Constructive recognition of black-box $F_4(q)$ in odd characteristic

by

Ian Gregor Dick

Submitted in partial fulfilment of the requirements of the Degree of
Doctor of Philosophy

School of Mathematical Sciences
Queen Mary, University of London
United Kingdom

July 2015

Statement of Originality

I, Ian Gregor Dick, confirm that the research included within this thesis is my own work or that where it has been carried out in collaboration with, or supported by others, that this is duly acknowledged below and my contribution indicated. Previously published material is also acknowledged below.

I attest that I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge break any UK law, infringe any third party's copyright or other Intellectual Property Right, or contain any confidential material.

I accept that the College has the right to use plagiarism detection software to check the electronic version of the thesis.

I confirm that this thesis has not been previously submitted for the award of a degree by this or any other university.

The copyright of this thesis rests with the author and no quotation from it or information derived from it may be published without the prior written consent of the author.

Signature:

Date: 04 July 2014

Details of collaboration and publications:

No part of this thesis has been published elsewhere. Chapter 1 contains an exposition of relevant material already in the literature; otherwise, except where noted, the thesis is entirely my own work, conducted under the supervision of Prof. Robert Wilson.

Abstract

Let G be a group, and let $\widehat{G}$ be a group isomorphic to G. The *constructive recognition problem* for G with respect to $\widehat{G}$ is to find an isomorphism ϕ from G to $\widehat{G}$ such that the images under ϕ and ϕ^{-1} of arbitrary elements may be computed efficiently. If the representation of $\widehat{G}$ is well-understood, then the representation of G becomes likewise by means of the action of ϕ.

The problem is of foundational importance to the *computational matrix group project* in its ambitious desire to find an algorithm to construct a composition series for an arbitrary matrix group over a finite field. This requires algorithms for the constructive recognition of all finite simple groups, which exist in the literature in varying degrees of practicality. Those for the exceptional groups of Lie type admit of improvement, and it is with these that we concern ourselves.

Kantor and Magaard in [31] give Monte Carlo algorithms for the constructive recognition of black-box (i.e. opaque-representation) exceptional groups other than ${}^2F_4(2^{2n+1})$. These run in time exponential in the length of the input at several stages. We specialise to the case of $F_4(q)$ for odd q, and in so doing develop a polynomial-time alternative to the preprocessing stage of the Kantor–Magaard algorithm; we then modify the procedure for the computation of images under the recognising isomorphisms to reduce this to polynomial running time also. We provide a prototype of an implementation of the resulting algorithm in MAGMA [10].

Fundamental to our method is the construction of involution centralisers using Bray's algorithm [11]. Our work is complementary to that of Liebeck and O'Brien [40] which also uses involution centralisers; we make a comparison of the two approaches.

Acknowledgments

Ad majorem Dei gloriam.

In the first place I wish to thank my supervisor, Robert Wilson, for his patient guidance over the course of the past four years. His ready offering of his wisdom and experience re-orientated me again and again at times when I would otherwise have floundered. I likewise thank John Bray for supplying in times of Rob's absence, and, together with Ivan Tomašić, for offering salutary and sobering advice on the realities of the writing-up process.

No less do I thank my wife, Alisa, for her good-natured exhortations to persistence and diligence in my research, and without whom I suspect I would have wearied of the whole enterprise. On the other hand, I must also thank our daughter, Judith, whose cheerful indifference to group theory, research and all grown-up matters ensured that I maintained a healthy perspective. To my parents I owe the irredeemable debt of their sacrifices and encouragement throughout my whole life, and for this I am most grateful.

There are many other people at Queen Mary to whom I owe my thanks. They are too numerous to list here, but I wish to mention Charles Leedham-Green who, although we spoke only occasionally, was generous with his expertise and unfailingly enthusiastic; and I also thank Yue Gao of the School of Electronic Engineering and Computer Science for the LaTeX class that formed the basis of that used to produce this document, and which saved me some considerable drudgery. I would also like to thank my contemporaries as research students in the School of Mathematical Sciences: the good-willed spirit that always prevailed amongst us as a group helped to make my time at Queen Mary a most enjoyable one.

London

On the vigil of Ss. Peter and Paul, 2014

This work was supported by the Engineering and Physical Sciences Research Council grant number MTHA1F6R, for which the author is very grateful.

Table of Contents

Chapter 1

Introduction

After stating some preliminary definitions, we will state the problem in which we are interested. Subsequently, we shall situate this problem within the wider context of computational group theory, and then proceed to survey existing work on this and closely-related problems. We shall conclude the chapter with a collection of results, for the most part straightforward, that will be useful in the sequel.

1.1 Preliminaries

We first define the notation that we will use throughout this thesis, and then proceed to review some of the standard properties of groups of Lie type that will be of use.

1.1.1 Notation and conventions

'Group' should always be understood to mean 'finite group'. For g and h in a group G, we will as usual denote left-conjugation by ${}^h g := hgh^{-1}$ and right-conjugation by $g^h := h^{-1}gh$. We define the commutator $[g,h]$ to be $g^{-1}h^{-1}gh$.

The groups denoted by the *names* of linear, classical and exceptional groups (for example,

$\mathrm{GL}_n(q)$ or $F_4(q)$) are the corresponding groups of matrices in the appropriate natural representation in the case of classical groups, or the finitely-presented group specified by the reduced Curtis–Steinberg–Tits presentation in the case of exceptional groups, as discussed below. Used as noun adjuncts, however, these names indicate *isomorphism* with the appropriate group: it will often be convenient to speak of the '$\mathrm{SL}_2(q)$-subgroups' of a given group, meaning its subgroups isomorphic to $\mathrm{SL}_2(q)$.

For an integer n and a prime p, we will denote by $\{n\}_p$ the highest power of p dividing n, and then define $\{n\}_{p'_1,\ldots,p'_m}$ to be $\frac{n}{\{n\}_{p_1}\ldots\{n\}_{p_m}}$ for any primes $p_1, \ldots, p_m$.

We will consistently let $q = p^e$ to be a prime power, but both p and e may vary.

1.1.2 Elementary definitions

In this section, we collect some definitions pertaining to finite simple groups in general, following [59].

Definition 1.1. A group G is *almost simple* if there exists a simple group S such that $S < G \leq \mathrm{Aut}(S)$.

Definition 1.2. A p-group G such that $G = G' = \Phi(G)$, where $\Phi(G)$ is the Frattini subgroup of G, is *special*. If, furthermore, $|Z(G)| = p$, then G is *extraspecial*.

Definition 1.3. An $n \times n$ matrix T is a *transvection* if $N := T - I_n$ has rank 1 and $N^2 = 0$.

It is a well-known result that $\mathrm{SL}_n(q)$ is generated by transvections. Similar results can be formulated for other classical groups. The concept of a transvection will be generalised below in Lie-theoretic terms to give the notion of a *root element*; these elements will be fundamental in our attempts to find generating sets for groups of Lie type.

1.1.3 Chevalley groups

The theory of the finite groups of Lie type is given a thorough and lucid treatment in [21]. We defer to that text on this subject, except to give an overview of some of the structural features of these groups that will concern us directly.

The finite *Chevalley groups* arise as subgroups of the automorphism groups of discrete analogues of the simple complex Lie algebras. Much of the apparatus of the Lie algebra, in particular its associated root system, encodes structural properties of the corresponding Chevalley group.

Let G be such a group with associated root system Φ and defined over the field $\mathbb{F}_q$. We will work with such groups frequently. Then G contains a system $\mathscr{X} = \{X_\phi\}_{\phi\in\Phi}$ of subgroups that together generate G. The subgroups, and all their conjugates in G, are the *root (sub)groups* of G, which it is sometimes helpful to distinguish as being *long* or *short* according to the length of the corresponding root in Φ. Each X_ϕ is isomorphic to the additive group of $\mathbb{F}_q$ (and so is an elementary abelian group of exponent p), and hence an element in an X_ϕ may be specified by an element t of $\mathbb{F}_q$, and denoted by $X_\phi(t)$. This labelling is not arbitrary, and arises from the representation of the group as Lie algebra automorphisms. We shall shortly describe the interaction of these elements in terms of their labels.

It should be emphasised at this point that neither the labelling of the elements of the root groups nor the assignment of a root in Φ to each root group is unique: pairs of different labellings induce automorphisms of G.

Two root groups X and Y are *opposite* if $\langle X, Y\rangle$ is isomorphic to $\mathrm{SL}_2(q)$; in particular, $X_{\pm\phi}$ are opposite for every $\phi \in \Phi$.

The elements $n_\phi(t) := X_\phi(t)X_{-\phi}(-t^{-1})X_\phi(t)$ generate a subgroup N of G acting on $\mathscr{X}$ by conjugation as the Weyl group of Φ, such that ${}^{n_\psi(t)}X_\phi = X_{w_\psi(\phi)}$, where w_ψ is the reflection through the hyperplane orthogonal to ψ.

We also define elements $h_\phi(t) := n_\phi(t)n_\phi(-1)$. The group H generated by these elements as ϕ ranges over the whole of Φ, together with all of the G-conjugates of H, are the *maximal split tori* of G, and are isomorphic to a direct product of a number of copies of the multiplicative group of $\mathbb{F}_q$ equal to the dimension of Φ. More generally, the *maximal tori* of G are the fixed-point subgroups of the maximal split tori of the overgroups of G induced by extending $\mathbb{F}_q$, under the action of the corresponding Frobenius automorphism φ (acting on the root groups by mapping $X_\phi(t)$ to $X_\phi(\varphi(t))$ for each $\phi \in \Phi$).

The last properties with which we are concerned are the following interactions of root elements (and other elements derived from them), where in each case $\phi, \psi \in \Phi$ and $t, u \in \mathbb{F}_q$:

$$ {}^{n_\psi(u)}X_\phi(t) = X_{w_\psi(\phi)}(\eta_{\psi,\phi}u^{-A_{\psi,\phi}}t) \tag{1.1} $$

$$ {}^{h_\psi(u)}X_\phi(t) = X_\phi(u^{A_{\psi,\phi}}t). \tag{1.2} $$

We also have the *Chevalley commutator relations:*

$$ [X_\psi(u), X_\phi(t)] = \prod_{\substack{i,j>0 \\ i\phi+j\psi\in\Phi}} X_{i\phi+j\psi}(C_{i,j,\phi,\psi}(-t)^i u^j), \tag{1.3} $$

where the product is taken in the order of increasing $i + j$, and the $A_{\psi,\phi}$, $\eta_{\psi,\phi}$ and $C_{i,j,\phi,\psi}$ are *structure constants* depending on Φ. In all practical settings we assume these to have been precalculated.

The *Curtis–Steinberg–Tits* presentation of G has as its generators the root elements and the elements $h_\phi(t)$, and as its relations those given in (1.2) and (1.3), together with the defining properties of the root elements. Full details are given in [21, p. 190]. This presentation contains much redundancy, and shorter presentations that are better suited to algorithmic purposes are given in [6].

The finite simple groups of Lie type that are not Chevalley groups arise as *twisted* groups, which are subgroups G of Chevalley groups $\tilde{G}$ induced by symmetries of the Dynkin diagram for $\tilde{G}$. We will not perform any calculations inside the twisted groups and so we do not explain

their structure further.

1.2 Group recognition

We now state the general case of the main problem in which we are interested. Let G be a group generated by a set X, and let $\widehat{G}$ be a group isomorphic to G and generated by a set $\widehat{X}$. We seek an *explicit* or *computable* isomorphism $\Psi : \widehat{G} \to G$, which is to say that Ψ must come equipped with algorithms for calculating images and preimages of arbitrary elements.

We call $\widehat{G}$ the *standard copy of* G and $\widehat{X}$ its *standard generators*. Typically these will be chosen so that the abstract group structure of the former is well understood in terms of the latter: when G is a group of Lie type, normally $\widehat{G}$ is the (shortened) Curtis–Steinberg–Tits presentation, or the natural representation in the case of classical groups, with the generators $\widehat{X}$ being a (basis over $\mathbb{F}_p$ of a) complete set of root elements. On the other hand, very few assumptions will be made about the representation of G and the particular choice of X (cf. Section 1.4 below). In this case, the problem of finding Ψ is the *constructive recognition problem* for G.

In order to choose $\widehat{G}$ we must of course know the isomorphism class of G. The process of determining this is *non-constructive recognition* of G, which according to taste may be considered to be either a sub-problem of or a distinct problem to the constructive recognition of G. A survey of the topic (as of 2005) is given in [48, Section 6]; in brief, algorithms exist for the non-constructive recognition of classical groups in their natural representations (e.g. in [45], [47]) and for black-box groups of Lie type, which we shall define later (where the algorithms appear e.g. in [8]). Subsequently, there has been further work on the prerequisite problem of determining the characteristic [1] of a black-box Lie-type group by Kantor and Seress [37] and Lytkin [43].

For our purposes, the most important feature of these algorithms is that they run in time

[1] Or the characteristic of *some* representation in the case of small groups exhibiting exceptional isomorphisms.

polynomial in the length of the input. Since it will transpire that this is at least as fast (asymptotically speaking) as the constructive algorithms with which we shall be working, we shall always assume that the isomorphism class of our input group G is known *a priori*. We will not now return to the problem of non-constructive recognition, and henceforth, 'recognition' should be understood to mean 'constructive recognition'.

Constructive recognition has several identifiable sub-problems. The first is to find images of the elements of $\widehat{X}$ under Ψ. This identifies elements of G with known group-theoretic properties, which one might hope to exploit to write arbitrary elements of G (given as words in X) as words in these images. It then remains only to provide an algorithm for writing elements of $\widehat{G}$ as words in $\widehat{X}$ for Ψ to be computable.

Before surveying the present state of the art, we situate the problem in the wider context of computational group theory, and then consider the assumptions that we might make about the representation of the input group G.

1.3 The computational matrix group project

The computational matrix group project seeks to develop algorithms for solving group-theoretic problems — for example, to find Sylow subgroups — posed on matrix representations of groups. Surveys of the project are given in [38] and (subsequently) in [48]. Both articles discuss a range of problems that fall under the umbrella of the project, including group recognition. Another of these is the construction of a *composition tree*, which we will now examine as it serves to motivate the present work.

A composition tree for a group G is a binary tree with groups as nodes. The root node is G. Non-leaf nodes labelled with some group H have as their left child a normal subgroup N of H, together with a membership test for N in H; as right child, they have H/N, with an epimorphism $H \to N$. A node is a leaf if and only if it is simple — these, of course, being the composition factors of the group in the usual sense — and such nodes are equipped with an

isomorphism with a standard copy of the group. Kantor in [32] gives an algorithm for finding the Sylow subgroups of a group, given its composition tree.

The production of a composition tree for a given group, then, requires algorithms for the recognition of each of its composition factors. This also allows a useful reduction of the general recognition problem to that for simple groups only (although it will prove to be profitable also to consider the covers of such when they exist).

There are broadly two approaches to the design of algorithms for computing with matrix groups. The first of these is the *geometric approach*, which might be said to belong most closely to the computational matrix group project. We postpone discussion of the second, the *black-box approach*, until the next section.

The geometric approach relies on Aschbacher's generic classification of the maximal subgroups of linear groups, which can be stated in outline thus:

Theorem 1.4 (Aschbacher, [2])**.** *Let G be a maximal subgroup of* $\mathrm{GL}(d,q)$ *acting linearly on a d-dimensional $\mathbb{F}_q$-vector space V. Then at least one of the following holds:*

$\mathscr{C}_1$. *G acts reducibly on V.*

$\mathscr{C}_2$. *$d = d'n$ and G acts imprimitively on V, with blocks determined by a decomposition of V into a sum of n subspaces of dimension d'.*

$\mathscr{C}_3$. *$d = d'n$ and G embeds in $\Gamma L(d', q^n)$.*

$\mathscr{C}_4$. *$d = mn$ and G preserves a tensor decomposition $V = U \otimes W$, with* $\dim(U) = n$ *and* $\dim(W) = n$.

$\mathscr{C}_5$. *Modulo scalars, G may be written over a subfield of $\mathbb{F}_q$.*

$\mathscr{C}_6$. *$d = r^m$ for some prime r dividing $q-1$, and G normalises either an extraspecial subgroup r^{1+2m}, or, if $r = 2$, a symplectic-type subgroup of order 2^{2+2m}.*

$\mathscr{C}_7$. *$d = d'^n$ and G acts imprimitively on a homogeneous tensor decomposition $V = \bigotimes_n U$,*

with $\dim(U) = d'$.

$\mathscr{C}_8$. *G normalises the natural representation of a classical group inside* $\mathrm{GL}(d,q)$.

$\mathscr{C}_9$. *G is almost simple modulo scalars.*

The labels in each case are the customary names for the classes of maximal subgroups (which are not disjoint). $\mathscr{C}_9$ is sometimes denoted by $\mathscr{S}$.

If an Aschbacher category can be identified for a subgroup of a linear group, then in most cases the corresponding geometric structure that is preserved induces a normal subgroup (for example, in $\mathscr{C}_1$, these are the centralisers of the sections of V induced by the reduction of the action). Recursive application of this process can then, in favourable circumstances, constitute one ingredient in the process of finding a composition tree for the group. Strategies for identifying the Aschbacher class(es) of a group are discussed in the surveys by Leedham-Green and O'Brien already mentioned.

Such a strategy still leaves the problem of recognising the composition factors. It is possible, when dealing with matrix groups, to take a geometric approach to this, also: one might, for example, wish to find the eigenspaces of a particular element in the course of recognising the ambient groups. Such a strategy has resulted in a range of algorithms for recognising matrix representations of classical groups. As some of these will prove to be useful for our current purposes, we examine them in some detail in Section 1.5.

1.4 Black-box group algorithms

An alternative approach to the development of group-theoretic algorithms adopts the principle that no representation-dependent properties of any input groups are to be used. To this end, Babai and Szemerédi [9] introduced the notion of a *black-box* representation of a group, whereby elements of a group are represented by uniform-length strings of bits, and functions are available to compute the product of two elements, to invert elements and to test whether

a given element is the identity element. Each of these functions is assumed to be computed in constant time. An algorithm using no more than these properties of its input groups is said to be a *black-box algorithm*.

It is common also to assume that an *oracle* is available that computes the order of an element, or some multiple thereof. We discuss this matter further in Section 1.4.3.

We now consider some existing black-box algorithms that will be of use.

1.4.1 Generating random elements

As we shall see, it is possible to generate elements at (pseudo-)random in a black-box group with a distribution arbitrarily close to being uniform. Given the limited array of operations permitted on black-box groups, this is a particularly (but not uniquely: see the discussion in Section 1.5.3.5 of Kantor and Kassabov's algorithm) useful foundation for other black-box algorithms. Such algorithms are of course then probabilistic, most frequently of Monte Carlo or Las Vegas type, in the senses defined in Section 1.5.1.

1.4.1.1 The product replacement algorithm

The *product replacement algorithm*, due to Leedham-Green and Soicher and published in [23], generates group elements at pseudo-random. Given a pseudo-random number generator and a group specified by a set of generators, the algorithm begins with a k-tuple of generators for the group and perturbs it t times successively by multiplying one element by another (or its inverse). A random selection from the tuple then provides the random element. The algorithm is analysed in [50], with special attention being paid to the choice of the parameters k and t, and is implemented in MAGMA; we use it whenever we require a random group element.

1.4.2 Finding involution centralisers

Bray in [11] gives an algorithm for generating the centraliser of an involution in a black-box group, extending an earlier algorithm due to Richard Parker. It constructs the elements of the centraliser specified by the following result:

Proposition 1.5 (Bray, [11])**.** *Let G be any group and let $x \in G$ be an involution; let $g \in G$ be arbitrary and let $c = [x, g]$. If c has even order $2n$, then c^n and $(c^n)^{g^{-1}}$ commute with x [this is essentially the principle underlying Parker's algorithm]. If instead c has odd order $2n+1$, then gc^n commutes with x.*

Proof. If c has order $2n$, then

$$xc^n = x(xg^{-1}xg)^n = x(g^{-1}xgx)^n = (xg^{-1}xg)^n x = c^n x$$

and

$$x(c^n)^{g^{-1}} = x(gxg^{-1}x)^n = (xgxg^{-1})^n x = (c^n)^{g^{-1}} x.$$

If c has order $2n+1$, then

$$xgc^n = xg(xg^{-1}xg)^n = xg(g^{-1}xgx)^{n+1} = gx(g^{-1}xgx)^n = g(xg^{-1}xg)^n x = gc^n x.$$

□

We use this algorithm, for which we provide an implementation, to find involution centralisers in $F_4(q)$, as described in Section 2.2.2. The probability of doing so successfully is analysed in Section 2.1.

1.4.3 Orders and pseudo-orders

The naïve algorithm for calculating the order of an element $g \in G$, viz. the evaluation of g^n for successive values $2 \leq n \leq |G|$ until the identity results, is certainly possible in a black-box group but is utterly impractical, having a complexity of $O(|G|)$, supposing that g^{n+1} is calculated by evaluating $g^n.g$. If the exponent of c of G is known we may replace $|G|$ by c, but asymptotically this is not in general any better. Then there is the standard trick, useful in itself, of evaluating a given power n of g by calculating g^{2^m} for $0 \leq m \leq n$ and observing that g^n is a product of a subset of these determined by the binary representation of n; but this is not applicable to the recursive evaluation of g^n, and used to calculate the powers directly it only achieves a complexity of $O(|G|\log|G|)$. These methods are all a long way from our desired $O(\log|G|)$ running time.

In [22], Celler and Leedham-Green consider the problem of calculating the order of an element of a *matrix* group of dimension d defined over $\mathbb{F}_q$, and give an algorithm for doing so with time complexity $O(d^3 \log q)$, with the factor of d^3 resulting from the cost of matrix multiplication, if the prime factorisation of $q^i - 1$ is known for each $i \leq d$. They also remark that if, instead of such prime factorisations, factorisations into pairwise coprime factors p_j (rather than into prime factors) are known, their algorithm yields a *pseudo-order* of g, i.e. the least integer l such that g^l is the identity and l is a product of the p_j (perhaps with repetitions). Babai and Beals in [7] formulate the notion of pseudo-orders for black-box group elements, and this will form the basis of our own approach. We implement the algorithm ORDER that they describe, using the divide-and-conquer optimisation set out in [22]. Further techniques for calculating pseudo-orders are described in [58].

Care is required when using pseudo-orders in place of genuine orders that the possible extra factors that arise do not interfere with any algorithmic assumptions. An informed choice of the "pretend primes" p_j can be helpful in this regard. We discuss such matters in their relation to our own work in the appropriate places in Chapter 4.

