COMPOSITION TREE IN MAGMA

HENRIK BÄÄRNHIELM, CHARLES LEEDHAM-GREEN, AND EAMONN O'BRIEN

Abstract. Design description and documentation for Magma implementation of `CompositionTree`.

1. Introduction

This is a description of the implementation of the `CompositionTree` Magma package. The motivation comes from the *matrix group recognition project* [LG01, O'B06], where the desire is to compute the structure of a given matrix group. Permutation group techniques (BSGS) can be used for matrix groups up to a certain degree and finite field size, but beyond that something else is necessary. This package is intended for that purpose.

One would like to fit into the *Soluble Radical* method of [HEO05, Chapter 10], since this enables the use of many algorithms that are implemented in Magma. The method of [HS08, Sta06] achieves this, but this package fit into an alternative and simpler approach.

(1) Compute a *composition tree* of the given matrix group, which is what this package does. The overall strategy for this is described in [LG01], but new ideas from [NS06] will be used.
(2) Compute a composition series of the group from the composition tree.
(3) Rearrange the composition factors in order to obtain a chief series.

This approach has the advantage that the algorithm of [LG01] is simpler than the one in [HS08], and therefore easier to implement and optimise. The rearranging is also a simple algorithm, and can use the composition tree as a black box with a well-defined interface, without knowing how it was produced. The disadvantage towards [HS08] is that certain groups will require a lot of rearranging[1].

A composition tree of a group can be seen as a data structure for storing composition factors, which can be built up recursively. It can also be seen as a data structure that facilitates constructive membership testing (rewriting) in a group, using the algorithm described in [LG01]. The ability to perform rewriting is the only thing that is immediately available when a composition tree has been constructed, and it is an essential feature, which is also necessary in the recursive construction algorithm. It depends on the availability of rewriting algorithms in the finite simple groups, a dependency which is only almost satisfied. To efficiently obtain composition series we also need constructive recognition algorithms for finite simple groups. In the cases where implementations of these are not available, we have to rely on BSGS computations and on [HLO$^+$08].

A *rewriting* algorithm for a group G is an algorithm for the *constructive membership problem* in G: given $g \in U \geqslant G = \langle X \rangle$, decide whether or not $g \in G$, and if so express g as a straight line program in X. The first component is the memberhip problem, and sometimes only the second component is called rewriting. The *constructive recognition problem* is as follows: construct an isomorphism φ from G to a *standard copy* H of G such that $\varphi(g)$ can be computed efficiently for every $g \in G$.

[1]The typical example is imprimitive matrix groups like $\mathrm{SL}(2,7) \wr \mathrm{Sym}(20)$.

Such an isomorphism is called *effective*. Sometimes one also demands that φ^{-1} is also effective.

Constructing the composition factors from the tree requires more work, occasionally much more, typically when the tree includes leaves where we have to compute BSGS. This is the reason that constructing the tree and the series are separate operations. One may wish to only have the tree, if one is only interested in rewriting and not in the composition series.

The overall strategy for building the composition tree is as follows.

(1) Given a group G, construct an effective homomorphism $\phi : G \to I$, for some group I, or notice that G is simple. If G is simple, it is a leaf. Assume henceforth that G is not simple.
(2) Construct a composition tree for I.
(3) Construct generators for $K := \mathrm{Ker}(\phi)$.
(4) Construct a composition tree for K.
(5) Put together the composition trees for I and K into a tree for G.

Here, the ability of rewriting in I is necessary in order to construct K.

The overall strategy above does not by itself assume that the input is a matrix group, but this assumption is needed in the first step. From [LG01], we can use Aschbacher's theorem [Asc84] for matrix groups. This requires algorithms for deciding if G lies in a certain Aschbacher class, and to construct the corresponding ϕ. Such algorithms exist and are implemented in Magma.

As a consequence, in the matrix group context a leaf is not necessarily simple. A matrix leaf is either a classical group in natural representation or a central extension of a non-abelian finite simple group in matrix representation, *i.e.* simple modulo scalars. Leaves can also be cyclic, not necessarily of prime order, but these are not stored as matrix groups.

One can proceed in a similar way for permutation groups, using the O'Nan-Scott theorem instead of the Aschbacher theorem in order to implement the first step. However, in the permutation group context there is already an extensive library of algorithms which only require a BSGS. For a small base permutation group one can easily find a BSGS, so our goal in the permutation group context is only to apply reductions to small base groups and then use the existing permutation group machinery.

2. Interface for factor rearranging

The algorithm that rearranges the composition factors requires the ability to construct a composition series together with a number of homomorphisms. For a group G, it requires the following.

- A composition series $\langle 1 \rangle = G_0 < G_1 < \cdots < G_k = G$, hence generating sets for each G_i.
- For each $i > 0$, an effective homomorphism $\varphi_i : G_i \to S_i$, where S_i is a *standard copy* of G_i/G_{i-1}. The standard copy should be chosen so that computations inside S_i are as easy as possible. The kernel of φ_i should be G_{i-1}. However, it is allowed that S_i is not simple, but a central extension of a finite simple group. In this case, $\mathrm{Ker}(\varphi_i) = (G_{i-1}.Z)/Z$ where $Z = \varphi_i^{-1}(\mathrm{Z}(S_i))$. The typical example where this is necessary is when $G_i/G_{i-1} \cong \mathrm{PSL}(d, q)$, then $S_i = \mathrm{SL}(d, q)$, because it is too expensive to explicitly write down $\mathrm{PSL}(d, q)$ (a permutation group) when d or q are large.
- For each $i > 0$, the effective inverse φ_i^{-1}.
- For each $i > 0$, a rewriting algorithm for S_i.

An additional requirement is an algorithm that, given G and $g \in G$, returns the minimal i such that $g \in G_i$. Given the above homomorphisms, this algorithm is as in Algorithm 1. It only relies on the observation that for any j, if $\varphi_j(g) = 1$ then $g \in \mathrm{Ker}(\varphi_j) = G_{j-1}$ and therefore the required minimal i is less than j.

Algorithm 1: FactorNumber(g)

```
1   for i := k − 1 downto 1 by −1
        do
2           h := φ_i(g)
3           if S_i is simple
              then if h ≠ 1
4                       then return i + 1
5             else  if h ∉ Z(S_i)
6                       then return i + 1
    end
7   return 1
```

3. Data structure

A composition tree is stored as a full binary tree. Each node represents a group and has various data attached to it. The right child represents the image of a reduction and the left child represents the kernel.

If G is the node group, the relationship is that the group I of the right child should be the image of G under a homomorphism ϕ, and the group K of left child is then $\mathrm{Ker}(\phi) \trianglelefteqslant G$. However, we need to generalise this slightly. It is allowed that the right child group is only the image of the homomorphism after dividing it by a subgroup of the group generated by its scalar flag. More details about this in Section 7. We also allow that the kernel is not strictly a subgroup, but isomorphic to a subgroup. The isomorphisms must be effective and should be constructed at the same time as ϕ. This abstracts away the details of the mappings between the node and its children from the rest of the algorithm.

This generalisation allows us to store the kernel in a different representation, which can be more efficient. For instance, if ϕ is a tensor decomposition of a matrix group, then we want to store both the image and the kernel as matrix groups of smaller dimension.

Here is a list of the important data elements stored in a node, after the composition tree rooted at the node is constructed.

- A list of input generators, generating the group G of the node. If the node is a right child, then these should be in bijection with the input generators of the parent.
- An instance of the product replacement algorithm [CLGM+95, LGM02, Pak01, Pak00] on the input generators.
- A list of reduction algorithms that can be used to find a reduction homomorphism, in a given order. If the group is a matrix group, this is the list of Aschbacher reduction algorithms, but the order may need to change at runtime. For example, if a matrix group is tensor induced, then the kernel is a tensor product, so the order of the reduction algorithms in the kernel can be changed so that the tensor decomposition algorithm is first.
- A list of *nice* generators, also generating G. In an internal node, these consist of the nice generators of the kernel and image. In a leaf the nice generators are not specified in general, but for example in a classical group,

they are the standard generators of the leaf group. The idea is to replace the input generators with a smaller generating set, which allows for having shorter SLPs in constructive membership testing.
- A list of SLPs of the nice generators, in the input generators.
- A rewriting algorithm, that expresses any element of G as an SLP in the nice generators.
- A list of elements used to verify the the subtree rooted at the node. This is done by rewriting the elements using the constructed rewriting algorithm, thus verifying that the elements lie in the node group. These elements, that serve the purpose of quality control, are called the *mandarins* (after the government officials in Imperial China.)
- A list of SLPs of the mandarins, in the nice generators.
- A list of SLPs of relators for a presentation on the nice generators.
- A list of lists of SLPs of generators for each composition factor of G, in the nice generators.
- The three lists of homomorphisms described in Section 2.
- The scalar flag for G. If G is a permutation group this is always the identity. If $G \leqslant \mathrm{GL}(d, q)$ it is a scalar matrix $\alpha \,\mathrm{Id}_d$ where $\alpha \in \mathbb{F}_q^\times$.

In the case that the node is not a leaf, some additional data is stored.
- A list of SLPs of the generators of the left child, the kernel, in the input generators of the node.
- The reduction homomorphism ϕ.

In the case that the node is a leaf, there are also some additional data.
- The name of the leaf group, from [BKPS02].
- Generators of the standard copy S. This is typically the corresponding finite simple group in a small degree representation. Hence it is not necessarily isomorphic to the leaf group, in the case where this is a central extension of the simple group.
- An effective isomorphism $\alpha : G \to S$.
- An effective isomorphism $\beta : S \to G$.

