Some topics in groups of matrices over finite fields

Daniel Rogers

Progress Report - Part A

Contents

1 Introduction

The first half of this report describes the Sims VERIFY Algorithm, an algorithm for verifying whether a proposed base and strong generating set for a finite permutation or matrix group G actually is, whilst simultaneously computing a presentation for G. We will describe the terminology required to understand the algorithm, give a detailed overview of how the algorithm works, including a worked example, before describing the implementation of the algorithm by the author for matrix groups.

The second half of the report describes the Aschbacher classification of all maximal subgroups of matrix groups over finite fields. We provide a concise introduction to the nine classes, defining each class and providing an example of a group which lies in each.

Note that all groups considered in this report will be assumed to be finite.

2 The Sims 'Verify' Algorithm

2.1 Base Strong Generating Sets

Throughout this section, let G be a group acting on a set X. Typically, the examples we will consider are $G \leq \mathrm{Sym}(X)$; or $G = \mathrm{GL}(n, \mathbb{F})$ and X the union of the set of all subspaces of $\mathbb{F}^n$ and the set of all vectors in $\mathbb{F}^n$, considering all vectors as row vectors and letting G act on X on the right by multiplication. Note that a single orbit under the action will consist of spaces of the same dimension (or vectors), and we are usually interested in the action of G on points of $\mathbb{F}^n$ or 1-dimensional subspaces.

Definition 2.1. A *base* for G is a sequence $B \subset X$ such that the pointwise stabilizer $G_{(B)}$ is trivial. Elements of B are called *base points*, and are indexed by their position in the sequence; in particular we may talk of the i^{th} *base point* of G.

In other words, $B = [\beta_1, .., \beta_n]$ is a base if, for $g \in G$

$$\beta_i^g = \beta_i \ \forall \ i \in [1, ..., n] \iff g = 1_G$$

Definition 2.2. The *basic stabilizer chain* associated with a base B for G is the chain given by

$$G \geq G_{(\beta_1)} \geq G_{(\beta_1,\beta_2)} \geq ... \geq G_{(\beta_1,...,\beta_n)} = 1$$

We use the notation $G^{(i+1)} := G_{(\beta_1,...,\beta_i)}$ to simplify notation for the basic stabilizer chain.

Definition 2.3. A *strong generating set* for G relative to a base B is a sequence $S \subseteq G$ such that $\forall i \in [1, ..., n+1]$, $\langle S \cap G^{(i)} \rangle = G^{(i)}$.

Thus, a strong generating set is a generating set for G that also contains generators for each group in the basic stabilizer chain.

Note that there is no requirement for a base or a strong generating set to be minimal, although in practice we seek to avoid superfluous base points or strong generators.

Example 2.4. (1) Let $G = S_n = \langle (1,2), (1,2,...,n) \rangle$ and $X = [1, ..., n]$. The smallest base we can obtain in this case is any $(n-1)$-point subset of X; it is clear that any element of G which fixes $n-1$ points in X must also fix the last one and thus must be the identity; conversely, if B is any $(n-2)$-point subset of X and $\{\alpha, \beta\} := X \setminus B$, then $(\alpha, \beta) \in G_{(B)}$ is a non-trivial element of the pointwise stabilizer of B. (A similar argument shows that the smallest base of A_n has $n-2$ elements.)

Take $B := [n, n-1, ..., 2]$. Then we clearly have $G^{(i)} \cong S_{n+1-i}$ (for $i \in [1, ..., n-1]$), and thus a strong generating set for G relative to B is given by $S := \{(1, ..., i) : i \in [2, ..., n]\}$

(2) Let $G \leq S_8$ be given by $G = \langle (1,2)(5,6)(7,8), (3,4)(5,7)(6,8) \rangle \cong C_2 \times C_2$. Both $\{1,3\}$ and $\{5\}$ are bases for G, and both are minimal in the sense that no base point is redundant. Thus the size of a given base, even if the base is minimal, does not give a well-defined notion of the dimension of a group. The minimal size of a base for G is an object which can be of interest to study - see [10] for examples of computations where this is relevant, or Remark 2.7 for a brief motivation.

(3) Given any matrix group G viewed as acting on the set $X = \mathbb{F}^n$, it is clear that one possible choice for a base of G is a basis of X. However, in [2] Butler considered the action of G on one-dimensional subspaces of X, which served to reduce the size of the basic orbits. The disadvantage of this method is that the action of G on the collection of subspaces of X may not be faithful, so we may not be able to find a base consisting entirely of subspaces. To avoid this problem, we often follow each subspace of X with a nontrivial vector contained in the subspace. This increases the size of the base but decreases the size of the orbits.

Definition 2.5. A *base strong generating set (BSGS)* for a group G is a pair (B, S) of a base B for G and an associated strong generating set S.

Definition 2.6. A *strong presentation* for a group G with a BSGS (B, S) is a presentation $\langle S|R \rangle$ on the strong generating set of G, with the additional property that within this presentation we have subpresentations for each of the entries in the basic stabilizer chain. In other words, given the basic stabilizer chain $G^{(1)} = G \geq G^{(2)} \geq ... \geq G^{(n)} \geq G^{(n+1)} = 1$, let $S^{(i)} := G^{(i)} \cap S$ denote the strong generators which lie in $G^{(i)}$, and let $R^{(i)}$ be those relations in R which only involve elements of $S^{(i)}$. Then for a strong presentation, we have for each i that $G^{(i)} \cong \langle S^{(i)}|R^{(i)} \rangle$.

Remark 2.7. A BSGS for G is a useful computational tool; we will briefly describe two uses here:

- Given a group $G \leq F$ for $F = \mathrm{Sym}(X)$ or $F = \mathrm{GL}(m, \mathbb{F})$, we can perform *constructive membership testing* for G; in other words, given an arbitrary $g \in F$, we can test whether $g \in G$. An algorithm is given in [7, p.89] which we will describe here. We write $B = [\beta_1, ..., \beta_k]$, and compute with each β_i in turn, constructing an element $h \in F$ which fixes the first i points in B. Assume this process has completed for the first $i-1$ entries in B (beginning with $h = g$ in the case $i = 1$), so we have an element $h \in F$ such that $\beta_j^h = \beta_j$ for all $1 \leq j \leq i-1$. We then let $\beta = \beta_i^h$, and check to see whether $\beta \in \beta_i^{G^{(i)}}$ (recalling that $G^{(i)}$ stabilises $\beta_1, ..., \beta_{i-1}$). If not, then h cannot be in G. Otherwise, we can find $u_i \in G^{(i)}$ such that $\beta_i^{u_i} = \beta$, for example using the function UBeta to be described in the following subsection. We then replace h with the element hu_i^{-1}, so h now stabilises $\beta_1, ..., \beta_i$. We continue this process for all i. If it completes, then h stabilises all of B. If $h \neq 1$ then $h \notin G$ since the only element of G which stabilises all of B is the identity. Conversely if $h = 1$ then we have $1 = gu_1^{-1}u_2^{-1}...u_k^{-1}$, so $g = u_k u_{k-1}...u_1$ and so $g \in G$. This process also allows us to write g as a word in the strong generators.

- Computing a base can be informative about the order of G. For example, given a base $B = [\beta_1, ..., \beta_n]$ of a group G, it is clear that we can uniquely determine any $g \in G$ by its action on the base points; hence we must have $|G| \leq |\beta_1^G||\beta_2^G|...|\beta_n^G|$. For instance if $G \leq \mathrm{Sym}(X)$ has a base B of size n, then we must have $|G| \leq |X|^n$, and it may be possible to reduce this bound further if G is not transitive. These bounds are very crude and more precise ones have been achieved; see for example [10]. In particular, for these types of computations minimal bases hold a particular interest.

2.2 Sims 'Verify' Algorithm Overview

We will describe the Sims Verify algorithm, which was first proposed by Charles C. Sims and is described in [7]. There, the algorithm is described in terms of permutation groups, but this can be generalised to any group G acting (faithfully) on any set X and we will describe it as such.

The VERIFY algorithm takes as input a proposed base $B = \{\beta_1, \beta_2, ..., \beta_n\}$ and strong generating set S for a group G, and produces a strong presentation for the group G, as well as verifying that the input is a BSGS, correcting it if not.

In contrast to the Todd-Coxeter-Schreier-Sims (TCSS) algorithm, which uses coset enumeration, VERIFY uses the orbit structure implicit in the basic stabilizer chain to perform this computation.

The algorithm works by considering each member of the basic stabilizer chain in turn, starting with the smallest group. Note that here, we construct the basic stabilizer chain by defining $G^{(i)} := \langle S \cap G_{\{\beta_1,...,\beta_{i-1}\}} \rangle$; thus this is the group generated by those strong generators in S which stabilise the proposed base points; this will generate a subgroup of the stabilizer we seek at each point, and it will generate the required stabilizer iff S is a strong generating set, by definition.

We will describe one step of this algorithm - thus, assume that K is a group in our basic stabilizer chain (in the above notation $K = G^{(i)}$ for some i) and that we have a strong presentation for K on its strong generators, $Y := S \cap K$. Let H denote the next step in the basic stabilizer chain, so $H = G^{(i-1)}$, and define $\alpha := \beta_{i-1}$. We seek to decide whether $H_\alpha = K$; if so, then $\{\beta_{i-1}, \beta_i, ..., \beta_n\}$ form a base for H with strong generating set $X := S \cap H$; otherwise we will need to extend either our strong generating set, so we will want to find an element of $H_\alpha \setminus K$ to append to our strong generating set, or our base, so we will want to find an additional base point.

We will denote by $\hat{y}$ the element $y \in Y$ considered as a letter in a word; similarly by $\hat{Y}$ we will denote the set of all such possible letters. Thus our assumption is that we have a presentation $\langle \hat{Y} \mid R \rangle \cong K$ for R a set of words in $\hat{Y}$, and the algorithm will, in the case where the algorithm correctly verifies the BSGS up to H, extend this to a presentation $\langle \hat{X} \mid R \cup R_1 \cup R_2 \rangle \cong H$ (where $\hat{X}$ is defined in an analogous way, and we abuse notation slightly to identify R as a set of words in $\hat{X}$, of which $\hat{Y}$ is a subset.)

2.2.1 Algorithms assumed

There are various computational tasks which we will assume we are able to do. We will not describe methods for computing these here, but the name of the functions corresponds to a corresponding function in [7] performing the required task. (Sometimes these programs are written in the context of permutation groups, but the process generalises naturally to any G-set). For all the below algorithms, suppose where necessary that we have groups $H \leq G$ acting on a set Ω.

- ORBITSV; given $\alpha \in \Omega$, compute α^G and produce a *Schreier vector* for α, which is a vector indexed by Ω which tracks how elements of α^G are constructed in terms of generators X of G.
- UBETA; given $\beta \in \alpha^G$, return an element $g \in G$ such that $\alpha^g = \beta$, possibly returning g as a word in the generators of G. This function uses the Schreier vector computed from ORBITSV and means that we do not need to store all of these elements in advance, making this computationally more efficient.
- MSTAB; given $\alpha \in \Omega$ and the order of $|G_\alpha|$, find generators of G_α; this proceeds by generating a certain number of random elements of the stabilizer (using RANDOMSTAB in [7]) and then checking to see if these generate a group of the expected size. This process is described in [7, p. 209], although the algorithm is not named; this algorithm is referred to as MSTAB in the implementation in the appendix.
- BLOCKVERIFY; given $\alpha \in \Omega$, determine whether α^H forms a block under the action of G on α^G.
- STRIP; given $g \in G$, determine if $g \in H$, and if so write it as a word in the generators of H.

2.3 The algorithm

We divide the calculation of the algorithm into a number of cases, depending on the size of the set $X \setminus Y$. If $X = Y$ then there is nothing to check and we can omit the base point; hence we can assume

without loss of generality that $|X \setminus Y| \neq 0$.

2.3.1 The single generator case

The case where $X \setminus Y$ consists of a single generator z is the simplest to describe.

We first set $\beta := \alpha^{z^{-1}}$ and $\Delta := \alpha^H$. We compute generators for K_β using MStab and choose orbit representatives of K_β on Δ; call these $[\gamma_1 := \alpha, \gamma_2 := \beta, \gamma_3, ..., \gamma_m]$. These will be used to compute the relations R_1.