1.5 Existing recognition algorithms for linear and classical groups

We survey here the algorithms for constructive recognition that have already been developed. The nature of the problem is such that the recognition of one group of Lie type typically reduces in essence to the recognition of Lie-type subgroups of smaller rank, especially those isomorphic to $\mathrm{SL}_3(q)$ or to $\mathrm{SL}_2(q)$. The separate algorithms tend to rely on one another, then, and indeed for our own use in recognising $F_4(q)$ we will require algorithms for recognising several low-rank classical and linear groups, to be discussed in due course.

1.5.1 Preliminaries

There are some common features of the algorithms that bear mention. Perhaps the most important of these is the use of *oracles*, by which we mean routines for solving some subproblem or performing a particular sort of calculation. In this category we might place the routines for performing group operations or generating random group elements, but more significantly, oracles can be defined so as to isolate computationally difficult parts of the algorithms. In some of the algorithms below, oracles of this sort will be assumed to exist for the recognition of $\mathrm{SL}_2(q)$ or for calculating discrete logarithms. Thus it will often be the case that a certain algorithm runs in time polynomial in the size of its input *but for the cost of invoking the* $\mathrm{SL}_2(q)$*-recognition oracle*. Where oracles are required in the following discussion, this will be made clear.

Recall that an algorithm for solving a decision problem is *Monte Carlo* if its running time is bounded and its result is incorrect with non-zero probability. A *one-sided Monte Carlo* algorithm has such a probability of error in only one or other of its results (i.e. either when it returns *true* or when it returns *false*). In contrast, a *Las Vegas* algorithm runs in unbounded time (or eventually fails), but when it produces a result, it does so with certainty.

With the exception of the Kantor–Kassabov algorithm, all of the algorithms below are randomised. In each case they can be arranged to be one-sided Monte Carlo or Las Vegas, but

the latter is preferred as corresponding implementations are more useful. In practice the random generation of elements will be performed using the product replacement algorithm, discussed in Section 1.4, and, since it may be taken to run in constant time, its contribution to the complexity of the algorithms below is ignored. For the same reason, we do likewise to the running times of the group-operation oracles.

1.5.2 Matrix algorithms

We turn our attention first to algorithms that take as input matrix representations of groups, whether in their natural representations or in others, but always in defining characteristic.

1.5.2.1 $\mathrm{SL}_2(q)$ in its natural representation

Conder and Leedham-Green in [26] give an algorithm for determining whether a group G of 2×2 unit-determinant matrices over $\mathbb{F}_q$ is the whole of $\mathrm{SL}_2(q)$, and recognising it if so. It searches for transvections in G by attempting to construct elements with a shared eigenvector, and then taking their commutator. It assumes the availability of a discrete logarithm oracle.

Theorem 1.6 (Conder–Leedham-Green, [26])**.** *Subject to the availability of a discrete logarithm oracle, there is a Las Vegas algorithm for recognising* $\mathrm{SL}_2(q)$ *in its natural representation. It runs in* $O(\log q)$ *time.*

1.5.2.2 $\mathrm{SL}_2(q)$ and $\mathrm{PSL}_2(q)$ in arbitrary characteristic-p representations

In [25], Conder, Leedham–Green and O'Brien provide an algorithm for constructing the natural module for a characteristic-p matrix group G that is isomorphic modulo scalars to $\mathrm{PSL}_2(q)$, and thus reduces the problem so that Theorem 1.6 can be applied. It does so by first reducing (using the MeatAxe [52, 30]) to an irreducible representation if necessary, and then expressing this, using the Steinberg twisted tensor product theorem [57], as a tensor product of

symmetric products of copies of the natural representation, and then decomposing it.

Theorem 1.7 (Conder *et al.*, [25])**.** *Subject to the availability of a discrete logarithm oracle, there is a Las Vegas algorithm for recognising* $\mathrm{SL}_2(q)$ *and* $\mathrm{PSL}_2(q)$ *(for known q) in matrix representations of degree d in defining characteristic. It runs in $O(d^5\tau(d)\log(q))$ time, where $\tau(d)$ is the number of divisors of d.*

1.5.2.3 Classical groups in their natural representations

Brooksbank in [14] gives an algorithm that, for an input group G generated by $d \times d$ matrices over $\mathbb{F}_q$, determines whether G is isomorphic to $\mathrm{Sp}_{2d}(q)$, $\mathrm{SU}_d(\sqrt{q})$ or to one of $\Omega^{\epsilon}_d(q)$, and if so, constructs an effective isomorphism between G and the standard copy of the group. It relies on Theorem 1.6 to recognise $\mathrm{SL}_2(q)$-subgroups. Its strategy is to construct elements allowing the performance of a form-preserving analogue of Gaussian elimination.

Theorem 1.8 (Brooksbank, [14])**.** *Subject to the availability of a discrete logarithm oracle, there is a Las Vegas algorithm for recognising the natural representations (when they are defined) of* $\mathrm{Sp}_{2d}(q)$, $\mathrm{SU}_d(q)$ *and* $\Omega^{\epsilon}_d(q)$ *for* $\epsilon \in \{0,-,+\}$*. It runs in* $O(\log^2 q)$ *time.*

1.5.3 Black-box algorithms

We now consider relevant algorithms for recognising black-box representations of groups of Lie type. It should be remarked at this point that, if the isomorphism produced by such an algorithm from the standard copy of the group is to be allowed to take an arbitrary matrix in the natural representation as input (rather than one given necessarily as a word in the standard generators), then the algorithm must also solve the natural-representation recognition problem explicitly.

1.5.3.1 The Kantor–Seress algorithms for black-box classical groups

Kantor and Seress gives algorithms in [34] for recognising $\mathrm{SL}_d(q)$, $\mathrm{Sp}_{2d}(q)$, $\mathrm{SU}_d(q)$, $\Omega^\epsilon_d(q)$ and their central quotients. Their running times are polynomial in q. There are now algorithms in the literature that improve upon each of these, and so we do not examine them in detail, but they do constitute an important milestone in the development of black-box recognition algorithms.

1.5.3.2 (P)$\mathrm{SL}_d(q)$ and (P)$\mathrm{Sp}_{2d}(q)$ in black-box representations

In [17], Brooksbank and Kantor improve on the algorithm in [34] for recognising $\mathrm{SL}_d(q)$ and its homomorphic images by eliminating factors of q from the running time. Indeed they remove all such factors besides any incurred by calling an $\mathrm{SL}_2(q)$-recognition oracle. They also outline an analogous modification to the algorithm of [34] for recognising $\mathrm{Sp}_{2d}(q)$ and $\mathrm{PSp}_{2d}(q)$.

Theorem 1.9 (Brooksbank–Kantor, [17])**.** *Subject to the availability of an oracle for recognising a black-box* $\mathrm{SL}_2(q)$*, there is a Las Vegas algorithm for recognising black-box groups isomorphic to* $\mathrm{SL}_d(q)$ *and* $\mathrm{PSL}_d(q)$ *for* $d \geq 3$ *and* $q \geq 17$*. As a function of* q*, its running time is polynomial in* $\log q$*, but for any dependence on* q *arising from the* $\mathrm{SL}_2(q)$ *oracle.*

Theorem 1.10 (*ibid.*)**.** *Subject to the availability of an oracle for recognising a black-box* $\mathrm{SL}_2(q)$*, there is a Las Vegas algorithm for recognising black-box groups isomorphic to* $\mathrm{Sp}_{2d}(q)$ *and* $\mathrm{PSp}_{2d}(q)$ *for* $2d \geq 4$ *and* $q \geq 9$*. As a function of* q*, its running time is polynomial in* $\log q$*, but for any dependence on* q *arising from the* $\mathrm{SL}_2(q)$ *oracle.*

1.5.3.3 $\mathrm{SL}_3(q)$ and $\mathrm{PSL}_3(q)$ in black-box representations

Lübeck, Magaard and O'Brien give an algorithm in [41] for recognising black-box representations of $\mathrm{SL}_3(q)$ and $\mathrm{PSL}_3(q)$. As the authors themselves note, this algorithm represents an

improvement upon Theorem 1.9 in the case when $d = 3$ in that it can handle smaller fields, and in that implementations are available in GAP and MAGMA. For such an input group G, it functions by constructing several subgroups of G isomorphic to $\mathrm{SL}_2(q)$ and recognising these using an oracle. The recognising isomorphisms are then adjusted to be mutually compatible, so as to map root groups in the natural $\mathrm{SL}_2(q)$ to root groups in G with respect to a common torus. A similar theme will occupy us in the recognition of $\mathrm{F}_4(q)$ later. The factor of q in the running time for the algorithm is due to the required $\mathrm{SL}_2(q)$-recognitions.

Theorem 1.11 (Lübeck *et al.*, [41])**.** *Subject to the existence of an* $\mathrm{SL}_2(q)$*-recognition oracle, there is a Las Vegas algorithm for recognising black-box groups isomorphic to* $\mathrm{SL}_3(q)$ *and* $\mathrm{PSL}_3(q)$*, where* $q \geq 7$*. If* χ *is the cost of invoking the* $\mathrm{SL}_2(q)$ *oracle, it runs in* $O(\chi \log q + \log^2 q)$ *time. Straight-line programs are found in* $O(\chi + \log q)$ *time.*

1.5.3.4 Black-box classical groups

Brooksbank in [15], Brooksbank and Kantor in [18] and Brooksbank again in [16] give algorithms for the recognition respectively of $\mathrm{SU}_d(q)$, $\Omega^\epsilon(q)$ and $\mathrm{Sp}_{2d}(q)$, and their central quotients, in black-box representations. These algorithms improve on those in [34] by restricting complexity dependence on q to $\mathrm{SL}_2(q)$ recognition, as in [17] and [41]. However, the algorithms have the further advantage of avoiding recursion (and thus they improve on the symplectic-group recognition algorithm in [17]). Apropos of our present endeavour, we note that [16] constructs a certain subgroup of $\mathrm{Sp}_4(q)$ by using Bray's algorithm to find an involution centraliser.

Theorem 1.12 (Brooksbank, [15], [16]; Brooksbank–Kantor, [18])**.** *Subject to the availability of an oracle for recognising black-box groups isomorphic to* $\mathrm{SL}_2(q)$*, there are Las Vegas algorithms for recognising black-box representations (when they are defined) of* $\mathrm{Sp}_{2d}(q)$*,* $\mathrm{SU}_d(q)$ *and* $\Omega^\epsilon_d(q)$ *(for* $\epsilon \in \{0, -, +\}$*) and their central quotients. In each case, they run in time polynomial in* d *and* $\log q$*, discounting the cost of invoking the* $\mathrm{SL}_2(q)$ *oracle.*

1.5.3.5 A deterministic algorithm for recognising black-box $\mathrm{SL}_2(2^e)$

Kantor and Kassabov give a constructive recognition algorithm for black-box $\mathrm{SL}_2(2^e)$ in [33] that marks a departure from the techniques employed by the above algorithms. It is remarkable firstly in that it is deterministic, and also in that it is specific to characteristic 2, which in previous literature had been the more *difficult* case to handle. Perhaps its most significant feature as part of the wider group recognition project, however, is that it runs in time polynomial in the length of the input, and thus, in use as an oracle for the relevant algorithms above, reduces these to polynomially-complex algorithms also, when restricted to even-order fields.

The crucial observation on which the algorithm relies is that, for $h \in \mathrm{SL}(2,q)$ with $q > 2$ even, $1 \neq h \in \mathrm{SL}(2,q)$ of odd order and $g \in \mathrm{SL}(2,q)$ such that h and h^g do not commute, either $[h,h^g]$ or $(hh^g)^k h$ is an involution, where $2k+1 = q^2-1$. Thus (by using the generators of the group themselves as candidates for h and g), root elements can be found deterministically. The fundamental obstacle to an extension of the algorithm to odd characteristic p is the lack of an analogue of this result for finding elements of order p deterministically.

The full result is as follows:

Theorem 1.13 (Kantor–Kassabov, [33])**.** *There is a deterministic algorithm for recognising black-box groups isomorphic to* $\mathrm{SL}(2,q)$*, where* $q = 2^e$ *is even. It runs in* $O(e^3 \log e)$ *time. The resulting isomorphism can be used to compute straight-line programs for elements of the black-box group in* $O(e^3)$ *time.*

1.6 Recognising exceptional groups of Lie type

We now turn to the problem with which we are concerned. The previous section demonstrated that practical algorithms for recognising classical and linear groups are in ample supply in the literature. For the exceptional groups, however, the machinery is less thoroughly developed. There are three published (families of) algorithms to this end, all of which are

Las Vegas. Of these, we mention first that Bäärnhielm gives natural-representation recognition algorithms for the Ree and Suzuki groups in [5, 4, 3] and, jointly with Bray, for black-box Suzuki groups in [12]. We will not consider these groups any further.

In [31], Kantor and Magaard provide algorithms for recognising black-box representations of all of the exceptional groups *other* than the Suzuki and Ree groups. These algorithms have complexity involving a factor of the size q of the field. The present work seeks to avoid this linear dependence on q in the running time in certain cases by using the strategy suggested by those authors themselves in [31][Section 4, Remark 6], viz. by exploiting the structure of involution centralisers in odd characteristic. We will explore this in detail in due course.

Thirdly, and of obvious relevance, is the recent paper [40] of Liebeck and O'Brien, which gives algorithms for recognising exceptional groups (other than the Suzuki and Ree groups and ${}^3D_4(2^e)$) that run in polynomial time, and that use involution centralisers in an essential way. Their approach is rather different from that of [31], and we describe it in Section 1.6.2 for the purposes of comparison.

Throughout this section, G will denote the exceptional group that we wish to recognise.

1.6.1 General concepts in the recognition algorithms

We first introduce some of the ideas and tasks that are common to these (and indeed other) recognition algorithms.

A *subsystem subgroup* H of G is itself a group of Lie type, whether linear, classical or exceptional, being generated by a subset of the root elements of G. Given two subsystem subgroups H_1 and H_2 of G, a common concern will be whether there exist sets of root elements generating H_1 and H_2 that extend to a set of root elements (with respect to a single root system) generating G. This property will be described as the *compatibility* of H_1 and H_2. If furthermore H_1 and H_2 have been recognised, we will also be interested in the stronger condition that they be compatible with respect to the sets of root subgroups induced by their recognis-

ing isomorphisms.

Let X be a root subgroup of G, so that X is isomorphic to the additive group of $\mathbb{F}_q$. To *label* X is to assign to each of its elements an element of $\mathbb{F}_q$ so that the induced mapping is an isomorphism (so in fact, this constitutes the recognition on X in its own right). In practice this involves identifying a generating set for X of cardinality e and assigning to each generator an element of an $\mathbb{F}_p$-basis for $\mathbb{F}_q$. The labelling of the various root groups of G must be arranged so as to respect the Chevalley commutator relations for G.

Remark 1.14. In light of this last requirement, the following should be observed: when, as is often the case, we have two recognised subsystem subgroups H_1 and H_2 of G that are compatible in the sense defined above, it does not follow that the labelling of the root groups of H_1 and H_2 induced by their recognising isomorphisms should respect the commutator relations.

Noting that a full labelled set of root subgroups of G immediately yields an isomorphism with the group in its Steinberg representation, the concepts just described justify the following general scheme for recognising G:

1. Find a collection $\mathscr{F}$ of subsystem subgroups of G incorporating between them a complete set of fundamental root subgroups of G.

2. Recognise the groups in $\mathscr{F}$.

3. Adjust the recognising isomorphisms just obtained for mutual compatibility.

4. From the fundamental root subgroups of G (and their opposites) induced by the now-compatible subgroup isomorphisms, extend $\mathscr{F}$ to a set $\mathscr{R}$ of all required root subgroups of G. Precisely which root groups are required depends to some extent on the particular algorithm.

5. Label the root groups in $\mathscr{R}$.

Remark 1.15. This much allows the specification of isomorphisms in each direction between G and its standard copy, by mapping the now-constructed standard generators in G to the

(known) standard generators in the standard copy. In order for these isomorphisms to be *effective*, however, algorithms are also necessary for writing arbitrary elements in G or its standard copy as words in the appropriate standard generators. This is the *straight-line program* problem, which requires its own treatment.

The algorithms of both [31] and [40] take this form. We first look at the particulars of the latter.

1.6.2 The Liebeck–O'Brien algorithms

This is the main result of [40]:

Theorem 1.16 (Liebeck–O'Brien, [40, Theorem 1])**.** *Let G be a black-box group isomorphic to an exceptional group of Lie type defined over $\mathbb{F}_q$ for $q > 2$, other than the Suzuki and Ree groups and ${}^3D_4(2^e)$. Then there is a Las Vegas algorithm that recognises G that runs in polynomial time.*

If G is isomorphic to ${}^3D_4(2^e)$ for $e > 1$, there is a Las Vegas algorithm that recognises G in $O(2^e)$ time.

If G is isomorphic to an exceptional group (other than the Suzuki and Ree groups) defined over $\mathbb{F}_2$, then there is a Las Vegas algorithm that recognises G in constant time.

(We formulate the theorem here in terms of black-box groups for comparison with [31]; the authors give the statement in terms of matrix groups to match their implementation, with a remark that a black-box formulation is possible. Also, we need not pay any further consideration to the special-case algorithms of the latter two paragraphs of the theorem.)

The strategy of the algorithms in Theorem 1.16 is to find a set of $\mathrm{SL}_2(q)$-subgroups of G, one for each node in the Dynkin diagram of G, all taken with respect to a fixed root system. These are found by constructing centralisers of various involutions, whether in the whole of G, inside a subsystem subgroup of G or indeed inside another involution centraliser in G.

The required $\mathrm{SL}_2(q)$-subgroups K_i of these are obtained using the algorithm KILLFACTOR in [7]. The root and toral elements in the K_i are then labelled carefully so as to ensure mutual compatibility.

In contrast, our use of involution centralisers as an adaptation to [31] requires only a single centraliser Γ to be constructed: this is the same centraliser as is formed in the initial stage of [40]. Thereafter, the approaches diverge. The centraliser is a commuting product [2] of a group R, isomorphic to $\mathrm{SL}_2(q)$, and a subsystem subgroup L of co-rank 1 in G: we proceed by recognising L and extending R to a group isomorphic to $\mathrm{SL}_3(q)$ and recognising this also; [40] instead finds further $\mathrm{SL}_2(q)$-subgroups inside L as (subgroups of) further involution centralisers, and a final $\mathrm{SL}_2(q)$-subgroup constructed as the centraliser of a random involution in Γ whose centraliser when intersected with L takes an appropriate form.

Thus we avoid several involution-centraliser constructions (and, most importantly, the verification that these centralisers take the desired forms) at the cost of invoking a recognition algorithm for L and of extending and then recognising R. At first blush this might appear to be a poor trade-off, but these recognitions also afford us the means to perform the greater part of the work involved in labelling the root elements of G; indeed, [40] in its labelling stage requires the recognition of multiple subgroups isomorphic to $\mathrm{SL}_3(q)$ or to $\mathrm{Sp}_4(q)$. That our approach involves only two subsystem groups also reduces the number or boundaries across which it is necessary to ensure compatibility.

We conclude this section with a consideration of the labelling procedures. For the fundamental subgroups, the Liebeck–O'Brien and the Kantor–Magaard algorithms take essentially the same approach: they label one element of the highest root group arbitrarily, while those of the roots orthogonal to this receive their labels from the recognition of the corresponding subsystem subgroups (with some additional linear algebra in [40] for compatibility). Standard properties of groups of Lie type, not least the Chevalley commutator relations, label the remaining root groups. In [31] the procedure for labelling *all* root groups is described, but in practice only the reduced Curtis–Steinberg–Tits generators (as described in [6]) are required.

[2] The techniques here and in [40] for finding these factors also differ, but this is of less significance.

[40] is arranged accordingly, and gives explicit constructions of these in appendices. We shall make use of these.

1.6.2.1 Straight-line programs

The procedure described above finds standard generators for G. In order to avoid a factor of q in the running time for the algorithm for finding straight-line programs for elements of G, Liebeck and O'Brien use the generalised row- and column-reduction algorithm due to Cohen, Murray and Taylor [24], which operates on sets of matrices representing G. This contrasts with the black-box but linear-in-q algorithm in [31].

1.6.3 The Kantor–Magaard algorithm

In the rest of this dissertation we will detail our own adaptations to the algorithm of Kantor and Magaard for recognising exceptional groups, but here we consider it as it is presented in [31] itself. It conforms to the pattern described at the end of Section 1.6.1, and so, after stating the full result, we shall examine it under corresponding headings.

We shall describe only what is necessary either for the purposes of implementation where we follow [31], or else for comparison where we do things differently. In particular we omit most of the justification for the algorithm. We shall also maintain the notation from that paper, even if the resulting body of symbols would in other circumstances be eccentric (for example, we shall define the root v', but not v). In the interests of comprehensibility, we prefer to specify entities descriptively rather than symbolically whenever reasonable.

Theorem 1.17. *Let G be a black-box representation of a perfect central extension of a simple exceptional group of Lie type of rank at least 2, defined over the field of order q, other than ${}^2F_4(q)$. Then there is a Las Vegas algorithm that recognises G as a homomorphic image of the universal cover $\widehat{G}$ in $O(q\log^2 q)$ time. There is a Las Vegas algorithm for calculating preimages under the resulting isomorphism that runs in $O(q\log^2 q)$ time, and a deterministic algorithm*

for calculating images in $O(\log q)$ *time.*

Remark 1.18. The running times just stated require some qualification. The times are straightforward in calculations proper to the algorithm itself, and these are discussed below. When a classical recognition algorithm is invoked, however, [31] supposes a cost to the running time of $O(q\log^2 q)$, once times for the group operation and the generation of random elements have been suppressed. In view of the timings for classical recognition algorithms given in Section 1.5, however, which are polynomial in the size of the input *subject to the availability of an* $\mathrm{SL}_2(q)$*-recognition oracle*, we wish to extend such oracle-dependent timings to the analysis of the algorithm for Theorem 1.17 with the intention of isolating the portions of that algorithm which run in times intrinsically linear (or worse) in q. We have *not* done so in the statement of the theorem, but in the following discussion we will refer back to this remark when appropriate, at which points it should be understood that the relevant process constitutes a $O(q\log^2 q)$-time bottleneck for the purposes of the theorem, but that a finer-grained analysis might be found by making reference to the appropriate theorem in Section 1.5. The exponential-time portions thus distinguished will constitute the principal areas that we seek to improve in the next chapter. Cf. the remark following Theorem 1.3 in [31].

Remark 1.19. Theorem 1.17 as stated holds for groups of rank at least 2. Kantor and Magaard remark that it may be extended to include the rank-1 groups, but that as there exist *ad hoc* algorithms in the literature for each of these, this is less interesting. In the discussion which follows, we will restrict ourselves to the case where the rank is at least 3: this will of course obtain when we focus later on $F_4(q)$, and the rank-2 version of the algorithm in [31] involves additional complications that need not concern us.