To be precise, α and β are allowed to be isomorphisms modulo scalars. Hence we require that if $G \cong Z.S$ where $Z = \mathrm{Z}(G)$ is a cyclic group, then $\beta(\alpha(g)) \in Z$ for every $g \in G$. If G is a matrix group, then Z consists of scalar matrices, because a matrix leaf is absolutely irreducible. If G is a permutation group leaf which is a central extension of a simple group, then it is in fact simple, by the O'Nan-Scott theorem, so $Z = \langle 1 \rangle$ in this case.

4. General algorithm

We now describe aspects of the general composition tree algorithm.

4.1. **Main recursion.** The main algorithm proceeds as follows.

(1) Try to find a reduction homomorphism $\phi : G \to I$. Hence we try each reduction algorithm in turn, until one succeeds, see Sections 5 and 8.
(2) If no reduction is found, the node is marked as a leaf and the main recursion is finished, see Section 9.
(3) Setup the data structure for the right child, and recursively find a composition tree.
(4) Find a probable generating set for K, see Section 4.2.
(5) Recursively find a composition tree for K. This may provoke a crisis, see Section 4.2.
(6) Setup the nice generators and the rewriting algorithm for the node.
(7) Verify the rewriting using the mandarins. This may provoke a crisis.

(8) Possibly construct a presentation from I and K and see if G satisfies it, see Section 4.3. This may provoke a crisis.

In the case of a crisis, if the node is safe we add more generators to K and proceed from that step, otherwise we send the crisis to the parent.

4.2. **Kernel generation.** We use two general methods for constructing a probable generating set for the kernel K. The first is the method described in [LG01]:

(1) Construct a set $Y = \{g_1, \ldots, g_n\}$ of random elements of G using the random process in the node.
(2) Obtain SLPs for $\{\phi(g) : g \in Y\} \subseteq I$, using the rewriting algorithm in the right child.
(3) We obtain the SLPs in the input generators of I, which are in bijection with the input generators of G. Hence we can evaluate the SLPs in G and obtain a set $\bar{Y} = \{\bar{g}_1, \ldots, \bar{g}_n\}$ corresponding to Y.
(4) Now $g_i/\bar{g}_i \in K$ for each i, and this is a random element of K.

This method has no provable probability of success in general, and is only a heuristic. For a given group it is a Monte Carlo algorithm, although with unknown success probability, and as a heuristic it is very practical.

The other method assumes that a presentation of $I \cong \langle Y \mid R \rangle$ is available. In our case, Y is the nice generating set of I, but we can easily rewrite the presentation on the input generators.

(1) Let $G = \langle X \rangle$. Evaluate the relators R on X. This relies on the bijection between the input generators of I and X, and produces $Z_1 \subseteq K$.
(2) Map X to I, obtain SLPs and divide, as in the previous method, to obtain $Z_2 \subseteq K$.
(3) Now $Z = Z_1 \cup Z_2$ is a normal generating set for K. In other words, K is the normal closure of $\langle Z \rangle$. Note that $|Z| = |R| + |X|$.
(4) Use the algorithm in [CF93] to construct probable generators of the normal closure of $\langle Z \rangle$. This is a Monte Carlo algorithm, and the number of generators it constructs depends on the given error probability and the length of the longest subgroup chain in G, usually denoted l_G.

Note that there are no practical upper bounds on l_G. If $G \leqslant \mathrm{GL}(d, q)$ then $l_G \leqslant d^2$, and this bound is optimal since it is achieved by p-groups. If $\mathrm{O}_p(G) = \langle 1 \rangle$ then $l_G \leqslant d$, which is better but still not useful in practice, and this bound is also optimal. If $G \leqslant \mathrm{Sym}(n)$ then $l_G \leqslant n$, again an optimal upper bound. In practice, we use a very small value of l_G.

We prefer the second method of generating the kernel, whenever it can be used. It depends upon our ability to obtain presentations of the leaves, because we can construct a presentation of a node if both its children have presentations. A drawback is that the number of relators can be very large, and hence the number of kernel generators will be large.

Note that both these methods of finding the kernel are Monte Carlo, so there is a possibility that we fail to find the full kernel. This has consequences for the design of the composition tree algorithm. It means that constructive membership testing is insecure, if it reports that an element does not lie in a group corresponding to some node, this may mean that some left child on the path from the node to the root was not calculated correctly. This situation is called a *crisis*, and if it occurs we must discard a part of the tree and recompute it. The mandarins are used to test constructive membership testing, to try to provoke a crisis.

If there are several kernel computations on the path to the root, we have no way of knowing which one of them that is incorrect. We therefore introduce the notion of a *safe* node.

- The root node is safe.
- A right child is safe if and only if its parent is safe.
- A left child is never safe unless it is calculated using some special method that is guaranteed to give the full kernel.

Now if we get into a crisis during rewriting in some node, we must backtrack along the path from the node to the root, until we reach a safe node. We must discard the whole left subtree of this node, except that we store the generators of its left child. Then we can add more generators of this kernel and try again.

4.3. **Verification.** The composition tree algorithm is inherently probabilistic and there are many computations along the way which return results that may be wrong with some positive probability. In effect it is a Monte Carlo algorithm, although the success probability is unknown, and it is not clear that there exists a constant upper bound on this probability if the input is an arbitrary matrix group over a finite field.

Therefore there is a positive probability that the composition tree is only a composition tree for a subgroup of the input group.

To verify the composition tree, and hence eventually turn it a Las Vegas algorithm, we calculate a presentation from the composition tree and see if this is a presentation of the input group. In other words, we test if all the relators, which must be relators for a subgroup, are also relators in the input group.

There are two options, either we verify during construction the composition tree, or afterwards. In the first case, when the composition trees of the right and left children have been constructed, they are immediately verified before putting them together.

Presentations can be constructed recursively, given presentations of the image and kernel. For a leaf, we require that the leaf algorithm constructs a presentation on the nice generators. This is possible in theory for all finite simple groups except the small Ree groups [GKKL08b, GKKL08a, Bra07, BCLGO06], but not always possible in practice for every type of leaf, in which case we cannot construct a presentation for the composition tree.

To construct a presentation recursively for G, assume that $I \cong \langle X_I \mid R_I \rangle$ and $K \cong \langle X_K \mid R_K \rangle$ are presentations for the image and kernel, where X_I and X_K are the nice generators and R_I and R_K are SLPs of the relators. Recall that the nice generators of G is $X_G = X_K \cup \phi^{-1}(X_I)$, where $\phi^{-1}(X_I)$ is calculated by evaluating the SLPs of the nice generators of I on the input generators of G. This relies on the bijection between the input generators of G and I. Now proceed as follows.

(1) Rewrite R_I on X_G to obtain $R_I^G = \{r_1^G, \ldots, r_n^G\}$ and evalute them on X_G to obtain elements $Y_I \subseteq G$.
(2) Since R_I are relators of I, $Y_I \subseteq K$, so obtain SLPs of Y_I in X_K and rewrite them on X_G, to obtain $R_I^K = \{r_1^K, \ldots, r_n^K\}$.
(3) Let $Z_I = \{r_i^G / r_i^K : i = 1, \ldots, n\}$.
(4) Rewrite R_K on X_G to obtain Z_K.
(5) Let $R_K^G = \{s_1^G, \ldots, s_m^G\}$ be the SLPs in X_G corresponding to

$$\{x^y : x \in X_K, y \in \phi^{-1}(X_I)\}.$$

Evaluate them on X_G to obtain elements $C \subseteq G$.
(6) Since K is normal in G, $C \subseteq K$ and we can obtain SLPs of C in X_K. Rewrite these on X_G to obtain $R_K^K = \{s_1^K, \ldots, s_m^K\}$.
(7) Let $Z_C = \{s_i^G / s_i^K : i = 1, \ldots, m\}$.
(8) Now $G \cong \langle X_G \mid Z_I \cup Z_K \cup Z_C \rangle$ is a presentation.

Note that the number of relators in the presentation for G is $|R_K| + |R_I| + |X_K|\,|X_I|$, which can grow large with the height of the tree.

5. Reduction database for matrix groups

The basic use of the Aschbacher reductions is already described in [LG01]. Here we describe our use of these. The order in which they are described here is the standard order in which they are tried at runtime.

5.1. **Unipotent.** A group $G \leqslant \mathrm{GL}(d, p^e)$ is unipotent if and only if every $g \in G$ has order $p^k \leqslant d$. Moreover, G is unipotent if and only if all composition factors of the module of G have dimension 1, and G acts trivially on each such factor.

We first employ a quick negative test to rule out most groups that are not unipotent: if some of the given generators of G do not have order a power of p, then G is not unipotent. If this test does not rule out G, then we use the MeatAxe to decompose the module of G and test if the composition factors behave as above.

If the group is unipotent, then MeatAxe provides a change of basis matrix c such that G^c is lower triangular. Each subdiagonal of G^c is then a vector space over $\mathbb{F}_p$ and hence an elementary abelian p-group, and the projection of G^c to each such group is a homomorphism. Hence we obtain a reduction to the first non-zero subdiagonal.

The image is elementary abelian and treated as PC-group, so it is a leaf. Hence we can readily obtain a presentation and therefore obtain the kernel using the presentation method.

