Journal of Symbolic Computation 68 (2015) 27–60

Contents lists available at ScienceDirect

Journal of Symbolic Computation

www.elsevier.com/locate/jsc

A practical model for computation with matrix groups [☆]

Henrik Bäärnhielm [a], Derek Holt [b], C.R. Leedham-Green [c], E.A. O'Brien [a]

[a] Department of Mathematics, University of Auckland, Auckland, New Zealand
[b] Mathematics Institute, University of Warwick, Coventry, CV4 7AL, United Kingdom
[c] School of Mathematical Sciences, Queen Mary University of London, Mile End Road, London, United Kingdom

A R T I C L E I N F O

Article history:
Received 8 February 2012
Accepted 30 April 2014
Available online 7 August 2014

Keywords:
Finite matrix groups
Structural computation
Composition tree

A B S T R A C T

We describe an algorithm to compute a composition tree for a matrix group defined over a finite field, and show how to use the associated structure to carry out computations with such groups; these include finding composition and chief series, the soluble radical, and Sylow subgroups.

© 2014 Elsevier Ltd. All rights reserved.

1. Introduction

In this article we describe in detail algorithms for computing with large matrix groups defined over finite fields, and their implementation in Magma (Bosma et al., 1997). The motivation comes from the *matrix group recognition project* (Leedham-Green, 2001): its aim is to understand the structure of $G = \langle Y \rangle \leqslant \mathrm{GL}(d, \mathbb{F}_q)$. Our algorithms construct a composition series for G, and facilitate membership testing and writing of elements of G as words in Y. Further functionality includes, for example, the computation of chief series, the centre, the soluble radical, and Sylow subgroups.

Comparable questions can already be answered readily for permutation groups of large degree. These algorithms rely on "base and strong generator" (BSGS) methods (Holt et al., 2005, Chapter 4).

☆ We thank Peter Brooksbank and Max Neunhöffer for helpful comments, and Colva Roney-Dougal for helpful discussions. Bäärnhielm was funded by a University of Auckland Postdoctoral Fellowship. All authors were partially supported by the Marsden Fund of New Zealand via grant UOA1015.

E-mail addresses: henrik@math.auckland.ac.nz (H. Bäärnhielm), D.F.Holt@warwick.ac.uk (D. Holt), c.r.leedham-green@qmul.ac.uk (C.R. Leedham-Green), e.obrien@auckland.ac.nz (E.A. O'Brien).

URLs: http://www.math.auckland.ac.nz/~henrik/ (H. Bäärnhielm), http://www.warwick.ac.uk/staff/D.F.Holt (D. Holt), http://www.maths.qmul.ac.uk/~crlg/ (C.R. Leedham-Green), http://www.math.auckland.ac.nz/~obrien/ (E.A. O'Brien).

http://dx.doi.org/10.1016/j.jsc.2014.08.006
0747-7171/© 2014 Elsevier Ltd. All rights reserved.

It is now possible to study the structure of permutation groups with moderately small bases having degrees up to about ten million. Variations of these techniques have only limited applicability to matrix groups.

A *composition tree* for a finite group is a data structure to store its composition factors; it is defined in Section 3. The basic strategy for computing a composition tree of a matrix group, based on a combination of a constructive version of Aschbacher's theorem (Aschbacher, 1984) and constructive recognition algorithms for finite simple groups, was proposed by Leedham-Green (2001), and a prototype was implemented in Magma by Leedham-Green and O'Brien in 2000. Various developments and refinements of the strategy were introduced in Neunhöffer and Seress (2006) and O'Brien, (2006, 2011).

Similar methods can also be applied to permutation groups, and are useful for the study of large base groups, where BSGS techniques are less effective. Neunhöffer and Seress (2006) present a uniform data structure for both matrix and permutation groups; an implementation is available as the GAP (GAP Group, 2014) `recog` packages (Neunhöffer and Seress, Neunhöffer et al.).

In summary, our objectives are four-fold.

