which is $c$ with $c_i$ zeroed. Then for each element $r$ of $R$ find the stabilizer $G_0(r)$ of the set $r$ under the action of $G$ and call $\texttt{Reps}(c', G_0(r))$. The result of this will be a set of subsets of $E_\tau$. For each such subset we append $r$. Once this is completed for all $r$ in $R$, return the collection of sets created by this procedure.

This algorithm is essentially a depth-first traversal of a tree, where a path in the tree corresponds to a unique representative degeneracy of a given configuration. Each level in the tree corresponds to choices of elements from a specific orbit of $E_\tau$ (so that the tree has depth at most $t$, the number of orbits). Note that nodes at depth $i$ do not necessarily correspond to elements in the $i$th orbit, because we skip over any orbits that do not feature in the configuration. We partition the orbit $\omega_i$ of $E_\tau$ under the action of the stabilizer of the level above for two reasons: we need to determine the orbits under a group to account for the difference mentioned in the previous example; and this group is the stabilizer of the level above so as to not interfere with the partition of the upper levels.

As an example, say there are 3 orbits of $E_\tau$: $\omega_1$, $\omega_2$, and $\omega_3$. Suppose we have configuration $(2, 2, 1)$. We first choose two elements from $\omega_1$. The options available to us are representatives from the orbits of 2-tuples of elements from $\omega_1$ under the action of $\mathcal{H}$. This is because the action of $\mathcal{H}$ on these elements must keep them in the same orbit, so they will forever be a 2-tuple of elements from $\omega_1$ under the action of $\mathcal{H}$. So if we select one of these 2-tuples, say $X_1$, we want to preserve this choice. Therefore our choices for elements in $\omega_2$ must be under the action of the stabilizer $G_0(1)$ of $X_1$. So we compute the orbits of 2-tuples of elements from $\omega_2$ and select a candidate $X_2$, and compute its stabilizer $G_0(2)$ in $G_0(1)$. This stabilizer (by construction) preserves the choice of both $X_1$ and $X_2$ (that is, they are both setwise-stable under the action of this group). Then we compute the orbits of $\omega_3$ under the action of $G_0(2)$. The complete list

of choices we could have made corresponds to the complete list of orbits of degeneracy sets of this configuration under the action of $\mathcal{H}$.

### 6.1.1 Utility and efficiency

One of the useful side-effects of this algorithm is that we can easily find the size of the orbit of any representative degeneracy set. When we choose the set of representatives from an orbit $\omega_i$, we have to compute the orbits. If we know this information for each node in the tree, then the size of an orbit of a degeneracy set (which is represented by a path in the tree) is the product of the size of orbits associated to each node. Another way of viewing it is that we store the number of options we had for a representative at each node. The product of number of options is the total number of options we had for a representative degeneracy set, that is, the size of the orbit we have chosen it from. The size of the orbit of a degeneracy set will be used later in our inclusion-exclusion algorithm.

The main point of efficiency in this algorithm is that we never have to store all of the subsets of $E_\tau$ at any one stage in the calculations. We only have to store the representative degeneracy sets themselves, as well as associated, useful information. We still need to do orbit calculations on sets of subsets, but these domains are quite restrictive (they restrict to subsets of elements of a fixed size, coming from a set that is smaller than $E_\tau$).

On a 3.2 GHz machine with 1 GB of memory running Windows XP, the algorithm can find the representative degeneracy sets for the first type in $\mathrm{GL}(3,3;p)$, of which there are about half a million representing $2^{24} \approx 16$ million sets, in under 15 minutes (including time for storage and other auxiliary functions). The naïve approach could take over a week on the same machine (although the memory requirements for such a thing would force frequent swapping to the disk).

This approach is easily suited to a parallel computing environment where at the very least the work could be distributed as different configurations for degeneracy sets as the calculations are independent. With a little more work, the workload could be

divided amongst processes, each operating on a node in a tree belonging to a particular configuration.

In practice there doesn't seem much need to worry about parallelizing or optimizing the representative degeneracy sets algorithm. The calculations in this algorithm may take a not-insignificant amount of time, but the time taken is of a much smaller magnitude than later calculations, especially inclusion-exclusion. Nevertheless, parallelizing by configurations can be fruitful.

## 6.2 Inclusion-Exclusion procedure

To compute the number of choices $d(S)$ for a given degeneracy set $S$, we require an inclusion-exclusion calculation where the terms are precomputed PORC functions $c(S)$. Symbolically this is just

$$d(S) = \sum_{S \subseteq T} (-1)^{|T-S|} c(T), \tag{6.1}$$

where $T$ is a subset of the master degeneracy set $E_\tau$ that also contains $S$. The main difficulty here is that there are many such sets. The worst case scenario is when $S$ is the empty set — in this case, if the master degeneracy set has size $N$, then there are $2^N$ subsets to consider.

A direct approach would be to create the Boolean lattice of sets and calculate inclusion-exclusion by traversing this lattice. This approach has three main problems:

1. Determining the sets involved in a part of our Boolean lattice is expensive;
2. There are, at worst, $2^N$ PORC functions required to be retrieved from storage and held in memory;
3. There are, at worst, $2^N$ additions of PORC functions.

Of course, there are cases when there are simple solutions to these problems, say in the case that $S$ is the empty set or the whole set $E_\tau$. However we need an efficient solution for *all* sets.

The main failing of the direct approach is that it does not recognize and utilize the high amount of redundancy in the PORC functions $c(S)$. Equivalent degeneracy sets have equal PORC functions. Compressing these equivalent degeneracy sets into a single vertex reduces the lattice we need to work with. The lattice itself is more complicated as it is not as uniform, but this reduces the number of vertices we need to look at for each inclusion-exclusion calculation. With this reduction we can rewrite (6.1) as

$$d(S) = \sum_T (-1)^{|T-S|} N_S(T) \cdot c(T),$$

where $T$ ranges over the set of representative degeneracy sets, $N_S(T)$ is the number of sets equivalent to $T$ under the action of $\mathcal{H}$ that also contain $S$, and as before $c(T)$ is the PORC function associated to a degeneracy set $T$.