We also choose orbit representatives $[\lambda_1 := \alpha.\lambda_2 := \beta, \lambda_3, ..., \lambda_l]$ of K on Δ, and simultaneously define transversals $\{\tau(\delta) : \delta \in \Delta\}$ of H_α in H. (By a *transversal* here, we mean a set of elements of H [stored as words in $\hat{X}$] such that for every δ, $\alpha^{\tau(\delta)} = \delta$.) We do this as follows:

We begin by setting $\lambda_1 := \alpha$ and $\lambda_2 := \beta$; simultaneously defining $\tau(\lambda_1) := \epsilon$ (the empty word) and $\tau(\lambda_2) := z^{-1}$, and proceed inductively. Upon defining λ_i we perform the following (dealing with the cases $i = 1$ and $i = 2$ together):

- Compute λ_i^K. For each $\lambda \in \lambda_i^K$, by definition we must have some $k \in K$, and thus corresponding word $\hat{k} \in \hat{Y}$, such that $\lambda_i^k = \lambda$; thus we set $\tau(\lambda) = \tau(\lambda_i)\hat{k}$.
- Check if $\cup_{j=1}^{i} \lambda_i^K = \Delta$. If so, then we are done; if not, then by definition of Δ we must have some $j \leq i$ and $\lambda \in \lambda_j^K$ such that $\lambda^z \notin \cup_{j=1}^{i} \lambda_i^K$. In this case, set $\lambda_{i+1} := \lambda^z$ and $\tau(\lambda_{i+1}) := \tau(\lambda)\hat{z}$.

We are now in a position to compute R_1 and R_2.

For R_1, consider all elements of the form $\tau(\gamma_k)\hat{z}\tau(\gamma_k^z)^{-1}$. We have

$$\alpha^{\tau(\gamma_k)\hat{z}\tau(\gamma_k^z)^{-1}} = \gamma_k^{\hat{z}\tau(\gamma_k^z)^{-1}} = \alpha$$

so we must have that $\tau(\gamma_k)\hat{z}\tau(\gamma_k^z)^{-1}$ represents an element of K if $K = H_\alpha$. Thus, we can test the element of H given by $\tau(\gamma_k)\hat{z}\tau(\gamma_k^z)^{-1}$ for membership of K. If this element is not in K, then we know that $K \neq H_\alpha$ and we can return the element as a proposed addition to the BSGS. Otherwise the element is in K, so we can write it as a word $\hat{u}$ in $\hat{Y}$, whereupon $\tau(\gamma_k)\hat{z}\tau(\gamma_k^z)^{-1}\hat{u}^{-1}$ is a word which evaluates in H to 1_H and so we can add this to R_1.

To construct R_2, we consider each λ_i for $1 \leq i \leq l$. We compute generators $y_1, ..., y_t$ for K_{λ_i} (we have done this already for K_{λ_1}). We then express each y_i as a word in $\hat{Y}$. Then we have that

$$\alpha^{\tau(\lambda_i)\hat{y_j}\tau(\lambda_i)^{-1}} = \lambda_i^{\hat{y_j}\tau(\lambda_i)^{-1}} = \lambda_i^{\tau(\lambda_i)^{-1}} = \alpha$$

so as before we can test this element for membership of K. If it is not an element of K, then we return this element as a new proposed strong generator; otherwise, it lies in K and thus can be written as a word $\hat{u}$ in $\hat{Y}$; whereupon $\tau(\lambda_i)\hat{y_j}\tau(\lambda_i)^{-1}u^{-1}$ evaluates to 1_H and we can add this to R_2.

If the algorithm completes these steps without rejecting any elements as not belonging to K, then we have that $\langle\ \hat{X} \mid R \cup R_1 \cup R_2\ \rangle \cong H$. From our choice of relations it is clear that the group we generate has a subgroup isomorphic to H. It is not obvious that these relations are enough to fully describe H; for a proof of this, see [7, p. 211].

2.3.2 Multiple generator case

In general, the generating sets X of K and Y of H may differ by more than one generator, in which case we are required to do more work. If $X = \{x_1, ..., x_t\}$ and $Y = \{x_1, ..., x_t, ...x_r\}$ with $r - t > 1$, we construct the chain of subgroups:

$$K = K(t) \leq K(t+1) \leq ... \leq K(r-1) \leq K(r) = H$$

where $K(s) = \langle x_1, ..., x_s \rangle$. Remember that we are seeking to show that $H_\alpha = K$. To check this, we will proceed up this chain, checking for $s = t+1, t+2, ..., r$ whether $K(s)_\alpha = K$ (for $s = t$ this will be

true by assumption). As before, if this is false then we will want to output an element of $K(s)_\alpha \setminus K$, which will therefore also be an element of $H_\alpha \setminus K$. If it is true, then we will seek to extend the set of relations for $K(s-1)$ as words in $\{x_1, ..., x_{s-1}\}$ to a set of relations for $K(s)$ as words in $\{x_1, ..., x_s\}$. If this completes for every s then we will have checked that $H_\alpha = K$ as required.

The tests here are reminiscent of those that we needed to do in the single generator case, and with a few adaptations we can reduce this to several applications of the single generator case.

Analogous to the single generator case, we define $\Delta(s) := \alpha^{K(s)}$ for $t \leq s \leq r$, giving us the chain

$$\{\alpha\} = \Delta(t) \subset \Delta(t+1) \subset ... \subset \Delta(r-1) \subset \Delta(r) = \Delta$$

We proceed up this chain; thus, fix s with $t < s \leq r$ and suppose that we have already verified that $K(s)_\alpha = K$ and we have a presentation of $K(s)$ given by $\langle \hat{x_1}, ..., \hat{x_s} | R(s) \rangle$. (For $s = t$ this is what we will have computed in the previous iteration of VERIFY). How we proceed depends on the relationship between $\Delta(s)$ and $\Delta(s+1)$; note that we must have $\Delta(s+1) \supseteq \Delta(s)$.

- Suppose $\Delta(s+1) = \Delta(s)$. Define $\beta := \alpha^{x_{s+1}}$. Then clearly $\beta \in \Delta(s+1) = \Delta(s)$ and so there exists an element $\theta \in K(s)$ (and corresponding word $\hat{\theta}$ in the generators of $K(s)$) such that $\alpha^\theta = \beta$, and so we have $\alpha^{x_{s+1}\theta^{-1}} = \alpha$. Hence $x_{s+1}\theta^{-1} \in K(s+1)_\alpha$, and we test this element for membership of K. As usual, if it fails we append this to the strong generating set. If we suceed, then we have that $x_{s+1}\theta^{-1} \in K \subset K(s)$ and $\theta \in K(s)$; hence $x_{s+1} \in K(s)$ and so $K(s+1) = K(s)$. Thus to compute a presentation of $K(s+1)$ we simply have to write the additional generator x_{s+1} as a word in $K(s)$. Since we know that $x_{s+1}\theta^{-1} \in K$ we can write it as a word $\hat{u}$ in the generators of K, and then set $R(s+1) = R(s) \cup \{\hat{x_{s+1}}\hat{\theta}\hat{u}^{-1}\}$, giving a presentation of $K(s)$.

- Suppose $\Delta(s+1) \supsetneq \Delta(s) = \{\alpha\}$. This case is precisely that of the single generator case described above, and we can apply it to $K(s+1)$ with no adaptation required.

- Suppose $\Delta(s+1) \supsetneq \Delta(s) \supsetneq \{\alpha\}$. To proceed here, we use the following result, adapted for our purposes from [7, p.25]:

  Theorem 2.8. *Using the above notation, suppose that we have $\Delta(s+1) \supsetneq \Delta(s) \supsetneq \{\alpha\}$ and $K(s+1)_\alpha = K$. Then $\Delta(s)$ is a block for the action of $K(s+1)$ on $\Delta(s+1)$.*

  Proof. Note that by assumption, $K(s+1)_\alpha = K \lneq K(s)$, and by construction we have $K(s) \lneq K(s+1)$. Recall that $\Delta(s) = \alpha^{K(s)}$. Suppose that we have $g \in K(s+1)$ such that $\alpha^{K(s)g} \cap \alpha^{K(s)} \neq \emptyset$; take $\beta \in \alpha^{K(s)g} \cap \alpha^{K(s)}$, and $k, l \in K(s)$ such that $\beta = \alpha^{kg} = \alpha^l$. Then $kgl^{-1} \in K(s+1)$ and $\alpha^{kgl^{-1}} = \alpha$ so $kgl^{-1} \in K(s+1)_\alpha \lneq K(s)$. Hence since $k, l \in K(s)$ we have $g \in K(s)$ and so $\alpha^{K(s)g} = \alpha^{K(s)}$. Hence $\alpha^{K(s)} = \Delta(s)$ is a block as required. □

  This gives us a necessary condition for $K(s+1)_\alpha = K$ to be satisfied; namely that $\Delta(s)$ is a block for the action of $K(s+1)$ on $\Delta(s+1)$. If not, then we will require an element of $K(s+1)_\alpha \setminus K$. If so, then we will want to return a block system Σ which contains $\Delta(s)$. The details of this can be found in the description of the function BLOCKVERIFY in [7, p.216].

  Suppose the above procedure successfully completed. Then we have an induced action of $K(s+1)$ on Σ, and, with respect to this action, $K(s) \leq K(s+1)_{\Delta(s)}$. Hence we have reduced the situation to that of the single generator case again, and we can perform the single generator computation to test whether $K(s+1)_{\Delta(s)} = K(s)$. If not, then as usual we will produce a new strong generator.

  Suppose we discover that $K(s+1)_{\Delta(s)} = K(s)$. Note that since $\Delta(s) = \alpha^{K(s)}$ we have $K(s+1)_\alpha \leq K(s+1)_{\Delta(s)}$, and recalling our assumption from the previous iteration of VERIFY that $K(s)_\alpha = K$, we get that

  $$K(s+1)_\alpha = K(s+1)_\alpha \cap K(s+1)_{\Delta(s)} = K(s+1)_\alpha \cap K(s)_\alpha = K(s+1)_\alpha \cap K = K$$

  Thus we have completed verification in this case.

2.4 Example

Let $G \leq \mathrm{GL}(2, \mathbb{F}_{11})$ be given by $G := \left\langle a := \begin{pmatrix} 10 & 0 \\ 0 & 1 \end{pmatrix}, b := \begin{pmatrix} 8 & 5 \\ 6 & 8 \end{pmatrix} \right\rangle$

Then $|G| = 24$; indeed, $G \cong D_{24}$, the dihedral group of symmetries of a dodecagon. We propose a BSGS given by $B := \{\beta_1 := (0,1), \beta_2 := (1,0)\}$, $S := \{a, b\}$. We describe how VERIFY will proceed in this case.

We let $K := \langle a \rangle$ and note that $|K|$=2, immediately giving the relation $a^2 = 1$. We also note that $\beta_2^a = (10, 0) \neq \beta_2$ (remembering that we view the underlying vector space as row vectors, with G acting on the vector space from the right by matrix multiplication). so that $\langle a \rangle_{\beta_2} = 1$. Thus the first step in VERIFY would have completed successfully and returned the presentation $\langle\ a \mid a^2\ \rangle$. We will check the next level.

Using the above notation we set $K := \langle a \rangle$, $H := G = \langle a, b \rangle$, $\alpha := \beta_1$, $z := b$, $R := \{a^2\}$, $\Delta \subset \mathbb{F}_{11}^2$, $|\Delta| = 12$.

Define $\beta := \alpha^{z^{-1}} = (5, 8)$. We note that $\beta^K = \{(5,8), (6,8)\}$, so that we know $|K_\beta| = |K|/|\beta^K| = 1$, so we have trivial stabilizer. We thus end up with 12 orbit representatives of K_β on Δ; we label these as follows:

$$\begin{array}{lll} \gamma_1 := (0,1) & \gamma_2 := (5,8) & \gamma_3 := (1,0) \\ \gamma_4 := (3,6) & \gamma_5 := (8,6) & \gamma_6 := (8,5) \\ \gamma_7 := (10,0) & \gamma_8 := (0,10) & \gamma_9 := (3,5) \\ \gamma_{10} := (5,3) & \gamma_{11} := (6,3) & \gamma_{12} := (6,8) \end{array}$$

which are the 12 elements of Δ (in the order produced by the MAGMA implementation of this algorithm).