(1) Construct a composition tree for a group G that is a subgroup of either $\mathrm{GL}(d, \mathbb{F}_q)$ or of $\mathrm{Sym}(n)$.
(2) Using the composition tree, construct a composition series for G.
(3) Adjust the generators of the subgroups in the composition series to obtain a new composition series that refines a particular characteristic series of G. We refer to this step as *rearranging* the composition series.
(4) Provide the infrastructure needed by the *Soluble Radical Model* of Holt et al. (2005, Chapter 10) to answer other structural questions about G.

Of these four tasks the first has required most effort. It may be divided into three subtasks: realising a constructive version of Aschbacher's theorem; constructive recognition for simple groups; and combining these to complete the task. We are now, after some 20 years of continuous development, at the point where, in practice, we can construct a composition tree for a matrix group defined in dimension up to 100 and over a large field in a reasonable length of time.

It is not our purpose here to describe the large body of work, by many authors, that enables us to make Aschbacher's theorem constructive, and to process simple groups. This work is described elsewhere; we give references. Our concern here is with the *strategic concepts* and *technical details* needed to combine these algorithms into a package that both constructs a composition tree, and, by carrying out our other objectives, can be used to analyse the structure of such groups. This superstructure has been developed by us over the past 15 years, and is described in sufficient detail to guide others who may wish to implement these ideas.

We barely touch on questions of complexity; the critical issues arise with the algorithms for Aschbacher's theorem and simple groups. Serious obstructions remain before we have a provably polynomial-time algorithm to compute a composition tree. We contrast this with the striking theoretical results obtained in the black-box context (Babai et al., 2009) and reported in Section 3.3.

Sections 5–9 describe the construction of the composition tree. Sections 10–11 describe the rearranging algorithm, which requires us to solve an identification problem for automorphisms of classical groups, which may be of independent interest. In Section 12 we explain how the resulting data structure is exploited by the Soluble Radical Model to carry out various structural computations in G such as finding its centre and its Sylow subgroups.

Our implementations of the resulting algorithms are available in Magma as the `CompositionTree` package, and form part of its standard machinery for computing with matrix groups. Section 13 discusses aspects of our implementations and their performance.

The algorithms of Holt and Stather (2008) and Stather (2006) construct directly a chief series of G that refines the characteristic series of G defined in Section 3.3. By contrast, we first construct a composition series. The algorithm to adjust the generators of the terms in the composition series is straightforward, and uses the composition tree as a "black box" with a well-defined interface, without relying on detailed knowledge of its construction. A disadvantage compared with Holt and Stather (2008) is that certain groups require many adjustments.

Algorithms for computing with linear groups defined over infinite fields often compute, as a preliminary step, an image over a finite field. Our algorithms, in particular those that produce a presentation for the image, are immediately applicable; see, for example, Detinko et al. (2011, 2013).

2. Background and notation

All groups in this paper are assumed to be finite. For background on aspects of the matrix group recognition project, we refer the reader to Leedham-Green (2001) and O'Brien (2006, 2011).

As outlined in Seress (2003, Chapter 6), an extensive library of highly efficient algorithms exists for small base permutation groups. Finite polycyclic groups are usually represented by PC-presentations; practical algorithms for their study appear in Holt et al. (2005, Chapter 8). High-quality implementations of both classes of algorithms are available in both GAP and MAGMA; we assume their availability, and refer the interested reader to these sources for details. We refer to these as *standard machinery*.

Many existing algorithms to compute structural information about $G \leqslant \mathrm{GL}(d, \mathbb{F}_q)$ – for example, its composition factors or conjugacy classes – rely on the availability of a BSGS for G; see Holt et al. (2005) for a discussion. We refer to these as *BSGS machinery*.

We assume that efficient algorithms are available to solve many basic problems for invertible matrices. These include finding orders (Celler and Leedham-Green, 1997; O'Brien, 2006); computing characteristic and minimal polynomials (Neunhöffer and Praeger, 2008; Holt et al., 2005, §7.3); and computing large powers (Holt et al., 2005, §3.1.1; Leedham-Green and O'Brien, 2009).

A *straight-line program* (SLP) is a compactly stored word in a set of group generators and their inverses; for a formal definition, see for example Holt et al. (2005, §3.1.3). While the length of a word in a given generating set constructed in n multiplications and inversions can increase exponentially with n, the length of the corresponding SLP is *linear* in n. Babai and Szemerédi (1984) prove that every element of a finite group G has an SLP of length $O(\log^2 |G|)$ in every generating set.