5.2. **MeatAxe reductions.** There are several reductions coming from the MeatAxe [Par84, HR94, IL00]. Let M be the module of $G \leqslant \mathrm{GL}(d, q)$. We first find the indecomposable summands of M, so $M \cong M_1 \oplus \cdots \oplus M_k$. Then we find a composition series of each M_i, so $M_i = M_{i,n_i} > M_{i,n_i-1} > \cdots > M_{i,1} > M_{i,0} = 1$. This also provides a change of basis matrix $c_i \in \mathrm{GL}(M_i)$ that exhibits this series. Then $c = \bigoplus_{i=1}^{k} c_i$ (diagonal join) exhibits the direct sum decomposition of M as well as the composition series of each M_i. The corresponding composition series of M is

$$1 = M_{1,0} < M_{1,1} < \cdots < M_{1,n_1} < M_1 \oplus M_{2,1} < \cdots < M_1 \oplus M_2 < \cdots < M \quad (5.1)$$

Now $G^{c^{-1}}$ is both block lower triangular, corresponding to the composition factors of M, and block diagonal, coresponding to the direct summands of M. We obtain a homomorphism by projecting onto the diagonal blocks corresponding to the composition factors. The kernel of this is $\mathrm{O}_p(G)$. Since this is the first reduction in the composition tree, unless G is unipotent from the start, the composition tree has the property that $\mathrm{O}_p(G)$ is the first kernel. Therefore the composition factors coming from $\mathrm{O}_p(G)$ will be at the bottom of the composition series constructed from the tree. This is a useful, because it avoids later rearranging of the (potentially large number of) composition factors of $\mathrm{O}_p(G)$.

The group of the right child is now contained in $\mathrm{GL}(M_1) \times \cdots \times \mathrm{GL}(M_k)$, embedded in $\mathrm{GL}(d, q)$. From this node we therefore obtain a homomorphism onto the first non-trivial summand block. The right child of this reduction is contained in $\mathrm{GL}(M_{1,1}).\,\mathrm{GL}(M_{1,2}).\,\ldots.\,\mathrm{GL}(1, n_1)$, embedded in $\mathrm{GL}(M_1)$, so from this node we obtain a homomorphism onto the first non-trivial composition factor block of M_1.

Continuing in this way, we obtain reductions to the groups acting on the composition factors of M. The reason for going via the summands is to ensure that the matrices are as sparse as possible, and to retain information on how G acts on M. Moreover, we know that the reduction to the last factor inside each summand must be an isomorphism, because the kernel is trivial. If we had mapped to the

composition factors immediately we would have lost this information and would had to calculate the kernel.

5.3. **Absolute reducibility.** Groups that act irreducibly but not absolutely irreducibly are part of the *semilinear* Aschbacher class. However, here we consider these separately, because we have a much faster reduction homomorphism in this special case. An irreducible group $G \leqslant \mathrm{GL}(d, q)$ is absolutely reducible if is reducible when embedded into $\mathrm{GL}(d, q^e)$ for some $e > 1$. The smallest such e gives the *splitting field* $\mathbb{F}_{q^e}$ for G. Then G has a faithful representation $G \to \mathrm{GL}(d/e, \mathbb{F}_{q^e})$, so we obtain an isomorphism. This isomorphism is in the MAGMA kernel and hence easy to calculate.

5.4. **Semilinearity.** For the general semilinear case we use part of [HLGOR96], known as SMASH. In this case there exists $N \lhd G = \langle X \rangle \leqslant \mathrm{GL}(d, q)$ that is not absolutely irreducible, so it has a splitting field $\mathbb{F}_{q^e}$. Moreover, there exists a non-scalar $C \in \mathrm{Z}(N)$ such that for every $g \in G$ there exists $1 \leqslant i = i(g) \leqslant e$ such that $Cg = gC^{q^i}$. The SMASH algorithm provides e and C, and we then obtain a homomorphism $G \to \mathrm{C}_e$, given by $g \mapsto i(g)$.

In this case we can also immediately obtain generators for the kernel K. Let the image $I = \langle Y \rangle \leqslant \mathcal{C}_e$, where $|Y| = |X|$. Let $X = \{x_1, \ldots, x_m\}$ and $Y = \{y_1, \ldots, y_m\}$.

(1) Use the extended Euclidean algorithm to obtain $n = \gcd(y_1, \ldots, y_m, |I|)$ and an expression $n = a_0 |I| + \sum_{i=1}^{m} a_i y_i$.
(2) Let $x = \prod_{i=1}^{m} x_i^{a_i}$ and $h_i = x_i x^{-y_i/n}$ for $i = 1, \ldots, m$. Then $\{h_1, \ldots, h_m\}$ is a normal generating set for K.
(3) Let $k_0 = x^{|I|/n}$ and $k_{i,j} = h_i^{x^j}$ for $i = 1, \ldots, m$ and $j = 1, \ldots, e$.
(4) Now $K = \langle \{k_0\} \cup \{k_{i,j} : 1 \leqslant i \leqslant m, 1 \leqslant j \leqslant e\} \rangle$.

5.5. **Imprimitivity.** A matrix group $G \leqslant \mathrm{GL}(d, q)$ is imprimitive if it permutes a non-trivial direct sum decomposition of $V = \mathbb{F}_q^d$. Hence $V \cong V_1 \oplus \cdots V_k$, where all V_i are isomorphic and G permutes these. This provides a homomorphism $G \to \mathrm{Sym}(k)$. The SMASH algorithm is used to construct this homomorphism.

The kernel consists of those elements that preserves the decomposition of V, and hence it is reducible. The SMASH algorithm provides a change of basis matrix such that the kernel is block diagonal, and hence we can immediately go to one of the MeatAxe reductions above, that map to the direct summands. However, we find generator for the kernel we have to use the general methods.

5.6. **Tensor products.** A matrix group $G \leqslant \mathrm{GL}(d, q)$ lie in this Aschbacher class if its natural module $V = \mathbb{F}_q^d$ has the structure of a tensor product $V \cong U \otimes W$, and G respects this tensor product. It follows that $G \cong H_1 \circ H_2$ where $H_1 \leqslant \mathrm{GL}(U)$ and $H_2 \leqslant \mathrm{GL}(W)$.

We use [LGO97], with implementation by Eamonn O'Brien, to obtain a change of basis matrix $c \in \mathrm{GL}(d, q)$ such that G^c is an explicit Kronecker product. It is straightforward to obtain the factors of the Kronecker product, and hence we obtain a homomorphism $G \to H_1$. This homomorphism is in fact only a homomorphism modulo scalars, see Section 7.

5.7. **Tensor induction.** A matrix group $G \leqslant \mathrm{GL}(d, q)$ is tensor induced if its natural module $V = \mathbb{F}_q^d$ has the structure of a tensor product $V \cong U_1 \otimes U_2 \otimes \cdots \otimes U_k$, where all U_i are isomorphic and G permutes the tensor factors. This defines a homomorphism $G \to \mathrm{Sym}(k)$.

We use the algorithm of [LGO02], with implementation by Eamonn O'Brien, to compute this homomorphism.

We have no special method of obtaining the kernel in this case, but we know that the kernel consists of those elements that preserves the tensor product of V, and hence it falls in the previous Aschbacher class. Hence we can immediately go to that reduction when processing the kernel.

5.8. **Extra-special normalisers.** A group $G \leqslant \mathrm{GL}(d, q)$ belongs to this Aschbacher class if it normalises an r-group R of order r^{2m+1} or 2^{2m+2}, where r is prime, $r^m = d$ and $r \mid q - 1$. If $r > 2$ then R is extraspecial, and if $r = 2$ then R is either extraspecial or of symplectic type, that is, a central product of an extraspecial 2-group and a cyclic group of order 4.

If $m = 1$ then the algorithm of [Nie05], with implementation by Alice Niemeyer and Eamonn O'Brien, constructs a homomorphism $G \to \mathrm{GL}(2, r)$. If $m > 1$ then the algorithm of [BNS06], with implementation by Henrik Bäärnhielm, constructs a homomorphism $G \to \mathrm{GL}(2n, r)$ or $G \to \mathrm{Sym}(r^n)$, where $1 \leqslant n \leqslant m$.

The latter algorithm also provides the kernel of the homomorphism, but in the former case we have no special method of obtaining the kernel.

5.9. **Smaller fields.** In this case, $G \leqslant \mathrm{GL}(d, \mathbb{F}_q)$ is conjugate to $H \leqslant \mathrm{GL}(d, \mathbb{F}_s)Z$ where $Z = \mathrm{Z}(\mathrm{GL}(d, \mathbb{F}_q))$ and $\mathbb{F}_s < \mathbb{F}_q$. If $H \leqslant \mathrm{GL}(d, \mathbb{F}_s)$ then this conjugation is an isomorphism $G \to H$, so there is no kernel. In the general case, we obtain a homomorphism $G \to H \cap Z$, so the image is a cyclic group. In fact, this is a homomorphism modulo scalars, see Section 7. The kernel is $H \cap \mathrm{GL}(d, \mathbb{F}_s)$, but we have no special method of obtaining this kernel.

We use [GLGO06], with implementation by Eamonn O'Brien, to compute the change of basis and to compute the homomorphism.

5.10. **C8.** The Aschbacher class C8 consists of the normalisers of classical groups in natural representation. These are therefore not leaves, but classical groups in natural representation are leaves. We use three reductions to remove the decorations and obtain the classical groups. All involves homomorphisms to cyclic groups. Unfortunately this adds the requirement to calculate discrete logarithms, potentially in the whole of $\mathbb{F}_q^\times$, since constructive membership testing in a cyclic group is the discrete logarithm problem.

To test if G contains a classical group in natural representation, we use [NP97, NP98, NP99] with implementation by Alice Niemeyer. If so, the node is marked as a leaf.

In fact, none of these reductions to cyclic groups are restricted to the Aschbacher C8 class. They only rely on the existence of elements with non-trivial determinants, and on a preserved classical form, but a group may have these properties without being in the C8 class.