Next we compute the orbit representatives of K on Δ, defining the transversals at the same time. We set $\lambda_1 := \alpha, \lambda_2 := \beta$, and we have

$$\begin{aligned} \lambda_1^K &= \{(0,1)\} \\ \lambda_2^K &= \{(5,8), (6,8)\} \end{aligned}$$

This allows us to define tranversals (removing the double brackets for notational convenience): $\tau(0,1) = \epsilon$, $\tau(5,8) = b^{-1}$, $\tau(6,8) = b^{-1}a$.

We are not finished yet, so we note that $(6,8) \in \lambda_2^K$ but $(6,8)^z = (8,6)$ which is not in any of the orbits we have to date. Thus we set $\lambda_3 = (8,6)$ and the corresponding transversal as $\tau(8,6) = \tau(6,8)b = b^{-1}ab$. Then $\lambda_3^K = \{(8,6), (3,6)\}$ and so we write $\tau(3,6) = b^{-1}aba$.

The computations of the remaining λ_i are similar and unenlightening; we end up with orbit representatives $\lambda_4 = (1,0)$, $\lambda_5 = (8,5)$, $\lambda_6 = (6,3)$ and $\lambda_7 = (0,10)$. We produce below a table of the transversals:

δ	$\tau(\delta)$
$(0,1)$	ϵ
$(5,8)$	b^{-1}
$(6,8)$	$b^{-1}a$
$(8,6)$	$b^{-1}ab$
$(3,6)$	$b^{-1}aba$
$(1,0)$	$b^{-1}ab^2$
$(10,0)$	$b^{-1}ab^2a$
$(8,5)$	$b^{-1}ab^3$
$(3,5)$	$b^{-1}ab^3a$
$(6,3)$	$b^{-1}ab^4$
$(5,3)$	$b^{-1}ab^4a$
$(0,10)$	$b^{-1}ab^5$

From here we can begin to produce our relations. In R_1 we consider elements of the form $\tau(\gamma_k)\hat{z}\tau(\gamma_k^z)^{-1}$. For $k = 1$, this gives the element $\tau(0,1)b\tau(6,8) = \epsilon b(b^{-1}a)^{-1} = ba^{-1}b$. Computing this in H, we compute $\begin{pmatrix} 8 & 5 \\ 6 & 8 \end{pmatrix}\begin{pmatrix} 10 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 8 & 5 \\ 6 & 8 \end{pmatrix} = \begin{pmatrix} 10 & 0 \\ 0 & 1 \end{pmatrix} = a \in K$.

Thus we have the relation $ba^{-1}b = a$, and so we add to R_2 the relation $ba^{-1}ba^{-1}$ (which, given we know $a^{-1} = a$, we can rewrite as $baba$).

If we consider $k = 2$, we get the relation $\tau(5,8)b\tau(5,8)^b = \tau(5,8)b\tau(0,1) = b^{-1}b\epsilon = \epsilon$, so this relation holds trivially and gives no new information.

Continuing as such, we get the collection of relations

$$R_1 = \{baba\ ,\ b^{-1}abab^2\ ,\ b^{-1}ab^2abab^{-1}a^{-1}b\ ,\ b^{-1}ab^6ab^{-4}aba\ ,\ b^{-1}ab^3ababab^{-2}ab\ ,\ b^{-1}ab^4ababab^{-3}ab\}$$

This illustrates one common feature of this algorithm; namely that it often produces a large number of redundant relations. For instance, given these relations (and $a^2 = 1$) we can reduce using Tietze transformations to the set $\tilde{R} = baba, b^{-1}ab^6ab^{-5}\}$.

We similarly compute R_2. Since $|K| = 2$, computing generators for the stabilizers is straightforward - the only λ_i on which K does not act faithfully are λ_1 and λ_7, so the stabilizers there are generated by a, and the other λ_i have trivial stabilizers, and hence there is nothing to test here.

For λ_1, we compute $\tau(0,1)a\tau(0,1)^{-1} = a$, which is self-evidently an element of K, and thus we get no new relation here. For λ_7 we compute $\tau(0,10)a\tau(0,10)^{-1} = b^{-1}ab^5ab^{-5}ab$. Performing the matrix calculation confirms that this is in K and is equal to a, so $R_2 = \{b^{-1}ab^5ab^{-5}aba\}$. It turns out that we can write this in terms of relations in $\tilde{R}$. Thus we have confirmed that (B, S) form a BSGS for G, and we have a presentation

$$\langle\ a, b \mid a^2\ ,\ baba\ ,\ b^{-1}ab^6ab^{-5}\ \rangle$$

which is indeed a presentation of D_{24}.

2.5 Adaptations

The appendix of this report describes an implementation by the author of the algorithm as described in [7] for matrix groups. A couple of adaptations were made (only the first of which is a change which only applies to matrix groups):

- Unlike with permutation groups, the natural G-set does not have a natural ordering; thus it is necessary at each stage of the VERIFY algorithm to produce the orbit α^H as a sequence, and use this throughout to allow reference to the underlying G-set.

- Using permutation groups, the algorithm computes generators for various stabilizers by taking random elements of the stabilizer and checking whether this generates the whole group (knowing the order of the group by Orbit-Stabilizer). For matrix groups, this is not quite as straightforward, since determining the size of a matrix group is computationally much more difficult. This is still a faster method than directly computing the stabilizer, though, and appears to be a large bottleneck in the effective running of the program. It may be possible to attempt to turn this into a Monte Carlo algorithm by generating some reasonable number of elements of the stabilizer and assuming without checking that these elements generate the stabilizer.

- As originally presented, the algorithm verified that S was a strong generating set and added elements to S if it was not. However no such explicit construction existed for confirming that B was a base, and the original implementation caused errors if the originally inputted base was insufficient. To correct this, a step was added to check that there is no element of the proposed strong generating set which is stabilized by each of the elements of the base. If this situation does occur, the program uses the MAGMA command RANDOMSCHREIER to correct the base. It is possible to be in the situation where none of the proposed strong generators stabilize all of the proposed base points and yet the stabilizer of all the base points in the whole group is non-trivial; however, if B is not a base VERIFY will not be able to complete until it has added a strong generator which stabilizes every element of B, so this step will be sufficient to ensure that an incorrect base is detected eventually.

2.6 Implementation

The implementation in the appendix includes an optional 'level' parameter; if this parameter is set to be l, then for the first $|B| - l$ base points a combination of the inbuilt MAGMA functions VERIFY (which, for matrix groups, actually uses the Sims-Todd-Coxeter-Schreier method, see the section on VERIFY in [11]) and FPGROUPSTRONG is used to verify the BSGS construction, and our VERIFY algorithm is used for the remaining l steps. By default $l = 3$.

We show below the improvement that this implementation has over the inbuilt algorithm in MAGMA for verifying a BSGS in some large cases.

To compare the two programs for a group G, we run RANDOMSCHREIER first to give a proposed base strong generating set, and then time the respective functions to see how long they take to verify. We will use INBUILTVERIFY and MATRIXVERIFY to refer to the VERIFY program in MAGMA and the implementation described above respectively.

In the table below, 'Group' is self-explanatory, and 'Order' gives the approximate size of the group. The remaining columns describe the time taken in MAGMA, in seconds, for the described algorithm to complete. In each case the algorithm was run five times, the fastest and slowest times discounted, and the remaining three were averaged. Situations where the running time were typically large (over 1000 seconds) were only repeated twice and these values were averaged; results where this happened are indicated by an asterisk in the below table. Results followed by a + indicate where the computation had not completed by the indicated time (typically when it had been running for long enough to give a good indication of its performance relative to other algorithms).

The speed of MATRIXVERIFY is heavily dependent on the choice of the 'level' parameter l as described. We will show both the 'default' use of MATRIXVERIFY where $l = 3$ and the 'optimal' use, where l is chosen based on the lengths of the basic orbits of the proposed BSGS; typically the algorithm works best on larger basic orbits, with INBUILTVERIFY working better on smaller basic orbits. The level used is given in brackets after the optimal time.

All calculations were performed on a Dell Inspiron 17R 7720, with a 2.40 GHz Intel i7 processor and 8GB of RAM.

Table 1: Speed comparison of algorithms verifying a BSGS

Group	Order	InbuiltVerify	MatrixVerify default	MatrixVerify optimal
SL(4,5)	3×10^{10}	0.073	0.156	0.135 (2)
SL(5,5)	6×10^{16}	2.40	3.01	2.16 (2)
SL(6,5)	3×10^{24}	277	230	27.1 (7)
SL(7,5)	3×10^{33}	50 000+	10200*	646 (7)
GU(4,3)	5×10^{7}	0.016	0.099	0.083 (2)
GU(5,3)	1×10^{12}	0.245	1.64	0.994 (2)
GU(6,3)	2×10^{17}	15.1	15.8	12.5 (2)
GU(7,3)	3×10^{23}	3409*	288	285 (5)

This shows that, although for examples of relatively small order InbuiltVerify was marginally quicker than MatrixVerify, as the groups got larger MatrixVerify became the faster tool. In particular therefore it is not surprising that the optimal implementation allowed InbuiltVerify to run on the lower levels of the function and used MatrixVerify to complete the final few stages; and indeed this approach can produce dramatic improvements in running times.

2.7 Further improvements

- The program currently both verifies a BSGS and produces a strong presentation for the group. It would be possible to introduce an additional optional parameter which simply returned the verified base strong generating set and did not compute the strong presentation; within the program, this would not perform any of the computations for relations. For example, this would reduce the number of times the function Strip is required. This is unlikely to substantially reduce the running time of the algorithm, but may give it additional functionality.
- Some of the computations may be improved by enumerating Δ and considering the action of G on Δ as a permutation action, which would then allow the use of the large number of efficient algorithms in Magma for performing computations of the type we require in permutation groups.

3 Aschbacher's Theorem

3.1 Classical groups and notation

Throughout we will assume knowledge of linear, symplectic, unitary and orthogonal groups - see [3], for example, for an introduction to these. We will use the following notation, roughly in line with [1]. We denote by:

- $\mathrm{GL}(n, \mathbb{F})$ the general linear group of $n \times n$ matrices over $\mathbb{F}$.
- $\mathrm{Sp}(n, \mathbb{F})$ the symplectic group of $n \times n$ matrices over $\mathbb{F}$.
- $\mathrm{GU}(n, \mathbb{F})$ the unitary group of $n \times n$ matrices over $\mathbb{F}^2$.
- $\mathrm{GO}^{\epsilon}(n, \mathbb{F})$ the orthogonal group of type ϵ of $n \times n$ matrices over $\mathbb{F}$, where $\epsilon \in \{+, -, \circ\}$.

All the fields we will be discussing in this section of the report will be finite. It is a well-known result that for every integer of the form $q = p^m$ with p a prime and $m \in \mathbb{Z}_{>0}$ there exists a unique field

$\mathbb{F}$ of order q; we will denote this field by $\mathbb{F}_q$ and, where convenient, we will write $\mathrm{GL}(n,q)$ in place of $\mathrm{GL}(n,\mathbb{F}_q)$ (and similarly for the other families of classical groups)

Associated to each of these families of groups we have the chain of groups $\Omega \leq S \leq G \leq C \leq \Gamma \leq A$, with brief descriptions below. This notation will be used throughout the chapter without further reference.

- G is the *general group*, and is given by $G = X(n,\mathbb{F})$, where $X \in \{\mathrm{GL}, \mathrm{Sp}, \mathrm{GU}, \mathrm{GO}^\epsilon\}$. We will refer to the *linear*, *symplectic*, *unitary*, *orthogonal plus*, *orthogonal minus* and *orthogonal* families when the general group relating to the series above is of the named type.

- S is the *special group*, which consists of those elements of G which have determinant 1. Note that in the symplectic family all matrices have determinant 1 and hence $S = G$ here.