A *rewriting* algorithm for a finite group G solves the *constructive membership problem*: given $u \in U \geqslant G = \langle Y \rangle$, decide whether $u \in G$, and if so express u as an SLP in Y. Here U is the *universal group* containing G: namely, U is $\mathrm{GL}(d, \mathbb{F}_q)$ or $\mathrm{Sym}(n)$ when G is a matrix group or permutation group, respectively.

A *constructive recognition algorithm* for a (quasi)simple group G solves the following problem: construct an isomorphism φ from G to a *standard copy* of G (see Section 4.2) such that $\varphi(g)$ can be computed efficiently for every $g \in G$. Such an isomorphism is said to be *effective*. We also demand that φ^{-1} is effective. All constructive recognition algorithms employed here are Las Vegas.

The concept of a *black-box group* was introduced in Babai and Szemerédi (1984). In this model, group elements are represented by bit-strings of uniform length; the only group operations permissible are multiplication, inversion, and checking for equality with the identity element. Permutation groups and matrix groups defined over finite fields are covered by this model.

Seress (2003, p. 17) defines a *black-box algorithm* as one that does not use specific features of the group representation, nor particulars of how group operations are performed; it uses only the operations permissible in a black-box group. A common assumption is that *oracles* are available to perform certain tasks – usually problems not known to be solvable in polynomial time. An example is an oracle to determine the order of an element in a black-box group. Number-theoretic oracles include a *discrete log oracle*: for a given non-zero $\mu \in \mathbb{F}_q$ and a fixed primitive element ω of $\mathbb{F}_q$, it returns the unique integer k in the range $1 \leqslant k < q$ for which $\mu = \omega^k$. Another is *integer factorisation*, usually of numbers of the form $q^i - 1$. The most efficient algorithms for both number-theoretic oracles run in sub-exponential time (see Shparlinski, 1999, Chapter 4).

Babai (1991) presents a black-box Monte Carlo algorithm to construct nearly uniformly distributed random elements of a finite group G in polynomial time. An alternative is the *product replacement algorithm* (Celler et al., 1995; Pak, 2000). We assume that a random element and its SLP in the generators of G are simultaneously constructed. Both algorithms satisfy this assumption.

The *central order* of $g \in G$ is the smallest power of g that lies in the centre of G. The cyclic group of order e is denoted by C_e.

3. Outline of the algorithms

A composition tree of a group G is a data structure – namely, a full binary tree – to store either its composition factors, or factors of a subnormal series that can be readily refined to a composition series. We describe both the structure and its construction in detail in Section 5; here we outline the algorithms to achieve the objectives identified in the introduction.

Each node of the tree has an associated group, a subnormal section of G, which is the group of the root node. For the associated group H of each non-leaf node, there is an epimorphism $\theta : H \to H_1$ and a monomorphism $\theta_0 : H_0 \to H$ with $\operatorname{Im}\theta_0 = \operatorname{Ker}\theta$, where H_0 and H_1 are respectively the groups associated with the left child and the right child of the node.

The tree is constructed recursively. The data structure facilitates rewriting in G. In particular, the ability to perform rewriting in a section of G is available once a composition tree has been constructed for this section. This ability is essential to the recursion, and depends on the availability of rewriting algorithms for the finite simple groups.

3.1. *Composition tree construction*

Here is an outline of the algorithm to construct a composition tree of a group G with a given generating set Y.

(1) Either:
   (i) construct an effective epimorphism $\theta : G \to G_1$, for some group G_1; or
   (ii) deduce that G is cyclic, elementary abelian, or "close" to being non-abelian simple. Now G becomes a *leaf* in the tree.

   In Case (i), θ must be a *reduction*: namely, G_1 is "smaller" than G in some respect – for example, its degree or field of definition. Assume henceforth that Case (i) applies.
(2) Recursively construct a composition tree for $G_1 = \langle\theta(Y)\rangle$.
(3) Construct generators for $G_0 := \operatorname{Ker}\theta$.
(4) Recursively construct a composition tree for G_0.
(5) Combine the composition trees for G_1 and G_0 into a tree for G.