This approach allows us to lessen the penalty of problems 2 and 3 since we only deal with the representative degeneracy sets. We already have a list of these representatives (as well as their respective sizes) so problem 1 is mostly contained in calculating $N_S(T)$.

To calculate $N_S(T)$ we will use three small results ([27] pg 142–143).

**Lemma 6.1.** *Let $\mathcal{H}$ be a finite automorphism group of a finite Boolean lattice $(L, \subseteq)$. Then the following results hold:*

1. *If sets $a$ and $b$ are in the same orbit under $\mathcal{H}$, then $a \not\subset b$ and $b \not\subset a$;*

2. *For any orbit of sets $\omega$ and a fixed set $a \in L$, then*

$$|\{b \in \omega \mid a \subseteq b\}|$$

   *is dependent only on the orbit to which $a$ belongs, not $a$ itself.*

3. *For any sets $a, b \in L$,*

$$|\mathcal{H}(a)| \cdot |\{c \in \mathcal{H}(b) \mid a \subset c\}| = |\mathcal{H}(b)| \cdot |\{d \in \mathcal{H}(a) \mid d \subset b\}| \tag{6.2}$$

   *where $\mathcal{H}(x)$ is the orbit of $x \in L$ under the action of $\mathcal{H}$.*

In short, $N_S(T)$ does not depend on which representatives we have chosen for our lattice reduction. Equation (6.2) in the previous lemma allows us to simplify how we calculate $N_S(T)$. By our construction of representative degeneracy sets, we already know $|\mathcal{H}(S)|$ and $|\mathcal{H}(T)|$. If $|\mathcal{H}(T)| < |\mathcal{H}(S)|$ then we can just find the orbit of $T$ and count how many elements in that orbit contain $S$. Alternatively, if $|\mathcal{H}(S)| < |\mathcal{H}(T)|$ then we can use Equation (6.2):

$$N_S(T) = \frac{|\mathcal{H}(T)|}{|\mathcal{H}(S)|}|\{S' \in \mathcal{H}(S) \mid S' \subset T\}|.$$

Either way, we always compute the smallest orbit and count inclusion over the smallest possible values. This approach also allows us to use any results that can give

$$|\{T' \in \mathcal{H}(T) \mid S \subset T\}| \quad \text{or} \quad |\{S' \in \mathcal{H}(S) \mid S' \subset T\}|$$

without computing the sets explicitly. At the moment no method is known to do this implicit calculation so we just use `IsSubset` to determine inclusion. This is admittedly expensive. If we stored the configuration of a representative degeneracy set we can short-circuit this calculation if the configuration of $T$ did not contain the configuration of $S$. In this case $N_S(T) = 0$. However this is yet another `IsSubset` calculation that is not helpful in the case that the configuration of $S$ is contained in the configuration of $T$.

Precomputing $N_S(T)$ is not worthwhile as the value is used only once for each pair $S$ and $T$. Precomputing the orbits of $T$ is also not worthwhile as it requires us to store $2^{|E_\tau|}$ sets which we tried to avoid in the first place. There is a possibility that $N_S(T)$ could be deduced from information gleaned in the construction of the representative degeneracy sets. As yet a method has not been found.

One of the remaining problems in calculating $d(S)$ is that computing the sum

$$d(S) = \sum_T (-1)^{|T-S|} N_S(T) \cdot c(T),$$

is expensive if done explicitly. Although multiplying a PORC function by an integer is cheap, adding PORC functions is not. A further reduction was suggested by Mike Newman: equivalent degeneracy sets have equal PORC functions, but furthermore two nonequivalent representative degeneracy sets may have the same PORC function. This arises because there are few "options" available when computing $c(S)$. The degrees of parameters in a type limits the cyclotomic polynomials that can appear in $c(S)$, as well as their multiplicity. Though the exact number of options for torsion parts of $c(S)$ are unknown, there are typically few. The number of unique functions $c(S)$ (even when roughly estimated) are far fewer than the number of representative degeneracy sets. Therefore if we calculate the correspondence between the representative degeneracy sets and which unique PORC function $c(S)$ it has, then we can simplify our summation significantly. This is done by storing a vector $v$ of length equal to the number of unique PORC functions $c(S)$. Once we compute $(-1)^{|T-S|}N_S(T)$ then we add this to the appropriate component of $v$ as a "frequency". We can store the correspondence between representative degeneracy sets and their unique PORC function via a lookup hash table. Obviously we do not store the PORC function itself, but a reference to a canonical list of unique PORC functions. The correspondence can be computed cheaply whilst calculating the functions $c(S)$.

Therefore when calculating

$$d(S) = \sum_T (-1)^{|T-S|} N_S(T) \cdot c(T),$$

we need only do integer additions to store the "frequency" of the unique PORC function $c(T)$. To complete the sum we only need multiply each unique PORC function by its frequency and add those PORC functions together. This list is small, so the total calculation is much cheaper than adding many PORC functions together[1]. Another benefit of this method is that we only need to store a small number of PORC functions, which is easily done in memory. Storing the explicit lists of $c(S)$ is very expensive

[1]One of the major complications with adding many PORC functions together is creating and destroying large numbers of lists which slows down garbage collection, particularly in GAP.

memory-wise, and is very slow to read off a hard-disk. Therefore we save on memory access time and space.

The most expensive procedure in calculating $d(S)$ is the explicit orbit calculations inside $N_S(T)$. This is exacerbated by having to calculate inclusion-exclusion on a large number of candidates. This large number is unavoidable, even with the reductions. However the reductions outlined here allow the procedure to be computable in a practical sense.