Since all these reductions have cyclic images, which are leaves, we can readily obtain presentations of the images and hence calculate the kernels using the presentation method.

5.10.1. *Determinant.* The determinant map for $G \leqslant \mathrm{GL}(d, q)$ is $g \mapsto \det(g) \in \mathbb{F}_q^\times$, so the image is a cyclic group.

5.10.2. *Form action.* If $G \leqslant \mathrm{GL}(d, q)$ normalises a classical group H which is not linear, then H preserves a classical form J. Hence $gJ\bar{g}^T = J$ for every $g \in H$, where $\bar{g} = g$ unless H is a unitary group, in which case $\bar{g} = g^{\sqrt{q}}$ (q is a square in this case). Then elements of G preserves J up to a scalar, so for each $g \in G$ there exists $\lambda_g \in \mathbb{F}_q^\times$ such that $gJ\bar{g}^T = \lambda_g J$. Hence we obtain a homomorphism $g \mapsto \lambda_g$, again with the image being a cyclic group.

To determine if the group preserves a classical form up to a scalar, one can use the MeatAxe, as described in [HEO05, Section ?]. This is implemented in MAGMA by Derek Holt.

5.10.3. *Spinor norm.* If $G \leqslant \mathrm{GL}(d,q)$ is an orthogonal group, then the previous C8 reductions make sure that $G \leqslant \mathrm{SO}(d,q)$. But $\mathrm{SO}(d,q)$ is not perfect, and we again obtain a homomorphism $G \to \mathrm{C}_2$ where $g \in G$ maps to its spinor norm. The kernel of this is $\Omega(d,q)$ which is simple modulo scalars.

To calculate spinor norms, we use [MRD, Theorem 5.6], with implementation by Derek Holt.

5.11. **C9.** The Aschbacher class C9 consists of groups that are almost simple modulo scalars. Hence $G \leqslant \mathrm{GL}(d,q)$ is a C9 group if $S \leqslant G/Z \leqslant \mathrm{Aut}(S)$, where $Z = \mathrm{Z}(G)$. It is well-known that the automorphism group of a finite simple group is a small extension of the simple group. In other words, $G \cong Z.S.E$ where Z consists of scalar matrices and E is a small soluble group, usually cyclic. Because of our determinant reduction, $|Z| \mid \gcd(d, q-1)$.

For groups in this class, we have to decide which finite simple group they contain and we have to split off the decorations on top in order to obtain a central extension of the simple group, which we treat as a leaf.

Here is an overview of our C9 method.

(1) Calculate the *stable derivative* $D = G^{(\infty)}$ of G. This is done by computing commutators, taking normal closures and testing for perfectness, see [HS08] for more details. By a well-known result, we have to compute at most 3 derived groups to obtain the stable derivative if G is almost simple. Then D is simple modulo scalars, but now the scalars are restricted to those in the Schur multiplier of G.
(2) Try to name D. This is done in several steps, and if anyone of them succeeds we are done. It is possible that all fail, in which case D is probably not a C9 group.
   (a) Use a heuristic implemented by Derek Holt to test if D may be $\mathrm{Alt}(n)$ or $2.\mathrm{Alt}(n)$ (this requires knowledge of the centre), and try to find n. This is done more efficiently if we know $\mathrm{Z}(D)$, so we first calculate it as in Section 6. If the heuristic is successful, use the constructive recognition algorithm [BP00], as verification. We use the implementation by Derek Holt.
   (b) Find a batch of random elements of D and look at element orders to see if D is a sporadic group, and find its name. If this appears to be the case, try to find projective standard generators of the sporadic group, as verification. The standard generators are given in the ATLAS [ABL$^+$].
   (c) Find the characteristic of D using [LO07]. This was implemented by Eamonn O'Brien.
   (d) Find the name of D and its defining field using [BKPS02] (this requires that the characteristic is known). This was implemented by Eamonn O'Brien and Gunter Malle.
(3) Now we want to construct a homomorphism $G \to E$. This also uses the same general method as in [HS08, Sta06]. We try to find a complete set of coset representatives of $Z.S = \langle Z, D \rangle$ in G, and then construct the permutation representation of G on these cosets. We employ a few observations that help us with this.

- If D is named as a sporadic, then G is at most the full automorphism group of this sporadic. We test this by trying to find projective standard generators (we calculate projective/central orders rather than precise orders of elements) of the automorphim group of the sporadic, again using black box algorithms from [ABL$^+$]. If this succeeds, then we assume that G is the automorphism group rather than the simple group. Now we can obtain generators for the simple group immediately using explicit words in the automorphism group standard generators, given in [ABL$^+$]. For a sporadic group, the simple group has index at most 2 in its automorphism group, so in this case the maximum number of coset representatives is 2. On the other hand, if we fail to find standard generators of the automorphim group, we assume that G is in fact already a central extension of the sporadic group, and hence the node is a leaf.
- If D was named as an alternating group, there are again at most 2 coset representatives.
- Since the naming algorithm provides the defining field of G, we know if G is a representation in defining characteristic. In this case a conjecture[2] states that $|G : D| \leqslant \gcd(q-1, m)e$ where m is the maximum sum of the highest weights of G, and $e = 2$ if G is an orthogonal group and 1 otherwise. Since $|G : D| \geqslant |E|$ this provides an upper bound on the number of coset representatives. Moreover, we can calculate the maximum highest weight sum for representations of degree up to 250 using [Lüb01].

If we learn that the maximum number of cosets is 1, then we know immediately that the node is a leaf, and when enumerating coset representatives, we can stop when we have reached the maximum.

The kernel of the map onto the coset representatives is found using the general methods.

6. Centre calculations

In the C9 context, and in leaves, we need to obtain generators for $\mathrm{Z}(G)$ as `SLP`s in the generators of G. In fact, since G is absolutely irreducible and contained in $\mathrm{SL}(d, q)$, the centre consists of scalar matrices and is therefore cyclic. We require a single centre generator.

We calculate the centre using the Monte Carlo algorithm [BS01, Theorem 4.15]. In this context we know that an upper bound on the order is $\gcd(d, q-1)$. Having found a generating set X for the centre, we can calculate orders and take powers of the elements of X to obtain elements c_i of order $p_i^{e_i}$, where $|\langle X\rangle| = \prod_{i=1}^{n} p_i^{e_i}$. Our single generator is then $c = \prod_i c_i$.

7. Scalar flags

As has been indicated above, in some of the Aschbacher reductions we only obtain homomorphisms modulo scalars. Superficially this is only a detail, but it has implications for the whole composition tree design, and it complicates matters.

In the tensor product case, we have a group $G \leqslant \mathrm{GL}(d, q)$ which preserves a tensor decomposition $V \cong U \otimes W$ of its natural module. This means that $G \cong H_1 \circ H_2$. The point is that G is a central rather than a direct product, and the homomorphism is not $G \to H_1$ but $G \to H_1/\mathrm{Z}(H_1)$, so the image lives in $\mathrm{PGL}(U)$ not $\mathrm{GL}(U)$. However, we have no means of writing down a quotient of matrix

[2]Scott Murray, personal communication

groups, so we need to stay with H_1 as the image and somehow perform the quotient indirectly.

As a consequence, the tensor decomposition algorithm [LGO97] which constructs the image of $G = \langle X \rangle$ under the tensor decomposition homomorphism ϕ, only guarantee that the image $H = \langle \phi(X) \rangle \leqslant H_1 Z$ where $Z = \mathrm{Z}(\mathrm{GL}(U))$. Moreover, this is true for every element g that we map under ϕ. It is only guaranteed that $\phi(g) \in H_1 Z$, not necessarily that $\phi(g) \in H_1$ and not even $\phi(g) \in H$. The actual image group of G is not H but $H/Z = H_1 Z/Z \cong H_1/(H_1 \cap Z) = H_1/\mathrm{Z}(H_1)$.

There is a similar behaviour in the smaller field, when the group $G \leqslant \mathrm{GL}(d, \mathbb{F}_q)$ can only be written over a smaller field $\mathbb{F}_s$ modulo scalars. The algorithm of [GLGO06] constructs a homomorphism $G \to \mathbb{F}_q^\times / \mathbb{F}_s^\times$, but each computed image is only guaranteed to lie in $\mathbb{F}_q^\times$

The method we have used to work around these obstacles is that each node in the composition tree with group G has an associated *scalar flag* λ. The whole issue only arises for matrix groups, so if G is a permutation group then $\lambda = 1$, but if $G \leqslant \mathrm{GL}(d, q)$ then $\lambda \in \mathrm{Z}(\mathrm{GL}(d, q))$, a scalar matrix. Now the matrix group we can work with in the node is G, but the group that the node really represents is $G \langle \lambda \rangle / \langle \lambda \rangle$. Hence the composition factors corresponding to the node are those of $G \langle \lambda \rangle / \langle \lambda \rangle$. The rewriting algorithm in the node must be able to take as input elements of $G \langle \lambda \rangle$, and the rewriting of $g \in G \langle \lambda \rangle$ is allowed to return an `SLP` that in G evaluates to gz for some $z \in \langle \lambda \rangle$.

Every algorithm for processing a leaf must now be able to behave like this. For every reduction, we also have to determine what the scalar flags for the image and kernel nodes are, given the scalar flag in the node.

All Aschbacher reduction algorithms are applied to $G = \langle X \rangle$, but the C8 reductions are applied to $G \langle \lambda \rangle$. This is because the C8 reductions will give different answers when applied to G and $G \langle \lambda \rangle$. The input generators of the image are still the images of X, because we must maintain the bijection between these.