- Ω is the *generally quasisimple classical group*. In the linear, symplectic and unitary families $\Omega = S$; in the remaining families this is defined as those elements whose *spinor norm* (if char $\mathbb{F}$ is odd) or *quasideterminant* (if char $\mathbb{F}$ is even) is 1, except for the special case where $G = \mathrm{GO}^+(4,\mathbb{F}_2)$ where there is a different definition. We will not cover the details here - see for example [1, p. 28]

- C is the *conformal group*. Let $V = \mathbb{F}^n$ and let β be the associated form. C is the group of *similarities* of β; i.e. the set of $g \in \mathrm{GL}(n,\mathbb{F})$ such that there exists $0 \neq \lambda \in \mathbb{F}$ satisfying $\beta(u^g, v^g) = \lambda\beta(u,v)$ for all $u, v \in V$. Note that in the linear case we have $\beta = 0$ and so this condition holds for all g, taking $\lambda = 1$, so $C = G$ in this case.

- Γ is the *conformal semilinear group*. With V and β as above, Γ is the group of *semi-similarities* of β; i.e. the set of *semi-linear maps* of V (maps $f : V \to V$ such that there exists $\theta \in \mathrm{Aut}\,\mathbb{F}$ with $(v+w)^f = v^f + w^f$ and $(\gamma v)^f = \gamma^\theta v^f$ for all $v, w \in V$, $\gamma \in \mathbb{F}$) such that there exists $0 \neq \lambda \in \mathbb{F}$ satisfying $\beta(u^f, v^f) = \lambda\beta(u,v)^\theta$ for all $u, v \in V$ and θ as above.

- $A = \Gamma$ in all but the linear case, where A is defined to be the split extension of Γ by the inverse-transpose map $\gamma = -T$, where γ commutes with the field automorphisms. Note that, after quotienting out by scalars, A is almost always the full automorphism group of Ω (modulo scalars); the exceptions to this are $\mathrm{O}^+(8,q)$ which has an additional *triality* automorphism, and $\mathrm{Sp}(4,q)$ for q even, which has an additional *graph* automorphism which squares to the field automorphism. We will not discuss the details of this here; see for example [1, p.34].

We will use the notation XY for $X \in \{\Omega, S, G, C, \Gamma, A\}$ and $Y \in \{L, Sp, U, O^+, O^-, O\}$ to denote the group of type Y relating to the element X of the chain; for example, SL denotes the special linear group. This provides an unambiguous list of names for these groups, although some groups may have more than one name; for instance GSp, SSp and ΩSp would all refer to the symplectic group. In this case we will use Sp to refer to all of these.

We are interested in the maximality of subgroups $K \leq H$ with $\Omega \trianglelefteq H \leq A$ in some case as described above. For simplicity we will assume that $H \leq \Gamma$, which is only a simplification in the linear case. Otherwise, typically the subgroups of H in $\mathcal{C}_i$ are given by $N_A(L) \cap H$ where L is a $\mathcal{C}_i$ subgroup of Γ, with the classes $\mathcal{C}_i$ as in Aschbacher's Theorem below.

3.2 Aschbacher's Theorem

Aschbacher, in 1984, provided a classification of all maximal subgroups of matrix groups.

Theorem 3.1 (Aschbacher)**.** *Let K be any subgroup of H for some $H \leq A$ ($H \in XY$ in the notation of the previous section). Then K is contained in a subgroup of H lying in (at least) one of the following classes:*

($\mathcal{C}_1$) subspace stabilisers

($\mathcal{C}_2$) imprimitive wreath products

($\mathcal{C}_3$) extension field groups

($\mathcal{C}_4$) simple tensor products

($\mathcal{C}_5$) subfield groups

($\mathcal{C}_6$) extraspecial types

($\mathcal{C}_7$) wreathed tensor products

($\mathcal{C}_8$) classical types

($\mathcal{S}$) (or $\mathcal{C}_9$) other almost simple groups

We will give approximate descriptions of each of these classes. Note that the true definitions of these classes as originally given by Aschbacher involve a lot of technical detail, and so the definitions given here do not match up exactly with the original ones. For a full and detailed treatment of the various classes and classifications of examples, see Chapters 2 and 3 of [1], and for detailed definitions of the classes see [9].

We will need the following definition repeatedly:

Definition 3.2. G is *absolutely irreducible* in $\mathrm{GL}(V, \mathbb{F})$ if, for any extension field $\mathbb{E} \supset \mathbb{F}$, we have that G is irreducible in $\mathrm{GL}(V \otimes \mathbb{E}, \mathbb{E})$.

Remark 3.3. There is an equivalent characterisation, described in [9, p.48], whereby G is absolutely irreducible iff $C_{\mathrm{GL}(V,\mathbb{F})}(G) \cong \mathbb{F}^* = \mathbb{F} \setminus \{0\}$.

3.3 Class $\mathcal{C}_1$

Definition 3.4. A matrix group K acting on a natural module V is *reducible* if it stabilises a proper non-zero subspace of V. [1, p.59].

Definition 3.5. A group $K \leq H$ is in *class* $\mathcal{C}_1$ if it is reducible.

Example 3.6. Let $K \leq \mathrm{SL}(2,5)$ be given by $K := \left\langle K_1 := \begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix}, K_2 := \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \right\rangle$. Let $V = \mathbb{F}_5^2$ with standard basis e_1, e_2. Then note that $e_2^{K_1} = 3e_2$ and $e_2^{K_2} = e_2$; hence K stabilises the subspace $W := \langle e_2 \rangle$ of V, so K is in class $\mathcal{C}_1$.

Membership of this class is easy to check computationally using the MeatAxe algorithm, described for example in [7].

3.4 Class $\mathcal{C}_2$

Definition 3.7. Let D and Q be groups, let Ω be a set on which Q acts, let $K := \{D_\omega : \omega \in \Omega\}$ be a family of isomorphic copies of D indexed by Ω, and define $K := \prod_{\omega\in\Omega} D_\omega$. Then the *wreath product* of D by Q, denoted by $D \wr Q$ is the (external) semidirect product of K by Q, where Q acts on K by $q \cdot (d_\omega) = (d_{q\cdot\omega})$ for $q \in Q$, $(d_\omega) \in K$. [13, p. 172]

Remark 3.8. We often take $Q = S_n$, whereupon $K = \underbrace{D \times D \times \ldots \times D}_{n \text{ times}}$, and the action of Q on K is that of permuting the entries, so $(d_1, ..., d_n)q = (d_{q^{-1}(1)}, ..., d_{q^{-1}(n)})$ for all $q \in S_n$.

Definition 3.9. A group $K \leq H$ is in *class* $\mathcal{C}_2$ if its natural module V is absolutely irreducible and we can decompose $V = \bigoplus_{j=1}^{m} V_j$ with $m \geq 2$ such that the V_j are isomorphic and we have $G \leq \mathrm{GL}(V_1) \wr S_m$. (In other words, for every $k \in K$ there exists a permutation $\pi_g \in S_m$ such that $V_j^k = V_{\pi_k(j)}$ for all j - this is the formulation used in [12]).

Example 3.10. Take $K = D_{24}$ considered as a matrix group over $\mathbb{F}_{11}$; $K := \left\langle \begin{pmatrix} 10 & 0 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 8 & 5 \\ 6 & 8 \end{pmatrix} \right\rangle$

Then we have $K \wr S_2 = \left\langle \begin{pmatrix} 10 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 8 & 5 & 0 & 0 \\ 6 & 8 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \right\rangle$

Note that $K \wr S_2$ acts on $V = \mathbb{F}_{11}^4$, which has a vector space direct sum decomposition $V = V_1 \oplus V_2$ where $V_1 = \mathrm{span}(e_1, e_2)$, $V_2 = \mathrm{span}(e_3, e_4)$ (for e_i the standard basis of V). We can check that $C_{\mathrm{GL}(4,11)}(K)$ is the set of all scalar matrices, so is isomorphic to $\mathbb{F}_{11}^*$. Hence $K \wr S_2$ is in class $\mathcal{C}_2$.

3.5 Class $\mathcal{C}_3$

Definition 3.11. The action of G on a module V is said to be $\mathbb{F}$-*semilinear* if for all $g \in G$ there exists $\alpha_g \in \mathrm{Aut}(\mathbb{F})$ such that for all $v, w \in V$, $\lambda \in \mathbb{F}$,

$$(v + \lambda w) \cdot g = v \cdot g + \lambda^{\alpha_g} w \cdot g$$

Remark 3.12. If $\alpha_g = \mathrm{Id}_{\mathbb{F}}$ for every $g \in G$, then this is linearity.

Definition 3.13. $G \leq \mathrm{GL}(n, \mathbb{F}_q)$ is in *class* $\mathcal{C}_3$ if the natural module V is irreducible and there exists a finite field $\mathbb{F}_{q^s}$ such that we can extend the $\mathbb{F}_q$-vector space structure of V to an $\mathbb{F}_{q^s}$-vector space structure of dimension $\frac{n}{s}$ and the action of G on V is $\mathbb{F}_{q^s}$-semilinear. [12]

Example 3.14. We follow the explicit construction of a group in this class as described in [8], for $G = \mathrm{GL}(2, \mathbb{F}_5)$. Let $\mathbb{F} = \mathbb{F}_{25}$ be given by $\mathbb{F}_{25} \cong \mathbb{F}_5[X]/(X^2 + 4X + 2)$. We extend the $\mathbb{F}_5$-vector space structure of $V = \mathbb{F}_5^2$ (the natural module acted upon by G) to the 1-dimensional vector space $W = \mathbb{F}_{25}$, by identifying $aX + b \in W$ with $(a, b) \in V$. This gives us an embedding of $\mathbb{F}_{25}$ into $\mathbb{F}_5^2$, which in turn induces an embedding $\psi : \Gamma\mathrm{L}(1, \mathbb{F}_{25}) \to \mathrm{GL}(2, \mathbb{F}_5)$.

Note that $X^2 = X + 3$, so acting on the basis $\{1, X\}$ of $\mathbb{F}_{25}$ considered as a 2-dimensional $\mathbb{F}_5$-vector space, the matrix $A := \begin{pmatrix} 0 & 3 \\ 1 & 1 \end{pmatrix}$ acts like multiplication by X. Hence we have that $\psi(X) = A$

Next, note that the field automorphism on $\mathbb{F}_{25}$ given by $\theta : \lambda \mapsto \lambda^5$ is such that $1 \mapsto 1$, $X \mapsto X^5 = 4X + 1$. Thus the matrix $B := \begin{pmatrix} 1 & 1 \\ 0 & 4 \end{pmatrix}$ with respect to the basis $\{1, X\}$ acts like conjugation (i.e. an application of the field automorphism) in $\mathbb{F}_{25}$, so $\psi(\theta) = B$. As $\Gamma\mathrm{L}(1, \mathbb{F}_{25}) = \langle X, \theta \rangle$, we have that $\Gamma\mathrm{L}(1, \mathbb{F}_{25}) \cong \langle A, B \rangle \leq \mathrm{GL}(2, 5)$. Hence $\langle A, B \rangle$ is in class $\mathcal{C}_3$.

3.6 Class $\mathcal{C}_4$

Definition 3.15. For V, W finite dimensional vector spaces over the same field $\mathbb{F}$ with bases $\{v_1, ..., v_n\}$ and $\{w_1, ..., w_m\}$ respectively, we define the *tensor product* $V \otimes W$ to be a vector space spanned by symbols $v \otimes w$ for $v \in V, w \in W$ which satisfy the properties that for all $k \in \mathbb{F}, v, v' \in V, w, w' \in W$:

- $(kv) \otimes (w) = (v) \otimes (kw) = k(v \otimes w)$
- $(v + v') \otimes (w) = (v \otimes w) + (v' \otimes w)$
- $(v) \otimes (w + w') = (v \otimes w) + (v \otimes w')$

It is immediate from this definition that $\dim(V \otimes W) = nm$ with basis $\{v_i \otimes w_j : 1 \leq i \leq n, 1 \leq j \leq m\}$.