There is one final, smaller optimizations we can employ. Firstly, when calculating $N_S(T)$, we end up looping over all $T$ of a given size. The sum of $N_S(T)$ over all $T$ of a given size $n$ is

$$\sum_{|T|=n} N_S(T) = \binom{|E_\tau| - |S|}{n - |S|}.$$

If we keep track of this sum, we can possibly break the loop early once the sum reaches the maximum value. This can avoid calculating parts that do not contribute to the answer. Of course this is not always avoidable (say if all the representative sets that have $\mathcal{H}$-equivalent sets containing $S$ are at the end of the queue), but this is a cheap optimization to include.

## 6.3 Other aspects of implementation

This section details some of the smaller aspects regarding implementation.

### 6.3.1 Representation of data

The whole algorithm has several main sources of data: the types, the master predegeneracy set, the master degeneracy set, representative degeneracy sets, and the PORC functions $c(S)$ and $d(S)$. Primarily these are sets of objects which are most often represented by sets or lists. In GAP, lists are considered to be arrays of objects (that do not have to be of the same data type) that are not sorted and allow duplicates. A set in GAP terminology is a sorted list with no duplicates. As such, sets are more computationally expensive. In the current implementation of the main algorithm, most

data that could either be a list or a set is implemented as a list.

Types are mathematically considered to be lists of type-parameters given as triples $(d, \Lambda, l)$, where $d$ and $l$ are positive integers, and $\Lambda$ is a partition. We have implemented types as lists of records. A record is a data object with named components. So a triple $(d, \Lambda, l)$ is represented in GAP by a list of objects

`rec( degree :=` $d$`, partition :=` $\Lambda$`, location :=` $l$`)`

The partition $\Lambda$ is given just as a list of positive integers. We acquire the partitions via the `Partitions` function which orders the elements in a partition (in descending order), but we do not rely on this and treat the partition just as a list of positive integers.

Predegeneracy and degeneracy sets are represented by sets of tuples, as described in Chapter 4. The polynomials $u_i$ involved in the elements of degeneracy sets are stored as lists of coefficients rather than polynomials.

A representative degeneracy set is a subset of the master degeneracy set that represents several others via the group action of $\mathcal{H}$. We do not store the entire coset, just the representative. In explicit form, a degeneracy set is a list of elements, each of these elements representing a list $(u_1, \ldots, u_w)$. However, this makes a degeneracy set a list of lists of lists of integers. Since the master degeneracy set $E_\tau$ is fixed for a type, and is indexed, we can store a degeneracy set as a set of integer indices from $\{1, \ldots, |E_\tau|\}$. This is a much more compact representation, especially since the indices do not get very large. One can use an even more compact representation via a Blist which is essentially a bit-list of length $|E_\tau|$ where the $i$th bit is 1 if the $i$th element of $E_\tau$ is in the degeneracy set, and zero otherwise. We do not use this as the previous representation is easier to use in permutation actions. GAP does not have extensive native support for bitlists (they are actually lists whose elements are true or false). A degeneracy set can be swapped to either of the two forms quite easily if need be. Blists are quite efficient for determining whether a set is a subset of another and so can be utilized in the inclusion-exclusion calculations.

As mentioned in Section 1.1 on page 5, a PORC function is a sum of terms of the form

$$a(p)b(p)$$

where $a(p)$ is a product of functions of the form $\gcd(k, f(p))$ (with $k$ a positive integer and $f(p)$ a polynomial in $p$ with integer coefficients) and $b(p)$ is a polynomial in $p$ with integer coefficients. We represent the polynomials (in both parts) as lists of coefficients, and gcds as pairs $(k, f(p))$ since they do not have to be functionally gcds. The lists are kept unique by combining the polynomials whenever the $a(p)$ parts match. The zero PORC function is represented as an empty list.

For the most part we only need to add PORC functions and multiply a PORC function by a rational number. To add PORC functions we determine the $a(p)$ parts involved and for each unique $a(p)$ we add the polynomials $b(p)$ that had a corresponding part $a(p)$. Adding the polynomials is just component-wise list addition (where a list of coefficients for a smaller-degree polynomial is enlarged to meet the length of the larger-degree polynomial). The slow part of this process is determining the unique $a(p)$ parts. Multiplying PORC functions by a rational number $r$ simply involves multiplying each list corresponding to $b(p)$ parts by $r$ (with the appropriate modifications for multiplying by zero).

Only in the final stages of calculating a type do we need to multiply and divide PORC functions by a polynomial. For the former we use special vector arithmetic to emulate multiplying the $b(p)$ parts by a polynomial (we multiply all such parts as their collection implies a summation of terms $a(p)b(p)$). For the latter we need to convert the coefficient lists to polynomials and do the explicit division. The resulting function will always be a PORC function, even in our restricted sense (that is, our divisor will always evenly divide the polynomials $b(p)$).

# Chapter 7

# Results

The algorithm discussed in this thesis has been implemented in GAP 4.4.7. It has successfully calculated the first few values of $g(r, s; p)$ and $f_2(n; p)$:

| $(r, s)$ | $g(r, s; p)$ |
|---|---|
| $(1, 1)$ | 1 |
| $(2, 1)$ | 3 |
| $(2, 2)$ | 3 |
| $(3, 1)$ | 4 |
| $(3, 2)$ | $p + 14$ |
| $(4, 1)$ | 6 |

| $n$ | $f_2(n; p)$ |
|---|---|
| 1 | 1 |
| 2 | 2 |
| 3 | 4 |
| 4 | 8 |
| 5 | $p + 22$ |

These results can be found in a reasonable amount of time. The $(3, 2)$ and $(4, 1)$ cases take approximately 25 minutes each on a 3.2 GHz machine with 1 gigabyte of RAM running GAP 4.4.7 on Windows XP. Results for larger values of $r$ and $s$ take more time. The case $(3, 3)$ takes several weeks of CPU time, most of which is spent in calculating inclusion-exclusion.