7.1. **Scalar flag propagation.** Let ω_q be a primitive element of $\mathbb{F}_q$. Assume the node group is G with scalar flag λ, ϕ is the reduction homomorphism, I is the group of the image and K is the group of the kernel. Let λ_I and λ_K be the scalar flags of the image and kernel. These are defined as in Table 7.1.

Reduction	λ_I	λ_K
Unipotent	$\phi(\lambda)$	1
MeatAxe	$\phi(\lambda)$	1
Absolute reducibility	$\phi(\lambda)$	λ
Imprimitivity	$\phi(\lambda)$ $(= 1)$	λ
Semilinearity	$\phi(\lambda)$ $(= 1)$	λ
Tensor product	ω_q	λ
Tensor induction	$\phi(\lambda)$ $(= 1)$	λ
Extraspecial	$\phi(\lambda)$	λ
Smaller field	$\phi(\lambda)$	λ
Smaller field modulo scalars	$\omega_q^{\lvert\omega_q\rvert/\operatorname{lcm}(\lvert\lambda\rvert,\lvert\omega_s\rvert)}$	$\lambda^{\lvert\lambda\rvert/\gcd(\lvert\lambda\rvert,n)}$
Determinant	$\phi(\lambda)$ $(= \lambda^n)$	$\lambda^{\lvert\lambda\rvert/\gcd(\lvert\lambda\rvert,n)}$
Form action	$\phi(\lambda)$ $(= \lambda^n)$	$\lambda^{\lvert\lambda\rvert/\gcd(\lvert\lambda\rvert,n)}$
Spinor norm	$\phi(\lambda)$ $(\in \{0, 1\})$	$\lambda^{2\lambda_I}$
C9 (coset action)	$\phi(\lambda)$ $(= 1)$	λ

TABLE 1. Scalar flag propagation

In the smaller field case, $\mathbb{F}_s < \mathbb{F}_q$ is the smaller field. We define n as follows.

- In the smaller field case, $n = |\omega_s| = s - 1$.
- In the determinant case, $n = d$.
- In the form action case, non-unitary case, $n = 2$. In the unitary case, $n = \sqrt{q} + 1$ (q is a square in this case).

Most of the entries in the table comes from the observation that we have a homomorphism $G \to I$, and we define the induced homomorphism from $G/(G \cap \langle \lambda \rangle)$. The image of this is $I/(I \cap \langle \phi(\lambda) \rangle)$ so $\lambda_I = \phi(\lambda)$. In the tensor product case, as described above, the image is a projective matrix group, so we must divide out by the full scalar subgroup. In the smaller field case, the image are the scalars in the larger field modulo the scalars in the smaller field. Hence we divide out by the smaller scalars, except that there already be a scalar flag which we need to take into account. The kernel is a projective matrix group over the smaller field, so we divide out by the part of the scalar flag which lies in the smaller field.

In many cases we know explicitly what $\phi(\lambda)$ is. If the image is a permutation group, the scalar flag will be the identity, because in each of these reductions a scalar will lie in the kernel of the homomorphism. It is easy to see that in the C8 cases, $\phi(\lambda) = \lambda^n$ where n is as above.

One can note that λ_K is not always guaranteed to lie in K. This is ok within our design, since the actual kernel group is $K \langle \lambda_K \rangle / \langle \lambda_K \rangle$. However, in the C8 cases we must make sure that $\lambda_K \in K$, because we will apply the C8 tests in the kernel to $K \langle \lambda_K \rangle$. Hence the same C8 test may be succesful in the kernel if $\lambda_K \notin K$, which will lead to an infinite loop.

In the MeatAxe cases, it is easy to see that the kernel cannot contain any scalars, so the scalar flag is the identity.

7.2. **Cyclic group element patches.** As noted above, a leaf must be able to handle the case when there is a non-trivial scalar flag. From some of the reductions we obtain a cyclic image, which will be a leaf. The following situation arise in all these.

Let $G = \langle X \rangle$ be the node group, $\phi : G \to I = \langle Y_I \rangle$ the reduction and $K = \mathrm{Ker}(\phi) = \langle Y_K \rangle$. Assume that I is cyclic and let λ, λ_I and λ_K be the scalar flags for G, I and K.

Consider $g \in G \langle \lambda \rangle$ which we want to express as an `SLP` in X. The first step is to express $\phi(g)$ as an `SLP` in Y_I. However, it is not guaranteed that $\phi(g) \in \langle Y_I \rangle$, only that $\phi(g) z_I \in I$ for some $z_I \in \langle \lambda_I \rangle$. Moreover, if we obtain an `SLP` w of $\phi(g) z_I$, the next step is to evaluate w on X to obtain an element h and then obtain an `SLP` of g/h in Y_K. But again, it is not guaranteed that $g/h \in \langle Y_K \rangle$. This is ok, we only require that $g/h \in K \langle \lambda_K \rangle$, but unfortunately this may not be true either.

To be more precise, if $g = \bar{g} \lambda^i$, where $\bar{g} \in G$, then $\phi(g) = \phi(\bar{g}) \lambda_I^j$. This follows from the definition of λ_I in Table 7.1. Note that we know neither i nor j, and we want to avoid computing them, as this is a discrete logarithm computation. We obtain an `SLP` for $\phi(\bar{g})$, so $h = k\bar{g}$ for some $k \in K$. Hence $g/h = k^{-1} \lambda^i$, but we see from Table 7.1 that it is possible that $\lambda^i \notin \langle \lambda_K \rangle$ and hence $g/h \notin K \langle \lambda_K \rangle$.

What is needed is therefore an algorithm that, given I and $\phi(g)$, finds $z_I \in \langle \lambda_I \rangle$ such that $\phi(g) z_I \in I$. Moreover, given λ it should also find $z \in \langle \lambda \rangle$ such that $gz/h \in K \langle \lambda_K \rangle$.

Proposition 7.1. *Let $C, \langle \lambda \rangle \leqslant \mathrm{C}_m$ and consider $x \in C \langle \lambda \rangle$. Given the list of primes dividing m, in polynomial time we can find $z \in \langle \lambda \rangle$ such that $xz \in C$.*

Proof. Let

$$|C| = \prod_{i=1}^{n} p_i^{a_i},$$
$$|c| = \prod_{i=1}^{n} p_i^{b_i},$$
$$|\lambda| = \prod_{i=1}^{n} p_i^{c_i}$$

where each p_i is prime, $a_i, b_i, c_i \geqslant 0$, $a_i + b_i + c_i > 0$ and $b_i \leqslant \max(a_i, c_i)$. Let $q_i = |x| / p_i^{b_i}$ for $i = 1, \ldots, n$, and observe that $\gcd(q_1, \ldots, q_n) = 1$. Using the extended Euclidean algorithm, we can find integers m_i such that $1 = \sum_{i=1}^{n} q_i m_i$.

Let $z_i = x^{m_i q_i}$ for $i = 1, \ldots, n$. Then

$$\prod_{i=1}^{n} z_i = \prod_{i=1}^{n} x^{m_i q_i} = x^{\sum_{i=1}^{n} m_i q_i} = x,$$

and also $|x^{q_i}| = p_i^{b_i}$. Since $|z_i| = (x^{q_i})^{m_i}$ it follows that $|z_i| = p_i^{b_i}$ unless $\gcd(|x^{q_i}|, m_i) > 1$. But the latter implies that $p_i \mid m_i$ and since $p_i \mid q_j$ for every $j \neq i$ it follows that $p_i \mid 1$, a contradiction. Therefore $|z_i| = p_i^{b_i}$.

Now let

$$z^{-1} = \prod_{b_i > a_i} z_i$$

and observe that if $b_i > a_i$ then $b_i \leqslant c_i$. This implies that $|z| \mid |\lambda|$ and hence $z \in \langle \lambda \rangle$.

Finally, $|xz| = \prod_{b_i \leqslant a_i} z_i$, and since $|z_i| = p_i^{b_i}$ it follows that $xz \in C$. Note that since we know a superset of the primes p_i, there is no need to factorise $|C|$ and $|c|$. □

Note that the result provides an algorithm for obtaining the patch element z_I. In this case, the cyclic groups under consideration all arise from some $\mathbb{F}_q^\times$, so we can factorise $q - 1$ once as preprocessing to obtain the list of possible primes.

The element z_I is unique up to multiplication by an element of $I \cap \langle \lambda_I \rangle$. It remains to find the other patch element $z \in \langle \lambda \rangle$. This is done differently depending on how λ_I arose from λ.

7.2.1. *Determinant case.* In this case, $\lambda_I = \lambda^d$, so λ^i is a d-th root of z_I^{-1}. However, not all roots suffice. Clearly, all dth roots of z_I are $\lambda^{-i}\sigma$ where $\sigma^d = 1$. Now $g/h\lambda^{-i}\sigma = k^{-1}\sigma$, so we must choose a root of z_I such that $\sigma \in \langle \lambda_K \rangle$.

We have no knowledge of λ^i or σ, only their product, which is the root of z_I. From Table 7.1 we see that $|\lambda_K| = \gcd(d, |\lambda|)$. Also, $|\lambda^i \sigma| \geqslant |\sigma| / \gcd(|\sigma|, |\lambda|)$ so if $|\lambda^i \sigma| \mid |\lambda|$, then $|\sigma| \mid |\lambda|$. But $|\sigma| \mid d$, so this implies that $|\sigma| \mid |\lambda_K|$, thus $\sigma \in \langle \lambda_K \rangle$.

The crucial condition on the dth root z of z_I that we choose is therefore that $z \in \langle \lambda \rangle$.