1.6.3.1 Finding subsystem subgroups

The algorithm begins by performing an $O(q)$-time search amongst random elements (or, for $F_4(q)$ with q odd, an $O(q^2)$-time search) for a long root element in G. An element is identified as having a power that is such by consideration of its order: [31, Lemma 2.24] details the

relevant criteria, but in most cases, the crux is that an element with factors in its order of p and certain primitive prime divisors of $|\widehat{G}|$ will, when raised to the power of the p'-part of $|\widehat{G}|$, yield a long root element.

If this long root element is z, the algorithm proceeds to construct two $\mathrm{SL}_3(q)$-subgroups S and S_2 of G by forming the subgroup generated by z and two random conjugates, and attempting to recognise the group as $\mathrm{SL}_3(q)$. A constant but large number of attempts is made. An $\mathrm{SL}_2(q)$-subgroup R containing z is taken inside S by means of its recognising isomorphism. The root group in R containing z is then constructed and denoted by Z, and decreed to be the highest root group of G. Random conjugates of Z are adjoined to S_2 (not cumulatively) until the resulting group J is isomorphic to $\mathrm{Spin}_8^-(q)$; this group is recognised, which allows the construction of an $\mathrm{SL}_2(q)$-subgroup R_1 of J centralising R. This is extended to a group L corresponding to the subsystem of the root system for G obtained by deleting the unique fundamental root not orthogonal to the highest root, by adjoining an element constructed from S and S_1 which we shall not further specify here.

This procedure has constructed a set $\mathscr{F}$ in the notation of Section 1.6.1, namely $\{S, L\}$. These groups are then recognised, and we examine in the next section how mutual compatibility is ensured.

1.6.3.2 Compatibility of S and L

To ensure that the root subgroups of S and L induced by their recognising isomorphisms are compatible, it is sufficient to ensure that the intersection of the two tori likewise induced, denoted respectively by T_S and T_L, is equal to the intersection of S and L themselves. By construction, $S \cap L = C_S(R)$ is a one-dimensional torus and so in fact it is sufficient to ensure that each of T_S and T_L contains $S \cap L$; moreover the former containment is automatic owing to the choice of R inside S. To arrange for T_L to lie in $S \cap L$, an element l of L is found that diagonalises $S \cap L$ on the natural module for L (or, when L is isomorphic to $E_7(q)$, on its action on its Lie algebra) via the recognising isomorphism, and S is replaced by S^l.

[31] also gives an alternative method, which is Las Vegas and hinges on arranging for $S \cap L$ and a conjugate of S to generate together $\mathrm{Spin}_8^-(q)$, but we not describe it now as our own approach will be an adaptation of the method above.

1.6.3.3 Root groups: finding and labelling

Once the compatibility of S and L has been accomplished, it is straightforward to find and label the remaining root groups. Those obtained from the just-mentioned groups can be extended to a full set $\mathscr{F}$ ([31] notes) either by the action of the Weyl group or via the Chevalley commutator relations. The latter method also labels the root groups, starting from the labelling induced by the recognition of L. This, together with an arbitrary choice of an element to label as '1' in the root group for the remaining fundamental root ν', determines the labelling of all of the root groups. The subgroup of G generated by the root groups so identified is denoted by G_0; later it will be verified that in fact $G = G_0$.

1.6.3.4 Straight-line programs

The strategy for finding a straight-line program for an element g of G takes the following form: firstly, a random y in G_0 is found such that Z^{gy} and Z are opposite. Now, let Q denote the unipotent radical of the parabolic subgroup of G corresponding to the root subsystem induced by L, let Z^- be the root group in $\mathscr{F}$ opposite to Z and let n be an element of $\langle Z, Z^- \rangle \cong \mathrm{SL}_2(q)$ conjugating Z^- to Z. Then elements u and u' in Q are constructed such that $g' = gyunu'$ normalises both Z and Z^-. We describe the construction of these elements shortly. Next, an element h corresponding to ν' in the maximal split torus is found that acts on Z and Z^- in the same as g'. This element h is found by exhaustive search, which takes $O(q)$ time. Then $g'h^{-1} \in L$, a straight-line program for which may be found by means of the recognition of L; but this also gives a straight-line program for g as h, y, u, u' and n were all constructed as straight-line programs.

The element u of Q is constructed by the following Las Vegas algorithm: random $v \in Q$

are generated until $S(v) := \langle Z, Z^-, Z^{gyv} \rangle$ is isomorphic to $\mathrm{SL}_3(q)$, which property is detected by attempting to recognise $S(v)$. The resulting isomorphism allows the identification of an element u of $O_p(C_{S(v)}(Z))$ such that $Z^{gyu} = Z^-$. Then in fact u must be in Q and is the desired element. The same algorithm allows the construction of u', with the defining property this time being that $Z^{-gyunu'} = Z^-$.

Finally, u and u' are written as products of elements of the root groups in $\mathscr{F}$ that generate Q, namely those corresponding to positive roots other than those in the subsystem for L. Fix an arbitrary order on these roots (which will also be the order in which the root groups appear in the word to be constructed) and let $\tilde{u}$ be either u or u'. We proceed iteratively. Let μ be the first root in the ordering whose root-group element has not yet been found. The element corresponding to μ is then obtained by taking the commutator of $\tilde{u}$ with an element of the root group for $\nu - \mu$; the result is an element of the root group for ν, the label for which, with adjustment for the structure constants as specified by the Chevalley commutator relations, yields the label for the desired root element x. By replacing $\tilde{u}$ by $\tilde{u}x^{-1}$ and repeating this process, all of the labels can be found.

The only part of the construction of these elements of Q requiring any significant computation is the $\mathrm{SL}_3(q)$-recognition, and so the running time is subject to the principles discussed in Remark 1.18.

Remark 1.20. The same procedure can be used to generate straight-line programs for arbitrary elements of $\widehat{G}$ in terms of root elements contained therein, as noted in [31, Remark 2.41]. Kantor and Magaard also give a deterministic algorithm for doing so using the action of $\widehat{G}$ on its Lie algebra, but we will not use this.

1.7 Counting elements in groups of Lie type

In group-recognition algorithms it is very often necessary to construct elements having certain desirable properties. If these elements should be scarce, we seek more abundant ele-

ments which can in turn be used to produce those we require. In either case, we generate elements in our group or some subgroup thereof (nearly) uniformly at random as discussed earlier, until we find an element of the sort needed. It might be the case that no such elements exist — either because the input group was not in fact isomorphic to the group that the algorithm recognises, or else because the existence of the elements was contingent on the success of an earlier probabilistic stage of the algorithm that had failed — and so to ensure that the algorithm terminates in finite time it is necessary eventually to abandon the search should it not have succeeded, with the precise point at which to do so being determined by the desired probability of the algorithm failing when given valid input and the abundance of the elements sought.

This leads us to the problem of counting the elements in a group of Lie type satisfying our specified properties, which, owing to the limited information that may be gleaned about an element in our black-box setting, will consist of conditions on the elements' orders, and in particular on the factors they share with certain polynomials in q. Parker and Wilson in [51] examine a variety of counting problems of this sort, with a view to analysing algorithms for finding involutions in groups of Lie type. The results that we detail here are of a similar flavour, except that, where Parker and Wilson give existence results concerning bounds on proportions of elements in general exceptional or classical groups, we are primarily concerned with finding explicit constant bounds in particular cases.

1.7.1 Semisimple elements, maximal tori and the Weyl group

We begin by recalling some standard results.

Let K be a field. An element x of a group acting linearly on a K-module V is said to be *semisimple* if, for every x-invariant submodule U of V, there exists an x-invariant complement W to U in V, or, equivalently, if x is diagonalisable over some extension of K. The latter characterisation will be more germane to our purposes. Maschke's theorem then gives the following result:

Lemma 1.21. *An element of a group of Lie type with defining characteristic p is semisimple if and only if its order is coprime to p.*

Let G be a Chevalley group defined over $\mathbb{F}_q$ with respect to a root system Φ. Recall from Section 1.1.3 that the *maximal split tori* of G are the conjugates of the subgroup $\langle h_\alpha(t) : \alpha \in \Phi, t \in \mathbb{F}_q \rangle$ of G, with notation as defined in that section. A subgroup T of G is a *maximal torus* of G if it is the intersection with G of a maximal split torus in an overgroup $\tilde{G}$ of G that is a Chevalley group of the same type as G but defined over an extension of $\mathbb{F}_q$. They are thus isomorphic to direct products of subgroups of the multiplicative groups of extensions of $\mathbb{F}_q$. The features of maximal tori that will be of most use to us is that their elements are semisimple, and that all semisimple elements of G lie in some maximal torus. An element of G lying in a *unique* maximal torus is said to be *regular semisimple*. An element is regular semisimple if and only if it has a full spectrum of distinct eigenvalues on the natural module for G (in an extension field if necessary).

It is a standard result (given, for example, in [56, p. E-20]), that the conjugacy classes of maximal tori in G are in bijective correspondence with the conjugacy classes in the Weyl group of G. Furthermore, [20, p. 46] gives the following lemma:

Lemma 1.22. *Let G be a Chevalley group with Weyl group W. Let T be a maximal torus of G corresponding to an element $w \in W$ in the sense remarked on above. Then, except for certain pairs (T, G) defined over small fields*[3]*, $N_G(T)/T$ is in bijective correspondence with $C_W(w)$.*

Corollary 1.23. *With notation as in Lemma 1.22, there are $\frac{|G|}{|C_W(w)||T|}$ conjugates of T in G.*

Proof. There are $\frac{|G|}{|N_G(T)|} = |G : N_G(T)| = \frac{|G:T|}{|N_G(T):T|} = \frac{|G|}{|C_W(w)||T|}$ such conjugates. □

Corollary 1.24. *With G and W as above, let $\mathcal{T}$ be a set of representatives for the conjugacy classes of maximal tori in G, and, for each $T \in \mathcal{T}$, let $w_T \in W$ be a corresponding element of*

[3] These cases will not arise on the occasions in which we use this result and so we do not enlarge on the matter.

the Weyl group, and let $A_T \subset T$ be an arbitrary subset of regular semisimple elements. Then

$$\frac{\left|\bigcup_{T\in\mathcal{T}} \bigcup A_T^G\right|}{|G|} = \sum_{T\in\mathcal{T}} \frac{|A_T|}{|C_W(w_T)||T|}.$$

Proof. Regular semisimplicity avoids double-counting. □

These results allow us to obtain estimates of the numbers of semisimple elements in G whose orders have given factors. The procedure, as in [51], is as follows:

1. We begin by identifying the conjugacy classes of maximal tori containing elements of the desired orders. The results allowing us to do so are presented in Section 1.7.2.

2. In each class of maximal torus, we count the number of regular semisimple elements having orders in which we are interested. An element can be identified as being regular semisimple by considering its eigenvalues, and restricting to such elements prevents double-counting.

3. Summing these element-counts over the identified classes of maximal tori, scaled by the factors occurring in Corollary 1.23 gives a lower bound on the number of desired elements in G.

1.7.2 Salient properties of some particular Weyl groups

In the course of the algorithm, we will seek elements in various tori of $G \cong F_4(q)$ itself, and in tori of subgroups of G isomorphic to $\mathrm{Spin}_9(q)$, $\mathrm{Sp}_6(q)$ and $\mathrm{SL}_2(q)$. In this section, we collect some (mostly elementary) results regarding the properties of the Weyl groups of the just-mentioned groups. Together with the results in the preceding section, these will prove useful for estimating the abundances of the desired elements in the respective subgroups.

The general structure of the maximal tori in classical groups is described in [55] in terms of Singer cycles acting on factors in decompositions of the natural modules into mutually orthogonal subspaces. This is in turn related explicitly to elements of the corresponding Weyl

groups in [19]. Complete lists of the conjugacy classes of maximal tori in exceptional groups are collated from the literature in [36].

1.7.2.1 The Weyl group $W(B_n) \cong W(C_n) \cong 2 \wr S_n$

Th Weyl groups of the root systems B_n and C_n, or respectively of the groups $\Omega_{2n+1}(q)$ and $\mathrm{Sp}_{2n}(q)$, are both isomorphic to $2 \wr S_n$. This group embeds naturally into the group S_{2n} acting on the set $X = \{\pm i : 1 \leq i \leq n\}$. When occasionally we will need to describe elements of $2 \wr S_n$ explicitly, we will write them as partial permutations of X, specifying the images only of the positive elements, which determine the entire permutation.

We proceed now to formulate the relationship between the conjugacy classes in $2 \wr S_n$ and the maximal tori in the classical groups.

Definition 1.25. Let n be a positive integer. Define a *signed partition of n* to be a multiset $\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_k\}$ of nonzero integers such that $n = \Sigma_{i=1}^{k} |\lambda_i|$.

Definition 1.26 (cf. [19, p. 80])**.** Let $\sigma \in 2 \wr S_n$ be such that the image ρ of σ under the natural quotient onto S_n is cyclic. Then σ is said to be a *cycle* in $2 \wr S_n$. If ρ has order k, then we say that σ is a *positive cycle* if σ^k is the identity, and otherwise that it is a *negative cycle*; the property of a cycle of being positive or negative is its *parity*.

Lemma 1.27 ([19, p. 80])**.** *The conjugacy classes in $2 \wr S_n$ are parameterised by signed partitions of n.*

Proof. [19] does not give a proof of this result, but for expository purposes we now supply a sketch.

Decomposition into disjoint cycles in the sense of definition 1.26 induces a map onto the set of signed partitions. Conjugation in $2 \wr S_n$ preserves parities as well as cycle lengths, so this map is a class function. Two elements mapping to the same partition are certainly conjugate in S_{2n} (considering the natural embedding), and can in fact be seen to be conjugate

in $2 \wr S_n$: without loss of generality, consider two cycles σ_1 and σ_2 in $2 \wr S_n$ of the same length and parity. If they are negative, choose elements m_1 and m_2 moved respectively by σ_1 and σ_2, and construct an element τ of S_{2n} conjugating σ_1 to σ_2 by mapping $\sigma_2^k(m_2)$ to $\sigma_1^k(m_1)$ for each k, and fixing all other elements. If instead the cycles are positive, they decompose as a product of two cycles in S_{2n}; modify τ so that it sends the distinct cycles of σ_2 to the distinct cycles of σ_1, with compatible orderings on the two cycles within each of σ_1 and σ_2. Then in both cases, τ is in fact in $2 \wr S_n$. □

We call the signed partition associated to an element of $2 \wr S_n$ its *cycle type*, by way of analogy with the symmetric group.

Lemma 1.28 ([19, Theorems 3 and 4])**.** *The conjugacy classes of maximal tori in $\Omega_{2n+1}(q)$ and in $\mathrm{Sp}_{2n}(q)$ are parameterised by signed partitions of n. Let T be a maximal torus in such a group corresponding to a partition $\{\lambda_1, \lambda_2, \ldots, \lambda_k\}$, and let ϵ_i be 1 or -1 respectively as λ_i is positive or negative. If necessary, reorder the λ_i such that $\left\{q^{\lambda_1} - \epsilon_1\right\}_2$ is minimal amongst the $\left\{q^{\lambda_i} - \epsilon_i\right\}_2$. Then in the case where $T \leq \Omega_{2n+1}(q)$,*

$$T \cong C_{\frac{q^{\lambda_1}-\epsilon_1}{2}} \times \prod_{i=2}^{k} C_{q^{\lambda_i}-\epsilon_i},$$

and when $T \leq \mathrm{Sp}_{2n}(q)$,

$$T \cong \prod_{i=1}^{k} C_{q^{\lambda_i}-\epsilon_i}.$$

Then, in view of Corollary 1.23, in order to determine the number of conjugates of a maximal torus corresponding to an element σ of $2 \wr S_n$, it is necessary to calculate the order of $C_{S_n}(\sigma)$, which has a straightforward combinatorial structure:

Lemma 1.29. *Let $x \in 2 \wr S_n$ have cycle type*

$$\{\lambda_{1_1}, \lambda_{1_2}, \ldots, \lambda_{1_{m_1}}, \lambda_{2_1}, \lambda_{2_2}, \ldots, \lambda_{2_{m_2}}, \ldots, \lambda_{l_1}, \lambda_{l_2}, \ldots, \lambda_{l_{m_l}}\},$$

where

$$\begin{aligned}
&\lambda_{1_1} = \lambda_{1_2} = \ldots = \lambda_{1_{m_1}} \geq \\
&\lambda_{2_1} = \lambda_{2_2} = \ldots = \lambda_{2_{m_2}} \geq \\
&\ldots \geq \\
&\lambda_{k_1} = \lambda_{k_2} = \ldots = \lambda_{k_{m_k}} > \\
&0 > \\
&\lambda_{(k+1)_1} = \lambda_{(k+1)_2} = \ldots = \lambda_{(k+1)_{m_{k+1}}} \geq \\
&\ldots \geq \\
&\lambda_{l_1} = \lambda_{l_2} = \ldots = \lambda_{l_{m_l}}.
\end{aligned}$$

Then the centraliser of x in $2 \wr S_n$ is isomorphic to

$$(2 \times \lambda_{1_1}) \wr S_{m_1} \times (2 \times \lambda_{2_1}) \wr S_{m_2} \times \ldots \times (2 \times \lambda_{k_1}) \wr S_{m_k} \times (2|\lambda_{(k+1)_1}|) \wr S_{m_{k+1}} \times \ldots \times (2|\lambda_{l_1}|) \wr S_{m_l}.$$

Proof. We consider the natural embedding of $2 \wr S_n$ in S_{2n}, wherein positive cycles of length λ in $2 \wr S_n$ become two cycles of length λ in S_{2n} and negative cycles of length μ become single cycles of length 2μ. Our desired centraliser, then, is the intersection of the centraliser in S_{2n}, which has the form

$$(\lambda_{1_1}) \wr S_{2m_1} \times (\lambda_{2_1}) \wr S_{2m_2} \times \ldots \times (\lambda_{k_1}) \wr S_{2m_k} \times (2|\lambda_{(k+1)_1}|) \wr S_{m_{k+1}} \times \ldots \times (2|\lambda_{l_1}|) \wr S_{m_l}, \quad (1.4)$$

with $2 \wr S_n$ itself. To determine this intersection, we consider positive and negative cycles separately.

Firstly, consider an element y in S_{2n} centralising (the image of) the product π of all negative cycles of a given length u (and so of length $2u$ in S_{2n}). Let $t \in \{1, 2, \ldots, n\}$ be moved by

π. Then y acts imprimitively on the the points moved by π with the cycles of π as its blocks, and cycling the elements within each block in the order determined by the cycle itself. Thus y acts on t, and on all other elements in the cycle containing t, by first mapping it to the corresponding element in another of the cycles, and then by cycling it some number of places within that cycle.

We now consider the image of $-t$ under y. Observe that t and $-t$ are u places apart in the cycle in S_{2n} which they inhabit. Thus their images under y are also u places apart in the cycle into which they are mapped. By the properties of the embedding of negative cycles in S_{2n} mentioned at the beginning of the proof, it follows that $y(t) = y(-t)$, so that $y \in 2 \wr S_n$. Thus the factors in the overgroup (1.4) corresponding to negative cycles — i.e., those in $(k+1)$-st and higher positions — are in fact attained, and are not strict overgroups.

Next, we proceed to perform an analogous analysis for a product ρ of all *positive* cycles (of which, let us say there are r) of a given length v, which split into *two* cycles each of length v when embedded in S_{2n}. *A priori*, we have that an element y in S_{2n} wreathes all $2r$ cycles with the intra-block action as in the previous paragraph, but we no longer automatically have that y is in $2 \wr S_n$ as t and $-t$ (where t is moved by ρ) are no longer members of the same cycle. In order to ensure that y is in fact in $2 \wr S_n$, we require that the wreathing operate on the *pairs* of cycles induced by the embedding in S_{2n}, possibly interchanging the two cycles within the pair. We thus have blocks consisting of these pairs of cycles, of which there are r, within which each of the elements of each cycle may be cycled as before, and the elements interchanged between the cycles respecting the orders, so that the action is of $2 \times v$. This completes the proof of the lemma. □

Chapter 2

Constructing and identifying involution centralisers in $F_4(q)$

In odd characteristic, $F_4(q)$ contains a conjugacy class of involutions whose centraliser has shape $(\mathrm{SL}_2(q) \circ \mathrm{Sp}_6(q)).2$. A concrete description of this centraliser is given in [59, p. 159], and in [28, Table 4.5.1] it is identified as part of an exposition of the structure of the exceptional groups in general. Our algorithm hinges on the construction of this subgroup. It will afford us with alternative methods to those given in [31] for finding various subsystem subgroups of $F_4(q)$. In this chapter, we look at the practicalities of constructing that centraliser using a randomised algorithm, including an analysis of the probability that that algorithm should succeed. We will also do likewise for an algorithm for finding the classical components of the index-two central product in $(\mathrm{SL}_2(q) \circ \mathrm{Sp}_6(q)).2$.

From now on, we will fix q to be odd.

2.1 Counting problems arising from the use of Bray's algorithm

Bray's algorithm for generating an involution centraliser was introduced in Section 1.4.2, and we will use it to construct $(\mathrm{SL}_2(q) \circ \mathrm{Sp}_6(q)).2$. We now turn our consideration to the problem of ensuring that the algorithm terminates with probability of error bounded by a given threshold, which will be fundamental to arranging that our own algorithm does likewise.

There is an observation in [11], attributed to Richard Parker, that, when c has odd order, the elements gc^n are uniformly distributed amongst the elements of the centraliser of x in G if g is uniformly distributed amongst those elements such that $c = [x, g]$ has odd order. This has the happy consequence that, *if such elements g are in sufficient supply*, Proposition 1.5 can be used to construct the centraliser of the involution x in G from pseudo-random elements of G constructed by the product replacement algorithm *in such a way that it is reasonably straightforward to analyse the probability of having generated the centraliser after a given number of iterations of the algorithm*. We make this property rigorous using the following lemma:

Lemma 2.1. *Let G be any group, let t be the smallest index of any of its proper (and hence maximal) subgroups, and let m be the number of conjugacy classes of maximal subgroups of G. If X is a non-empty subset of G whose elements have been sampled uniformly at random from G, then the probability that X generates G is at least $1 - mt^{1-|X|}$.*

Proof. [11, §4] contains a weaker version of the following argument, calculating the probability that a constant number of elements lie in a *given* subgroup of G.

In the first place we note that of course X generates G if and only if X is not contained in any maximal subgroup of G. Let M be a fixed maximal subgroup of G of index $s \geq t$, so that the probability that an arbitrary element of G is contained in M is s^{-1}. Then X is contained in M with probability $s^{-|X|}$. Now there are at most s conjugates of M in G, and therefore X is contained in some conjugate of M with probability at most $s^{1-|X|} \leq t^{1-|X|}$. Since there are m conjugacy classes of maximal subgroups in G, X is contained in some maximal subgroup of G with probability at most $mt^{1-|X|}$, and the result follows. □

The involution centralisers with which we are concerned have low-index subgroups that are central products of low-dimensional classical groups. It is in fact these subgroups, rather than the entire centraliser, that we wish to generate, and so, in applying Lemma 2.1 to them to estimate the probability of generating them using Bray's algorithm, the values of m and t in the lemma may be calculated by perusing [13].