This exponential increase in time is an unfortunate side-effect of finding an answer in the form of a PORC function. The problem suffers from an inherent combinatorial explosion of data, even with major algorithmic improvements to tame this. In general, the algorithm could run in a reasonable time if it weren't for inclusion-exclusion. For example, the data in $(3, 3)$ case (not including inclusion-exclusion) can be computed

for all 64 types in under half an hour with the machine described above. The inclusion-exclusion calculation takes weeks.

The bottleneck in the inclusion-exclusion calculation is determining the number of degeneracy sets in the orbit of a given set $T$ that contain another degeneracy set $S$. If this calculation could be sped up by a factor of $K$ then the entire inclusion-exclusion calculation would be sped up equivalently. Preferably such a calculation would not find the orbit of $T$ and detect whether a set contained $S$ (as both processes are expensive). It is known that the answer we should get out is only dependent on the orbit of $T$, not on $T$ itself. It stands to reason that structural information of the action of $\mathcal{H}$ on $T$ and $S$ should give us the answer we need. At present, such a method has not been found.

A method has been proposed that allows us to try to get explicit PORC answers in a short amount of time without computing inclusion-exclusion. This is the *hybrid method* which we will discuss next.

## 7.1 Hybrid method

Inclusion-exclusion is a prohibitively expensive calculation. It gives us exact, functional results, but at the cost of a exponential increase in calculation time. Given that all the other parts of the main algorithm are reasonably fast, we would like to use them to find $g(r, s; p)$ without calculating inclusion-exclusion.

This approach is called the *hybrid method* as it takes the functional answers gained from the main algorithm of this thesis (without inclusion-exclusion), finds the degree of the polynomial part of $g(r, s; p)$ and then uses O'Brien's `ClassTwo` function to interpolate $g(r, s; p)$ and thus determine it. Note that the degree of $g(r, s; p)$ is at most the degree of $h(r, s; p)$. Thus we need a degree estimate on the following equation:

$$h(r, s; p) = \frac{1}{|\mathrm{GL}(r, s; p)|} \sum_{\tau} \sum_{S \subseteq E} |c_\tau| \cdot d(S) \cdot \frac{p^{fs}}{|\mathcal{H}_0(S)|}.$$

The PORC function $d(S)$ is the sum of PORC functions $c(T)$ via inclusion-exclusion. The degree of $|\mathrm{GL}(r, s; p)|$ is $r^2 + s^2$, so the degree of $g(r, s; p)$ will be the degree of the

sum minus this value (as $|\mathrm{GL}(r,s;p)|$ divides the sum evenly).

Therefore we need to find the degree of the sum. The degree will be the maximum degree over all types $\tau$. Each type is a term

$$\sum_{S\subseteq E_\tau} |c_\tau| \cdot d(S) \cdot \frac{p^{f_S}}{|\mathcal{H}_0(S)|}.$$

The polynomial $|c_\tau|$ (the number of elements of any conjugacy class of type $\tau$) is fixed for a type. We can easily calculate $|c_\tau|$, and thus its degree. Therefore we want to find

$$\max_{S\subseteq E_\tau} \left\{ \deg\left( d(S) \cdot p^{f_S}/|\mathcal{H}_0| \right) \right\}$$

which is equal to

$$\max_{S\subseteq E_\tau} \{\deg(d(S)) + f_S\}.$$

We can quickly find the value of $f_S$ for all degeneracy sets $S$, so all we need to do is find the degree of $d(S)$ without calculating it explicitly.

If we actually calculated inclusion-exclusion, $d(S)$ would be an integer linear combination of the PORC functions $c(T)$ for $T \supseteq S$. Consider the following lemma:

**Lemma 7.1.** *Let $S$ be a degeneracy set, and $T$ a subset of $S$. Then the polynomial part of $c(S)$ divides the polynomial part of $c(T)$.*

*Sketch of proof.* The degeneracy set $S$ is the degeneracy set $T$, with added constraints. Therefore the number of choices of elements restricted by the equations from $S$ is at most as many as there are for $T$.

More specifically, the associated matrix used to compute $c(S)$ has rank at least as large as the rank of the associated matrix used to compute $c(T)$. Therefore the associated nullity for $c(S)$ must be at most the nullity for $c(T)$ and thus $c(S)$ must divide $c(T)$. □

Since all terms have degree at most the degree of $c(S)$, this is an upper bound on the degree of $d(S)$. In other words, the number of choices of finite field elements satisfying

equations in $S$ and certain inequalities can be estimated by ignoring the inequalities. Unfortunately this gives too much weight to some sets $S$ where the degree of $c(S)$ and the number of fixed points is high, but $d(S)$ is actually zero. This occurs when the equalities in $S$ contradict the inequalities that also need to hold. We have a process to deal with these exceptions.

**Lemma 7.2.** *For a given degeneracy set $S$ of equations, and a set $T$ of inequalities, we can determine whether these conditions can hold simultaneously, that is, whether the system of equations and inequalities are consistent.*

*Proof.* Our equations and inequalities involve type-parameters $\beta_1, \ldots, \beta_w$ and their Galois-conjugates. As before, an equation

$$\beta_1^{u_1} \cdots \beta_w^{u_w} = 1$$

can be written additively as a vector $(u_1, \ldots, u_w)$. If we replace the polynomials $u_i$ by vectors of length given by the degree of the corresponding type-parameter $\beta_i$, then we have an equivalent vector $(v_1, \ldots, v_{k'})$ representing this equation.