Definition 3.16. A group $G \leq \mathrm{GL}(V \otimes W, \mathbb{F})$ *preserves* the tensor product decomposition if for all $g \in G$ there exists $g_1 \in \mathrm{GL}(V, \mathbb{F}), g_2 \in \mathrm{GL}(W, \mathbb{F})$ such that for all $v \in V, w \in W$, $(v \otimes w)^g = v^{g_1} \otimes w^{g_2}$.

Definition 3.17. A group G is in *class* $\mathcal{C}_4$ if it acts on a space $V \otimes W$ which is absolutely irreducible, and G preserves this tensor product decomposition

We will construct an example of a group in $\mathcal{C}_4$; to do so, we will use the following construction:

Definition 3.18. Given $A = (a_{i,j}) \in \mathrm{GL}(n, \mathbb{F})$ and $B = (b_{i,j}) \in \mathrm{GL}(m, \mathbb{F})$, we define the *Kronecker product* of A and B, denoted $A \otimes B \in \mathrm{GL}(nm, \mathbb{F})$, by

$$A \otimes B := \begin{pmatrix} a_{1,1}B & \cdots & a_{1,n}B \\ \vdots & \ddots & \vdots \\ a_{n,1}B & \cdots & a_{n,n}B \end{pmatrix}$$

If we view A and B as linear transformations, $A : \mathbb{F}^n \to \mathbb{F}^n$ and $B := \mathbb{F}^m \to \mathbb{F}^m$ with respect to bases $v_1, ..., v_n$ of $\mathbb{F}^n$ and $w_1, ..., w_m$ of $\mathbb{F}^m$, then $A \otimes B$ represents the linear transformation $\mathbb{F}^n \otimes \mathbb{F}^m \to \mathbb{F}^n \otimes \mathbb{F}^m$, $v \otimes w \to (Av) \otimes (Bw)$ with respect to the ordered basis $\{v_1 \otimes w_1, \cdots, v_1 \otimes w_m, v_2 \otimes w_1, \cdots, v_2 \otimes w_m, v_3 \otimes w_1, \cdot, v_n \otimes w_m\}$.

Example 3.19. Let $G = \mathrm{GL}(2, \mathbb{F}_3) = \left\langle A_1 := \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}, A_2 := \begin{pmatrix} 2 & 1 \\ 2 & 0 \end{pmatrix} \right\rangle$, which both act on the vector space $\mathbb{F}_3^2$. Let $V = \mathbb{F}_3^4 \cong \mathbb{F}_3^2 \otimes \mathbb{F}_3^2$, and let H be the subgroup of $\mathrm{GL}(4, \mathbb{F}_3)$ given by $H := \langle\, A_i \otimes A_j \mid 1 \leq i \leq 2, 1 \leq j \leq 2 \,\rangle$.

We can check computationally that $C_{\mathrm{GL}(V, \mathbb{F}_3)}(H) \cong \mathbb{F}_3^*$ so that V is absolutely irreducible. Further, if we write $A = A_i \otimes A_j \in H$ as $\sum_{i,j=1}^{2} a_{ij} A_i \otimes A_j$ then it is clear from the definition of the Kronecker product that, for any $v \otimes w \in V$, we have $A(v \otimes w) = (A_i v) \otimes (A_j w)$. Extending linearly we see that H preserves the tensor product; thus H is in class $\mathcal{C}_4$.

Indeed, the above construction can be generalised to the Kronecker product of any two matrix groups, and the resulting construction will preserve the tensor product.

3.7 Class $\mathcal{C}_5$

Definition 3.20. A subgroup $H \leq \mathrm{GL}(n, \mathbb{F})$ is said to be *subfield* if H is absolutely irreducible and there exists a proper subfield $\mathbb{K}$ of $\mathbb{F}$ and a matrix $g \in \mathrm{GL}(n, \mathbb{F})$ such that $H^g \leq \langle \mathrm{Z}(\mathrm{GL}(n, \mathbb{F})), \mathrm{GL}(n, \mathbb{K}) \rangle$

In other words, a group H is subfield if, after conjugation by an appropriate matrix, you can write it (up to scalars) over a smaller field. The groups in class $\mathcal{C}_5$ are essentially of this form.

Remark 3.21. Let $\mathbb{K}$ be a subfield of $\mathbb{F}$. Then there is a natural way to embed $\mathrm{GL}(n, \mathbb{K})$ into $\mathrm{GL}(n, \mathbb{F})$ as follows; let $\{e_1, ..., e_n\}$ be a basis of $V = \mathbb{F}^n$, so that a typical element of V is of the form $\sum_{i=1}^n f_i e_i$ for $f_i \in \mathbb{F}$. We can then define a $\mathbb{K}$-subspace $\hat{V}$ of V by setting $\hat{V} = \{\sum_{i=1}^n k_i e_i | k_i \in \mathbb{K}\}$. Then we have a natural action of $\mathrm{GL}(\hat{V}) \cong \mathrm{GL}(n, \mathbb{K})$ on $\mathrm{GL}(V) \cong \mathrm{GL}(n, \mathbb{F})$.

Definition 3.22. Let $K \leq H \leq \mathrm{GL}(n, \mathbb{F})$, and let $Z \leq H$ be the subgroup of scalar matrices in H. K is in class $\mathcal{C}_5$ if $K = N_H(\hat{V})Z$ with $\hat{V}$ as above and $N_H(\hat{V})$ consisting of those matrices in H which stabilise $\hat{V}$.

Remark 3.23. All members of class $\mathcal{C}_5$ are subfield groups.

Example 3.24. Let $H = \mathrm{GL}(2, \mathbb{F}_{27})$ where $\mathbb{F}_{27}^* = \langle \theta \rangle$. Note that we can define a subfield $\mathbb{K}$ of $\mathbb{F}_{27}$ by $\mathbb{K} = \{0, \theta^0 = 1, \theta^{13} = -1\}$ so $\mathbb{K} \cong \mathbb{F}_3$

Consider the subgroup of G given by $K := \left\langle \begin{pmatrix} \theta^8 & \theta^{25} \\ \theta^{12} & \theta^2 \end{pmatrix}, \begin{pmatrix} \theta^{25} & \theta^{25} \\ \theta^4 & \theta^2 \end{pmatrix}, \begin{pmatrix} \theta & 0 \\ 0 & \theta \end{pmatrix} \right\rangle$.

Note that the centres of K and H coincide as the 26 scalar matrices, and we can check via Magma that K is self-normalising, so K is in class $\mathcal{C}_5$. It is in fact subfield; conjugating by the matrix $g = \begin{pmatrix} \theta^{24} & 0 \\ \theta^{11} & \theta^{25} \end{pmatrix}$ we get the group

$K^g := \left\langle \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 2 & 1 \end{pmatrix}, \begin{pmatrix} \theta & 0 \\ 0 & \theta \end{pmatrix} \right\rangle$.

The centre of K^g is the same as the centre of K, and modulo scalars (i.e. removing the third generator of K^g) we have a subgroup of $\mathrm{GL}(2, \mathbb{K})$; indeed we actually have a copy of $\mathrm{SL}(2, \mathbb{K})$. Thus K is subfield.

3.8 Class $\mathcal{C}_6$

Definition 3.25. A *special* group is a p-group G (p prime) such that $Z(G) = [G, G] = \Phi(G)$, i.e. the centre of G, the commutator subgroup of G, and the Frattini subgroup of G (defined as the intersection of all maximal subgroups of G) are equal.

A p-group G is *extraspecial* if it is special, and in addition $|Z(G)| = p$.

A p-group G is of *symplectic type* if it is extraspecial, and in addition every characteristic abelian subgroup of G is cyclic; in other words, for any abelian subgroup $H \leq G$ which is not cyclic, there exists an automorphism in $\mathrm{Aut}(G)$ which does not fix H.

Example 3.26. The dihedral group D_8 and the quarternion group Q_8 are the two extraspecial groups of order 8.

For instance, consider $H = Q_8 \leq \mathrm{SL}(2, \mathbb{F}_5)$ given by $H := \left\langle i := \begin{pmatrix} 0 & 4 \\ 1 & 0 \end{pmatrix}, j := \begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix} \right\rangle$.

Then $Z(G) = \langle i^2 \rangle$ (note that $i^2 = \begin{pmatrix} 4 & 0 \\ 0 & 4 \end{pmatrix}$ which is diagonal and therefore contained in the centre; the fact that $ij = -ji$ means that no other non-identity element of H is in the centre).

The commutator subgroup of H is thus easy to compute as well. We know that $Z(H)$ is normal and $|Z(H)| = 2$; hence $H/Z(H)$ is abelian as it has order 4. Hence we have $[H, H] \leq Z(H)$, and, since H itself is not abelian, $[H, H]$ cannot be trivial. Hence $[H, H] = Z(H)$.

Finally the maximal subgroups are all those of order 4; namely $\langle i \rangle$, $\langle j \rangle$ and $\langle ij \rangle$; the intersection of all these is $\langle i^2 \rangle$, a group of order 2; hence H is extraspecial.

Remark 3.27. In general, a result due to Hall fully describes the extraspecial groups (see for example [14, p.83] for a proof); they have order p^{1+2n} for p prime and n positive, and for each such order there are precisely two such groups. They are constructed from central products of extraspecial p-groups of order p^3. In particular, for $p = 2$, every extraspecial group is a central product of some combination of D_8 and Q_8; more specifically, if $|G| = 2^{1+2n}$ then G is either a central product of n copies of D_8, or $n - 1$ copies of D_8 and one copy of Q_8.

Definition 3.28. A group $K \leq \mathrm{GL}(n, \mathbb{F})$ is in *class* $\mathcal{C}_6$ if there exists an absolutely irreducible group E such that $E \trianglelefteq K \leq N_{\mathrm{GL}(n,\mathbb{F})}(E)$, where E is either extraspecial, or a 2-group of symplectic type. [5]

Remark 3.29. It is known, for example in [9, p.149], that the 2-groups of symplectic type are central products of C_4, the cyclic group of order 4, with an extraspecial 2-group.

Example 3.30. Let H be as above, considered as a subgroup of $\mathrm{SL}(2,\mathbb{F}_5)$, and let $a = \begin{pmatrix} 2 & 3 \\ 4 & 4 \end{pmatrix}$. Note that $i^a = -j$ and $j^a = ji$, so $a \in N_{\mathrm{SL}(2,\mathbb{F}_5)}(H)$, and so $M := \langle i, j, a\rangle \leq N_{\mathrm{SL}(2,\mathbb{F}_5)}(H)$. We can check using Magma that actually $M = N_{\mathrm{SL}(2,\mathbb{F}_5)}(H)$, so M is in class $\mathcal{C}_6$. Noting that $a^3 = i^2$ we can see that $|M| = 24$. Since $|\mathrm{SL}(2,\mathbb{F}_5)| = 120$, it follows that $|\mathrm{SL}(2,\mathbb{F}_5) : M| = 5$ so M is a maximal subgroup of $\mathrm{SL}(2,\mathbb{F}_5)$.

3.9 Class $\mathcal{C}_7$

Definition 3.31. A matrix $g \in \mathrm{GL}(V,\mathbb{F})$ *preserves* a tensor induced decomposition $V = V_1 \otimes V_2 \otimes ... \otimes V_t$ if there exists $g_i \in \mathrm{GL}(V_i,\mathbb{F})$ and $\sigma \in S_t$ such that for all $v_i \in V_i$

$$(v_1 \otimes ... \otimes v_t)^g = v_{\sigma(1)}^{g_{\sigma(1)}} \otimes ... \otimes v_{\sigma(t)}^{g_{\sigma(t)}}$$

A group $G \leq \mathrm{GL}(V,\mathbb{F})$ *preserves* a tensor induced decomposition $V = V_1 \otimes V_2 \otimes ... \otimes V_t$ if it is absolutely irreducible and every $g \in G$ preserves the given tensor induced decomposition (equivalently if every generator of G preserves the tensor induced decomposition).

Note the difference between this definition and Definition 3.16 is the relaxed requirement that the group merely preserves the structure of the tensor product - in particular, here we allow the tensor factors to be permuted.

Definition 3.32. A subgroup G of $\mathrm{GL}(n,\mathbb{F})$ is a *tensor induced group* if it preserves a tensor induced decomposition $\mathbb{F}^n = V_1 \otimes ... \otimes V_t$ with $\dim V_i = m$ for all i (and hence $n = m^t$).