It remains then to estimate the abundance of elements whose commutator with the involution z has odd order so that the resulting elements of the centraliser may be analysed using Lemma 2.1. Parker and Wilson prove in [51, Theorem 1] that, asymptotically, the elements of an exceptional group having this property occur in proportion bounded below by a constant that is independent of the size of the field. We require an explicit bound on this proportion for each group that we consider, however (although it is not necessary that this constant should be absolute, merely that it be $O(\log q)$), and so we now adapt the arguments in the proof of [51, Theorem 13] to this end.

Lemma 2.2. *Let G be a simple exceptional group of Lie type and let $x \in G$ be an involution inverting an odd-order maximal torus T in G when acting by conjugation. Then the proportion of elements $g \in G$ such that $g \neq x$ and $[x, g]$ has odd order is bounded below by*

$$\frac{r}{|C_W(w)||T|},$$

where r is the number of regular semisimple elements in T and w is an element in the Weyl group W of G corresponding to T (in the sense of Lemma 1.22 and the remark preceding it).

Proof. The group $D := \langle T, x\rangle$ is a dihedral group of twice odd order, so that D contains $|T|$ involutions which are all conjugate, and any of two of which have a product of odd order. In particular, D contains $|T| - 1$ involutions meeting the criteria for the desired elements g in the statement of the lemma. Moreover, taking the products of each of these with x traverses the non-identity elements of T, and so r of these products are regular semisimple. Thus, for each of the (by Corollary 1.23) $\frac{|G|}{|C_W(w)||T|}$ conjugates T' of T, the group $\langle T', x\rangle$ contains r involutions g, unique to that conjugate, such that $[x, g]$ has odd order. The result then follows

immediately. □

Such tori T exist in the situations with which we are concerned, and specimens are given in [51, Table 1].

In view of Lemmas 2.1 and 2.2, we adopt the following strategy for terminating Bray's algorithm when generating an involution centraliser in a group G: In iterating the algorithm, we count the number of generators constructed from odd-order commutators. If, after n iterations, k such generators have been found, then the entire centraliser has been generated with probability $1 - mt^{1-k}$ in the notation of Lemma 2.1. By finding a value K of k such that this probability is bounded below by a predetermined threshold p_1, we may allow the search to succeed when k reaches K with the probability of a false positive bounded above by $1 - p_1$. On the other hand, to ensure that the algorithm terminates, we pre-compute a bound N such that n exceeds N before k reaches K with probability at most some parameter p_2. In practice we find such an N using the Chernoff bound (in the form given in [1]); that this yields a rather sub-optimal N does not present a problem. We calculate the relevant proportions and bounds in the appropriate places later in this chapter.

Our final remark regarding Bray's algorithm is that, if we intend to construct elements at random from the resulting centraliser, we may use the generators given by Bray's algorithm even before the entire centraliser has been found (restricting ourselves to those arising from odd-order commutators if a uniform distribution is required). We will make use of this observation when attempting to distinguish between classes of involution centraliser.

2.2 Constructing the involution centraliser

In order to obtain a complete set of fundamental root groups for G, Kantor and Magaard construct a subgroup $S \cong \mathrm{SL}_3(q)$ generated by long root elements of G and choose within S a subgroup $R \cong \mathrm{SL}_2(q)$. The centraliser L of R in G is then a Levi subgroup, isomorphic to $\mathrm{Sp}_6(q)$,of the parabolic subgroup of G corresponding to the $\mathtt{C}_3$-subsystem $\mathtt{L}$ of $\mathtt{F}_4$ having fun-

Figure 2.1: Extended Dynkin diagram of $\mathtt{F}_4$, with fundamental subsystems indicated.

damental roots as illustrated in Figure 2.1. The same figure also indicates the $\mathtt{A}_2$-subsystem $\mathtt{S}$ corresponding to S.

Making compatible choices, in the sense defined in Section 1.6.1, amongst the root subgroups of S and L affords a set of root groups corresponding to the fundamental roots of $\mathtt{F}_4$ with respect to a common torus, and in turn these (together with their opposite root subgroups) furnish us with a complete set of $\mathtt{F}_4$-root groups via the action of the Weyl group, which can itself be constructed from S and L.

Whereas Kantor and Magaard construct S and L by building generators for the subgroups from random elements in G of appropriate orders — an approach that works independently of the characteristic — we opt instead to exploit the structure of the involution centraliser $(\mathrm{SL}_2(q) \circ \mathrm{Sp}_6(q)).2$, at the cost of restricting ourselves to odd characteristic. This section describes our strategy.

2.2.1 The involution centraliser $(\mathrm{SL}_2(q) \circ \mathrm{Sp}_6(q)).2$

Consider the central product $\mathrm{SL}_2(q) \circ \mathrm{Sp}_6(q)$ lying inside the involution centraliser $(\mathrm{SL}_2(q) \circ \mathrm{Sp}_6(q)).2$. It contains a unique subgroup $L \cong \mathrm{Sp}_6(q)$, as $\mathrm{Sp}_6(q)$ may be generated by elements of orders such that they cannot lie in any group isomorphic to $\mathrm{SL}_2(q)$; then there is a unique subgroup $R \cong \mathrm{SL}_2(q)$ of $\mathrm{SL}_2(q) \circ \mathrm{Sp}_6(q)$ commuting with L. R and L each contain a unique central involution, and it is precisely these two elements that are identified by the central product. The resulting involution is, of course, that which $(\mathrm{SL}_2(q) \circ \mathrm{Sp}_6(q)).2$ centralises.

2.2.2 Generating and identifying the desired involution centraliser

As described in [59], $F_4(q)$ has two conjugacy classes of involutions, one having centraliser $(\mathrm{SL}_2(q) \circ \mathrm{Sp}_6(q)).2$ as discussed above, and the other having centraliser $\mathrm{Spin}_9(q)$. The latter contains elements of order q^4+1 while the former does not, and it is in this manner that we distinguish the two in our black-box setting. In fact the presence of any element of order greater than 2 and dividing q^4+1 is sufficient to identify such a centraliser as $\mathrm{Spin}_9(q)$, as $\gcd(|(\mathrm{SL}_2(q) \circ \mathrm{Sp}_6(q)).2|, q^4+1) = 2$.

We first find an involution t in G by searching at random for an element of even order and then raising it to the appropriate power. The proportions of even-order elements in groups of Lie type are discussed at length in [42], where we find that the proportion of even-order elements in $F_4(q)$ having an involutory power with centraliser $(\mathrm{SL}_2(q) \circ \mathrm{Sp}_6(q)).2$ is at least 0.3338, and with centraliser $\mathrm{Spin}_9(q)$ at least 0.4060.

Next, we generate the centraliser C of t using Bray's algorithm, as discussed above in Sections 1.4.2 and 2.1. We wish to search C for elements whose order shares a common factor greater than 2 with q^4+1, the probability of finding which in $\mathrm{Spin}_9(q)$ being given by the following result:

Lemma 2.3. *The proportion of elements in* $\mathrm{Spin}_9(q)$ *having order n such that* $\gcd(n, q^4+1) > 2$ *is at least* $\frac{1}{16}$.

Proof. By Lemma 1.28, the only maximal tori of $\Omega_9(q)$ whose order shares a common factor greater than 2 with q^4+1 are those isomorphic to C_t, where $t = \frac{q^4+1}{2}$, of which there is a single conjugacy class corresponding to (the conjugacy class of) the element $\sigma = \left(\begin{smallmatrix} 1 & 2 & 3 & 4 \\ -2 & -3 & -4 & -1 \end{smallmatrix}\right)$ of $2 \wr S_4$. By Lemma 1.29, $|C_{2 \wr S_4}(\sigma)| = 8$, and then by Corollary 1.23, the conjugacy class consists of $\frac{|\Omega_9(q)|}{8t}$ such tori. By [51, Lemma 17], at least $\frac{1}{2}t$ of the elements in each of these tori are regular semisimple, and so the proportion of elements in $\Omega_9(q)$ having order of the desired form is at least $\frac{1}{16}$. The same is true of $\mathrm{Spin}_9(q) \cong 2 \cdot \Omega_9(q)$, as this group maps naturally onto $\Omega_9(q)$. □

We search for these elements in two phases. In the fashion remarked on at the close of Section 2.1, we test the orders of each of the generators of C as they are constructed by Bray's algorithm, and can reject the centraliser immediately if a suitable common factor with q^4+1 is encountered. Once the whole of C has (probably) been generated, Bray's algorithm is terminated and the search for such elements continues using the product replacement algorithm. In determining when to end this search, tests from the first phase are only counted towards the total number of trials attempted when the elements arose as powers of odd-order commutators.

The following results allow us to calculate values for the parameters m, t, T and r of Lemmas 2.1 and 2.2.

Lemma 2.4. *There are at most* 34 *conjugacy classes of maximal subgroups in* $\mathrm{SL}_2(q)\circ\mathrm{Sp}_6(q)$.

Proof. The maximal subgroups of $\mathrm{SL}_2(q)\circ\mathrm{Sp}_6(q)$ are the natural quotients of those of $\mathrm{SL}_2(q)\times\mathrm{Sp}_6(q)$; these, by Goursat's lemma, are of the form $X\times\mathrm{Sp}_6(q)$ or $\mathrm{SL}_2(q)\times Y$, where X and Y are respectively maximal in $\mathrm{SL}_2(q)$ and $\mathrm{Sp}_6(q)$. The maximal subgroups of these classical groups are enumerated in [13, Tables 8.1, 8.2, 8.28 and 8.29], and in total, restricting to q odd and discounting mutually-exclusive possibilities, there are at most 9 conjugacy classes of maximal subgroups in $\mathrm{SL}_2(q)$, and 25 in $\mathrm{Sp}_6(q)$. □

Lemma 2.5. *The minimal index of any proper subgroup of* $\mathrm{SL}_2(q)\circ\mathrm{Sp}_6(q)$ *for any odd* q *is* 3.

Proof. This follows from the same tables in [13], and is attained by the subgroup Q_8 of $\mathrm{SL}_2(3)$ (under the projection from the central product). □

Lemma 2.6. *$F_4(q)$ contains a maximal torus $T\cong C_{q^4-q^2+1}$ inverted under conjugation by an involution having centraliser* $(\mathrm{SL}_2(q)\circ\mathrm{Sp}_6(q)).2$ *and corresponding to an element in the Weyl group having centraliser of order* 12. *In proportion, at least*

$$\prod_{\pi\in\mathscr{P}}\left(1-\frac{1}{\pi}\right)>0.8297$$

of the elements of T are regular semisimple, where $\mathscr{P}$ is the set of the first twelve primes congruent to 1 *modulo* 12.

Proof. The first sentence follows from [51, Table 1].

To prove the second, let g be a non-identity element of T, and let α_1 be a non-identity eigenvalue in a suitable extension field of a fixed element representing g in the 26-dimensional representation [1] of $F_4(q)$ (we will henceforth abuse terminology and speak of "eigenvalues of g"). Since $q^4 - q^2 + 1$ is the twelfth cyclotomic polynomial, α_1 is a root of an irreducible polynomial P_g over $\mathbb{F}_q$ of degree 12, the remaining (distinct) roots of which are obtained by iterating the automorphism $x \mapsto x^q$ of $\mathbb{F}_{q^{12}}$ on α_1.

By the realisation of $F_4(q)$ as the group of automorphisms of the Albert algebra over $\mathbb{F}_q$, the argument on [59, p. 153] gives us that any element in a maximal split torus of $F_4(q)$ has unity as an eigenvalue twice, corresponding to the action on the diagonal part of the algebra, with the remaining twenty-four eigenvalues being determined (in general) by four arbitrary $\mathbb{F}_q$-parameters. Over a suitable extension field of degree m, every semisimple element of $F_4(q)$ lies in a maximal split torus of $F_4(q^m)$, however, and so in fact this property on the eigenvalues holds for *all* semisimple elements of $F_4(q)$. Thus our element g has as its eigenvalues the twelve described in the previous paragraph, unity twice, and a further twelve, the set of which we shall denote by A, that are either all unity or are the roots of an irreducible degree-12 polynomial Q_g over $\mathbb{F}_q$, not necessarily distinct from P_g.

Suppose further now that g generates T, so that without loss of generality α_1 has the same order as g. We wish to determine which powers g^n of g have centraliser in $F_4(q)$ equal to T, although for the sake of a straightforward argument we may instead seek the stronger condition that the centralisers of g^n and T in $\mathrm{GL}_{26}(q)$ be equal. It will then be sufficient for g^n to have pairwise distinct eigenvalues other than the two unity eigenvalues, for then any element outside T centralising g^n (by acting non-diagonally on the 1-eigenspace) centralises the whole of T. Furthermore, if all of the eigenvalues A are unity, the same argument shows

[1] That this representation is reducible when $q = 3$ does not pose a problem.

that every element of T is regular semisimple. This leaves the case that A consists of the roots of Q_g. If $P_g = Q_g$, every power of g is regular semisimple by the same argument once again; otherwise, to prove that g^n is regular semisimple it will be sufficient to show that P_{g^n} has at least one root not a root of Q_{g^n}, and hence that the two polynomials share no roots.

Let α_2 be a root of Q_g. For each $0 \leq k \leq 11$, let $n_k > 1$ be minimal subject to the condition that $\alpha_1^{n_k} = \alpha_2^{n_k q^k}$, so that if t is such that g^t has a repeated (non-unity) eigenvalue, it follows that $n_k \mid t$ for some k. Since each n_k also divides $|T|$, the subset of non-regular-semisimple elements of T is then

$$\{g^{n_k t} : 0 \leq k \leq 11, 0 \leq n_k t < |T|\},$$

which is a subset of

$$\{g^{p_k t} : 0 \leq k \leq 11, 0 \leq p_k t < |T|\},$$

where p_k is the smallest prime dividing n_k. The cardinality of this set is weakly bounded below by that of the set

$$\{\pi_k t : 0 \leq k \leq 11, 0 \leq \pi_k t < |T|\},$$

where π_0, …, π_{11} are the twelve smallest primes dividing $|T|$. But every prime dividing $|T|$ is congruent to 1 modulo 12 (see, e.g., [49, A068228]), and the result follows. □

We are seeking the central product $\mathrm{SL}_2(q) \circ \mathrm{Sp}_6(q)$, which is a subgroup of index two in the centraliser whose elements we construct. The following elementary result provides the practical means of bridging this gap.

Lemma 2.7. *Let H be a subgroup of index two of an arbitrary group G. If a uniformly-distributed random subset of H of size n generates H with probability P, then a uniformly-distributed random subset S of G of size $2n-1$ generates at least H with probability at least $\frac{1}{2}P$.*

Proof. The number X of elements of S that lie in H is binomially distributed with parameter $\frac{1}{2}$, so that the probability that $X \geq n$ is $\frac{1}{2}$. □

Lemmas 2.4 to 2.7 together give the following bounds on the required number of iterations of Bray's algorithm:

Lemma 2.8. *Let G be $F_4(q)$ and let $x \in G$ be an involution with centraliser $(\mathrm{SL}_2(q) \circ \mathrm{Sp}_6(q)).2$. Suppose that we use Bray's algorithm to generate elements of $C_G(x)$, and let X_n be the set of these elements after n iterations of the algorithm, and k_n the number of elements of X_n arising from an odd-order commutator. Then if*

$$k_n \geq K := \lceil 1 - 2\log_3\left(\frac{p_1}{102}\right) \rceil$$

for some n, X_n generates the subgroup $\mathrm{SL}_2(q) \circ \mathrm{Sp}_6(q)$ of $C_G(x)$ with probability at least $1 - p_1$. If

$$n \geq N := \frac{1}{P}(K - 1 - P_2 + \sqrt{P_2(P_2 - 2)}),$$

where $12P$ is the proportion given in the statement of Lemma 2.6, then $k_n < K$ with probability at most $\exp(P_2)$.

Proof. The results follow from Lemmas 2.1 and 2.2 and the constants calculated in Lemmas 2.4 to 2.7. The calculation for N additionally uses the Chernoff bound on k_n. □

Accordingly, we run Bray's algorithm until we have found K odd-order commutators, at which point we conclude that we have probably generated $\mathrm{SL}_2(q) \circ \mathrm{Sp}_6(q)$. If we do not find K such after N iterations, we abandon the search and return failure. In the event of a false positive, our algorithm will fail at a later stage.

2.3 Finding the components of the central product in $(\mathrm{SL}_2(q) \circ \mathrm{Sp}_6(q)).2$

Having obtained the involution centraliser $C \cong (\mathrm{SL}_2(q) \circ \mathrm{Sp}_6(q)).2$ (or at least its subgroup $\mathrm{SL}_2(q) \circ \mathrm{Sp}_6(q)$), we require its subgroups isomorphic to $\mathrm{SL}_2(q)$ and $\mathrm{Sp}_6(q)$ occurring in the central product. These are respectively the groups R and L described at the start of this sec-

tion, and we shall denote them as such. We construct generating sets for each by taking random elements of appropriate orders, as described below, terminating when the subgroups defined by these generating sets contain elements of orders proving that the entirety of the desired subgroup is in fact generated.

Lemma 2.9. *Write N for the order of $\mathrm{SL}_2(q)$. Then, given any element $x \in (\mathrm{SL}_2(q) \circ \mathrm{Sp}_6(q)).2$ whose order does not divide N, the element x^N must lie in L.*

Proof. Write $N = 2M$. Notice that x^2 lies in the normal subgroup $\mathrm{SL}_2(q) \circ \mathrm{Sp}_6(q)$ of the extension $(\mathrm{SL}_2(q) \circ \mathrm{Sp}_6(q)).2$, and so raising the element to further powers respects the central product. The maximal tori in $\mathrm{SL}_2(q)$ have orders $q \pm 1$, so that no element of $\mathrm{SL}_2(q)$ has order divisible by $\{N\}_2$, and thus $(x^2)^M = x^N$ has trivial $\mathrm{SL}_2(q)$-component and so lies in L. □

Using this lemma, we construct elements of L repeatedly from random elements of C whose order does not divide that of $\mathrm{SL}_2(q)$, appending these elements successively to a list $\mathscr{L}$. After the construction of each of the second and subsequent of these, samples of random elements are taken from the group generated by $\mathscr{L}$ to check for the existence of elements having orders sharing factors other than 2 with q^2+1 or q^2-q+1. However, there are no maximal subgroups of $\mathrm{Sp}_6(q)$, the complete list of which can be found in [13, pp. 391–2], having order divisible by factors of both these sorts, and so the presence of elements of both such orders guarantees that in fact $\mathscr{L}$ generates all of L.

In order to find the abundances of the elements that we seek in $\mathrm{Sp}_6(q)$, we first enumerate its maximal tori:

Lemma 2.10. *The maximal tori of $\mathrm{Sp}_6(q)$ have the following structures:*

C_{q^3+1}, C_{q^3-1}, $C_{q^2-1} \times C_{q-1}$,
$C_{q^2+1} \times C_{q-1}$, $C_{q^2-1} \times C_{q+1}$, $C_{q^2+1} \times C_{q+1}$,
$C_{q-1} \times C_{q-1} \times C_{q-1}$, $C_{q-1} \times C_{q-1} \times C_{q+1}$, $C_{q+1} \times C_{q+1} \times C_{q-1}$, *and*
$C_{q+1} \times C_{q+1} \times C_{q+1}$.

Proof. This follows immediately from Lemmas 1.27 and 1.28. □

Thus the only tori containing elements with orders having a common factor greater than 2 with q^2+1 are of shapes $C_{q^2+1} \times C_{q-1}$ and $C_{q^2+1} \times C_{q-1}$, and those containing elements having orders sharing factors greater than 3 with q^2-q+1 all have shape C_{q^3+1}. This allows us to count such elements.

Lemma 2.11. *The proportion of elements in* $\mathrm{Sp}_6(q)$ *whose order has a common factor greater than* 3 *with* q^2-q+1 *is at least* $\frac{1}{7}$. *The proportion of elements in* $\mathrm{Sp}_6(q)$ *sharing such factors with* q^2+q+1 *is at least* $\frac{2}{13}$.

Proof. We consider the first case first. Such an element, if it is semisimple, must lie in a torus of shape C_{q^3+1}. By Lemma 1.29, the corresponding centraliser in the Weyl group has order 6. Any element in a torus of this shape whose order does not divide $q+1$ is of the form required, and moreover is regular semisimple, as its eigenvalues on the natural module are distinct (being, as they are, images of one another under the Frobenius automorphism of $\mathbb{F}_{q^6}/\mathbb{F}_q$, and lying in no proper subfield). Thus there are $(q^2-q)(q+1)$ regular semisimple elements of the desired sort in each C_{q^3+1}-torus, and the proportion of such elements in G is

$$\frac{(q^2-q)(q+1)}{6(q^3+1)} = \frac{1}{6}\left(1-\frac{1}{q^2-q+1}\right)$$

by Corollary 1.24, which is at least $\frac{1}{7}$ as $q \geq 3$.

If instead we consider elements having factors of the required sort in common with q^2+q+1, the torus is now of shape C_{q^3-1} and the eigenvalues fall into two orbits under the Frobenius automorphism (which this time is of $\mathbb{F}_{q^3}/\mathbb{F}_q$), but as these must exist in inverse pairs they must all be distinct, and thereafter the argument proceeds as before, with the slightly different orders of the tori accounting for the difference in the estimate of the proportion. □

Lemma 2.12. *The proportion of elements in* $\mathrm{Sp}_6(q)$ *whose order has a common factor greater than* 2 *with* q^2+1 *is at least* $\frac{1}{20}$.

Proof. The argument proceeds as in the preceding lemma. Semisimple such elements lie in $C_{q^2+1} \times C_{q\pm1}$-tori; these both have Weyl-group centralisers of order 8. They are regular semisimple if neither their C_{q^2+1}- nor their $C_{q\pm1}$-component is ±1, and furthermore all such elements in these tori have orders with desired factors. By Corollary 1.24, the proportion of such elements in G is

$$\frac{(q^2-1)(q-3)}{8(q^2+1)(q-1)} + \frac{(q^2-1)(q-1)}{8(q^2+1)(q+1)} = \frac{1}{4}\left(1 - \frac{2(q+1)}{q^2+1}\right),$$

which is at least $\frac{1}{20}$ as $q \geq 3$. □

We are now armed with sufficient information to determine whether any given set of generators of a subgroup of L in fact generates L. However, we also wish to determine when to abandon the construction of further such generators if, at each attempt, we fail to find that the whole of L has been generated. To this end, we use the next result:

Lemma 2.13. *A set $\mathscr{L}$ of size n consisting of elements of L constructed according to the procedure described after Lemma 2.9 generates L with probability at least* $1 - 3(\frac{7}{20})^n$.