The set of all equations formed by $S$ is a $\mathbb{Z}[\sigma]$-module. Suppose it has rank $r$ as a $\mathbb{Z}$-module. If the equations $S$ and inequalities $T$ were consistent, then if $t$ is a vector in $T$, then $t$ cannot be contained in the span of the equations from $S$. That is, if an equation is derivable from $S$ then it must hold as an equality. But if $t$ is in the span of the equations from $S$ then it is derivable from the equations.

Therefore for each equation in $T$, we need a process to show that the vector corresponding to this equation is not contained in the $\mathbb{Z}$-module derived from $S$. Let the vectors obtained from $S$ be the rows of a matrix $U$. Compute the Sigma-closed Hermite Normal Form ($\sigma$HNF) of $U$ and store it in a matrix $U'$. The number of rows of $U'$ is the rank of the associated $\mathbb{Z}$-module. Now if we append to $U'$ a vector $t$ from the inequalities in $T$, and compute the $\sigma$HNF of the enlarged matrix. If $t$ is in the span of the rows of $U$, then the rank will be $r$ and the set of equations and inequalities will be inconsistent. Otherwise if the rank is $r + 1$ for every vector $t$ in $T$, then the system of

equations and inequalities will be consistent. □

This now allows us to find a reasonable estimate for the degree of $d(S)$. Interestingly, this estimate is actually an exact answer:

**Theorem 7.3.** *The hybrid algorithm gives the exact degree of* $g(r, s; p)$.

*Proof.* The hybrid algorithm computes the precise values for the size of a conjugacy class of a type, and the number of fixed points for an element of a conjugacy class of given degeneracy. The only estimated parameter is the degree of $d(S)$. The degree of $d(S)$ is estimated as the degree of $c(S)$.

Recall that

$$d(S) = \sum_{S \subseteq T} (-1)^{|T-S|} c(T).$$

By Lemma 7.1, the degree of each term in this sum is at most the degree of $c(S)$. Therefore if we estimate the degree of $d(S)$ by $c(S)$, we never under-estimate the degree. Since this holds for all degeneracy sets of a type, we never underestimate the maximum degree in a type, and since this holds for all types, we never underestimate the degree of $g(r, s; p)$.

For a given type, let $N$ be the maximum degree of $c(S)$ over all consistent representative degeneracy sets $S$ that would have inclusion-exclusion calculated for it. Let $S'$ be a largest such degeneracy set such that the degree of $c(S')$ is $N$. Since it has the largest possible cardinality for a set with this degree, it follows that for any subset $T$ of $S'$, the degree of $c(T)$ is strictly less than the degree of $S'$. Therefore the degree of $d(S')$ must be $N$. Since we are assured of a set $S'$ whose $d(S')$ is of degree $N$ in this type, and the $d(S)$ are always positive (and so cannot cancel the leading terms of one another), the maximum degree in this type must be $N$. Therefore the hybrid algorithm never overestimates the degree in a given type, and so never overestimates the degree of $h(r, s; p)$. Therefore the hybrid algorithm must calculate the degree of $h(r, s; p)$ exactly. □

The filtering process that determines whether a set $S$ is consistent or not is not too

expensive. Without it the estimates given are too high (they are roughly as bad as the degree estimates in Higman [20]), and so it is worthwhile to take the performance hit for a much more accurate answer.

Having found the degree of $g(r, s; p)$ we need to interpolate it to get the exact PORC function. Here we need to consider that $g(r, s; p)$ is not just polynomial, but PORC. From the $c(S)$ data we can determine which periodic parts could possibly show up in $g(r, s; p)$. In practice there seems to be very few options for periodic parts. Using this, we can determine which residue classes we need perform interpolation for. For example, if our only periodic part (other than the trivial one) is $\gcd(3, p-1)$ and the degree of $g(r, s; p)$ is $D$, then we will need at most $D + 1$ primes for each residue class modulo 3. For each such prime we calculate the appropriate `ClassTwo` value. By Lagrange's interpolation theorem we can find the unique polynomial through these points, and thus obtain $g(r, s; p)$.

A limiting factor in the hybrid algorithm is that the `ClassTwo` function has running time roughly exponential in the size of its inputs. If the primes we provide it in the interpolation step are too large, the calculation may take some time. In most cases, however, this will be much less than the time it would take to compute inclusion-exclusion. Inclusion-exclusion will give you an exact, functional answer with easily extractable (and checkable) terms for a severe cost in computation time. The hybrid algorithm is much faster yet still precise, but does not allow for the fine-scale book-keeping.

## 7.2 Verification of results

In this section we present a variety of checks to ensure that the data we calculate is correct. Some of them are simple checks to make sure the calculations are not obviously wrong, while other provide a much stronger standard of correctness. The checks also provide heuristics that may be useful in determining performance, expected calculation times, and bounds for the hybrid method.

### 7.2.1 The group $\mathcal{H}$

The group $\mathcal{H}$ is defined simply as the group of permutations of type-parameters, keeping degree, partition and location invariant. The order of $\mathcal{H}$ is typically small and so $\mathcal{H}$ can be inspected by eye. We can algorithmically check that the order of $\mathcal{H}$ is correct. Suppose we have type-parameters $y_1, \ldots, y_w$ with $y_i = (d_i, \lambda_i, l_i)$. Let $k(d, \lambda, l)$ be the number of type-parameters that have degree $d$, partition $\lambda$, and location $l$ (exactly). Then

$$|\mathcal{H}| = \prod_{(d,\lambda,l)\in\tau} k(d, \lambda, l)! \cdot d,$$

for all applicable triples $(d, \lambda, l)$.