7.2.2. *Form action case.* In this case it depends on the form type. In the non-unitary case, $\lambda_I = \lambda^2$, so we choose z to be a square root of z_I. The argument why this works is the same as in the determinant case.

The unitary case is more difficult. Recall that the *norm* of $\mathbb{F}_q$ over $\mathbb{F}_{\sqrt{q}}$ is a homomorphism $\mathrm{N} : \mathbb{F}_q^\times \to \mathbb{F}_{\sqrt{q}}^\times$ defined by $\omega_q \mapsto \omega_q^{\sqrt{q}+1}$.

In this case $\lambda_I = \mathrm{N}(\lambda)$, so we want to choose $z \in \mathbb{F}_{\sqrt{q}}^\times$ such that $\mathrm{N}(z) = z_I$. This is a norm equation, which we can solve [MRD, Proposition 2.2] and obtain a solution z_0. Again, the crucial condition is $z \in \langle \lambda \rangle$, but we may have $z_0 \notin \langle \lambda \rangle$. Each solution to the norm equation can be written in the form $z_0 z_1$ where $\mathrm{N}(z_1) = 1$. Clearly,

$|z_0 z_1| \geqslant |z_0| / \gcd(|z_0|, |z_1|)$, and since $\mathrm{N}(z_1) = 1$ it follows that $|z_1| \mid (\sqrt{q}+1)$. Hence $|z_0 z_1| \geqslant |z_0| / \gcd(|z_0|, \sqrt{q}+1)$.

There exists at least one solution $z \in \langle \lambda \rangle$ to the norm equation, so the above implies that $|z_0| / \gcd(|z_0|, \sqrt{q}+1)$ divides $|\lambda|$. Therefore $|z_0| \mid \mathrm{lcm}(|\lambda|, \sqrt{q}+1)$.

Thus, $z_0 \in \left\langle \lambda, \omega_q^{\sqrt{q}-1} \right\rangle$, since $\left| \left\langle \omega_q^{\sqrt{q}-1} \right\rangle \right| = \sqrt{q}+1$, and we want to find $z_1 \in \left\langle \omega_q^{\sqrt{q}-1} \right\rangle$ such that $z_0 z_1 \in \langle \lambda \rangle$. This is precisely the same situation we had above, when we were finding z_I. Therefore we can use the patching method of Proposition 7.1 to obtain z_1.

7.2.3. *Spinor norm case.* In this case, $\lambda_I \in \{0, 1\}$. Accordingly, $z = \lambda^{z_I}$.

7.2.4. *Smaller field case.* In this case there is no need for a second patch element, so $z = 1$.

8. Reduction database for permutation groups

Our goal here is to provide reductions from general permutation groups to small base groups, which can be treated as leaves using BSGS methods. We do not quite do this, since we are missing the "jellyfish" algorithm.

8.1. **Intransitivity.** If $G \leqslant \mathrm{Sym}(n)$ is intransitive, then $\mathcal{X} = \{1, \ldots, n\}$ is split up into at least two orbits under the action of G. If $\mathcal{O} \subset \mathcal{X}$ is such an orbit, there is an induced action of G on $\mathcal{O}$. It is straightforward to construct $\phi : G \to \mathrm{Sym}(\mathcal{O})$. We have no special method of obtaining the kernel.

8.2. **Imprimitivity.** Similarly as with matrix groups, a transitive $G \leqslant \mathrm{Sym}(n)$ is imprimitive if it permutes the blocks in a non-trivial partition of $\mathcal{X} = \{1, \ldots, n\}$. Here Magma provides all machinery for setting up the reduction homomorphism. We have no special method of obtaining the kernel.

8.3. **Alternating groups natural representation.** Another large base case are the alternating group in natural representation. We use the existing algorithm in Magma to see if $G = \mathrm{Alt}(n)$, and if so it is marked as a leaf, and handled using our special algorithm.

9. Leaf algorithm database

When we have a leaf, we first look at the type of the group. If it is not a matrix or a permutation group, we handle it accordingly. Otherwise, the group has either been named during the C9 reduction, or it is an unknown group.

If it has been named, we consider the list of possible leaf algorithms for the family, and try each one in turn until one succeeds.

The library of algorithms for handling the various types of leaves is in a sense the most critical component of the package and the one where most improvements are needed.

9.1. **Cyclic groups.** The cyclic leaves have been discussed above. In Magma they arise as `GrpAb`, a finitely presented group, and are stored as such. It would have been slightly easier if they could be converted into PC-groups, but this is not possible if the order is larger than 2^{30}.

The gold copy of a cyclic group is the group itself. Magma always introduces a minimal generating set for a `GrpAb`, in this case a single generator, and we take the nice generator to be this generator.

A presentation is straightforward to obtain: a single relation is required. If G is the group with scalar flag λ, we can easily form $H = G \langle \lambda \rangle / \langle \lambda \rangle$ since the groups are finitely presented. The presentation is then $\langle x | x^{|H|} \rangle$.

The composition factors are also straightforward to obtain using existing MAGMA functionality.

9.2. **Elementary abelian groups.** These arise from the unipotent groups, and we store them in MAGMA as `GrpPC`, a finitely presented group where the presentation is a power-commutator (PC) presentation. For `GrpPC` one can define homomorphisms slightly more freely. This is useful here, since MAGMA introduces a new generating set, which we take as nice generators, and we need to express these as words in our input generators. The gold copy is the group itself, on the nice generators.

A presentation for a `GrpPC` is easy to obtain, since the group is already stored as a presentation. Since we have chosen the nice generators to be the MAGMA generators, the stored presentation is the one we need.

The composition factors are straightforward to obtain using existing MAGMA functionality.

9.3. **Classical groups natural representation.** These leaves are quasi-simple matrix groups, the families $\mathrm{SL}(d,q)$, $\mathrm{Sp}(d,q)$, $\mathrm{SU}(d,q)$ and $\Omega^\epsilon(d,q)$. They form the leaves in the C8 Aschbacher class. If $G = \langle X\rangle \leqslant \mathrm{SL}(d,q)$ is a leaf group of this type, we use the algorithms in [LGO07], with implementations by Eamonn O'Brien, to find standard generators Y for G. This also provides a change of basis matrix $c \in \mathrm{GL}(d,q)$ such that $\langle Y\rangle^c = G$. We define the gold copy of G to be $\langle Y\rangle$, so that c also gives us isomorphisms to and from the gold copy. The nice generators are Y^c.

The algorithms in [Cos08], with implementation by Elliot Costi, provides constructive membership testing in the nice generators.

We have presentations for the classical groups on the standard generators, hence on our nice generators.

The composition factors consist of the simple group and the factors corresponding to $\mathrm{Z}(G)$. In this case we do not write down the simple group explicitly, since it is a permutation group of large degree. The generators of the simple group factor is instead just the nice generators, so the `SLP`s are immediate. Since G is a classical group in natural representation, we can write down an explicit scalar matrix that generates $\mathrm{Z}(G)$. Then we can obtain this as an `SLP` in the standard generators using the rewriting algorithm. The centre can easily be converted to a PC-group, we can take the quotient with the scalar flag, and then use existing MAGMA functionality to obtain the composition factors.

Let ω be a primitive element of $\mathbb{F}_q$. Then $\mathrm{Z}(G) = \left\langle \omega^{(q-1)/n} \right\rangle$ where

$$n = \begin{cases} \gcd(d, q-1) & G = \mathrm{SL}(d,q) \\ \gcd(2, q-1) & G = \mathrm{Sp}(d,q) \\ 3 & G = \mathrm{SU}(3,4) \\ \gcd(d, \sqrt{q}+1) & G = \mathrm{SU}(d,q), (d,q) \neq (3,4) \\ 1 & G = \Omega^\epsilon(d,q), 2 \mid q \\ 1 & G = \Omega^0(d,q) \\ 2 & G = \Omega^+(d,q), 2 \mid (d(q-1)/4) \\ 1 & G = \Omega^+(d,q), 2 \nmid (d(q-1)/4) \\ 2 & G = \Omega^-(d,q), 2 \nmid (d(q-1)/4) \\ 1 & G = \Omega^-(d,q), 2 \mid (d(q-1)/4) \end{cases}$$

9.4. **Quasi-simple groups, C9.** The C9 leaves are the most difficult ones, and those where we have fewest special leaf algorithms. We do have access to a number of algorithms, but not all work in the context we need.

In the C9 leaf case we have $G \leqslant \mathrm{GL}(d,q)$, with scalar flag λ, and $G \cong Z.S$ where S is a finite simple group and Z is cyclic. Since G is absolutely irreducible, $Z \leqslant \mathrm{Z}(\mathrm{GL}(d,q))$ and hence consists of scalar matrices. Since we have applied the determinant map, $G \leqslant \mathrm{SL}(d,q)$, so $|Z| \mid \gcd(d, q-1)$. This is the only thing we can say about Z in the general case. The extension $Z.S$ may split, it can even be a direct product, so Z is not guaranteed to lie in the Schur multiplier of S. Hence G is not necessarily perfect.

We now want the leaf algorithms to handle such a G. Hence a leaf algorithm should provide maps $\alpha : G\langle\lambda\rangle \to H$, where $H \cong S$ or $H \cong Z_S.S$ is the gold copy. Here Z_S are the scalars in the Schur multiplier of S, and we need to allow this situation, to correctly handle the case when $S \cong \mathrm{PSL}(d,q)$. Then we want $H = \mathrm{SL}(d,q)$ because we want to work with matrix groups.

The leaf algorithm should also provide $\beta : H \to G$, which should be the inverse modulo scalars to α. In other words, $\beta(\alpha(g)) \in Z\langle\lambda\rangle$ for every $g \in G\langle\lambda\rangle$.