Definition 3.33. $G \leq \mathrm{GL}(n,q)$ is in *class* $\mathcal{C}_7$ if the natural module V is absolutely irreducible and G is a tensor induced group.

The condition that the tensor factors must be of the same dimension prevents this from being a direct generalisation of the class $\mathcal{C}_4$. Here, if $t = 2$ and we have that, for every $g \in G$, the associated $\sigma \in S_2$ is the identity, we have a member of $\mathcal{C}_4$ (but not every member of $\mathcal{C}_4$ can be produced in such a way.)

Example 3.34. Let $R := \mathrm{SL}(3,2) = \left\langle R_1 := \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}, R_2 := \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} \right\rangle$.

Let $V = \mathbb{F}_2^3$ denote the underlying vector space, and let V' be an isomorphic copy of V. We will construct a group $K \leq \mathrm{SL}(9,2)$ which preserves the tensor induced decomposition $V \otimes V'$.

Let $\{v_1, v_2, v_3\}$ be a basis of V, $\{w_1, w_2, w_3\}$ a basis of V', and construct the ordered basis of $V \otimes V'$ as in Definition 3.18. We then let the first two generators of K be $K_i = R_i \otimes I_3$ $(i = 1, 2)$. Then each K_i preserves the given tensor induced decomposition, with $g_1 = g_2 = R_i$ and $\sigma = \mathrm{Id}$. Note that if these were the only generators of K then we would have an element of $\mathcal{C}_7$ which is also an element of $\mathcal{C}_4$.

The third generator K_3 of K is given by

$$K_3 := \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

This matrix maps $v_i \otimes w_j$ to $v_j \otimes w_i$; hence taking $g_1 = g_2 = I_3$ and $\sigma = (1,2)$, we see that K_3 also preserves the given tensor-induced decomposition; hence $K = \langle K_1, K_2, K_3 \rangle$ is in class $\mathcal{C}_7$. Note that K_3 cannot be expressed as the Kronecker product of two 3×3 matrices, so K is not in class $\mathcal{C}_4$.

3.10 Class $\mathcal{C}_8$

This class consists of those subgroups of a classical group which happen to be a member of a different class of classical groups; i.e. those groups on which it is possible to define a second form.

Definition 3.35. A group $K \leq H$ lies in *class* $\mathcal{C}_8$ if K is the intersection of H with the Γ-group of another classical group.

Example 3.36. The group $G = \mathrm{Sp}(4, \mathbb{F}_2)$ of 4-dimensional symplectic matrices over $\mathbb{F}_2$ can be viewed as the group of matrices $M \in \mathrm{GL}(4, \mathbb{F}_2)$ such that $MBM^T = B$ for some symplectic form B. In Magma the standard symplectic form is $B = \mathrm{antidiag}(1,1,1,1)$ and with respect to this form we have

$$G = \left\langle \begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix} \right\rangle$$

Let $H = \mathrm{GO}^+(4, \mathbb{F}_2)$, the set of matrices which preserve the quadratic form $A = \mathrm{antidiag}(1,1,0,0)$. From [1, p.18] the matrices which do this are precisely those which satisfy $(MAM^T)^{UT} = A$, where C^{UT} denotes an upper triangular matrix relating to C constructed in the following way: if $C = (c_{ij}) \in \mathrm{M}_n(\mathbb{F})$ then $C^{UT} = (d_{ij}) \in \mathrm{M}_n(\mathbb{F})$ is given by $d_{ij} = \begin{cases} c_{ii} & \text{if } i = j \\ c_{ij} + c_{ji} & \text{if } i < j \\ 0 & \text{if } i > j \end{cases}$

In Magma we can see that $H = \left\langle \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \right\rangle$ and we could check directly that $H \leq G$ (and hence H is in Class $\mathcal{C}_8$) here. In fact, we can prove a more general result:

Lemma 3.37. *Suppose* $H = \mathrm{GO}^+(2n, \mathbb{F})$ *where* $\mathrm{char}\ \mathbb{F} = 2$*, with* H *preserving the quadratic form* $A = \mathrm{antidiag}(\underbrace{1, ..., 1}_{n}, \underbrace{0, .., 0}_{n})$*. Then* $H \leq G = \mathrm{Sp}(2n, \mathbb{F})$*, where* G *preserves the symplectic form* $B = \mathrm{antidiag}(1, 1, ..., 1)$.

Proof. Take $M \in H$; then, since $(MAM^T)^{UT} = A$ and $\mathrm{char}\ \mathbb{F} = 2$ we must have that $\mathrm{diag}(MAM^T) = (0, ..., 0)$ and MAM^T is symmetric, except for the antidiagonal, which has the form $\mathrm{antidiag}(MAM^T) = (a_1, a_2, ..., a_n, a_n{+}1, ..., a_2{+}1, a_1{+}1)$. Hence $\mathrm{antidiag}((MAM^T)^T) = (a_1{+}1, a_2{+}1, ..., a_n{+}1, a_n, ..., a_2, a_1)$ and in all other entries MAM^T and $(MAM^T)^T$) agree.

Next note that $A^T = \mathrm{antidiag}(\underbrace{0, ..., 0}_{n}, \underbrace{1, .., 1}_{n})$, so that $B = A + A^T$. Then

$$MBM^T = M(A + A^T)M^T = MAM^T + MA^TM^T = MAM^T + (MAM^T)^T = B$$

so $M \in G$. □

3.11 Class $\mathcal{S}$

This class covers all groups which do not fall into any of the other 8 classes. Although groups in this class do not share any nice geometric structure as those in the other classes do, we have a number of properties that groups in this class must satisfy, given in the definition below.

Definition 3.38. A group H is *almost simple* if there exists a nonabelian simple group T such that $T \leq H \leq \mathrm{Aut}(T)$.

Definition 3.39. We define $G^\infty = \bigcap_{n\geq 0} G^{(n)}$ where $G^{(1)} = [G, G]$ is the derived subgroup of G and inductively, $G^{(n)} = [G^{(n-1)}, G^{(n-1)}]$.

Definition 3.40. Suppose we have $\Omega \leq G \leq A$ defined over a field $\mathbb{F}$ and suppose $H \leq G$. Let Z denote the group of scalar matrices in H. Then H is in *class* $\mathcal{S}$ if H/Z is almost simple, and all of the following properties hold:

(i) H does not contain Ω

(ii) H^∞ acts absolutely irreducibly

(iii) There does not exist a $g \in \mathrm{GL}(n, \mathbb{K})$ for $\mathbb{K}$ a field extension of $\mathbb{F}$ such that H^∞ is defined over a proper subfield of $\mathbb{K}$ modulo scalars.

(iv) Depending on the type of groups we are covering, we have a result regarding forms preserved by the group:

- H^∞ preserves a non-zero unitary form iff we are in the unitary case
- H^∞ preserves a non-zero quadratic form iff we are in the orthogonal plus, orthogonal minus or orthogonal case.
- H^∞ preserves a non-zero symplectic form and no non-zero quadratic form iff we are in the symplectic case
- H^∞ preserves no non-zero classical form iff we are in the linear case

These conditions limit the intersections of $\mathcal{S}$ with some of the classes $\mathcal{C}_i$;

- Condition (ii) prevents any intersection between $\mathcal{S}$ and $\mathcal{C}_1$ from the definition. It can also be shown (see [1, p.66]) that condition (i) is not satisfied by any elemnt of $\mathcal{C}_3$, preventing any intersection there also.
- Condition (iii) prevents any intersection between $\mathcal{S}$ and $\mathcal{C}_5$.
- Condition (iv) prevents any intersection between $\mathcal{S}$ and $\mathcal{C}_8$, as this means that no element of $\mathcal{S}$ can preserve more than one type of form.

It is, however, possible for an element of $\mathcal{S}$ to intersect with any of the other remaining classes ($\mathcal{C}_i$ for $i = 2, 4, 6, 7$).

4 Future directions

It is possible to further divide the class $\mathcal{S}$ into two subclasses, $\mathcal{S}_1$ and $\mathcal{S}_2$. See [1, p.153] for a definition of the *characteristic* of a group of Lie type.

Definition 4.1. Let G be a subgroup in Class $\mathcal{S}$ of a classical group C in characteristic p. Then G lies in *Class* $\mathcal{S}_2$, called the *defining characteristic* case, if G^∞ is isomorphic to a group of Lie type in characteristic p, and G lies in *Class* $\mathcal{S}_1$, called the *cross characteristic* case, otherwise.

Following the publication of Aschbacher's classification of maximal subgroups (the classes of which are sketched in the previous chapter), Kleidman and Liebeck in [9] describe and classify the structure of those groups in the *geometric* classes ($\mathcal{C}_1$ to $\mathcal{C}_8$). Bray, Holt and Roney-Dougal in [1] classifiy all subgroups in classes $\mathcal{S}_1$ and $\mathcal{S}_2$ for matrix groups of dimension up to 12. There is ongoing research to investigate the situation in higher-dimensional cases, and one possible avenue of further research would be to investigate the situation in dimension 16, which is likely to be the next interesting case.

5 References

[1] Bray JN, Holt DF, Roney-Dougal CM (2013) The Maximal Subgroups of the Low-Dimensional Finite Classical Groups. Cambridge University Press, Croydon.

[2] Butler G (1979) Computational Approaches to Certain Problems in the Theory of Finite Groups. PhD thesis, University of Sydney.

[3] Cameron PJ (2000) Classical Groups. Lecture notes, Queen Mary University of London. `http://www.maths.qmul.ac.uk/~pjc/class_gps/cg.pdf`. Accessed 17 February 2014.

[4] Conway JH, Curtis RT, Norton SP, Parker RA, Wilson RA (1985, republished 2003) Atlas of Finite Groups: Maximal Subgroups and Ordinary Characters for Simple Groups. Oxford University Press, New York.

[5] Coutts HJ (2010) Topics in Computational Group Theory: Primitive permutation groups and matrix group normalisers. PhD thesis, University of St Andrews.

[6] Gill N (2007) Polar spaces and embeddings of classical groups. New Zealand Journal of Mathematics 36, 175-184.

[7] Holt DF, Eick B, O'Brien EA (2005) Handbook of Computational Group Theory. Chapman and Hall, ebook.

[8] Holt DF, Roney-Dougal CM (2005) Constructing Maximal Subgroups of Classical Groups. LMS J. Comput. Math. 8 46-79.

[9] Kleidman P, Liebeck M (1990) The Subgroup Structure of the Finite Classical Groups. Cambridge University Press.

[10] Liebeck MW, Shalev A (2003) Bases of primitive permutation groups. Groups, Combinatorics and Geometry: Durham, 2001, World Scientific.

[11] Magma Computer Algebra handbook. `http://magma.maths.usyd.edu.au/magma/handbook/`. Accessed 26 February 2014.

[12] Neunhöffer M (2006) Aschbacher's Theorem. Slides from a presentation given at the University of St Andrews. `http://www-groups.mcs.st-and.ac.uk/~neunhoef/Publications/pdf/aschbacher.pdf`. Accessed 7 February 2014.

[13] Rotman JJ (1995) An introduction to the theory of groups (4th edition). Springer-Verlag New York, United States of America.

[14] Wilson RA (2009) The Finite Simple Groups. Springer-Verlag London.