Another concern is the action of $\mathcal{H}$ on degeneracy sets, and more specifically, on elements of such sets. The action of $\mathcal{H}$ on elements of degeneracy sets is given as a matrix multiplication on a vector. Due to the symmetry in our particular representation $\Gamma$, the image of the set of degeneracy sets under the action of $\mathcal{H}$ must be exactly $E_\tau$.

### 7.2.2 Predegeneracy sets

The master predegeneracy set describes the Jordan canonical form of a matrix under the representation $\Gamma^{(s)}$ for Chapter 2. The size of a Jordan block corresponding to a predegeneracy element is the data we computed in Chapter 4. So if $T$ is the master predegeneracy set, and $\mu(t)$ for $t \in T$ is the size of the block corresponding to $t$ then

$$\sum_{t\in T} \mu(t) = \frac{1}{2} r(r+1)s, \tag{7.1}$$

where $\Gamma$ is the representation of $\mathrm{GL}(r; \mathbb{F}_p) \oplus \mathrm{GL}(s; \mathbb{F}_p)$ induced by the action on subspaces of $V \oplus V \wedge V$ of codimension $s$, where $V = (\mathbb{F}_p)^r$. This is a simple check that equates to the matrix representation being of the correct size. The dimension of $V \oplus V \wedge V$ is $\frac{1}{2}r(r+1)s$ so a matrix acting naturally on it must have the same dimension.

### 7.2.3 Base type inequalities

In our setup, we never use base type equalities for our degeneracy sets, but we do use the base type inequalities. Following the notation in Section 7.2.1, the set $I$ of base type inequalities satisfies the following equation:

$$|I| = \sum_{(d,\Lambda,l)} \binom{k(d,\Lambda,l)}{2} \cdot d.$$

### 7.2.4 Master degeneracy set

Since the master degeneracy set $E_\tau$ is a simple translation from the master predegeneracy set $T$, Equation (7.1) in Section 7.2.2 holds for $E_\tau$ as well. As a degeneracy set (in a broader sense) can contain elements from the base type inequalities $I$ and those from $E_\tau$, the set of all degeneracy sets is of size $2^N$ where

$$N = \frac{1}{2}r(r+1)s + \sum_{(d,\Lambda,l)} \binom{k(d,\Lambda,l)}{2} \cdot d,$$

with the notation as specified in previous subsections.

### 7.2.5 Representative degeneracy sets

One of the main bases of our algorithm is to construct degeneracy sets that will represent *all* degeneracy sets. If we assume that the function to compute orbits of elements is trustworthy, then we have a few straightforward checks we can employ.

Whilst finding the representative degeneracy sets, we store the size of the orbit for each representative. If we keep a running total of these, then they should be equal to the total number of degeneracy sets. If our total differs, then this indicates redundant or omitted sets. These totals can also be broken down by size of the degeneracy set, so we have a finer grain of verification. That is, the total number of represented degeneracy sets of size $i$ should be $\binom{|E_\tau|}{i}$. If any one of these differs, we know where to look.

For a given configuration, the representative degeneracy sets of that configuration cannot be redundant as the algorithm to generate them necessarily separates subsets

via the orbit algorithm. Specifically, for each part in a configuration we recurse based on the distinct options given to us by the orbit calculation. By definition these options have to be distinct (because the orbit calculation partitions the set). Furthermore, two representative sets of different configurations cannot represent the same set. As an example, if one configuration required two elements from orbit 1 and the other configuration required three, then there is no way for the induced action of $\mathcal{H}$ on these sets to take any set of one configuration to the other.

If we want more assurance on the validity of our representative degeneracy sets, then we can explicitly check them. It is easy (but extremely memory-intensive) to generate degeneracy sets from the representatives (just by invoking the `Orbit` command in GAP). There are ways to reduce the memory workload, say by considering sets of a set size, and using iterators to cycle through the sets. This method can check both whether all sets are generated by the representatives, and whether two representatives generate the same set. In any case, this is an expensive check but gives absolute assurance that the representative degeneracy sets are valid.

### 7.2.6 The PORC functions $c(S)$ and $d(S)$

In essence the PORC function $c(S)$ counts the number of choices of elements from a finite field with constraints given by the set $S$. Specifically, if we have type-parameters $y_1, \ldots, y_w$, then $c(S)$ gives us the number of choices of vectors $(\beta_1, \ldots, \beta_w)$ from $(\mathbb{F}_q)^w$, where $q$ is a specified power of $p$ based on the degrees of the type parameters, and where the vectors satisfy equations

$$\beta_1^{u_1} \cdots \beta_w^{u_w} = 1$$

with $(u_1, \ldots, u_w) \in S$.

It is simple to construct the field $(\mathbb{F}_q)^w$ for a given $q$. The structure of the elements of $S$ can be readily used to create filters, so we can filter the elements of $(\mathbb{F}_q)^w$ according to whether they satisfy an equation given by an element of $S$. The number of elements

satisfying all such filters should be equal to the value of $c(S)$ at $p$. If we check $c(S)$ for enough primes $p$, we can validate the PORC function.

This check is slow as we are filtering large sets, and have to do a lot of finite field arithmetic. This method also runs into the problem that we may try to construct finite fields $\mathbb{F}_q$ larger than the system can handle (not so much in memory requirements but because there are internal limitations on such things). For a PORC function with gcd parts $\gcd(k_i, f_i(p)$ and polynomial $g(p)$ of degree $d$, we need to test at most

$$\left(\prod_i k_i\right) \cdot d$$

primes $p$. In practice we can limit the $k_i$ to distinct integers.

We can similarly check $d(S)$ using the same filters appropriately.

### 7.2.7 Fixed points

From Section 7.2.2 we already have one check on our fixed point data. For a given degeneracy set $S$ and a fixed prime $p$, it is possible to explicitly construct a matrix from the representation and a set of finite field elements chosen from $\mathbb{F}_{p^n}$ in much the same manner as the explicit check for $c(S)$. Given this matrix we can explicitly find the number of fixed points and compare. This is incredibly slow and should only be used sparingly.