Proof. By Lemmas 2.11 and 2.12, in proportion, at least $\frac{1}{7}$ of the elements of $\mathscr{L}$ have orders sharing a factor greater than 3 with q^2-q+1, at least $\frac{2}{13}$ do likewise with q^2+q+1, and at least $\frac{1}{20}$ have orders having a factor greater than 2 in common with q^2+1. Furthermore, for a given element, these three cases are mutually exclusive and exhaustive. Since for $\mathscr{L}$ to generate L it is sufficient for the former to contain two elements corresponding to distinct such cases, the probability that $\mathscr{L}$ *does not* generate L is bounded above by the sum of the probabilities that all the elements of $\mathscr{L}$ correspond to a single case, which is itself bounded above by

$$(\frac{1}{7}+\frac{2}{13})^n + (\frac{2}{13}+\frac{1}{20})^n + (\frac{1}{20}+\frac{1}{7})^n < 3(\frac{1}{7}+\frac{2}{13}+\frac{1}{20})^n < 3(\frac{7}{20})^n$$

□

Once we have found a set of generators for L, we proceed to do likewise for R. We wish

to produce elements of R in a manner analogous to that employed for L above, but since the order of R divides that of L, we cannot identify elements of C straddling both factors in quite the same way. Instead, we search for elements whose orders have sufficiently large prime factors in common with orders of tori from both $\mathrm{SL}_2(q)$ and $\mathrm{Sp}_6(q)$ in combinations *not* occurring inside $\mathrm{Sp}_6(q)$. The orders of the almost-coprime cyclic subgroups of tori are listed in Table 2-A, together with their pairwise greatest common divisors.

Order	$q-1$	$q+1$	q^2+1	q^2+q+1	q^2-q+1
$q-1$	—	2	2	$\gcd(q-1,3)$	1
$q+1$	2	—	2	1	$\gcd(q+1,3)$
q^2+1	2	2	—	1	1
q^2+q+1	$\gcd(q-1,3)$	1	1	—	1
q^2-q+1	1	$\gcd(q+1,3)$	1	1	—

Table 2-A: Orders, and pairwise GCDs thereof, of cyclic subgroups of tori in $\mathrm{Sp}_6(q)$.

The next result describes the orders of the elements that we desire.

Lemma 2.14. *Let n be the order of a semisimple element $c \in C$. Suppose that n has factors in common with*

Case 1: q^2-q+1 *and* $q-1$, *or*

Case 2: q^2+q+1 *and* $q+1$, *or*

Case 3: q^2+1, $q+1$ *and* $q-1$,

where in each case the common factors are greater than the maximum entry in the corresponding row of Table 2-A. In Case 1, $c^{\gcd(n,q^2-q+1)}$ is in R. In Case 2, $c^{\gcd(n,q^2+q+1)}$ is in R. In Case 3, either $c^{\gcd(n,(q^2+1)(q-1))}$ or $c^{\gcd(n,(q^2+1)(q+1))}$ lies in R.

Proof. In each case, requiring that the common factors be greater than the maximum entries of the appropriate table row ensures that c powers up to an element lying in a torus having the heading of that row as a factor of its order.

From Lemma 2.10 we observe that $Sp_6(q)$ has no tori of orders $(q^2-q+1)(q-1)$ or $(q^2+q+1)(q+1)$, so that the linear factors in the orders must correspond to a torus in $\mathrm{SL}_2(q)$, whence

raising c to the power $\gcd(n, q^2 \pm q + 1)$ as appropriate results in an element of R in Cases 1 and 2.

In Case 3, c lies in a torus of rank 4, but the Lie rank of L is only 3 and so c must power up to an element of R of order dividing $q \pm 1$. This completes the proof. □

Remark 2.15. Since there *are* tori of orders $(q^2+1)(q-1)$ and $(q^2+1)(q+1)$ in L, in Case 3 of Lemma 2.14 we do not know *a priori* which of the two powers of c lies in R. However, they can be distinguished by checking for commutativity with L.

Remark 2.16. When $q = 3$, Cases 1 and 3 cannot arise, but this does not present a problem.

Using Lemma 2.14, we construct elements of R by generating random elements of C (discarding any that are not semisimple in the unlikely event that such should appear), and testing their orders against the cases mentioned. As we already have a set of generators for L, we resolve the ambiguous Case 3 by testing for commutativity with all of these generators.

When $q = 3$, this method fails as the C_{q-1}-torus is central, and indeed the semisimple elements of $\mathrm{SL}_2(3)$ only generate its subgroup isomorphic to Q_8. In this case, we also use the 3-singular random elements x that we construct, by raising x to the appropriate power to ensure that its order is 3, and then testing whether it lies in R by checking for commutativity with L.

To determine when the whole of R has been generated, we proceed analogously to the method employed for L above. First, suppose that $q \neq 3$. If the set of elements constructed thus far is $\mathscr{R}$, then the presence of elements (other than the unique involution) in both C_{q+1}- and C_{q-1}-tori in $\langle \mathscr{R} \rangle$ proves that $\langle \mathscr{R} \rangle$ is not a proper subgroup of R. An analysis similar to that in Lemmas 2.11 and 2.12 above shows that the elements of order greater than 2 in C_{q+1}-tori account for at least $\frac{1}{3}$ of the elements in $\mathrm{SL}_2(q)$, and those in C_{q-1}-tori for at least $\frac{1}{4}$.

If $q = 3$, then again the centrality of the C_{q-1}-torus forces us to employ different tactics. Instead, we check $\langle \mathscr{R} \rangle$ for the presence of an element of order 4 (the proportion of which in $\mathrm{SL}_2(3)$ is $\frac{1}{4}$) and another of order divisible by 3 (which occur in proportion $\frac{2}{3}$), which together

generate the whole of R (for otherwise, they would generate a subgroup of index 2, whereas no such subgroup exists).

These proportions also afford us with bounds on the prior probabilities that $\mathscr{R}$ generates R as a function of the size of $\mathscr{R}$, and so allow us to decide when to abandon the construction of further generators.

Lemma 2.17. *The probability that $\mathscr{R}$ fails to generate R is at most $\left(\frac{19}{20}\right)^n + \left(\frac{47}{50}\right)^n$ when $q \neq 3$, and when $q = 3$ it is at most $\left(\frac{79}{81}\right)^n + \left(\frac{25}{26}\right)^n$, where $n = |\mathscr{R}|$.*

Proof. The arguments are similar to those in the proof of Lemma 2.13: we find upper bounds for the proportions of elements in $\mathscr{R}$ lying in a torus of a given shape (or which are 3-singular when $q = 3$), and observe that the whole of R is generated so long as there exist two elements of $\mathscr{R}$ representing distinct such cases. We suppress intermediate numerical calculations this time, and simply refer back to results giving the relevant proportions that are the sources of the final results.

Lemmas 2.11 and 2.12 give lower bound on the proportions of the elements of $\mathrm{Sp}_6(q)$ lying in the tori appearing in the three cases of Lemma 2.14; the same for the $\mathrm{SL}_2(q)$-tori in that Lemma are given in the paragraphs preceding the current result. When $q \neq 3$, we then have that the elements lying in C_{q-1}-tori occur in proportion at most $\frac{3}{50}$, and those in C_{q+1}-tori at most $\frac{1}{20}$, giving the claimed result.

When $q = 3$, the elements of $\mathscr{R}$ are of order either 3 or 4. The results already cited give that the expected proportion in $\mathscr{R}$ of the former sort of elements is at most $\frac{25}{26}$. It then remains to find a *lower* bound on the expected proportion of these 3-singular elements so as to find an upper bound for that of the others. [2] Precisely $\frac{2}{3}$ of the elements of $\mathrm{SL}_2(3)$ are 3-singular, as observed previously; but to count those in $\mathrm{Sp}_6(3)$, we count instead the regular semisimple elements of the group and notice that the proportion of the complement of these elements gives an upper bound on the proportion of the 3-singular elements.

[2] For more general arguments of this sort, see [46] and [29].

Then ten classes of tori in $\mathrm{Sp}_6(q)$ are listed in Lemma 2.10. All elements in those with factors of C_{q-1} must have repeated eigenvalues and so (at least *prima facie*) need not be regular semisimple, so we may discount those tori entirely. The proportions of regular semisimple elements in the remaining tori are given by the calculations in (or similar to those in) the proofs of Lemmas 2.11 and 2.12, and we find at length that at least $\frac{1}{27}$ of the elements of $\mathrm{Sp}_6(3)$ are regular semisimple. Thus in expectation at least $\frac{2}{81}$ of the elements of $\mathscr{R}$ are 3-singular, and the result follows. □

Chapter 3

Root groups and straight-line programs

In the previous chapter, we constructed the involution centraliser $(\mathrm{SL}_2(q) \circ \mathrm{Sp}_6(q)).2$ in $F_4(q)$ and used it to obtain the groups R and L of [31]. At this point we can return to the algorithm set out there, but we make substantial changes to it. Some of these are necessary or useful consequences of our alternative constructions of R and L, but we also make other improvements. We discuss all of these matters in this chapter.

3.1 Finding a complete set of root subgroups in G

The group L obtained in the previous chapter contains root subgroups corresponding to three of the four fundamental roots of $\mathtt{F}_4$. The remaining fundamental root is that which is not perpendicular to the highest root, which itself forms a fundamental system for the $\mathtt{A}_1$ system corresponding to the group R. Overgroups S of R isomorphic to $\mathrm{SL}_3(q)$ are constrained in several useful ways exhibited perspicaciously in Lemma 2.21 of [31]. A suitable such S — which as we shall see is, by that lemma, *any* such S — affords the remaining fundamental root group; thereafter an entire system of root subgroups in G may be constructed. This section

sets forth the details.

3.1.1 Obtaining S

As mentioned in Section 2.2, Kantor and Magaard generate S directly from long root elements, which are themselves constructed from random elements having certain orders. They then find R as a subgroup of S by recognising the latter and applying the resulting isomorphism to an $\mathrm{SL}_2(q)$-subgroup of the natural representation of $\mathrm{SL}_3(q)$. However, since we have already found R as a subgroup of the involution centraliser, we find S by adjoining elements to R.

The procedure for doing so runs as follows: first, R is recognised constructively, which allows us to find a long root element x in R as the image of a transvection in the natural representation. Next, x is conjugated randomly in G to give another long root element of G, which is adjoined to R to give a putative S. The probability that this group is in fact isomorphic to $\mathrm{SL}_3(q)$ is at least $\frac{1}{3}$ by [31, Lemma 2.36].

Finally, S is itself recognised constructively, to give isomorphisms $\Psi_S : \mathrm{SL}_3(q) \to S$ and $\Psi_S^{-1} : S \to \mathrm{SL}_3(q)$. Should S in fact not be isomorphic to $\mathrm{SL}_3(q)$, this recognition will fail and the process is repeated with a different conjugate of x.

Remark 3.1. While there are certainly faster ways to notice that we have generated the whole of $\mathrm{SL}_3(q)$ than employing the full recognition machinery, we will require the isomorphisms Ψ_S and Ψ_S^{-1} later in the algorithm.

We have thus found an overgroup of R that is isomorphic to $\mathrm{SL}_3(q)$, but we have yet to verify that it is compatible with L: we require that the union of the sets of root subgroups (with respect to fixed tori) of S and L extend to a complete set of root subgroups for G. As mentioned at the start of this section, [31, Lemma 2.21(ii)] shows that S and L, as constructed here, are always compatible. We restate this lemma now, specialised to $F_4(q)$.

Lemma 3.2. *Let $\{X_1^+, X_2^+, X_3^+, X_4^+\}$ be a set of fundamental root subgroups in $F_4(q)$, and let X_0^+ be the root subgroup corresponding to the highest root (with respect to the ordering induced by*

the fundamental system). In each case, define X_i^- to be the root subgroup corresponding to the root opposite to that for X_i^+. Suppose that X_1^+ is the unique fundamental root subgroup not commuting with X_0^+. Let $\widehat{S} = \langle X_0^\pm, X_1^\pm \rangle$ and $\widehat{L} = \langle X_2^\pm, X_3^\pm, X_4^\pm \rangle$.

Suppose that $\widehat{L_1}$ is conjugate to $\widehat{L}$ in $F_4(q)$, and that $\widehat{S_1} \leq F_4(q)$ is isomorphic to $\mathrm{SL}_3(q)$*. If furthermore $\widehat{L_1}$ centralises a long* $\mathrm{SL}_2(q)$*-subgroup of $\widehat{S_1}$, then the pair $(\widehat{S_1}, \widehat{L_1})$ is conjugate to $(\widehat{S}, \widehat{L})$ in $F_4(q)$.*

Since our subgroups R and L lie in a central product $R \circ L$ in G, L centralises R, which is a long $\mathrm{SL}_2(q)$-subgroup of S, and so the premises of the lemma are satisfied. This gives us our desired compatibility property.

3.1.2 Finding compatible systems of root groups in S and L

We have established that S and L contain compatible systems of root subgroups, but it remains to find them. We begin by recognising L constructively; this induces a choice of $\mathtt{C}_3$-root subgroups in L. Denote these recognising isomorphisms by $\Psi_L : \mathrm{Sp}_6(q) \to L$ and $\Psi_L^{-1} : L \to \mathrm{Sp}_6(q)$. We have already recognised S, and likewise the isomorphism induces $\mathtt{A}_2$-root subgroups in S. We proceed to adjust the isomorphisms so that the choices of root groups they induce are compatible with one another.

3.1.2.1 The compatibility condition

The union of sets of root subgroups for S and L will extend to one for G precisely when the maximal split tori in S and L with respect to which the root groups were taken (which we shall call T_S and T_L respectively, following [31]) lie in a common maximal split torus T_G in G. This is equivalent to $S \cap L$ lying in $T_S \cap T_L$: the latter is a one-dimensional torus if and only if T_S and T_L both lie in a T_G, in which case $S \cap L = C_S(R)$ certainly lies in T_S, which is the intersection of the induced opposite Borel subgroups of R, and moreover T_S is a direct product of $C_S(R)$ and a one-dimensional torus *not* centralising R, so that $C_S(R)$ lies in T_L also. This is in turn

equivalent to $S \cap L$ being diagonal when mapped to the natural representations of both S and L under their recognising isomorphisms, and it is this property that we endeavour to arrange.

We begin by finding $S \cap L = C_S(R)$. In [31] this is trivial owing the authors' constructions of R and L (as noted in Section 1.6.3, they obtain R via the natural representation of S and so can find $C_S(R)$ similarly, and what is more, this already ensures that T_S lies in $S \cap L$), but for the present purposes we require a further observation in order to do so. This is as follows:

Lemma 3.3. *Let V be the natural module for $\mathrm{SL}_3(q)$ and let $K \leq \mathrm{SL}_3(q)$ be isomorphic to $\mathrm{SL}_2(q)$. Then V has a K-invariant decomposition $U \oplus W$ such that $\dim(U) = 2$ and W is centralised by K. Furthermore, if $1 \neq k \in K$ is semisimple, then k has eigenvalues 1, λ and λ^{-1} on V (considered as an $\mathbb{F}_{q^2}$-space) for some $\lambda \in \mathbb{F}_{q^2}$, and moreover the λ- and λ^{-1}-eigenspaces of k span U. The centraliser of K in $\mathrm{SL}_3(q)$ consists of those elements respecting the decomposition $U \oplus W$ and acting as scalars on U.*

Proof. It is necessary to establish that $\mathrm{SL}_2(q)$ has no faithful, irreducible, three-dimensional representation; then K must be conjugate to the subgroup

$$\begin{pmatrix} \mathrm{SL}_2(q) & 0 \\ 0 & 1 \end{pmatrix} \tag{3.1}$$

of $\mathrm{SL}_3(q)$, and the rest is straightforward. That this is the case may be seen by perusing the list of maximal subgroups of $\mathrm{SL}_3(q)$ in [13, p. 378]: the only maximal subgroup containing an $\mathrm{SL}_2(q)$ is the parabolic 2-space normaliser $q^2 : \mathrm{GL}_2(q)$. Then K with respect to some basis has the form above, which gives the decomposition result immediately, as well as the claimed properties of k. The centraliser of K, with respect to that same basis, is given by the matrices of the form

$$\begin{pmatrix} \mu & 0 & 0 \\ 0 & \mu & 0 \\ 0 & 0 & \mu^{-2} \end{pmatrix}, \tag{3.2}$$

where $0 \neq \mu \in \mathbb{F}_q$. This completes the proof. □

This result suggests the following algorithm for finding $C_S(R)$:

1. Take an arbitrary generator x of R. Recall that this element is semisimple by construction.

2. Let $\mathscr{B}$ be a basis of eigenvectors of $\Psi_S^{-1}(x)$ on the natural module for $\mathrm{SL}_3(q)$. We will suppose that the 1-eigenvector comes in the last place in $\mathscr{B}$. Let $\Phi_{\mathscr{B}}$ be the corresponding change-of-basis map.

3. Replace Ψ_S henceforth by $\Phi_{\mathscr{B}}\Psi_S$ and Ψ_S^{-1} by $\Psi_S^{-1}\Phi_{\mathscr{B}}^{-1}$. Then $\Psi_S^{-1}(x)$ is diagonal.

4. Take a matrix A of the form given in (3.2) with μ primitive in $\mathbb{F}_q$. Then $\Psi_S(A)$ generates $C_S(R)$.

We have thus found $S \cap L = C_S(R)$ in the black-box representation, and furthermore have modified Ψ_S^{-1} so that the maximal split torus it induces in $\mathrm{SL}_3(q)$ meets our compatibility condition. We next do the same for L, by working in its natural representation and diagonalising $S \cap L$ there.

Remark 3.4. We have also arranged that the image of (3.1) under Ψ_S is R, which will be helpful later when labelling root subgroups.

At this point (having found $S \cap L$ and diagonalised it under Ψ_S^{-1}, that is), we can return to the algorithm as given in [31]. We wish to diagonalise $\tilde{A} := \Psi_L^{-1}\Psi_S(A)$ on the natural module for L. This amounts to finding a hyperbolic basis for the module with respect to which $\tilde{A}$ is diagonal; this much is noted in [31], without a particular algorithm for doing so being specified. We use an algorithm that is similar in spirit to the Gram–Schmidt process, but adapted to work with an alternating form and, more significantly, to produce a basis of eigenvectors of T — where T is a linear map with a full spectrum of eigenvalues over $\mathbb{F}_q$ — when given another such basis as its input. We now describe this algorithm in a setting of arbitrary dimension. We will assume that the natural module V for $\mathrm{Sp}_{2n}(q)$ is equipped with an alternating form $(\cdot,\cdot)$.

Let $T \in \mathrm{Sp}_{2n}(q)$ have order dividing $q-1$, and suppose that T has the following eigenvalues:

$$\underbrace{1,\ldots,1}_{2\nu_1},\ \underbrace{-1,\ldots,-1}_{2\nu_{-1}},\ \underbrace{\lambda_1,\lambda_1^{-1},\ldots,\lambda_1,\lambda_1^{-1}}_{2\nu_{\lambda_1}},\ \ldots,\ \underbrace{\lambda_k,\lambda_k^{-1},\ldots,\lambda_k,\lambda_k^{-1}}_{2\nu_{\lambda_k}},$$

where the $\lambda_i^{\pm 1}$ are pairwise distinct and are necessarily in $\mathbb{F}_q$.

Begin by diagonalising T in $\mathrm{GL}_{2n}(q)$ on V by choosing a basis $\mathscr{B}$ of eigenvectors. We denote the elements of $\mathscr{B}$ by $v_{\lambda,i}$, where λ is any eigenvalue of T and i is an integer between 1 and the multiplicity of λ, inclusive. Let P be the matrix with columns given by the elements of $\mathscr{B}$ taken in some order. Then P diagonalises T.

We need to arrange that, if λ and μ are any eigenvalues of T, then $(v_{\lambda,i}, v_{\mu,j})$ is ± 1 iff $\lambda = \mu^{-1}$ and $i = j$ (or, when $\lambda = \pm 1$, for some unique pair of distinct i and j, which we call the "distinguished pair"), and is zero otherwise. In fact, if $\lambda \neq \mu^{-1}$, we already have that

$$(v_{\lambda,i}, v_{\mu,j}) = (Tv_{\lambda,i}, Tv_{\mu,j}) = (\lambda v_{\lambda,i}, \mu v_{\mu,j}) = \lambda\mu(v_{\lambda,i}, v_{\mu,j}), \tag{3.3}$$

so that $(v_{\lambda,i}, v_{\mu,j}) = 0$. The problem reduces, then, to ensuring that, for $\lambda = \lambda_l$, $(v_{\lambda,i}, v_{\lambda^{-1},j})$ is equal to one if $i = j$, and is otherwise zero; while for $\lambda = \pm 1$, we must arrange the same for $(v_{\lambda,i}, v_{\lambda,j})$ with respect to the distinguished pair of values of i and j.

In order to handle the ± 1-eigenspaces uniformly with the others, we need to modify our notation slightly. Let $\lambda \in \{\pm 1, \lambda_i : 1 \leq i \leq k\}$, and define $\overline{v_{\lambda,j}}$ as follows:

$$\overline{v_{\lambda,j}} = \begin{cases} v_{\lambda,j+\frac{\nu_\lambda}{2}} & \text{if } \lambda = \pm 1 \\ v_{\lambda^{-1},j} & \text{otherwise.} \end{cases}$$

If we further define $\lambda_0 = 1$ and $\lambda_{-1} = -1$, then

$$\bigcup_{-1 \leq i \leq k} \left\{ v_{\lambda_i,j}, \overline{v_{\lambda_i,j}} : 1 \leq j \leq \nu_i \right\}$$

is equal to $\mathscr{B}$. We may now describe the algorithm, in which we will abuse notation system-

atically by redefining the various $\overline{v_{\lambda,j}}$, but no ambiguity will result.

Begin by choosing an eigenvalue $\lambda = \lambda_l$ for some integer l such that $-1 \leq l \leq k$, and restrict to the subspace U of V spanned by the λ- and λ^{-1}-eigenspaces. Suppose inductively for $0 \leq m < \nu_\lambda$ that we have already arranged that

$$(v_{\lambda,i}, \overline{v_{\lambda,i}}) = 1, \tag{3.4}$$

$$(v_{\lambda,i}, \overline{v_{\lambda,j}}) = 0, \tag{3.5}$$

$$(v_{\lambda,i}, v_{\lambda,j}) = 0, \text{ and} \tag{3.6}$$

$$(\overline{v_{\lambda,i}}, \overline{v_{\lambda,j}}) = 0 \tag{3.7}$$

for $1 \leq i \leq m$ and $1 \leq j \leq \nu_\lambda$, with $i \neq j$. We next attempt to do the same for $i = m+1$.