Appendices

This is the code for the author's implementation of the VERIFY algorithm for matrix groups, consisting of two programs - MPRES, the top-level function, and VERIFY, the Verify algorithm itself. There are a number of other programs referenced here that are included in the running of this program, which are identical or near-identical to the ones in [7]; page references for these are listed below:

- MORBITSV, the matrix version of OrbitSv. [7, p. 80]
- RANDOMSTAB generates a random element of a stabilizer. [7, p. 81]
- MSTAB uses RANDOMSTAB to quickly find generators of the stabilizer, given the expected order. [7, p. 209]
- MUBETAWORD takes $\beta \in \alpha^G$ and returns an element $g \in G$ such that $\alpha^g = \beta$, g expressed as a word in generators. [7, p. 80]
- MUBETA does the same, except it returns the element $g \in G$ itself rather than a word. [7, p. 80]
- MSTRIP takes a putative element h of $H \leq G$, checks if $h \in H$ and if so writes it as a word in the generators of H. [7, p. 89]

```
MVerify:=function(H,K,free,rels,z,alpha,X,Y,delta,B)
// Input: - K a group generated by Y, H generated by X, with H transitive under
// delta.
// - H contains K, and is generated by one additional element z
// - free, a free group of rank #X, and rels, a set of relations on the first #Y
// (=#X-1) generators.
// - rels is such that the the free group of rank #Y quotiented by rels is
// isomorphic to K.
// - alpha is an element of the proposed base
// - delta is the orbit of alpha under H.
// Output: A pair of outputs t,out. t is either true or false. If t is false, then
// we do not have a BSGS and out is an element to be appended to our strong
// generating set. If t is true, then we have a BSGS here, and output is given by
// rels, relations which give a finitely presented group isomorphic to H.

// Notes
// - We feed in both X and Y, and H and K, because this allows us to set up maps
// between free and H easily. It also avoids MAGMA issues where elements in the
// group generated by X and elements in H were deemed to not be from the
// same universe, for example.
// - We also feed in delta directly rather than computing it here, to allow this to
// work in more general settings.
// - Adaptations by Derek Holt denoted by DFH.

// PRELIMINARY SET-UP

// We produce a list of generators of each of the basic stabilizers of K.
// If a generator in Y isn't needed we replace it with the identity, to keep the
// ordering consistent with the ordering of Y.
Sgen:=[];
```

```
Sgen[1]:=Y;
Stemp:=Sgen[1];
for i in [1..#B] do;
        for j in [1..#Stemp] do;
                if B[i]^Stemp[j] ne B[i] then;
                        Stemp[j]:=Identity(K);
                end if;
        end for;
        Sgen[i+1]:=Stemp;
end for;


// This will be used in MStrip later on;
vstrip:=<>;
delstrip:=<>;
for i in [1..#B] do;
        Append(~delstrip,SetToIndexedSet(B[i]^sub<Generic(K)|Sgen[i]>));
        deltemp,vtemp:=MOrbitSv(B[i],Sgen[i],delstrip[i]);
        Append(~vstrip,vtemp);
end for;


beta:=alpha^(z^(-1));
deltasize:=#delta;

// Kbeta is the stabilizer of beta under K, generated by the set Kbetagens.
// MStab is a new program written here, which computes the correct stabilizer
// when MVerify is used in the general case.

betaorb,betav:=MOrbitSv(beta,Y,delta);
Kstabord:=#betaorb;

Kbeta:=MStab(beta,Y,K,#K div Kstabord,betav,delta);

// Set up orbit representatives and transversals:
// - orbKbeta stores the orbit representatives of Kbeta on delta, including
// alpha and beta.
// - orbKbetaunion is the subset of delta which is reached by orbits of
// elements of orbKbeta.
//   Hence, we stop once this orbKbetaunion = delta.

orbKbeta:= {@ alpha,beta @};
orbKbetaunion:= SetToIndexedSet((alpha^Kbeta) join (beta^Kbeta));

// Compute orbit representatives of Kbeta on delta

while (#orbKbetaunion lt deltasize) do;
        gamma:=Representative(delta diff orbKbetaunion);
        Include(~orbKbeta,gamma);
        //orbKbeta[#orbKbeta+1]:=gamma;
```

```
        orbKbetaunion:=orbKbetaunion join SetToIndexedSet(gamma^(Kbeta));
end while;

// The map phi allows us to transfer elements from free to H consistently with the
// isomorphism between quo<free|rels> and K.

phi:=hom<free -> Generic(H)|X>;

// Compute orbit representatives of K on delta, along with associated transversals
// - orbK stores the orbit representatives of K on delta, orbKunion is as above
// - trans stores the necessary transversals, word in trans[i] relates to the
// element in delta[i]

trans:=[];
orbK:= {@ alpha,beta @};
orbKunion:= SetToIndexedSet((alpha^K) join (beta^K));

// MOrbitSv simultaneously computes the orbit under alpha and a Schreier vector
// for alpha. This is used to compute the words in MUBetaWord (both functions
// adapted from respective permutation ones)

trans[Index(delta,alpha)]:=Identity(free);
trans[Index(delta,beta)]:=free![-#X];

orb,v:=MOrbitSv(alpha,Y,delta);
for a in orb do;
        if IsDefined(trans,Index(delta,a)) eq false then;
       // This means we only define new transversals, no overwriting old ones.
                w:=MUBetaWord(a,v,X,free,delta);
                trans[Index(delta,a)]:=trans[Index(delta,alpha)]*w;
        end if;
end for;

orb,v:=MOrbitSv(beta,Y,delta);
for a in orb do;
        if IsDefined(trans,Index(delta,a)) eq false then;
                w:=MUBetaWord(a,v,X,free,delta);
                trans[Index(delta,a)]:=trans[Index(delta,beta)]*w;
        end if;
end for;

// We know that every element of orbK can be written as (an element of orbKbeta)^z.
// However the proof is existence only, not constructive, so here we check each
// element of orbKbeta in turn. If it is a good choice for an element of orbK, then
// we append and compute the relevant transversals. Since we require choosing an
// element in orbKunion, this may cycle through a number of times. This could
// perhaps be improved to computing all elements orbKbeta^z at the start, then
// choosing a good one from that list each time.
track:=1;
while (#orbKunion lt deltasize) do;
```

```
    if orbKbeta[track] in orbKunion then;
      temporb:=orbKbeta[track]^z;
//DFH: introduced two "if" loops here to avoid unnecessary computations
      if temporb notin orbKunion then;
                orb,v:=MOrbitSv(temporb,X,delta);
/*
        if (temporb notin orbKunion) and (orbKbeta[track] in orbKunion) then;
   //Only enter this if it gives us a new orbit rep.
*/
        Include(~orbK, temporb);
        //orbK[#orbK+1]:=temporb;
        trans[Index(delta,temporb)]:=trans[Index(delta,orbKbeta[track])]*free![#X];
        for a in orb do;
                if IsDefined(trans,Index(delta,a)) eq false then;
                        w:=MUBetaWord(a,v,X,free,delta);
                        trans[Index(delta,a)]:=trans[Index(delta,temporb)] * w;
                 end if;
        end for;
        orbKunion:=orbKunion join orb;
      end if;
    end if;
    track:=track+1;
        if track gt #orbKbeta then;
                track:=1;
     //Resets the process if we haven't defined all transversals by this point.
        end if;
end while;

// CONSTRUCTION OF NEW RELATORS
// The new relators consist of three distinct sets of relators.
// The first set, rels, have been computed already.
// Further relators will be appended to this set.

// CONSTRUCTION OF SECOND SET OF RELATORS


for i in [1..#orbK] do;
        orb,v:=MOrbitSv(orbK[i],Y,delta);
        Kbetai,GenKbetai:=MStab(orbK[i],Y,K,#K div #orb,v,delta);
        //Kbetai is an output of MStab but isn't actually needed here.
        if IsEmpty(GenKbetai) eq false then;
                for g in GenKbetai do;
// The long element here fixes alpha, so we check it for membership of K. If it is
// not in K, then we append this to the strong generating set in the top level
// function. If it is in K, then Strip writes it as a word in the generators of K,
// and we append the whole word to rels.


               h,length,w:=MStrip(phi(trans[Index(delta,orbK[i])])*g*
              phi(trans[Index(delta,orbK[i])])^(-1),B,K,Y,Sgen,free,vstrip,delstrip);
```

```
                if h ne Identity(H) then;
                        return false, h;
                else;
                       // Need to write the long element in K as a word in the
                      // generators of K. We use Strip again to rewrite g as a
                     // word in K, and then combine everything together.
                     h2,length2,w2:=MStrip(g,B,K,Y,Sgen,free,vstrip,delstrip);
                      rels:=Append(rels,trans[Index(delta,orbK[i])]*w2*
                      (trans[Index(delta,orbK[i])])^(-1)*w^(-1));
                end if;
            end for;
      end if;
end for;


// CONSTRUCTION OF THIRD SET OF RELATORS

for i in [1..#orbKbeta] do;
// As before, this element should fix alpha. We proceed as in the previous section.
        h,length,w:=MStrip(phi(trans[Index(delta,orbKbeta[i])])*z
       *phi(trans[Index(delta,orbKbeta[i]^z)])^(-1),B,K,Y,Sgen,free,vstrip,delstrip);

        if h ne Identity(H) then;
                return false, h;
        else;
                rels:=Append(rels,trans[Index(delta,orbKbeta[i])]*free![#X]*
                (trans[Index(delta,orbKbeta[i]^z)])^(-1)*w^(-1));
        end if;
end for;

// If we reach this far, then the relators we have defined together give us a
// presentation of H.
return true, rels;

end function;


MBlockVerify:=function(H,X,Y,alpha)
// X (r generators) generates the group H, which contains the group L generated by Y
// (s generators).

//delta:=[a: a in alpha^H];
delta := SetToIndexedSet(alpha^H);
p:=[];
for beta in delta do;
        p[Index(delta,beta)]:=0;
end for;
B:= {@ @}; u:=[];
u[Index(delta,alpha)]:=Identity(H);
Btemp,v:=MOrbitSv(alpha,Y,delta);
```

```
//B[1]:=Setseq(Btemp);
Include(~B, Btemp);
for a in B[1] do;
        p[Index(delta,a)]:=1;
        if a ne alpha then;
                w:=MUBeta(a,v,X,delta);
                u[Index(delta,a)]:=u[Index(delta,alpha)]*w;
        end if;
end for;
rho:=1; m:=1;
while rho le m do;
        beta:=B[rho][1];
        for i in [1..#X] do;
                epsilon:=beta^X[i];
                sigma:=p[Index(delta,epsilon)];
                if sigma eq 0 then;
                        m:=m+1;
                        sigma:=m;
                        u[Index(delta,epsilon)]:=u[Index(delta,beta)] * X[i];
                        p[Index(delta,epsilon)]:=sigma;
                        //DFH:had to rewrite this slightly, form newblock first
                        newb := {@ epsilon @};
                        for j in [2..#B[rho]] do;
                                gamma:=B[rho][j];
                                epsilon:=gamma^X[i];
                                if p[Index(delta,epsilon)] eq 0 then;
                                        u[Index(delta,epsilon)]:=
                                           u[Index(delta,gamma)]*X[i];
                                        p[Index(delta,epsilon)]:=sigma;
                                        Include(~newb,epsilon);
                                        //Append(~B[sigma],epsilon);
                                else;
                                        return false,u[Index(delta,gamma)]*X[i]
                                        *u[Index(delta,epsilon)]^(-1);
                                end if;
                        end for;
                        Include(~B, newb );
                else;
                        for j in [2..#B[rho]] do;
//DFH: avoid use of Index
                                if not B[rho][j]^X[i] in B[sigma] then;
                                //if p[Index(delta,B[rho][j]^X[i])] ne sigma then;
                                        return false,u[Index(delta,beta)]*X[i]
                                        *u[Index(delta,epsilon)]^(-1);
                                end if;
                        end for;
                end if;
        end for;

        rho:=rho+1;
```

```
end while;
//Bset:=[Seqset(b) : b in B];
return true, B;
end function;

MPres:=function(G:B:=[],S:=[],level:=3)
// This is the top level function
// Inputs: - G, a matrix group.
// - B, a base for G. (Tuple) (Optional)
// - S, a proposed strong generating set for G. (Sequence) (Optional)
// - level determines at what level MAGMA uses the inbuilt Verify to complete.
// Outputs: - Gpres, a finitely presented group isomorphic to G.
// - B, a confirmed base for G.
// - S, a confirmed strong generating set for G.
// Gpres will be a quotient of a free group, constructed below.

// PRELIMINARY

// If no base or strong generating set is provided, produce one using Random Schreier.
if (#B eq 0 or #S eq 0) then;
        if #B eq 0 then;
                RandomSchreier(G);
                B:=Base(G);
        end if;
        if #S eq 0 then;
                RandomSchreier(G);
                S:=Setseq(StrongGenerators(G));
        end if;
end if;

if Identity(G) in Seqset(S) then;
        S:=Setseq(Seqset(S) diff {Identity(G)});
end if;

//Compute basic stabilizer chain for B
SChain:=[];
SChainGens:=[];
SChain[1]:=G;
Stemp:=Seqset(S);
SChainGens[1]:=Stemp;
for i in [1..#B] do;
        for s in Stemp do;
                if B[i]^s ne B[i] then;
                        Exclude(~Stemp,s);
                end if;
        end for;
        SChain[i+1]:=sub<Generic(G)|Stemp>;
        SChainGens[i+1]:=Stemp;