### 7.2.8 Other checks

The Cauchy-Frobenius theorem

$$\#\text{ number of orbits} = \frac{1}{|G|}\sum_{g\in G}\chi(g)$$

implicitly ensures that our final answer for the total number of fixed points over all types is divisible by the size of $G$. If anything were to go wrong in any one of the types, one would expect this to be visible at the level of the Cauchy-Frobenius theorem. This would detect if we omitted a type (since the result for any type is strictly positive) or

overcounted our degeneracy sets or their respective number of fixed points. It is unlikely (but still possible) that an unfortunate series of errors could conspire to produce a false positive. In any case, this check is automatic as our PORC functions always have polynomial parts, never strictly rational functions so our data structures will detect this immediately.

Furthermore, if we perform our calculations but ignore the number of fixed points for a given degeneracy of a given type, then the sum of this must equal the number of elements in $\mathrm{GL}(r;K) \oplus \mathrm{GL}(s;K)$ since this is essentially an element count. That is, if we reinterpret Theorem 2.5 we find that

$$|G| = \sum_{\tau} |c_\tau| \sum_{S \subseteq E_\tau} d(S), \tag{7.2}$$

where $|c_\tau|$ is the size of any conjugacy class of type $\tau$ (à la Theorem 3.3). Throughout our calculations we can compute this sum and check that it indeed equals $|G| = |\mathrm{GL}(r;K) \oplus \mathrm{GL}(s;K)|$ (for which we have a well-known formula). If the calculations pass this check but not the divisibility by $|G|$ then we know that the number of fixed points for a given degeneracy set is to blame (as it is the only element in the latter check but not the former). Combined with the previous checks, we can readily pinpoint corrupt data.

All of the results presented at the beginning of this chapter passed the following checks:

1. The size of the group $\mathcal{H}$;
2. The size of the predegeneracy set $T$;
3. The size of the master degeneracy set and the base type inequalities;
4. The numerical check that the representative degeneracy sets represented all degeneracy sets;
5. The PORC functions $c(S)$ and $d(S)$, checked for the primes $2, 3, 5$, and $7$;

6. The divisibility of the penultimate result by $|G|$;

7. All elements were accounted for (Equation (7.2)).

Explicit checks on the representative degeneracy sets and the number of fixed points for a given degeneracy set were too expensive to perform for all data. The above checks form a reasonably strong assertion of correctness in of themselves. The fact that the results match those already calculated by other methods strengthens this opinion.

# Bibliography


[1] Sequence A003606, The On-Line Encyclopedia of Integer Sequences. Published electronically at `http://www.research.att.com/~njas/sequences/A003606`, 2006.

[2] George E. Andrews. *The theory of partitions*. Cambridge Mathematical Library. Cambridge University Press, Cambridge, 1998. Reprint of the 1976 original.

[3] Tom M. Apostol. *Introduction to analytic number theory*. Springer-Verlag, New York, 1976. Undergraduate Texts in Mathematics.

[4] Giuseppi Bagnera. La composizione dei gruppi finiti il cui grado è la quinta potenza di un numero primo. *Ann. Mat. Pura Appl.*, 1898.

[5] F. Bergeron, G. Labelle, and P. Leroux. *Combinatorial species and tree-like structures*, volume 67 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, Cambridge, 1998. Translated from the 1994 French original by Margaret Readdy, With a foreword by Gian-Carlo Rota.

[6] P. B. Bhattacharya, S. K. Jain, and S. R. Nagpaul. *Basic abstract algebra*. Cambridge University Press, Cambridge, second edition, 1994.

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

[8] Arthur Cayley. On the theory of groups, as depending on the symbolic equation $\theta^n = 1$. *Philos. Mag. (4)*, 7:40–47, 1854.

[9] Henri Cohen. *A course in computational algebraic number theory*, volume 138 of *Graduate Texts in Mathematics*. Springer-Verlag, Berlin, 1993.

[10] F.N. Cole and J.W. Glover. On groups whose orders are products of three prime factors. *Amer. J. Math.*, 15:191–220, 1893.

[11] Gabrielle A. Dickenson. On the enumeration of certain classes of soluble groups. *Quart. J. Math. Oxford Ser. (2)*, 20:383–394, 1969.

[12] Leonard Eugene Dickson. *Linear groups: With an exposition of the Galois field theory*. with an introduction by W. Magnus. Dover Publications Inc., New York, 1958.

[13] Marcus du Sautoy. Counting $p$-groups and nilpotent groups. *Inst. Hautes Études Sci. Publ. Math.*, (92):63–112 (2001), 2000.

[14] Marcus du Sautoy. Zeta functions of groups: the quest for order versus the flight from ennui. In *Groups St. Andrews 2001 in Oxford. Vol. I*, volume 304 of *London Math. Soc. Lecture Note Ser.*, pages 150–189. Cambridge Univ. Press, Cambridge, 2003.

[15] Bettina Eick and E. A. O'Brien. Enumerating $p$-groups. *J. Austral. Math. Soc. Ser. A*, 67(2):191–205, 1999. Group theory.

[16] Joseph A. Gallian and James Van Buskirk. The number of homomorphisms from $Z_m$ into $Z_n$. *Amer. Math. Monthly*, 91(3):196–197, 1984.

[17] J. A. Green. The characters of the finite general linear groups. *Trans. Amer. Math. Soc.*, 80:402–447, 1955.

[18] J. A. Green. *Polynomial representations of* $\mathrm{GL}_n$, volume 830 of *Lecture Notes in Mathematics*. Springer-Verlag, Berlin, 1980.