We first correct the degenerate case, should it arise: If $(v_{\lambda,m+1}, \overline{v_{\lambda,m+1}}) = 0$, attempt to find an integer $k > m+1$ such that $(v_{\lambda,m+1}, \overline{v_{\lambda,k}}) \neq 0$, and then swap $\overline{v_{\lambda,m+1}}$ and $\overline{v_{\lambda,k}}$. We have chosen k large enough so that this redefinition of notation does not interfere with the inductive hypothesis. If λ is not ± 1 then such a k must exist by the non-singularity of the form, but otherwise, in the event that there is no such k, there is certainly an $l \neq m+1$ such that $(v_{\lambda,m+1}, v_{\lambda,l})$ is nonzero, in which case replacing $\overline{v_{\lambda,m+1}}$ by $\overline{v_{\lambda,m+1}} + v_{\lambda,l}$ has the desired effect, and in particular preserves (3.5) and (3.7).

We can now replace $\overline{v_{\lambda,m+1}}$ by

$$\frac{\overline{v_{\lambda,m+1}}}{(v_{\lambda,m+1}, \overline{v_{\lambda,m+1}})},$$

so that $(v_{\lambda,m+1}, \overline{v_{\lambda,m+1}}) = 1$. Define

$$\overline{v_{\lambda,j}}\prime = \overline{v_{\lambda,j}} - (v_{\lambda,m+1}, \overline{v_{\lambda,j}})\overline{v_{\lambda,m+1}}$$

for $j \neq m+1$. Then $(v_{\lambda,m+1}, \overline{v_{\lambda,j}}\prime) = 0$, and furthermore, for $1 \leq i \leq m$,

$$(v_{\lambda,i}, \overline{v_{\lambda,j}}\prime) = (v_{\lambda,i}, \overline{v_{\lambda,j}}) - (v_{\lambda,m+1}, \overline{v_{\lambda,j}})(v_{\lambda,i}, \overline{v_{\lambda,m+1}});$$

but by the inductive hypothesis, $(v_{\lambda,i}, \overline{v_{\lambda,m+1}})$ is zero, so that $(v_{\lambda,i}, \overline{v_{\lambda,j}}\prime) = (v_{\lambda,i}, \overline{v_{\lambda,j}})$. Then we may replace $\overline{v_{\lambda,j}}$ by $\overline{v_{\lambda,j}}\prime$, with the result that $(v_{\lambda,i}, \cdot)$ now satisfies (3.4) and (3.5) for $1 \leq i \leq m+1$.

If $\lambda \neq \pm 1$ then (3.6) and (3.7) also hold for these values of i by (3.3), but otherwise we must also ensure that $(v_{\lambda,m+1}, v_{\lambda,j}) = (\overline{v_{\lambda,m+1}}, \overline{v_{\lambda,j}}) = 0$ for $m+2 \leq j \leq \nu_\lambda$ (with smaller values of j having been dealt with by the inductive hypothesis). This can be achieved as in the preceding paragraph, i.e. by replacing $v_{\lambda,j}$ with

$$v_{\lambda,j} - (v_{\lambda,m+1}, v_{\lambda,j})\overline{v_{\lambda,m+1}} \tag{3.8}$$

and $\overline{v_{\lambda,j}}$ with

$$\overline{v_{\lambda,j}} + (\overline{v_{\lambda,m+1}}, \overline{v_{\lambda,j}})v_{\lambda,m+1}, \tag{3.9}$$

noting that in each case the inductive results (3.4)–(3.7) are preserved.

It remains only to remark that the above adjustments to the various $v_{\lambda,j}$ and $\overline{v_{\lambda,j}}$ preserve the property that each is a λ- or λ^{-1}-eigenvector of T, as appropriate (in the case of the replacement in (3.8), this relies on the fact that $\lambda = \lambda^{-1}$). Then by induction we can use this algorithm to adjust $\mathscr{B}$ to behave as desired with respect to $(\cdot,\cdot)$ on the whole of U, and then by performing the algorithm for each eigenvalue λ_l in turn the behaviour is extended to the whole of V, and we have obtained a hyperbolic basis diagonalising T. Redefining P to be the matrix with columns given by the elements of $\mathscr{B}$ taken in the order corresponding to the matrix representation of $(\cdot,\cdot)$, we have that $P \in \mathrm{Sp}_{2n}(q)$ and that P diagonalises T.

We use this algorithm to obtain a matrix $P \in \Psi_L^{-1}(L) \cong \mathrm{Sp}_6(q)$ diagonalising $\tilde{A}$. Defining $\Phi_P : \mathrm{Sp}_6(q) \to \mathrm{Sp}_6(q)$ to be the map that conjugates by P^{-1}, and replacing Ψ_L by $\Psi_L\Phi_P$ and Ψ_L^{-1} by $\Phi_P^{-1}\Psi_L^{-1}$, we have at last arranged that $S \cap L$ is diagonal under both Ψ_L and Ψ_S.

3.1.3 Building the remaining root subgroups

The next task at hand is to extrapolate from the structure of S and L made available by the respective isomorphisms Ψ_S and Ψ_L to the structure of the whole of G and an isomorphism $\Psi : F_4(q) \to G$. The algorithm for doing so is described in [31, Sections 2.11–2.12] and summarised here in Section 1.6.3.3. We deviate from this process in one lesser and one greater respect. In order to discuss these we will require names for the fundamental and highest roots of $F_4(q)$, which are given in Figure 3.1.

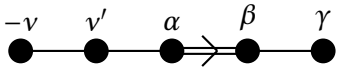

Figure 3.1: Extended Dynkin diagram of $\mathtt{F}_4$, with roots named.

Denote the root group that we construct in G corresponding to a root ρ by X_ρ, and its elements by $X_\rho(t)$, where $t \in \mathbb{F}_q$ is the element's label in the sense of Section 1.6.1.

The elements $X_\rho(t)$ for $\rho \in \mathtt{L}$ are taken to be those induced by [1] Ψ_L. As a result of their construction of the groups R and S, Kantor and Magaard also have the group X_ν — in fact it is the group Z of Section 1.6.3.1 — but we do not. However, the six root subgroups of S induced by Ψ_S are those of the root subsystem generated by ν and ν'; from amongst these we find the *set* $\{X_\nu, X_{-\nu}\}$ by those groups' characteristic property that they commute with both X_α and $X_{-\alpha}$. We may then take either of the groups just found to be X_ν, as $R = \langle X_\nu, X_{-\nu}\rangle$ centralises L. This choice, together with commutativity or otherwise with X_α, determines the roots of $\mathtt{F}_4$ corresponding to the other root subgroups of S also. In fact, since by Remark 3.4 we know the preimage of R under Ψ_S, we may identify the images of the standard root groups of $\mathrm{SL}_3(q)$

[1] This phrasing is somewhat imprecise, although it is hoped that its meaning is clearer than that of the more careful "those induced by the images under Ψ_L of any system of root subgroups in the natural representation with respect to the diagonal maximal split torus and being labelled by the corresponding off-diagonal elements of the representing matrices".

with those in S by the following assignment of roots to the corresponding subgroups:

$$\begin{pmatrix} 1 & \nu & \nu - x \\ -\nu & 1 & -x \\ x - \nu & x & 1 \end{pmatrix},$$

where x is $\nu - \nu'$ if the corresponding root subgroup commutes with X_α, and is otherwise ν'.

We take our arbitrary choice of the element $X_{\nu'}(1)$ in $X_{\nu'}$ to be the image of the element labelled as '1' in the corresponding root group in S. Then we know immediately that $X_{-\nu'}(1)$ is the corresponding image of the opposite root element in S (for the difference in the labellings of the root groups of S via Ψ_S on the one hand and via our own labelling as part of the $\mathtt{F}_4$ root system on the other must be induced by a field automorphism of S). This will be helpful when labelling the root groups, and avoids the use of the linear-algebraic method for labelling $X_{-\nu'}$ given in [31].

More significantly, in that it simplifies the procedure substantially, we take an approach similar to that of Liebeck and O'Brien in [40] when constructing and labelling the root groups. Those authors use the reduced Curtis–Steinberg–Tits presentation as described in [6] so as to require a smaller generating set than the entire set of root subgroups. Beyond the root subgroups of L that we have already identified and labelled, it transpires that we need only label a further four root groups. Indeed, by [40, Section 13.2], the roots whose root groups we require are, in our notation,

$$\begin{matrix} \nu', & \alpha, & \beta, & \gamma, \\ \nu' + \alpha, & \alpha + \beta, & \alpha + 2\beta \text{ and} & \beta + \gamma \end{matrix},$$

together with their negatives. We shall denote the set of these roots by D. Of these, only those with a term in ν' do not lie in the $\mathtt{C}_3$-subsystem $\mathtt{L}$.

Our strategy for constructing the isomorphism $\Psi : F_4(q) \to G$ is then as follows:

1. Define $\Psi(\widehat{X}_\rho(t))$ to be $X_\rho(t)$ for each $\rho \in D \cap \mathtt{L}$ and $t \in \mathbb{F}_q$.

2. Identify $X_{v'}$ as one of the root groups X in the set $\mathscr{R}$ of such induced by Ψ_S as described above.

3. Define $\Psi(\widehat{X}_{v'}(1))$ to be $X(1)$ and $\Psi(\widehat{X}_{-v'}(1))$ to be $Y(1)$, where Y is the root subgroup opposite to X in $\mathscr{R}$.

4. Label the remaining elements of $X_{v'}$ via the action of the maximal split torus of L.

5. Label the remaining elements of $X_{-v'}$ and all of those of $X_{\pm(v'+\alpha)}$ using the Chevalley commutator relations.

3.2 Straight-line programs

The procedure detailed above yields an isomorphism from the standard copy of $F_4(q)$, given by the Steinberg presentation, to the black-box copy G, and allows the calculation of images of arbitrary elements under this isomorphism. This is straightforward, as elements of the standard copy are given as words in the standard generators, black-box elements for all of which are known. To do the same for the inverse isomorphism, however, is more involved: it requires that an arbitrary black-box element $g \in G$ be expressed as a product of the (images of the) standard generators in G. The algorithm for doing so given in [31] takes $O(q)$ time and is described in Sections 2.13 to 2.15 of that paper; we also summarise it here in Section 1.6.3.4. The running-time linear dependence on q is not particularly strong, however, and the changes required to eliminate it are less substantial than was the case for the preprocessing stage that has occupied us thus far in this chapter. Indeed, other than the dependence on smaller-rank recognition algorithms, which we are assuming to run in time polynomial in $\log q$ subject to the existence of an $\mathrm{SL}_2(q)$-oracle, the only $O(q)$ contribution to the running time is the search for a toral element acting in a specified way on an opposite pair of root groups, for which purpose we will give a more efficient method. There are also several points at which we deviate somewhat from the algorithm in [31] for practical reasons, often arising from our different approach to the preprocessing stage.

As already described in Section 1.6.3.4 (the notation from which section we now resume), the straight-line-program algorithm consists of components for performing the following tasks:

- finding an element $q \in Q$ conjugating one long root group opposite X_ν to another;
- finding a toral element $h_{\nu'}(t)$ acting on $X_{\pm\nu}$ in the same way as an element $g \in G$ that normalises those two root groups;
- finding an SLP for a black-box element $q \in Q$; and
- modifying an arbitrary $g \in G$ so that it normalises $X_{\pm\nu}$.

We do not depart materially from [31] in the latter two respects. We now treat the first two in turn.

3.2.1 Conjugating in Q between root groups opposite to X_ν

Given root groups A and B in G both opposite to X_ν, B is conjugated randomly by $\upsilon \in Q$ until $S(\upsilon) := \langle X_\nu, X_{-\nu}, B^\upsilon \rangle$ is isomorphic to $\mathrm{SL}_3(q)$; this group is then recognised and the resulting isomorphism used to find an element conjugating $X_{-\nu}$ to B^υ. This last step is not described further in [31], except to refer readers to earlier works, the most relevant of which to us being [41, Lemma 5.2]. The procedure given there operates by conjugating the maximal split tori in the $\mathrm{SL}_2(q)$-subgroups $\langle X_\nu, X_{-\nu} \rangle$ and $\langle X_\nu, B^\upsilon \rangle$ into one another (where, in the latter case, $\mathrm{SL}_2(q)$-recognition is used in order to find the torus). However, those authors have earlier ensured that the images of X_ν and $X_{-\nu}$ in $\mathrm{SL}_3(q)$ are standard root groups and make use of this fact; we do not have this property and so we must perform some additional work before following the procedure that they describe. It will be convenient in the rest of this section to let Y_ν denote the subgroup $\left\{ \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x & 0 & 1 \end{pmatrix} : t \in \mathbb{F}_q \right\}$ of $\mathrm{SL}_3(q)$, and $Y_{-\nu}$ its transpose.

Having obtained a recognising isomorphism $\Psi_{S(\upsilon)} : \mathrm{SL}_3(q) \to S(\upsilon)$, we first find a matrix $M \in \mathrm{GL}_3(q)$ conjugating $\Psi^{-1}_{S(\upsilon)}(X_\nu)$ to Y_ν, and replace $\Psi_{S(\upsilon)}$ by $\Psi^M_{S(\upsilon)}$. This gets us half-way

to the prerequisites for applying the algorithm in [41], with the remaining requirement being that Y_{-v} should be mapped by $\Psi_{S(v)}$ to X_{-v}. However, we may use that same algorithm to achieve precisely this property, yielding an element $u_1 \in \mathrm{SL}_3(q)$ conjugating $\Psi_{S(v)}^{-1}(X_{-v})$ to Y_{-v}. Applying the algorithm again to obtain $u_2 \in \mathrm{SL}_3(q)$ conjugating $\Psi_{S(v)}^{-1}(B^v)$ to Y_{-v}, we have that $\Psi_{S(v)}(u_2 u_1^{-1})$ conjugates B^v to X_{-v}, and is in Q by construction. [2]

3.2.2 Finding $h_{v'}(t)$ with a given action on $X_{\pm v}$

Recall that the element $g \in G$ for which an SLP pre-image is desired is modified so that it normalises $X_{\pm v}$. Then the action of g on these root groups is the action of the canonical maximal split torus of G, and so there exists an $h_{v'}(t)$ such that $h_{v'}(t)g^{-1}$ centralises those groups. [3] [31] does not detail how such an element is found, although it does remark that the process runs in $O(q)$ time, so that, at least asymptotically, it performs no better than a search through $\mathbb{F}_q$ for the appropriate t. We use $\mathrm{SL}_2(q)$-recognition instead (or more precisely, the image $\Psi_S^{-1}(R)$ inside $\mathrm{SL}_3(q)$, which is already available and requires no further recognition), and simply read the value of t from an appropriately-constructed matrix. [4] Indeed the salient property is that

$$\Psi_S^{-1}(X_v(1)^g) = \begin{pmatrix} 1 & t & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

[2] In [41] the property required is that the elements generated are in the Borel subgroup normalising X_v, and this is what is proven there; but in fact they do indeed lie in the unipotent part.

[3] The existence in general of such an element relies on the Killing form (v, v') being ± 1.

[4] The original paper assumes $\mathrm{SL}_2(q)$-recognition to have a factor of q in its running time, which might explain why this approach was not adopted there.

Chapter 4

Analysis of the algorithm

The purpose of our entire undertaking is to provide an algorithm for recognising $F_4(q)$ after the pattern set out in [31] that is amenable to a practical implementation. Having discussed in earlier chapters our efforts to that end, we now analyse the resulting algorithm in its complexity and reliability. We then conclude the chapter in Section 4.4 with some general commentary on our work.

4.1 A pseudo-order oracle for the algorithm

The use of pseudo-orders was introduced in Section 1.4.3 as a practical alternative to the calculation of genuine orders when implementing group-theoretic algorithms. We now describe a suitable oracle for calculating pseudo-orders of the elements that we encounter in the course of our own algorithm.

4.1.1 The pretend primes

Recall from Section 1.4.3 that the pseudo-order of an element g in some group G is defined with respect to a set of pairwise-coprime integers $\{p_i : 1 \leq j \leq n\}$ referred to as "pretend

primes", and is the least m such that $g^m = 1$ and m is a product of powers of the p_i. For this notion to be well-defined, all primes dividing the exponent of G must occur as a factor of some p_i.

Having observed this, we set out the pretend primes that we shall use, and then proceed to demonstrate their suitability.

Proposition 4.1. *With $q = p^e$ as always, and $p > 2$, the following is a set of pretend primes for $F_4(q)$:*

$$\Pi = \left\{ \begin{array}{ccc} p, & 2, & 3, \\ \{q+1\}_{2',3'}, & \{q-1\}_{2',3'}, & \\ \frac{q^2+1}{2}, & \{q^2+q+1\}_{3'}, & \{q^2-q+1\}_{3'}, \\ \frac{q^4+1}{2}, & q^4-q^2+1 & \end{array} \right\} \setminus \{1\}.$$

Proof. That the elements are pairwise coprime follows by straightforward calculation. Now all semisimple elements lie in some maximal torus, and the exponent of any maximal torus of $F_4(q)$, as listed in [36], may be formed from a product of elements of Π; but since p is also in Π, the same is true of the exponent of $F_4(q)$. □

Remark 4.2. With the exception of 2 and 3, each element of Π occurs with multiplicity at most 1 in the factorisation of the exponent of $F_4(q)$.

4.1.2 Substituting pseudo-orders for orders

These are all the stages of our algorithm at which we must calculate an element's order:

1. finding an involution (Section 2.2.2);
2. centralising the involution (*ibid.*);
3. testing the isomorphism class of the centraliser (*ibid.*);
4. finding classical subgroups of the centraliser (Section 2.3), and in particular:

(a) finding elements with components in both the $\mathrm{SL}_2(q)$- and $\mathrm{Sp}_6(q)$-factors, and

(b) testing whether the whole of $\mathrm{SL}_2(q)$ or $\mathrm{Sp}_6(q)$ has been generated.

We now take each of these stages in turn and show that pseudo-orders may be substituted for orders without modification to the algorithm.

4.1.2.1 Finding an involution

The analysis here is straightforward: the presence of 2 in the set Π of pretend primes guarantees that the 2-part of the calculated pseudo-order n of g is that of the genuine order, and thus $g^{n/2}$ is still an involution when n is even. We shall use this "exactness" property of the 2-part again, and shall refer to it as such.

4.1.2.2 Centralising the involution

We show that Bray's algorithm, including Parker's result on the distribution of the elements it generates, works just as well when pseudo-orders are used in place of actual orders, so long as the 2-part is exact.

Proposition 4.3. *Proposition 1.5, wherein elements of the centraliser of an involution $x \in G$ are generated, holds with the word "pseudo-order" substituted for all occurrences of "order". Furthermore, those elements $\kappa(x,g) := gc^m$ where $c := [x,g]$ has pseudo-order $2m+1$ are uniformly distributed in $C_G(x)$ if the random elements g are uniformly distributed in G.*

Proof. If the 2-part of an element's pseudo-order is exact, then that element has even order if and only if it has even pseudo-order. This ensures that the identification of the two distinct cases in the algorithm remains unchanged. Thereafter, the proof of Proposition 1.5 proceeds as before, as the minimality of the order is not used.

The proof of the uniformity result when using genuine orders is that of Theorem 3.1 in [11]. This holds without change when using pseudo-orders, subject to one minor subtlety:

the definition of $\kappa(x,g)$ depends now on the pseudo-order of c, but as this is an invariant of any element of G, κ remains well-defined. □

Thus our probabilistic results in Section 2.1 stand.

4.1.2.3 Testing the isomorphism class of the centraliser

We detect that the portion of the centraliser that we have generated lies in $\mathrm{Spin}_9(q)$ by testing for the presence of elements of order having a common factor (greater than 2) with q^4+1. An element has such a factor (say n) in its order if and only if it has a factor of $\frac{q^4+1}{2}$ in its pseudo-order, for n is coprime to all of our other pretend primes; these conditions are then equivalent to the element having a factor of n in its *pseudo*-order, and so the test remains the same.

4.1.2.4 Finding elements in $\mathrm{SL}_2(q)$- and $\mathrm{Sp}_6(q)$-subgroups of the centraliser

First we wish to find an element in the centraliser C having factors in common both with $q\pm1$ and with the order of one of the two-dimensional tori in $\mathrm{Sp}_6(q)$, in a combination not occurring in any torus of $\mathrm{Sp}_6(q)$. Now each of the orders of these one- and two-dimensional tori (without the factors of 2 and 3 which we wish to discount in any case) occurs as a pretend prime, and so by essentially the same argument as in the previous paragraph, the sought-out factors exist in the genuine order if and only if the corresponding pretend primes appear in the pseudo-order.

The powers of elements so identified that are the subjects of Lemmas 2.9 and 2.14 continue to lie in the subgroups as asserted by those results when they are stated with respect to pseudo-orders. In the latter case, the resulting elements are in general *different* from those obtained when using genuine orders, but as we only require that the elements lie in the appropriate subgroups, this is of no consequence.

4.1.2.5 Testing whether the entire $\mathrm{SL}_2(q)$- and $\mathrm{Sp}_6(q)$-subgroups have been generated

The tests for the generation of the entirety of these classical subgroups hinge on searches for elements of certain orders. We show now that all of these tests may stand without change.

For $\mathrm{SL}_2(3)$, we search for elements of order 4 and others of order divisible by 3: this works equally well for pseudo-orders owing to the exactness of the 2- and 3-parts.

For $\mathrm{SL}_2(q)$ for larger q, we search for elements having orders sharing factors greater 2 with $q-1$ and others likewise with $q+1$. The presence of $\{q\pm 1\}_{2'}$ as pretend primes ensures that an element has a factor *other than* 2 in its order in common with $q\pm 1$ if and only if its pseudo-order has such, and then the exactness of the 2-part allows the detection of factors of 2^n for $n>1$ in the order, and thus our desired factors occur in an element's pseudo-order precisely when they occur in its order.

Finally, for $\mathrm{Sp}_6(q)$, the sought-out elements have factors in common with $\{q^2+1\}_{2'}$ on the one hand and with q^2-q+1 on the other, but these are both pretend primes.

4.2 Overall reliability

Here we collect the probabilities that, given correct input, each stage of the algorithm succeeds correctly, succeeds falsely or fails (in this last case, necessarily falsely). This being done, we proceed to analyse the overall reliability of the algorithm, both in its preprocessing stage and in finding straight-line programs.

4.2.1 Preprocessing

Table 4-A lists each step in the preprocessing stage of the algorithm, i.e. up to the labelling of the identified root groups in $G\cong F_4(q)$. For each, cross-references are given to the description of that stage and to any relevant probability results, and the probabilities of false success

and failure are described in notation explained below and, where appropriate, in the cross-references.