We need a notion of standard generators for G. This is given implicitly by the images under β of the generators of H. Then the leaf algorithm should provide a rewriting facility that expresses any $g \in G\langle\lambda\rangle$ as an `SLP` in the standard generators, which evaluates to $h \in G\langle\lambda\rangle$ with the property that $g/h \in Z\langle\lambda\rangle$.

Our nice generators for G are the standard generators together with a generator for Z. We can find the centre of G as described in Section 6. This allows us to obtain a rewriting algorithm for G which is precise. We use the above leaf algorithm rewriting, so we obtain an element $x \in Z\langle\lambda\rangle$. If $Z = \langle c\rangle$ then $|x| \mid \mathrm{lcm}(|c|, |\lambda|)$. Observe that $y = x^{|x|/\gcd(|c|,|x|)} \in Z$, so we can express y as an `SLP` in c. This is an instance of the discrete logarithm problem.

Finally, we want the leaf algorithm to provide a presentation of S on the standard generators. This allows us to obtain a presentation for G. The presentation $\langle x_1, \ldots, x_n \mid r_1, \ldots, r_m\rangle$ for S is a presentation for G/Z. We want to obtain a presentation of G on $\{x_1, \ldots, x_n, c\}$. Each r_i evalutes to $c_i \in Z$, and we can express each c_i as an `SLP` w_i in c, as above. The relators for G are then $\{r_i/w_i : 1 \leqslant i \leqslant m\} \cup \{[x_i, c] : 1 \leqslant i \leqslant n\} \cup \{c^{|c|/\gcd(|c|,|\lambda|)}\}$.

To obtain composition factors in the C9 case is easy, since we can find the centre. We then proceed as in the C8 case.

9.4.1. *Alternating groups.* In this case we use [BP00] with implementation by Derek Holt, extended by Eamonn O'Brien. This works exactly as we require.

9.4.2. *Sporadic groups.* For the sporadic groups we have to proceed in a slightly different way. For each sporadic group we designate a small-dimensional permutation or matrix representation as the gold copy S. Then we use the black box algorithms from [ABL$^+$] to obtain standard generators in G and in S. These algorithms work projectively and hence can be applied to G without knowledge of Z.

To obtain a rewriting algorithm, we either use [HLO$^+$08] or BSGS methods. This also allows us to define the maps α and β: express an element as an `SLP` in the standard generators in one group and evalute the `SLP` on the standard generators in the other group.

For most sporadic groups we also have presentations on the standard generators, but in some cases we must use brute force methods.

9.4.3. *Classical groups non-natural representation.* This is the most difficult class of leaves. In the case where $S \cong \mathrm{PSL}(2,q)$ we use [CLGO06], with implementation by Eamonn O'Brien, which works exactly as we require.

In the case where $S \cong \mathrm{PSL}(3,q)$, and G is a representation in defining characteristic, we use [LMO07], with implementation by Henrik Bäärnhielm and Eamonn O'Brien, to obtain the map β. This algorithm cannot handle the case when Z is

not contained in the Schur multiplier of G. We work around this by computing the derived group $D \cong S$ of $G\langle\lambda\rangle$ and executing the algorithm on D. The map β allows us to obtain standard generators for G, and we define the nice generators as above. From [LMO07] we obtain rewriting ability in D, and all the standard generators lie in D, so can obtain them as SLPs in the input generators of D and hence in the input generators of G. For rewriting in $G\langle\lambda\rangle$ we use [Cos08], with implementation by Elliot Costi, which works as we require. This also provides the map α, in the same way as with the sporadic groups.

When $S \cong \mathrm{PSL}(d,q)$, $d > 3$ and G is a representation in defining characteristic, we use either [MOÁS08], with implementation by Eamonn O'Brien or [KS01] with implementation by Peter Brooksbank, Bruce Cox and Derek Holt, to obtain β. Similarly as above we first have to compute the derived group D of $G\langle\lambda\rangle$. In this case we have no rewriting ability in D. However, we obtain $\alpha_0 : D \to \mathrm{SL}(d,q)$, so we can obtain the images of our input generators in the natural representation. Then we can use [LGO07] to obtain the standard generators as SLPs in the input generators. For rewriting in $G\langle\lambda\rangle$ we can again use [Cos08], which also provides α.

When G is not in defining characteristic, or if G is a permutation group, then we use [KS01]. We can obtain the nice generators as SLPs in the input generators as above. To obtain rewriting ability in $G\langle\lambda\rangle$, we use BSGS methods or [HLO$^+$08]. Since the map to $\mathrm{SL}(d,q)$ is from D, not $G\langle\lambda\rangle$, we cannot use it to map elements to the natural representation and use our natural representation rewriting ability.

When $S \cong \mathrm{PSp}(d,q)$ and q is odd, we use [Bro03, Bro08] with implementation by Peter Brooksbank and Derek Holt. This works in the same way as the black box $\mathrm{PSL}(d,q)$ case.

In all these cases we have presentations on the standard generators.

9.4.4. *Exceptional groups.* The algorithms of [Bää06, Bää07, Bää08], with implementation by Henrik Bäärnhielm, handle the cases of Suzuki and Ree groups in most ways we require. The representations of the Suzuki groups in defining characteristic (characteristic 2) have dimension 4^n for some $n \geqslant 1$, so $|Z| \mid \gcd(4^n, 2^k - 1)$ and hence $Z = 1$. In black box context the Suzuki algorithms cannot handle the case when $Z \neq 1$.

The algorithms for the Big Ree groups only handle the natural representation, dimension 26, with defining field of size $q = 2^{2m+1}$ for some $m > 0$.

The algorithms for the small Ree groups only handle representations in defining characteristic (characteristic 3). Such representations have dimension $7^n 3^{3k}$ where $n \geqslant 0$, $k \geqslant 0$ and $n + k > 0$.

It follows from the following result that $Z = 1$ in the Ree group cases.

Lemma 9.1. (1) *For every $m > 0$, $13 \nmid (2^{2m+1} - 1)$.*
(2) *For every $k > 0$, $7 \nmid (3^k - 1)$.*

Proof. □

In the Suzuki case we have a presentation on the standard generators, but we have no presentations for the Ree groups, so in these cases we are forced to use existing Magma machinery.

9.5. **Unknown groups.** Because we do not have special algorithms for all classes of finite simple groups, it is inevitable that we need a backup strategy for groups we cannot handle. It is also possible that some Aschbacher algorithms fail to deduce that a give group lies in its category, which can result in that it falls into the C9 case. But then the naming algorithms may fail, since they assume that the group is almost simple modulo scalars. Hence we end up with a group which really is unknown.

In the first of these cases we either use BSGS techniques or [HLO$^+$08] together with recursive applications of composition tree. In the second case we always use BSGS methods.

More specifically, if $G = \langle X \rangle$ is the input group with scalar flag λ, and we have successfully named G, then we use [HLO$^+$08] if the field or degree is larger than some specified bound. In this case the gold copy is G and X are the nice generators. For membership testing we apply [HLO$^+$08] to $G\langle\lambda\rangle$, and membership testing in the centralisers are done using recursive applications of composition tree. Note that we then obtain `SLP`s in $X \cup \{\lambda\}$, so we have to rewrite this to an `SLP` on X (we set $\lambda = 1$). Presentations and composition factors are found using existing Magma machinery, which may be very expensive.

If we cannot name G, or the the field and degree are small, we compute a BSGS of $G\langle\lambda\rangle$ using the random Schreier-Sims algorithm and possibly verify it using the Todd-Coxeter-Schreier-Sims algorithm. Then Magma provides constructive membership testing facilities in $G\langle\lambda\rangle$. As above, we need to rewrite the `SLP`s to obtain them on X, without λ. The gold copy is G, and the nice generators are the strong generators. We can find a presentation on the strong generators, which is easier than in general, but can still be expensive. Composition factors are found using existing Magma machinery, which can be very expensive.

References

[ABL$^+$] R. Abbot, J. Bray, S. Linton, S. Nickerson, S. Norton, R. Parker, S. Rogers, I. Suleiman, J. Tripp, P. Walsh, and R. Wilson, *Atlas of Finite Group Representations*, `http://brauer.maths.qmul.ac.uk/Atlas/`.

[Asc84] M. Aschbacher, *On the maximal subgroups of the finite classical groups*, Invent. Math. **76** (1984), no. 3, 469–514. MR MR746539 (86a:20054)

[Bää06] Henrik Bäärnhielm, *Recognising the Suzuki groups in their natural representations*, J. Algebra **300** (2006), no. 1, 171–198. MR MR2228642

[Bää07] ———, *Algorithmic problems in twisted groups of Lie type*, Ph.D. thesis, Queen Mary, University of London, 2007.

[Bää08] ———, *Recognising the Ree groups in their natural representations*, submitted, 2008.

[BCLGO06] John Bray, M. D. E. Conder, C. R. Leedham-Green, and E. A. O'Brien, *Short presentations for alternating and symmetric groups*, submitted, 2006.

[BKPS02] László Babai, William M. Kantor, Péter P. Pálfy, and Ákos Seress, *Black-box recognition of finite simple groups of Lie type by statistics of element orders*, J. Group Theory **5** (2002), no. 4, 383–401. MR MR1931364 (2003i:20022)

[BNS06] Peter Brooksbank, Alice C. Niemeyer, and Ákos Seress, *A reduction algorithm for matrix groups with an extraspecial normal subgroup*, Finite geometries, groups, and computation, Walter de Gruyter GmbH & Co. KG, Berlin, 2006, pp. 1–16. MR MR2257997 (2007k:20111)

[BP00] Sergey Bratus and Igor Pak, *Fast constructive recognition of a black box group isomorphic to S_n or A_n using Goldbach's conjecture*, J. Symbolic Comput. **29** (2000), no. 1, 33–57. MR MR1743388 (2001c:11137)

[Bra07] John N. Bray, *Presentations of the Suzuki groups*, preprint, 2007.