[19] Marshall Hall, Jr. and James K. Senior. *The groups of order* $2^n$ $(n \leq 6)$. The Macmillan Co., New York, 1964.

[20] Graham Higman. Enumerating $p$-groups. I. Inequalities. *Proc. London Math. Soc. (3)*, 10:24–30, 1960.

[21] Graham Higman. Enumerating $p$-groups. II. Problems whose solution is PORC. *Proc. London Math. Soc. (3)*, 10:566–582, 1960.

[22] Peter John Hilton and Urs Stammbach. *A course in homological algebra*. Springer-Verlag, New York, 1971. Graduate Texts in Mathematics, Vol. 4.

[23] Otto Hölder. Die Gruppen der Ordnungen $p^3$, $pq^2$, $pqr$, $p^4$. 43:301–412, 1893.

[24] Derek F. Holt, Bettina Eick, and Eamonn A. O'Brien. *Handbook of computational group theory*. Discrete Mathematics and its Applications (Boca Raton). Chapman & Hall/CRC, Boca Raton, FL, 2005.

[25] Nathan Jacobson. *Lectures in abstract algebra. Vol. II. Linear algebra*. D. Van Nostrand Co., Inc., Toronto-New York-London, 1953.

[26] Rodney James. The groups of order $p^6$ ($p$ an odd prime). *Math. Comp.*, 34(150):613–637, 1980.

[27] Adalbert Kerber. *Applied finite group actions*, volume 19 of *Algorithms and Combinatorics*. Springer-Verlag, Berlin, second edition, 1999.

[28] Peter Lancaster and Miron Tismenetsky. *The theory of matrices*. Computer Science and Applied Mathematics. Academic Press Inc., Orlando, FL, second edition, 1985.

[29] Rudolf Lidl and Harald Niederreiter. *Finite fields*, volume 20 of *Encyclopedia of Mathematics and its Applications*. Addison-Wesley Publishing Company Advanced Book Program, Reading, MA, 1983. With a foreword by P. M. Cohn.

[30] Dudley E. Littlewood. *The Theory of Group Characters and Matrix Representations of Groups*. Oxford University Press, New York, 1940.

[31] Saunders Mac Lane. *Categories for the working mathematician*. Springer-Verlag, New York, 1971. Graduate Texts in Mathematics, Vol. 5.

[32] I. G. Macdonald. *Symmetric functions and Hall polynomials*. The Clarendon Press Oxford University Press, New York, 1979. Oxford Mathematical Monographs.

[33] I. G. Macdonald. Numbers of conjugacy classes in some finite classical groups. *Bull. Austral. Math. Soc.*, 23(1):23–48, 1981.

[34] Marvin Marcus. *Finite dimensional multilinear algebra. Part 1.* Marcel Dekker Inc., New York, 1973. Pure and Applied Mathematics, Vol. 23.

[35] Marvin Marcus. *Finite dimensional multilinear algebra. Part II.* Marcel Dekker Inc., New York, 1975. Pure and Applied Mathematics, Vol. 23.

[36] Annabelle McIver and Peter M. Neumann. Enumerating finite groups. *Quart. J. Math. Oxford Ser. (2)*, 38(152):473–488, 1987.

[37] Eugen Netto. *Substitutionentheorie und ihre Anwendungen auf die Algebra*. Teubner, Leipzig, 1882.

[38] Peter M. Neumann. An enumeration theorem for finite groups. *Quart. J. Math. Oxford Ser. (2)*, 20:395–401, 1969.

[39] M. F. Newman. Determination of groups of prime-power order. In *Group theory (Proc. Miniconf., Australian Nat. Univ., Canberra, 1975)*, pages 73–84. Lecture Notes in Math., Vol. 573. Springer, Berlin, 1977.

[40] M. F. Newman, E. A. O'Brien, and M. R. Vaughan-Lee. Groups and nilpotent Lie rings whose order is the sixth power of a prime. *J. Algebra*, 278(1):383–401, 2004.

[41] E. A. O'Brien. The $p$-group generation algorithm. *J. Symbolic Comput.*, 9(5-6):677–698, 1990. Computational group theory, Part 1.

[42] E. A. O'Brien and M. R. Vaughan-Lee. The groups with order $p^7$ for odd prime $p$. *J. Algebra*, 292(1):243–258, 2005.

[43] Eamonn A. O'Brien. Bibliography on the determination of finite groups. Available at `http://www.math.auckland.ac.nz/ obrien/research/bibliography.pdf`.

[44] L. Pyber. Enumerating finite groups of given order. *Ann. of Math. (2)*, 137(1):203–220, 1993.

[45] L. Pyber. Group enumeration and where it leads us. In *European Congress of Mathematics, Vol. II (Budapest, 1996)*, volume 169 of *Progr. Math.*, pages 187–199. Birkhäuser, Basel, 1998.

[46] Ronald C. Read. Every one a winner or how to avoid isomorphism search when cataloguing combinatorial configurations. *Ann. Discrete Math.*, 2:107–120, 1978. Algorithmic aspects of combinatorics (Conf., Vancouver Island, B.C., 1976).

[47] Charles C. Sims. Enumerating $p$-groups. *Proc. London Math. Soc. (3)*, 15:151–166, 1965.

[48] Charles C. Sims. *Computation with finitely presented groups*, volume 48 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, Cambridge, 1994.

[49] Richard P. Stanley. *Enumerative combinatorics. Vol. 1*, volume 49 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, Cambridge, 1997. With a foreword by Gian-Carlo Rota, Corrected reprint of the 1986 original.

[50] R. Steinberg. A geometric approach to the representations of the full linear group over a Galois field. *Trans. Amer. Math. Soc.*, 71:274–282, 1951.

[51] J.W.A. Young. On the determination of groups whose order is a power of a prime. *Amer. J. Math.*, 15:124–178, 1893.

# Index