Stage	Cross-references	Upper bound on probability of false Failure	Success
Find involution x in desired class	Section 2.2.2	0	P
Construct $C := C_G(x)$	Lemma 2.8	p_1	$\exp(P_2)$
Find $\mathrm{Sp}_6(q)$-subgroup L of C	Section 2.3, Lemmas 2.9–2.12	$\frac{1}{2}P$	$\frac{1}{2}P$
Find $\mathrm{SL}_2(q)$-subgroup R of C	Section 2.3, Lemma 2.14	$\frac{1}{2}P$	$\frac{1}{2}P$
Recognising R and L	Section 3.1.1	0	Q
Find $\mathrm{SL}_3(q)$-overgroup S of R	Section 3.1.1	0	Q
Constructing and labelling root groups	Section 3.1.3	*Deterministic*	

Table 4-A: Probabilities of false success and failure for each stage of the algorithm up to root-group labelling.

Remark 4.4. Should the algorithm succeed falsely at any of the stages where this is possible, it will subsequently fail when attempting to recognise R or L. It is thus, even before verifying the Steinberg presentation, a *one*-sided Monte Carlo algorithm.

The probability P where it occurs in Table 4-A may be arranged to be arbitrarily small by performing a suitable number of random trials subject to the analyses given in the relevant sections, and for convenience may in all these cases be taken to be equal. The same is true for the probability Q, except that here there is a dependence on the probability of success of an implementation of a one-sided Monte Carlo algorithm for recognising an appropriate classical group. We will arrange in practice for Q, and also for the probability $p_1 + \exp(P_2)$, to equal P, and so henceforth we shall use the latter notation for all of these.

The table then yields the following result:

Theorem 4.5. *Given correct input, the preprocessing algorithm described in this chapter fails with probability at most* $(1-P)^6$.

In our implementation, we take P to be $\frac{1}{1024}$, taking our lead from the probabilistic analy-

ses in [31]. Successive executions of the algorithm are independent, so the algorithm may be re-run in the event that it fails in order to reduce the probability of a false negative.

Remark 4.6. In fact the probability of failure is much lower than the above analysis shows. See Section 4.4.2.

4.2.2 Straight-line programs

There is less on which to remark regarding the algorithm for finding straight-line programs. There are only two stages at which the algorithm is not deterministic: namely, the construction of a random element y conjugating X_v^g to a long root group opposite X_v, and the attempts at recognising $S(v) := \langle X_v, X_{-v}, X_v^{gyv} \rangle$ as $\mathrm{SL}_3(q)$, where g is the element of G for which a straight-line program is desired. We do not make changes to the procedures set out in [31] for either of these stages, not even to the bounds on the numbers of trials, and so the result in that paper still obtains:

Theorem 4.7 (Cf. [31, Section 2.15])**.** *Given $g \in G$, the algorithm described in this chapter for calculating a straight-line program in $F_4(q)$ for $\Psi^{-1}(g)$ fails with probability at most $\frac{1}{2^8}$.*

4.3 Complexity

The most important aspect of the complexity of the algorithm described in this chapter is that all linear dependencies on q in the running time of the original algorithm in [31] have been eliminated. In this section we give an overview of the running times of each stage of our algorithm.

We reiterate at this point that we are suppressing the costs of performing group operations and of constructing random elements. Equivalently, we may take the assumption to be that these operations can be performed in constant time. Thus, in order to arrive at a running time for the algorithm, we need only consider in the first place any explicit iterations made

by the algorithm of some length dependent on the size of the input, and then also we must take account of the cost of any oracles that we invoke. We look at these oracles now, and then integrate our observations into an analysis of the whole algorithm.

4.3.1 The pseudo-order oracle

As discussed in Section 1.4.3, the Celler–Leedham-Green algorithm for calculating pseudo-orders runs (for our purposes) in $O(\log q)$ time. We can refine this result by considering our particular choice of pretend primes given in Section 4.1.1: except for 2 and 3, each of these occurs with multiplicity one in the exponent of $F_4(q)$, and so the corresponding factors of a pseudo-order are calculated in constant time, so that only the 2- and 3- parts of the exponent of $F_4(q)$ contribute to the running time. Asymptotically, of course, this quantity is still $O(\log q)$.

4.3.2 $\mathrm{Sp}_6(q)$-, $\mathrm{SL}_3(q)$- and $\mathrm{SL}_2(q)$-recognition oracles

The running times of algorithms available in the literature for recognising black-box linear and classical groups are collected in Section 1.5.3. We are, per the standard approach, discounting the cost of $\mathrm{SL}_2(q)$-recognition, so this may be taken to be constant. By Theorems 1.11 and 1.12, recognition both of $\mathrm{Sp}_6(q)$ and of $\mathrm{SL}_3(q)$ is possible in time polynomial in $\log q$. This also holds for the calculation of natural-representation preimages of black-box elements by means of the resulting isomorphisms. We will denote by χ_6 and χ_3 the costs of recognising these respective groups, and likewise by σ_6 and σ_3 the costs of finding preimages.

4.3.3 Analysis of each stage of the algorithm

We are now in a position to find the complexity of each stage of the algorithm in turn.

- The procedure for finding R and L described in Section 2.3 performs a number of operations bounded by an absolute constant. These operations all run in constant time

with the exception of the calculation of pseudo-orders. This stage thus runs in $O(\log q)$ time.

- In Sections 3.1.1 and 3.1.2, the groups R, L and S are recognised. In the latter case, multiple attempts might be required, but the number of these is bounded by a constant. Thus this stage runs in $O(\chi_3 + \chi_6)$ time.

- The algorithm in Section 3.1.2 for finding $C_S(R)$ is essentially a diagonalisation inside $\mathrm{SL}_3(q)$, which takes a constant number of field operations and so $O(\log q)$ time. Beforehand it also evaluates the preimage of a (black-box) generator of R in $\mathrm{SL}_3(q)$, giving a total running time of $O(\log q + \sigma_3)$.

- Similarly, in the same section, the procedure for finding hyperbolic eigenvectors of elements of $\mathrm{Sp}_6(q)$ runs in $O(\log q + \sigma_6)$ time.

- The algorithm at the end of Section 3.1.3 for identifying and labelling the root groups is a little more involved than the above stages. The identification of the (bases for the) root groups lying inside L is a simple assignment and takes $O(\log q)$ time for each of the constant number of root groups. Distinguishing between X_ν and $X_{\nu'}$ takes constant time; assigning the generators of each of these as the standard generators again takes $O(\log q)$ time. The labelling of each of the $O(\log q)$ generators of $X_{\nu'}$ requires the construction of a toral element as a product of six root elements. The labels of each of these that are required are already basis elements, however, so each root element is obtained in constant time. Thus this step takes $O(\log q)$ time. Finally, the remaining three root groups are labelled using the Chevalley commutator relations. Again, the only field elements involved are basis elements, so that the $O(\log q)$ generators are labelled in a total of $O(\log q)$ time. The labelling procedure thus takes $O(\log q)$ time overall.

This completes the preprocessing stage, and so we have the following result:

Theorem 4.8. *Given $G \cong F_4(q)$, the algorithm described in this chapter constructs an isomorphism $\Psi : F_4(q) \to G$ together with its inverse in $O(\log q + \chi_3 + \chi_6 + \sigma_3 + \sigma_6)$ time.*

We next analyse the algorithm for calculating straight-line programs, which is described in Sections 1.6.3.4 and 3.2. We make use here of the notation defined in those sections.

- The construction of an element conjugating X_v^g to a root group opposite X_v requires up to a constant number of random straight-line program constructions in G, which takes $O(\log q)$ time, as this is dominated by the choice of random exponents for the root-group generators.

- Finding the standard form of an element of Q takes $O(\log q)$ time by [31, Proposition 2.37].

- Given two root groups opposite X_v, an element $u \in Q$ is found conjugating one to the other. We require two invocations of this procedure. It begins by making up to a constant number of attempts at $\mathrm{SL}_3(q)$-recognition. Next, the basis of the recognised $\mathrm{SL}_3(q)$ is changed in $O(\log q)$ time, and then the algorithm of Lemma 5.2 of [41] is applied twice, which, discounting $\mathrm{SL}_2(q)$-recognition, also runs in $O(\log q)$ time.

- The procedure described in Section 3.2.2 for constructing the element $h_{v'}(t)$ requires one $\mathrm{SL}_3(q)$-preimage and, once t has been found, the construction of $h_{v'}(t)$ as a product of root elements. In contrast to the toral elements constructed in the preprocessing stage, there is no guarantee that t is a canonical basis element for $\mathbb{F}_q$, and so in total this stage takes $O(\sigma_3 + \log q)$ time.

- These ingredients combine to give an element in L, a preimage for which is then found in $O(\sigma_6)$ time.

This amounts to:

Theorem 4.9. *Given $g \in G$, the algorithm described in this chapter calculates a straight-line program for $\Psi^{-1}(g)$ in $O(\log q + \sigma_3 + \sigma_6)$ time.*

Remark 4.10. Images of elements $x \in F_4(q)$ under Ψ are found in time proportional to the word-length of x, specified as it is by a word in the Steinberg generators, and images of each

of these in G were calculated at the preprocessing stage. If we further assume that x is in Bruhat form, then this improves to constant time: cf. [31, Remark 2.40].

4.4 Concluding comments

We end this chapter with some observations on the algorithm described in the preceding chapters.

4.4.1 Applicability to other exceptional groups

The algorithm in [31] that we took as our starting point applies to exceptional groups of Lie rank greater than 2. In principle, much of our own work either applies or could be adapted to apply to other such groups than $F_4(q)$.

There are analogues of the involution centraliser in $F_4(q)$ having a subgroup of shape $R \circ L$ in the other exceptional groups G, where R is a long $\mathrm{SL}_2(q)$ as before, but the group L is not $\mathrm{Sp}_6(q)$ but a different classical or exceptional group; these are listed in [31, Table 2.15]. The method by which the desired class of centraliser is identified would need to be adjusted to account for the classes occurring in each group. The different structures of these centralisers are set out in [28, Table 4.5.1].

Separating the factors R and L in the manner of Section 2.3 would require an easy analysis of the tori in L. This (as well as the orders of the tori in G) would also inform the choice of pretend primes for the order oracle.

Ensuring the compatibility of $S > R$ and L would need an analogue of the symplectic algorithm in Section 3.1.2, the difficulty of which being dependent on the structure of L: when $G \cong E_6(q)$ then $L \cong \mathrm{SL}_6(q)$ and the matter is straightforward, but when $G \cong E_8(q)$, then $L \cong E_7(q)$ and the process would be rather more involved.

Having adjusted Ψ_S and Ψ_L for mutual compatibility, the rest of the algorithm, viz. the

conclusion of the preprocessing stage discussed in Section 3.1.3 and the straight-line-program algorithm of Section 3.2, applies essentially without modification, with the only additional consideration necessary being the choice of appropriate pairs of roots (in order that they have appropriate structure constants) when finding labels for root elements.

The probability estimates of Chapter 2, which are of a distinctly *ad hoc* flavour, would require analogues for each of the other exceptional groups. The asymptotic arguments of [51] might in many cases suggest suitable strategies. It is also conceivable that results generic to all of the relevant exceptional groups could be formulated.

Kantor and Magaard also give an algorithm for recognising exceptional groups of rank 2, which mimics their algorithm for the larger-rank groups but requires some additional care, for example, in the identification of elements in Q. In this respect it is more distant from our own algorithm, but many of the principles that we use still apply: in particular, analogous involution centralisers still exist. In $G_2(q)$ there is the complication that $R \cong L \cong \mathrm{SL}_2(q)$, where L is *short*, so that these groups are harder to distinguish than is the case in the present work. A strategy for attacking this problem is described by Magaard in the unpublished manuscript notes in [44].

4.4.2 Probability estimates

Many of our probability estimates are wildly pessimistic. In most cases these results are used to obtain an upper bound on the number of attempts to make at forming some construction with a certain desired property, and as such affect the running time in practice only when the input group G is not in fact $F_4(q)$, which is not a case for which we are especially concerned to optimise. On the other hand, Lemma 2.8 and the results on which it depends determine the number of generators constructed to define the involution centraliser $(\mathrm{SL}_2(q) \circ \mathrm{Sp}_6(q)).2$, and the various results on element proportions in $\mathrm{SL}_2(q)$ and $\mathrm{Sp}_6(q)$ determine the number of iterations required to conclude that a given group is probably isomorphic to one of these groups, and so improvements in these estimates would result in lower running times even

when $G \cong F_4(q)$. In some cases the existing results yield notably better estimates if q is restricted to be larger than a small constant, say 3 or 5; and so these better probabilities could be used in general with special cases for small fields. More sophisticated arguments could also be employed to give better estimates: for example, we at times restrict ourselves to considering single classes of tori when in fact multiple such contain suitable elements.

Chapter 5

Implementation commentary

We provide an implementation of the algorithm described in the preceding chapters in the computer algebra system MAGMA [10]. The code is available on the enclosed CD, but this chapter contains a discussion of that implementation, and constitutes the module's documentation.

5.1 Overview

The organisation of package is described in the table below. Only the more significant files are mentioned.

`src/`	**Source files.**
`src/bray.m`	Implements Bray's algorithm for centralising an involution.
`src/pseudo-order.m`	Functions for calculating pseudo-orders.
`src/randomelt.m`	Random-element generation functions.
`src/util.m`	General-purpose utility functions.
`src/F4/`	**Source files specific to $F_4(q)$.**
`src/F4/f4.m`	General routines for handling presentations and elements of $F_4(q)$.
`src/F4/find-central-product.m`	Implementation of algorithm for finding the desired central product of subsystem subgroups. Cf. Section 2.2.2.
`src/F4/sl2-sp6-separator.m`	Functions for finding the subsystem subgroups of the involution centraliser $(\mathrm{SL}_2(q) \circ \mathrm{Sp}_6(q)).2$. Cf. Section 2.3.
`src/F4/slp.m`	Functions for finding SLPs for black-box elements. Cf. Section 3.2.
`src/F4/subsystem-groups.m`	Functions for ensuring compatibility between subsystem subgroups. Cf. Section 3.1.2.
`t/`	**Test scripts.**

5.2 Function documentation

In this section we document the functions that are implemented in the module, organised according to the files in which they are contained.

5.2.1 `bray.m`

5.2.1.1 `InvCentGen`

Parameters:

`g`	GrpFinElt	Element to centralise.
`RandomGenerator`	UserProgram	Random-element generator for group in which to centralise g.
`OrderOracle`	UserProgram	Order oracle for group in which to centralise g.

Returns a sequence consisting either of a single element or of two elements in the centraliser of g in the group from which `RandomGenerator` generates random elements, constructed using Bray's algorithm: if the commutator used to construct the elements had odd order, only one element is returned; otherwise there are two. This distinction is important in our use of this function.

5.2.2 `pseudo-order.m`

5.2.2.1 `PseudoOrder`

Parameters:

`g`	GrpFinElt	Element.
`PretendPrimes`	SeqEnum	Sequence of pretend primes.

Returns the pseudo-order of g with respect to `PretendPrimes`.

5.2.2.2 `PrimePowerOrder`

Parameters:

`g`	GrpFinElt	Element. Its order must be a power of `p`.
`p`	RngIntElt	Prime.
`MaxPower`	RngIntElt	An integer such that $g^{p^{\texttt{MaxPower}}} = 1$. Used as an upper bound in the search.

Finds the smallest l such that $g^{p^l} = 1$, and returns p^l.

5.2.3 `randomelt.m`

5.2.3.1 `MakeRandomGenerator`

Parameters:

`G`	GrpFin	Group for which to construct a random-element generator.

Returns a zero-argument function that itself returns pseudo-random elements of `G` constructed according to the product replacement algorithm.

5.2.3.2 `FindRandomElt`

Parameters:

`G`	GrpFin	Group from which to construct random elements.
`Condition`	UserProgram	Function testing elements for the desired property.
`MaxTrials`	RngIntElt	Maximum number of random elements to generate in attempting to find one satisfying `Condition`.
`RaiseError`	BoolElt	(Default: `false`.) Determines the behaviour when `MaxTrials` unsuccessful trials have been made: if `true`, an error is raised; if `false`, the identity of `G` is returned.
`RandomGenerator`	UserProgram	(Default: `false`.) Function to generate random elements of `G`. If `false`, `MakeRandomGenerator(G)` is used.

This function generates elements at random from the group `G` until one satisfying the condition `Condition` is found, and returns that element.

5.2.3.3 `FindEltDivisible`

Parameters:

`G`	GrpFin	Group from which to construct random elements.
`n`	RngIntElt	Integer that must divide the random element's order.
`MaxTrials`	RngIntElt	Cf. `FindRandomElt`.
`RaiseError`	BoolElt	(Default: `false`.) Cf. `FindRandomElt`.
`RandomGenerator`	UserProgram	(Default: `false`.) Cf. `FindRandomElt`.

This function generates elements at random from the group `G` until one is found with

order divisible by n.

5.2.3.4 FindEltNotDividing

This function takes the same parameters as FindEltDivisible, but the sense of n is changed so that a random element is sought whose order *is not a factor of* n.

5.2.3.5 FindEltSharingFactors

Parameters:

G	GrpFin	Group from which to construct random elements.
n	RngIntElt	Integer that must share factors with the random element's order.
MaxTrials	RngIntElt	Cf. FindRandomElt.
GCDThreshold	RngIntElt	(Default: 1.) Strict lower bound for the desired greatest common divisor of n and the element's order.
RaiseError	BoolElt	(Default: false.) Cf. FindRandomElt.
RandomGenerator	UserProgram	(Default: false.) Cf. FindRandomElt.

This function searches G for a random element having order x such that $\gcd(x, \mathtt{n}) >$ GCDThreshold, and returns that element.

5.2.4 util.m

5.2.4.1 CoprimePart

Parameters:

a, b	RngIntElt

Suppose that a is a product of prime powers $p_1^{e_1} \dots p_k^{e_k} p_{k+1}^{e_{k+1}} \dots p_n^{e_n}$, where $p_i | b$ if and only if $i > k$. Then `CoprimePart(a, b)` returns $p_1^{e_1} \dots p_k^{e_k}$, which is the largest factor of a coprime to b.

5.2.4.2 `Swap`

Parameters:

`seq`	SeqEnum
`i, j`	RngIntElt

Modifies `seq` so that `seq[i]` and `seq[j]` are exchanged.

5.2.5 `F4/f4.m`

We first define some terminology relating to the internal representation of the reduced root system: each positive root is assigned a 1-based *root number*. The root numbers of negative roots are the negations of that of the corresponding positive roots. In contrast, we also define *root indices*, which index locations (after being scaled) in an internal table of commutator relations.

Several functions take a parameter `G` that is a finitely-presented group which we quotient successively until it is isomorphic to $F_4(q)$. Their use on other groups is undefined. Likewise, there is an implicit basis of $\mathbb{F}_q$ encoded in the generators of `G` and to which some of the functions below refer. The basis is used explicitly only in `ShortF4` itself, but indices of basis vectors are used by other functions. We will sometimes refer to this representation of $F_4(q)$ as the "white-box" representation.

5.2.5.1 `IsSumRoot`

Parameters:

`RootNum1, RootNum2`	RngIntElt	Root numbers of two roots in the reduced root system.

Returns `true` if the sum of the two specified roots is itself a root, and `false` otherwise.

5.2.5.2 `NumberedGen`

Parameters:

`G`	GrpFP	The group being constructed. See introduction to this section.
`e`	RngIntElt	The dimension of $\mathbb{F}_q$ as a vector space over $\mathbb{F}_p$.
`Base`	RngIntElt	Root number.
`Index`	RngIntElt	Index of the desired basis element.

Returns the generator of `G` specified by the corresponding root and basis element of $\mathbb{F}_q$.

5.2.5.3 `RootEltIndex`

Parameters:

`RootNumber`	RngIntElt	Root number.
`e`	RngIntElt	The dimension of $\mathbb{F}_q$ as a vector space over $\mathbb{F}_p$.
RngIntElt `Index`	RngIntElt	Index of the desired basis element.

Returns the index corresponding to the given root element, specified by its root number and basis index, in the sense outlined at the start of this section.

5.2.5.4 `ShortF4`

Parameters:

`p, e`	RngIntElt	Integers such that `p` is prime and $q = \mathtt{p}^{\mathtt{e}}$.

Returns the reduced Curtis–Steinberg–Tits presentation of $F_4(q)$ as a group in the category `GrpFP`.

5.2.5.5 `CommutatorLHS`

Parameters:

`G`	GrpFP	The group being constructed. See introduction to this section.
`e`	RngIntElt	The dimension of $\mathbb{F}_{\mathtt{q}}$ as a vector space over $\mathbb{F}_p$.
`Base1`	RngIntElt	Root number of first term in commutator.
`Index1`	RngIntElt	Index of the desired basis for the first term in the commutator.
`Base2`	RngIntElt	Root number of second term in commutator.
`Index2`	RngIntElt	Index of the desired basis for the second term in the commutator.

Returns the commutator $[X_a(s), X_b(t)]$, where a and b are respectively the roots specified by `Base1` and `Base2`, and s and t are respectively the basis elements of $\mathbb{F}_{\mathtt{q}}$ specified by `Index1` and `Index2`.

5.2.5.6 `CommutatorRHSTerm`

Parameters:

`G`	GrpFP	The group being constructed. See introduction to this section.
`e`	RngIntElt	The dimension of $\mathbb{F}_q$ as a vector space over $\mathbb{F}_p$.
`RootBase`	RngIntElt	Root number of desired root group.
`ParamSeq`	SeqEnum	Co-ordinates of the parameter t of the root element with respect to the fixed basis of $\mathbb{F}_q$.

Returns a root element $X_a(t)$, where a is the root specified by `RootBase` and t is as defined above.

5.2.5.7 `EvalF4RootElt`

Parameters:

`F4`	GrpFP/SeqEnum	$F_4(q)$.
`rt`	RngIntElt	Root index in the sense of `NumberedGen`.
`t`	FldFinElt	Parameter in $\mathbb{F}_q$ of the root element to evaluate.
`e`	RngIntElt	The dimension of $\mathbb{F}_q$ as a vector space over $\mathbb{F}_p$.

Returns a root element $X_{\mathtt{rt}}(t)$. This function works both for the white-box case, when `F4` is a `GrpFP`, and also for the black-box case, when `F4` is a sequence of basic root elements in the usual order.

5.2.5.8 `RecogniseF4Odd`

Parameters:

`G`	GrpFin	Group isomorphic to $F_4(q)$.
`p`	RngIntElt	Parameter such that $q = \mathtt{p}^{\mathtt{e}}$.
`e`	RngIntElt	Parameter such that $q = \mathtt{p}^{\mathtt{e}}$.

Recognises `G` and returns a pair of isomorphisms, the first being from $F_4(q)$ to `G` and the second being its inverse.