end for;
```

```
// New part - verifying that the given set B is actually a base
if #SChainGens[#B+1] ne 0 then;
        RandomSchreier(G);
        B:=Base(G);
        SChain:=[];
        SChainGens:=[];
        SChain[1]:=G;
        Stemp:=Seqset(S);
        SChainGens[1]:=Stemp;
        for i in [1..#B] do;
                for s in Stemp do;
                        if B[i]^s ne B[i] then;
                                Exclude(~Stemp,s);
                        end if;
                end for;
                SChain[i+1]:=sub<Generic(G)|Stemp>;
                SChainGens[i+1]:=Stemp;
        end for;
end if;

// Additional check: at this point, we can check that no base points are
// superfluous. If they are, then we can run MVerify again without said base point.
// checkgens is a vector with #B entries, one for each base point.
// Each is a subset of {1..#S}, with n in the set iff S[n] stabilizes B;
// We reject the base point B[j] if there exists a k such that checkgens[j] is
// contained in checkgens[k].

checkgens:=[];
for i in [1..#B] do;
        checkgens[i]:={0};
        for j in [1..#S] do;
                if B[i]^S[j] eq B[i] then;
                        Include(~checkgens[i],j);
                end if;
        end for;
end for;

// Introduce level - only the top few levels of SChain will be computed by MPres,
// the rest will be done by MAGMA.

if level gt #B then;
        X:=[];
        H:=sub<Generic(G)|X>;
        free:=FreeGroup(1);
        rels:=[];
else;
        for s in SChainGens[level] do;
                Exclude(~S,s);
        end for;
        AssertAttribute(G,"Base",B);
```

```
        Btemp:=<>;
        for i in [1..level] do;
                Append(~Btemp,B[i]);
        end for;
        AssertAttribute(SChain[level],"Base",Btemp);
        Verify(SChain[level]);
        //DFH: need to know current order
        ordH := #SChain[level];
        X:=[];
        pres,presmap:=FPGroupStrong(SChain[level]);
        for i in [1..NumberOfGenerators(pres)] do;
                X[i]:=presmap(pres.i);
                Include(~S,X[i]);
        end for;
// This is included to prevent computational issues with MAGMA not recognising X
// as coming from the correct group.
        SChain:=[];
        SChainGens:=[];
        SChain[1]:=G;
        Stemp:=Seqset(S);
        SChainGens[1]:=Stemp;
        for i in [1..#B] do;
                for s in Stemp do;
                        if B[i]^s ne B[i] then;
                                Exclude(~Stemp,s);
                        end if;
                end for;
                SChain[i+1]:=sub<Generic(G)|Stemp>;
                SChainGens[i+1]:=Stemp;
        end for;
        Xtemp:=[];
        for s in SChainGens[level] do;
                if s ne Identity(G) then;
                        Xtemp[Index(X,s)]:=s;
                end if;
        end for;
        X:=Xtemp;

// End of additions to correct MAGMA

        free:=FreeGroup(NumberOfGenerators(pres));
        presphi:=hom<pres -> free|[free.i : i in [1..NumberOfGenerators(free)]]>;
        rels:=presphi([LHS(q): q in Relations(pres)]);
        H:=sub<Generic(G)|X>;
        AssertAttribute(H,"Order",ordH);
end if;

//In the for loop, we work up the stabilizer chain SChain, performing Verify at
// each stage beyond that specified above.
for i in [Minimum(#B,level-1)..1 by -1] do;
```

```
        alpha:=B[i];
        Y:=X;
        K:=H;
        // As we work up the chain, Y and K will be the same as X and H
        // from previous step.
        for x in SChainGens[i] do;
                if (x notin SChainGens[i+1]) then;
                        X[#X+1]:=x;
//Set up X so that new generators of SChain[i] are at the end of X.
//This ensures that the relations in rels stay mapped to the same elements in G.
                end if;
        end for;
        H:=sub<Generic(G)|X>;
        AssertAttribute(H,"Order",#K * #BasicOrbit(G,i));

        if (#X-#Y) eq 1 then;
                // SINGLE GENERATOR CASE
                // Here we directly use MVerify to compute our relators

                freetemp:=free;
//Here free will be a free group on #Y generators, reusing the previous free group
                free:=FreeGroup(#Y+1);
//Set up new free group with one additional generator
                f:=hom<freetemp -> free | [free.i: i in [1..#Y]]>;
                rels:=[f(r):r in rels];
//Transfer generators to new free group with extra generators
                //delta:=[a: a in alpha^H];
                delta:= SetToIndexedSet(alpha^H);
                z:=X[#X]; // The single new generator must be the last entry in X.

                // Use MVerify
                Btemp:=<>;
                for btem in [i..#B] do;
                        Append(~Btemp,B[btem]);
                end for;
                t,out:=MVerify(H,K,free,rels,z,alpha,X,Y,delta,Btemp);
// t=false corresponds to S not being a strong generating set, whence out will be
// an element to add to the set.
// We then rerun the algorithm with the new element of the strong generating set.

                if t eq false then;
                        Include(~S,out);
                        return $$(G:B:=B,S:=S);
                else;

// t=true means process ran successfully, and out is the relations that generate the
// current level of the chain

                        rels:=out;
                end if;
```

```
        else;
                // MULTIPLE GENERATOR CASE
                s:=#X-#Y;

 // We effectively do s applications of MVerify here, with some cases to consider
// Kmult stores the chain of subgroups generated by adding one element of X at a
// time. It will have length s+1 if the algorithm completes.
// The first element of KMult is K, the last will be H, and as the index increases,
// the groups get larger.
// deltamult stores the orbit of alpha under the various elements of Kmult
// vmult stores Schreier vectors corresponding to the elements of deltamult,
// used in MUBetaWord later on.

                Kmult:=[K];
                //delta:=Setseq(alpha^H); deltamult:=[]; vmult:=[];
                delta:= SetToIndexedSet(alpha^H); deltamult:=[]; vmult:=[];
                deltamult[1],vmult[1]:=MOrbitSv(alpha,X[1..#Y],delta);
                for j in [1..s] do;

                        freetemp:=free;
//Here freetemp will be a free group on #Y+j-1 generators
                        free:=FreeGroup(#Y+j);
//Set up new free group with one additional generator;
                        f:=hom<freetemp -> free | [free.i: i in [1..#Y+j-1]]>;
                        rels:=[f(r):r in rels];
//Transfer generators to new free group with extra generators#

                        Kmult[j+1]:=sub<Generic(H)|X[1..#Y+j]>;
                        deltamult[j+1],vmult[j+1]:=MOrbitSv(alpha,X[1..#Y+j],delta);

//We again consider a number of cases

                        if deltamult[j+1] eq deltamult[j] then;

//NO INCREASE IN ORBIT SIZE
// In this situation, adding the extra generator to Kmult[j] has not
// changed the orbit at all. Here, there is only one term to check
// for membership of K - we follow the methods of MVerify here.

                            phi:=hom<free -> Generic(Kmult[j])|X[1..#Y+j]>;
                            Sgen:=[];
                            Sgen[1]:=X;
                            Stemp:=Sgen[1];
                            for i in [1..#B] do;
                                    for j in [1..#Stemp] do;
                                            if B[i]^Stemp[j] ne B[i] then;
                                                    Stemp[j]:=Identity(K);
                                            end if;
                                    end for;
                                    Sgen[i+1]:=Stemp;
```

```
                                end for;


                                beta:=alpha^X[#Y+j];
                                tbeta:=MUBetaWord(beta,vmult[j],X[1..#Y+j],free,delta);

                                vstrip:=<>;
                                delstrip:=<>;
                                for i in [1..#B] do;
                                       Append(~delstrip,SetToIndexedSet
                                       (B[i]^sub<Generic(G)|Sgen[i]>));
                                       deltemp,vtemp:=MOrbitSv(B[i],Sgen[i],delstrip[i]);
                                       Append(~vstrip,vtemp);
                                end for;

                                h,length,w:=MStrip(phi(free.(#Y+j)*tbeta^(-1)),B,K,Y,
                                Sgen,free,vstrip,delstrip);
                                if h ne Identity(K) then;
                                        Include(~S,h);
                                        return $$(G:B:=B,S:=S);;
                                else;
                                        rels:=Append(rels,free.(#Y+j)*tbeta^(-1)*w^(-1));
                                end if;

// We could be in the situation where we have a redundant free generator.
// This will remove it.

                                vals:={};
                                valsmod:={};
                                for i in [1..#rels] do;
                                     vals:=vals join Seqset(ElementToSequence(rels[i]));
                                end for;

                                for v in vals do;
                                       if v ne Abs(v) then;
                                                Include(~vals,Abs(v));
                                                Exclude(~vals,v);
                                        end if;
                                end for;

                                if #vals ne NumberOfGenerators(free) then;
                                        Exclude(~S,X[#X]);
                                        return $$(G:B:=B,S:=S);;
                                end if;
                                AssertAttribute(Kmult[j+1],"Order",#Kmult[j]);
                        elif deltamult[j] eq {@ alpha @} then;

// TRIVIAL ORBIT
// This covers the case when deltamult increases for the first time, which is
// computationally easier than subsequent times. We run MVerify on the
```

```
// group Kmult[j+1] which contains Kmult[j], exactly as in the single generator
// case. These groups are generated by X[1..#Y+j+1] and X[1..#Y+j] respectively.
                                z:=X[#Y+j];
                                Btemp:=<>;
                                for btem in [i..#B] do;
                                        Append(~Btemp,B[btem]);
                                end for;
                                AssertAttribute(Kmult[j],"Order",#K);

                                t,out:=MVerify(Kmult[j+1],Kmult[j],free,rels,z,
                                alpha,X[1..#Y+j],X[1..#Y+j-1],deltamult[j+1],Btemp);

                                if t eq false then;
                                        Include(~S,out);
                                        return $$(G:B:=B,S:=S);;
                                else;
                                        rels:=out;
                                        AssertAttribute(Kmult[j+1],"Order",
                                        #Kmult[j] * #deltamult[j+1]);
                                end if;
                        else;
// NONTRIVIAL ORBIT
// This is the most complex verification stage, and proceeds in two parts.
// The first is to check whether alpha^Kmult[j] is a block for Kmult[j+1] on delta[j]
// We use MBlockVerify to check this

                                t,block:=MBlockVerify(Kmult[j+1],X[1..#Y+j]
                                ,X[1..#Y+j-1],alpha);

// If we do not have a block, then MBlockVerify gives us an element to add to the
// strong generating set.

                                if t eq false then;
                                        Include(~S,block);
                                        return $$(G:B:=B,S:=S);;

                                else;

// If we do have a block, then we can use MVerify on the induced action.

                                        act:=block;
                                        z:=X[#Y+j];
                                        Btemp:=<>;
                                        for btem in [i..#B] do;
                                                Append(~Btemp,B[btem]);
                                        end for;

                                        t,out:=MVerify(Kmult[j+1],Kmult[j],free,rels,
                                        z,act[1],X[1..#Y+j],X[1..#Y+j-1],act,Btemp);
```

```
                                        if t eq false then;
                                                Include(~S,out);
                                                return $$(G:B:=B,S:=S);;
                                        else;
                                                rels:=out;
                                                AssertAttribute(Kmult[j+1],"Order"
                                                ,#Kmult[j] * #act);
                                        end if;
                                end if;
                        end if;
                end for;
        end if;
end for;

// If we escape the for loop, it means we have all the required relations in rels
//  and we construct the presentation.

Gpres:=quo<free|rels>;
return Gpres,B,X;
end function;
```