[Bro03] Peter A. Brooksbank, *Constructive recognition of classical groups in their natural representation*, J. Symbolic Comput. **35** (2003), no. 2, 195–239. MR MR1958954 (2004c:20082)

[Bro08] ———, *Fast constructive recognition of black box symplectic groups*, J. Algebra **320** (2008), no. 2, 885–909. MR MR2422320

[BS01] László Babai and Aner Shalev, *Recognizing simplicity of black-box groups and the frequency of p-singular elements in affine groups*, Groups and computation, III (Columbus, OH, 1999), Ohio State Univ. Math. Res. Inst. Publ., vol. 8, de Gruyter, Berlin, 2001, pp. 39–62. MR MR1829470 (2002f:20021)

[CF93] Gene Cooperman and Larry Finkelstein, *Combinatorial tools for computational group theory*, Groups and computation (New Brunswick, NJ, 1991), DIMACS Ser. Discrete Math. Theoret. Comput. Sci., vol. 11, Amer. Math. Soc., Providence, RI, 1993, pp. 53–86. MR MR1235795 (94h:20004)

[CLGM+95] Frank Celler, Charles R. Leedham-Green, Scott H. Murray, Alice C. Niemeyer, and E. A. O'Brien, *Generating random elements of a finite group*, Comm. Algebra **23** (1995), no. 13, 4931–4948. MR MR1356111 (96h:20115)

[CLGO06] M. D. E. Conder, C. R. Leedham-Green, and E. A. O'Brien, *Constructive recognition of* PSL$(2, q)$, Trans. Amer. Math. Soc. **358** (2006), no. 3, 1203–1221 (electronic). MR MR2187651 (2006j:20017)

[Cos08] Elliot Costi, *Constructive membership testing in classical groups*, Ph.D. thesis, Queen Mary, University of London, 2008.

[GKKL08a] R. M. Guralnick, W. M. Kantor, M. Kassabov, and A. Lubotzky, *Presentations of finite simple groups: A computational approach*, submitted, 2008.

[GKKL08b] R. M. Guralnick, W. M. Kantor, M. Kassabov, and A. Lubotzky, *Presentations of finite simple groups: a quantitative approach*, J. Amer. Math. Soc. **21** (2008), no. 3, 711–774. MR MR2393425

[GLGO06] S. P. Glasby, C. R. Leedham-Green, and E. A. O'Brien, *Writing projective representations over subfields*, J. Algebra **295** (2006), no. 1, 51–61. MR MR2188850 (2006h:20002)

[HEO05] Derek F. Holt, Bettina Eick, and Eamonn A. O'Brien, *Handbook of computational group theory*, Discrete Mathematics and its Applications (Boca Raton), Chapman & Hall/CRC, Boca Raton, FL, 2005. MR MR2129747 (2006f:20001)

[HLGOR96] Derek F. Holt, C. R. Leedham-Green, E. A. O'Brien, and Sarah Rees, *Computing matrix group decompositions with respect to a normal subgroup*, J. Algebra **184** (1996), no. 3, 818–838. MR MR1407872 (97m:20021)

[HLO+08] P. E. Holmes, S.A. Linton, E. A. O'Brien, A. J. E. Ryba, and R. A. Wilson, *Constructive membership in black-box groups*, J. Group Theory **11** (2008), no. 6, 747–763.

[HR94] Derek F. Holt and Sarah Rees, *Testing modules for irreducibility*, J. Austral. Math. Soc. Ser. A **57** (1994), no. 1, 1–16. MR MR1279282 (95e:20023)

[HS08] Derek F. Holt and Mark J. Stather, *Computing a chief series and the soluble radical of a matrix group over a finite field*, LMS J. Comput. Math. **11** (2008), 223–251. MR MR2429998

[IL00] Gábor Ivanyos and Klaus Lux, *Treating the exceptional cases of the MeatAxe*, Experiment. Math. **9** (2000), no. 3, 373–381. MR MR1795309 (2001j:16067)

[KS01] William M. Kantor and Ákos Seress, *Black box classical groups*, Mem. Amer. Math. Soc. **149** (2001), no. 708, viii+168. MR MR1804385 (2001m:68066)

[LG01] Charles R. Leedham-Green, *The computational matrix group project*, Groups and computation, III (Columbus, OH, 1999), Ohio State Univ. Math. Res. Inst. Publ., vol. 8, de Gruyter, Berlin, 2001, pp. 229–247. MR MR1829483 (2002d:20084)

[LGM02] C. R. Leedham-Green and Scott H. Murray, *Variants of product replacement*, Computational and statistical group theory (Las Vegas, NV/Hoboken, NJ, 2001), Contemp. Math., vol. 298, Amer. Math. Soc., Providence, RI, 2002, pp. 97–104. MR MR1929718 (2003h:20003)

[LGO97] C. R. Leedham-Green and E. A. O'Brien, *Recognising tensor products of matrix groups*, Internat. J. Algebra Comput. **7** (1997), no. 5, 541–559. MR MR1470352 (98h:20018)

[LGO02] ———, *Recognising tensor-induced matrix groups*, J. Algebra **253** (2002), no. 1, 14–30. MR MR1925006 (2003g:20089)

[LGO07] ———, *Constructive recognition of classical groups in odd characteristic*, submitted, 2007.

[LMO07] F. Lübeck, K. Magaard, and E. A. O'Brien, *Constructive recognition of* $\mathrm{SL}_3(q)$, J. Algebra **316** (2007), no. 2, 619–633. MR MR2356848 (2008j:20039)

[LO07] Martin W. Liebeck and E. A. O'Brien, *Finding the characteristic of a group of Lie type*, J. Lond. Math. Soc. (2) **75** (2007), no. 3, 741–754. MR MR2352733 (2008i:20058)

[Lüb01] Frank Lübeck, *Small degree representations of finite Chevalley groups in defining characteristic*, LMS J. Comput. Math. **4** (2001), 135–169 (electronic). MR MR1901354 (2003e:20013)

[MOÁS08] Kay Magaard, E. A. O'Brien, and Ákos Seress, *Recognition of small dimensional representations of general linear groups*, J. Austral. Math. Soc. (2008).

[MRD] Scott Murray and Colva Roney-Dougal, *The spinor norm and homomorphism algorithms for classical groups*, preprint.

[Nie05] Alice C. Niemeyer, *Constructive recognition of normalizers of small extra-special matrix groups*, Internat. J. Algebra Comput. **15** (2005), no. 2, 367–394. MR MR2142090 (2006k:20100)

[NP97] Alice C. Niemeyer and Cheryl E. Praeger, *Implementing a recognition algorithm for classical groups*, Groups and computation, II (New Brunswick, NJ, 1995), DIMACS Ser. Discrete Math. Theoret. Comput. Sci., vol. 28, Amer. Math. Soc., Providence, RI, 1997, pp. 273–296. MR MR1444141 (98g:20002)

[NP98] ———, *A recognition algorithm for classical groups over finite fields*, Proc. London Math. Soc. (3) **77** (1998), no. 1, 117–169. MR MR1625479 (99k:20002)

[NP99] ———, *A recognition algorithm for non-generic classical groups over finite fields*, J. Austral. Math. Soc. Ser. A **67** (1999), no. 2, 223–253, Group theory. MR MR1717416 (2000i:20080)

[NS06] Max Neunhöffer and Ákos Seress, *A data structure for a uniform approach to computations with finite groups*, ISSAC 2006, ACM, New York, 2006, pp. 254–261. MR MR2289128

[O'B06] E. A. O'Brien, *Towards effective algorithms for linear groups*, Finite geometries, groups, and computation, Walter de Gruyter, Berlin, 2006, pp. 163–190. MR MR2258009 (2008c:20022)

[Pak00] Igor Pak, *The product replacement algorithm is polynomial*, FOCS '00: Proceedings of the 41st Annual Symposium on Foundations of Computer Science (Washington, DC, USA), IEEE Computer Society, 2000, pp. 476–485.

[Pak01] ———, *What do we know about the product replacement algorithm?*, Groups and computation, III (Columbus, OH, 1999), Ohio State Univ. Math. Res. Inst. Publ., vol. 8, de Gruyter, Berlin, 2001, pp. 301–347. MR MR1829489 (2002d:20107)

[Par84] R. A. Parker, *The computer calculation of modular characters (the meat-axe)*, Computational group theory (Durham, 1982), Academic Press, London, 1984, pp. 267–274. MR MR760660 (85k:20041)

[Sta06] Mark James Stather, *Algorithms for computing with finite matrix groups*, Ph.D. thesis, University of Warwick, August 2006.

Department of Mathematics, University of Auckland, Auckland, New Zealand
URL: `http://www.math.auckland.ac.nz/~henrik/`
E-mail address: `henrik@math.auckland.ac.nz`

School of Mathematical Sciences, Queen Mary, University of London, Mile End Road, London, United Kingdom
URL: `http://www.maths.qmul.ac.uk/~crlg/`
E-mail address: `c.r.leedham-green@qmul.ac.uk`

Department of Mathematics, University of Auckland, Auckland, New Zealand
URL: `http://www.math.auckland.ac.nz/~obrien/`
E-mail address: `obrien@math.auckland.ac.nz`