5.2.5.9 `F4PretendPrimes`

Parameters:

`p`	RngIntElt	Parameter such that $q = \mathtt{p}^{\mathtt{e}}$.
`e`	RngIntElt	Parameter such that $q = \mathtt{p}^{\mathtt{e}}$.

Returns a set of pretend primes suitable for the algorithm for recognising $F_4(q)$. These are described in Section 4.1.

5.2.5.10 `MakeF4OrderOracle`

Parameters:

`p`	RngIntElt	Parameter such that $q = \mathtt{p}^{\mathtt{e}}$.
`e`	RngIntElt	Parameter such that $q = \mathtt{p}^{\mathtt{e}}$.

Returns an order oracle for $F_4(q)$ suitable for the algorithm for recognising that group.

See also: F4PretendPrimes

5.2.5.11 `MakeQ`

Parameters:

`F4`	RngIntElt	White-box $F_4(q)$.
`F4RootElements`	SeqEnum	Basic root elements in the usual order.
`p`	RngIntElt	Parameter such that $q = \mathtt{p}^{\mathtt{e}}$.
`e`	RngIntElt	Parameter such that $q = \mathtt{p}^{\mathtt{e}}$.

Constructs the group Q as defined in Section 1.6.3.4.

5.2.5.12 `SLPQ`

Parameters:

`Q`	GrpFin	The group returned by `MakeQ`.
`R`	GrpFin	The standard subgroup $R \cong \mathrm{SL}_2(q)$.
`PsiSL3`	UserProgram	Isomorphism from the standard group S to $\mathrm{SL}_3(q)$.
`x`	GrpFinElt	Element of `Q` whose straight-line program is desired.

Returns a straight-line program for `x` as an element of the white-box representation of $F_4(q)$.

5.2.5.13 `LMO`

Parameters:

`X`	GrpFin	Root subgroup of $\mathrm{SL}_3(q)$ opposite to the bottom-left root subgroup.
`p`	RngIntElt	Parameter such that $q = \mathtt{p}^{\mathtt{e}}$.
`e`	RngIntElt	Parameter such that $q = \mathtt{p}^{\mathtt{e}}$.

Implements the Lübeck–Magaard–O'Brien algorithm for constructing elements conjugating between root subgroups of $\mathrm{SL}_3(q)$, as detailed in [41]. Returns an element conjugating `X` to the top-right root subgroup of $\mathrm{SL}_3(q)$. See Section 3.2.

5.2.5.14 `FindQOpposer`

Parameters:

`Q`	GrpFin	The group returned by `MakeQ`.
`X`	GrpFin	Root subgroup of $F_4(q)$ opposite X_ν.
`X_nu, X_minusnu`	GrpFin	X_ν and $X_{-\nu}$ respectively, in the usual notation.
`p`	RngIntElt	Parameter such that $q = \mathtt{p}^{\mathtt{e}}$.
`e`	RngIntElt	Parameter such that $q = \mathtt{p}^{\mathtt{e}}$.

Returns an element of `Q` conjugating `X` to `X_minusnu`.

5.2.6 `F4/slp.m`

5.2.6.1 `SLP`

Parameters:

`g`	GrpFinElt	The element for which a straight-line program is required.
`G`	GrpFin	Group isomorphic to $F_4(q)$.
`F4`	RngIntElt	White-box $F_4(q)$.
`Q`	GrpFin	The group returned by `MakeQ`.
`R`	GrpFin	The standard subgroup $R \cong \mathrm{SL}_2(q)$.
`X_nu, X_minusnu`	GrpFin	X_ν and $X_{-\nu}$ respectively, in the usual notation.
`z`	GrpFinElt	$X_\nu(1)$.
`PsiSL3`	UserProgram	Isomorphism from the standard group S to $\mathrm{SL}_3(q)$.
`PsiSp6`	UserProgram	Isomorphism from the standard group L to $\mathrm{Sp}_6(q)$.
`PsiF4Inv`	UserProgram	Isomorphism from `F4` to `G`.
`p`	RngIntElt	Parameter such that $q = \mathtt{p}^{\mathtt{e}}$.
`e`	RngIntElt	Parameter such that $q = \mathtt{p}^{\mathtt{e}}$.
`n`	GrpFinElt	Image of $n_\nu(1)$ in `G`.
`n_inv_wb`	GrpFinElt	$n_\nu(-1)$.
`h_nuprime`	UserProgram	Function mapping t to $h_{\nu'}(t)$.

Returns a pair x, y such that $g = xy$, where $x \in L \cong \mathrm{Sp}_6(q)$ and $y \in F_4(q)$ is expressed as a straight-line program in the generators of $F_4(q)$.

5.2.7 `F4/find-central-product.m`

5.2.7.1 `FindCentralProduct`

Parameters:

`G`	GrpFin	$F_4(q)$.
`q`	RngIntElt	The order q of the field.
`OrderOracle`	UserProgram	Order oracle for $F_4(q)$.

This function attempts to find an involution isomorphic to $(\mathrm{SL}_2(q) \circ \mathrm{Sp}_6(q)).2$ or the subgroup thereof of index 2, according to the algorithm described in Section 2.2.2. It will raise an error if it demonstrably fails to do so after a fixed number of iterations, but the algorithm is *two*-sided Monte Carlo, and even when the function returns, its return value is a subgroup of `G` which is probably but not guaranteed to be of the desired form: it may also be a proper subgroup of such, or even in the wrong class.

5.2.7.2 `MustBeInSpin`

Parameters:

`g`	GrpFinElt	An element in an involution centraliser in $F_4(q)$.
`q`	RngIntElt	The order q of the field.
`OrderOracle`	UserProgram	Order oracle for $F_4(q)$.

This function checks the order of g for factors larger than 2 in common with q^4+1 which, as discussed in Section 2.2.2, indicates that the centraliser in which g lies must be isomorphic to $\mathrm{Spin}_9(q)$.

See also: FindCentralProduct

5.2.8 F4/SL2Sp6Separator.m

5.2.8.1 SL2Sp6Separator

Parameters:

InvCent	GrpFin	An involution centraliser in $F_4(q)$ isomorphic to $(\mathrm{SL}_2(q) \circ \mathrm{Sp}_6(q)).2$.
q	RngIntElt	The order q of the field.
OrderOracle	UserProgram	Order oracle for $F_4(q)$.

This function implements the algorithm described in Section 2.3 to find the commuting subgroups Sp6 and SL2 isomorphic respectively to $\mathrm{Sp}_6(q)$ and $\mathrm{SL}_2(q)$ in the involution centraliser $(\mathrm{SL}_2(q) \circ \mathrm{Sp}_6(q)).2$. It returns the two-element sequence Sp6, SL2 on success, or raises an error on failure.

See also: FindSL2Element, IsAllOfSL2, IsAllOfSp6.

5.2.8.2 FindSL2Element

Parameters:

InvCent	GrpFin	An involution centraliser in $F_4(q)$ isomorphic to $(\mathrm{SL}_2(q) \circ \mathrm{Sp}_6(q)).2$.
Sp6	GrpFin	A subgroup of InvCent isomorphic to $\mathrm{Sp}_6(q)$.
q	RngIntElt	The order q of the field.
OrderOracle	UserProgram	Order oracle for G.
RandomIC	UserProgram	Random-element generator for InvCent.

This function finds an returns a semisimple element of the unique $\mathrm{SL}_2(q)$-subgroup of InvCent commuting with Sp6, and is called by SL2Sp6Separator for this purpose when

building this $SL_2(q)$-subgroup according to the procedure outlined in Section 2.3, using the results of Lemma 2.14. It raises an error if it cannot find such an element.

See also: `SL2Sp6Separator`.

5.2.8.3 `IsAllOfSL2`

Parameters:

`G`	GrpFin	A group isomorphic to a subgroup of $SL_2(q)$.
`q`	RngIntElt	The order q of the field.
`OrderOracle`	UserProgram	Order oracle for `G`.

This function attempts to determine whether `G` is isomorphic to $SL_2(q)$, under the assumption that it is at least isomorphic to a subgroup of $SL_2(q)$. It does so by searching for non-central elements in C_{q+1}- and C_{q-1}-tori, the justification for which being given at the end of Section 2.3, and returns `true` if elements of the form given there are found, or `false` otherwise. It is called by `SL2Sp6Separator` on a list of elements constructed by `FindSL2Element`.

See also: `SL2Sp6Separator`.

5.2.8.4 `IsAllOfSp6`

Parameters:

`G`	GrpFin	A group isomorphic to a subgroup of $Sp_6(q)$.
`q`	RngIntElt	The order q of the field.
`OrderOracle`	UserProgram	Order oracle for `G`.

This function, analogously to `FindSL2Element`, attempts to determine whether `G` is isomorphic to $Sp_6(q)$, returning `true` or `false` accordingly, using the criteria expounded imme-

diately after Lemma 2.9. It is called by `SL2Sp6Separator` on a list of elements constructed according to the specification of that lemma.

See also: `IsAllOfSL2`, `SL2Sp6Separator`.

5.2.9 `F4/SubsystemGroups.m`

5.2.9.1 `MakeCompatibleSubsystemGroups`

Parameters:

`SL2, Sp6`	GrpFin	The two respective components of the involution centraliser $(SL_2(q) \circ Sp_6(q)).2$.
`G`	GrpFin	The ambient group isomorphic to $F_4(q)$ that we are attempting to recognise.
`p`	RngIntElt	Parameter such that $q = p^e$.
`e`	RngIntElt	Parameter such that $q = p^e$.

This function returns `SL3, Phi, PhiInv, Psi, PsiInv`, where `SL3` is an $SL_3(q)$-overgroup of `SL2`; `Phi` is an isomorphism from `SL3` to the natural representation of $SL_3(q)$, and `PhiInv` is its inverse; and `Psi` is an isomorphism from `Sp6` to $Sp_6\, q$, and `PsiInv` is its inverse. `Phi` and `Psi` are such that the images under each of the intersection of `SL3` and `Sp6` are diagonal. The algorithm employed is described in Sections 3.1.1 and 3.1.2.

See also: `SL2Sp6Separator`, `SymplecticDiagonalisingMatrix`.

5.2.9.2 `SymplecticDiagonalisingMatrix`

Parameters:

`x`	GrpMatElt	An element of `SymplecticGroup(d, q)`, for some `d` and `q`, having a full spectrum of eigenvalues in `GF(q)`.

This function constructs and returns an element `P` of `SymplecticGroup(d, q)` such that `P * x * P^(-1)` is diagonal. The algorithm employed is described in Section 3.1.2.

5.3 The testing framework

The implementation is supplemented by an automated testing framework to aid regression testing in future development. It is contained in the `t/` directory, and consists of a collection of routines for testing conditions and logging the results, as well as batches of tests for some components of the system. This section documents the framework.

5.3.1 Test operation

Each test script, when loaded, immediately runs all of the tests that it contains. The results are printed according to the Test Anything Protocol (TAP) [39]. Although the test scripts may be be run directly, they are designed in such a way that it is also possible to use the functionality in the Perl `TAP::Parser` module [53] to produce a test harness to automate the running of the tests and parsing of the results.

5.3.2 Overview of the tests

Each test script `RoutineName.t` tests one aspect of the module as comprehensively as is practical. In this section, the tests for each routine are summarised briefly, under the filename of

the test script.

5.3.2.1 `SymplecticDiagonalisingMatrix.m`

For simplicity, the tests are restricted to the six-dimensional case, as this is all that is used by the algorithm. For each field F of degree less than or equal to five over a prime field of order less than 100, the function is tested on random matrices in $\mathrm{Sp}_6(F)$ (with respect to a fixed seed to allow the results to be reproduced) having an assortment of eigenvalues in that field; it is tested that the resulting matrix does indeed diagonalise the input matrix, but not whether it is symplectic, as `SymplecticDiagonalisingMatrix` coerces its return value into the appropriate symplectic group.

5.3.2.2 `PseudoOrder.m`

With $q = p^e$ as usual, for each prime p less than 50 and each value of e between 1 and 5, the set of pretend primes for $F_4(q)$ is constructed using `F4PretendPrimes` and these are verified to be pairwise coprime. Then a random element of $F_4(q)$ is constructed and its genuine order o and its pseudo-order o' are calculated, and for each pseudo-prime t, it is checked that $\gcd(o, t) > 1$ if and only if $\gcd(o', t) > 1$.

5.3.3 Testing support routine documentation

The routines in `t/simple.m`, which handle the generation of TAP output from test results, are documented here.

5.3.3.1 `ok`

Parameters:

`TestResult`	BoolElt	A flag indicating whether the test passed.
`Description`	MonStgElt	(Default: `""`.) Description of the test for the TAP line.
`TestNum`	RngIntElt	(Default: `0`.) Number of the test. Set to zero to suppress its output.

This procedure generates a TAP line for a single test. It prints "`ok`" or "`not ok`" respectively as `TestResult` is `true` or `false`, and also prints the test number and description if these are supplied. It mimics the behaviour of the `ok` subroutine in the `Test::Simple` CPAN module, which is documented at [54].

5.3.3.2 `PrintPlan`

Parameters:

`NumTests`	RngIntElt	The number of intended tests.

This procedure prints a TAP plan specifying the number of tests that are intended to be performed by the current script. The TAP protocol requires that this appear either at the very beginning or the very end of the TAP output (but not both), and so this procedure should be called at the appropriate time.

Bibliography

[1] D. Angluin and L. G. Valiant. Fast probabilistic algorithms for Hamiltonian circuits and matchings. *J. Comput. System Sci.*, 18(2):155–193, 1979.

[2] M. Aschbacher. On the maximal subgroups of the finite classical groups. *Invent. Math.*, 76(3):469–514, 1984.

[3] H. Bäärnhielm. Recognising the Ree groups in their natural representations. Unpublished.

[4] H. Bäärnhielm. Tensor decomposition of the Suzuki groups. Unpublished.

[5] H. Bäärnhielm. Recognising the Suzuki groups in their natural representations. *J. Algebra*, 300(1):171–198, 2006.

[6] L. Babai, A. J. Goodman, W. M. Kantor, E. M. Luks, and P. P. Pálfy. Short presentations for finite groups. *J. Algebra*, 194(1):79–112, 1997.

[7] L. Babai and R. Beals. A polynomial-time theory of black box groups. I. In C. M. Campbell, E. F. Robertson, N. Ruskuc, and G. C. Smith, editors, *Proceedings of the International Conference held at the University of Bath, Bath, July 26–August 9, 1997*, volume 260 of *London Mathematical Society Lecture Note Series*, pages 30–64, Cambridge, 1999. Cambridge University Press.

[8] L. Babai, W. M. Kantor, P. P. Pálfy, and Á. Seress. Black-box recognition of finite simple groups of Lie type by statistics of element orders. *J. Group Theory*, 5(4):383–401, 2002.

[9] L. Babai and E. Szemerédi. On the complexity of matrix group problems I. In *Proc. 25th IEEE Symp. Found. Comp. Sci. (FOCS'84)*, pages 229–240. IEEE Computer Society, 1984.

[10] W. Bosma, J. Cannon, and C. 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).

[11] J. N. Bray. An improved method for generating the centralizer of an involution. *Arch. Math. (Basel)*, 74(4):241–245, 2000.

[12] J. N. Bray and H. Bäärnhielm. Standard generators for the Suzuki groups. Unpublished.

[13] J. N. Bray, D. F. Holt, and C. M. Roney-Dougal. *The maximal subgroups of the low-dimensional finite classical groups*, volume 407 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, Cambridge, 2013.

[14] P. A. Brooksbank. Constructive recognition of classical groups in their natural representation. *J. Symbolic Comput.*, 35(2):195–239, 2003.

[15] P. A. Brooksbank. Fast constructive recognition of black-box unitary groups. *LMS J. Comput. Math.*, 6:162–197, 2003.

[16] P. A. Brooksbank. Fast constructive recognition of black box symplectic groups. *J. Algebra*, 320(2):885–909, 2008.

[17] P. A. Brooksbank and W. M. Kantor. On constructive recognition of a black box PSL(d, q). In Kantor and Seress [35], pages 95–111.

[18] P. A. Brooksbank and W. M. Kantor. Fast constructive recognition of black box orthogonal groups. *J. Algebra*, 300(1):256–288, 2006.

[19] A. A. Buturlakin and M. A. Grechkoseeva. The cyclic structure of maximal tori in finite classical groups. *Algebra Logika*, 46(2):129–156, 2007.

[20] R. W. Carter. Conjugacy classes in the Weyl group. *Compositio Math.*, 25:1–59, 1972.

[21] R. W. Carter. *Simple groups of Lie type*. Wiley Classics Library. John Wiley & Sons Inc., New York, 1989. Reprint of the 1972 original, A Wiley-Interscience Publication.

[22] F. Celler and C. R. Leedham-Green. Calculating the order of an invertible matrix. In Finkelstein and Kantor [27], pages 55–60.

[23] F. Celler, C. R. Leedham-Green, S. H. Murray, A. C. Niemeyer, and E. A. O'Brien. Generating random elements of a finite group. *Comm. Algebra*, 23(13):4931–4948, 1995.

[24] A. M. Cohen, S. H. Murray, and D. E. Taylor. Computing in groups of Lie type. *Math. Comp.*, 73(247):1477–1498, 2004.

[25] M. D. E. Conder, C. R. Leedham-Green, and E. A. O'Brien. Constructive recognition of PSL(2, q). *Trans. Amer. Math. Soc.*, 358(3):1203–1221 (electronic), 2006.

[26] M. Conder and C. R. Leedham-Green. Fast recognition of classical groups over large fields. In Kantor and Seress [35], pages 113–121.

[27] L. Finkelstein and W. M. Kantor, editors. *Groups and Computation II*, volume 28 of *DIMACS Ser. Discrete Math. Theoret. Comp. Sci.* Amer. Math. Soc., Providence, RI, 1997.

[28] D. Gorenstein, R. Lyons, and R. Solomon. *The classification of the finite simple groups. Number 3. Part I. Chapter A*, volume 40 of *Mathematical Surveys and Monographs*. American Mathematical Society, Providence, RI, 1998. Almost simple K-groups.

[29] R. M. Guralnick and F. Lübeck. On p-singular elements in Chevalley groups in characteristic p. In *Groups and computation, III (Columbus, OH, 1999)*, volume 8 of *Ohio State*

Univ. Math. Res. Inst. Publ., pages 169–182. de Gruyter, Berlin, 2001.

[30] D. F. Holt and S. Rees. Testing modules for irreducibility. *J. Austral. Math. Soc. Ser. A*, 57(1):1–16, 1994.

[31] W. M. Kantor and K. Magaard. Black box exceptional groups of Lie type. *Trans. Amer. Math. Soc.*, 365(9):4895–4931, 2013.

[32] W. M. Kantor. Sylow's theorem in polynomial time. *J. Comput. System Sci.*, 30(3):359–394, 1985.

[33] W. M. Kantor and M. Kassabov. Black box groups isomorphic to $\mathrm{PGL}(2,2^e)$. *J. Algebra*, 421:16–26, 2015.

[34] W. M. Kantor and Á. Seress. Black box classical groups. *Mem. Amer. Math. Soc.*, 149(708):viii+168, 2001.

[35] W. M. Kantor and A. Seress, editors. *Groups and Computation III*, volume 8 of *Ohio State Univ. Math. Res. Inst. Publ.* de Gruyter, 2001.

[36] W. M. Kantor and Á. Seress. Prime power graphs for groups of Lie type. *J. Algebra*, 247(2):370–434, 2002.

[37] W. M. Kantor and Á. Seress. Large element orders and the characteristic of Lie-type simple groups. *J. Algebra*, 322(3):802–832, 2009.

[38] C. R. Leedham-Green. The computational matrix group project. In Kantor and Seress [35], pages 229–247.

[39] A. Lester. TAP specification. `http://testanything.org/tap-specification.html`, 2006.

[40] M. W. Liebeck and E. A. O'Brien. Recognition of finite exceptional groups of Lie type. Preprint.

[41] F. Lübeck, K. Magaard, and E. A. O'Brien. Constructive recognition of $\mathrm{SL}_3(q)$. *J. Algebra*, 316(2):619–633, 2007.

[42] F. Lübeck, A. C. Niemeyer, and C. E. Praeger. Finding involutions in finite Lie type groups of odd characteristic. *J. Algebra*, 321(11):3397–3417, 2009.

[43] D. Lytkin. Large element orders and the characteristic of finite simple symplectic and orthogonal groups, 2013.

[44] K. Magaard. Constructive recognition of $G_2(q)$ and ${}^3D_4(q)$. Manuscript notes.

[45] P. M. Neumann and C. E. Praeger. A recognition algorithm for special linear groups. *Proc. London Math. Soc. (3)*, 65(3):555–603, 1992.

[46] P. M. Neumann and C. E. Praeger. Cyclic matrices over finite fields. *J. London Math. Soc. (2)*, 52(2):263–284, 1995.

[47] A. C. Niemeyer and C. E. Praeger. Implementing a recognition algorithm for classical groups. In Finkelstein and Kantor [27], pages 273–296.

[48] E. A. O'Brien. Towards effective algorithms for linear groups. In *Finite geometries, groups, and computation*, pages 163–190. Walter de Gruyter, Berlin, 2006.

[49] OEIS Foundation Inc. The on-line encyclopedia of integer sequences. `http://oeis.org/`, 2014.

[50] I. Pak. What do we know about the product replacement algorithm? In Kantor and Seress [35], pages 301–347.

[51] C. W. Parker and R. A. Wilson. Recognising simplicity of black-box groups by constructing involutions and their centralisers. *J. Algebra*, 324(5):885–915, 2010.

[52] R. A. Parker. The computer calculation of modular characters (the meat-axe). In *Computational group theory (Durham, 1982)*, pages 267–274. Academic Press, London, 1984.

[53] C. Poe. TAP::Parser. `http://perldoc.perl.org/TAP/Parser.html`, 2008.

[54] M. G. Schwern. Test::Simple. `http://search.cpan.org/~exodist/Test-Simple-1.001003/lib/Test/Simple.pm`, 2008.

[55] G. M. Seitz. On the subgroup structure of classical groups. *Comm. Algebra*, 10(8):875–885, 1982.

[56] T. A. Springer and R. Steinberg. Conjugacy classes. In *Seminar on Algebraic Groups and Related Finite Groups (The Institute for Advanced Study, Princeton, N.J., 1968/69)*, Lecture Notes in Mathematics, Vol. 131, pages 167–266. Springer, Berlin, 1970.

[57] R. Steinberg. Representations of algebraic groups. *Nagoya Math. J.*, 22:33–56, 1963.

[58] A. V. Sutherland. *Order computations in generic groups*. ProQuest LLC, Ann Arbor, MI, 2007. Thesis (Ph.D.)–Massachusetts Institute of Technology.

[59] R. A. Wilson. *The finite simple groups*, volume 251 of *Graduate Texts in Mathematics*. Springer-Verlag London, Ltd., London, 2009.