If $G \leqslant \mathrm{GL}(d, \mathbb{F}_q)$, then we exploit Aschbacher's theorem (Aschbacher, 1984) in Step (1). This requires algorithms to decide whether G lies in a certain Aschbacher class, and to construct the corresponding θ. Other homomorphisms, such as the determinant map, may also be used.

If $G \leqslant \mathrm{Sym}(n)$, then we could exploit the O'Nan-Scott theorem (Holt et al., 2005, Chapter 10) in Step (1). However, as outlined in Seress (2003, Chapter 6), an extensive library of highly efficient algorithms exists which only require a BSGS. For a small base permutation group one can easily construct a BSGS. Thus our goal is more limited: we apply certain reductions to large base groups to obtain small base images, and then use the existing algorithms to complete the investigation.

The group associated with a leaf need not be simple. It may be cyclic or elementary abelian, a simple or soluble primitive permutation group, or an absolutely irreducible matrix group that is simple modulo its centre. Our decisions on what groups may be treated as leaves are partly dictated by complexity considerations, and partly based on the quality of algorithms available to process a leaf. For example, we observe no practical advantage from refining a cyclic group to its composition factors.

3.2. *The composition series*

The composition tree suffices if we are interested only in rewriting and membership testing in G, but for many other structural calculations we require a composition series. To construct such a series, we need to find composition series of the groups associated with the leaves of the tree. This may require additional work, particularly when the group associated with a node is insoluble but not simple. To do this efficiently, we need constructive recognition algorithms for the non-abelian composition factors of G.

We use the tree and associated constructive recognition algorithms to construct the following.

- A composition series $\langle 1\rangle = G_0 < G_1 < \cdots < G_m = G$ and, for each k, a subset X_k of G_k that generates G_k modulo G_{k-1}.
- For each $k > 0$, an effective epimorphism $\tau_k : G_k \to S_k$, where S_k is the standard copy of G_k/G_{k-1}, and $\operatorname{Ker} \tau_k = G_{k-1}$.
- For each $k > 0$, a rewriting algorithm for S_k.
- For each $k > 0$, an effective map $\phi_k : S_k \to G_k$ with $\phi_k \circ \tau_k = \mathrm{Id}_{S_k}$.

If G is not a leaf, then it is straightforward to construct this data for G from the corresponding data for its image and kernel. Hence the task reduces to constructing such data for the leaves of the composition tree for G.

3.3. *Rearranging the composition series*

Recall, for example from Holt et al. (2005, §10.1), that a finite group G has a characteristic series of subgroups

$$1 \leqslant L \leqslant M \leqslant K \leqslant G$$

defined as follows.

The group L is the soluble radical (the largest normal soluble subgroup) of G. The group M is defined to be the complete inverse image in G of the socle $\mathrm{soc}(G/L)$ of G/L (the product of its minimal normal subgroups). Thus M/L is the direct product of a uniquely defined set Δ of non-abelian simple groups. This set is permuted by G, by conjugation, and K is the kernel of this action.

Observe that, since the centraliser of M/L in G/L is trivial, the factor K/M is isomorphic to a subgroup of the direct product of the outer automorphism groups of the simple factors of M/L, and is therefore soluble. The factor G/K can be regarded as a permutation group on the (generally small) set Δ.

Babai and Beals (1999) initiated the *black-box approach* to the study of matrix groups: focusing on the abstract structure of $G \leqslant \mathrm{GL}(d, q)$, it seeks to construct precisely this characteristic series for G. In 2009, as a culmination of 25 years of work, Babai et al. (2009) proved that, subject to the existence of a discrete log oracle and the ability to factorise integers of the form $q^i - 1$ for $1 \leqslant i \leqslant d$, there exist black-box Las Vegas polynomial-time algorithms to construct this series for a large class of matrix groups. The reliance on the discrete log oracle arises from the application of the algorithm of Conder et al. (2006) to recognise constructively central quotients of $\mathrm{SL}(2, q)$. Kantor and Kassabov (2014) and Borovik and Yalçınkaya (2013) propose new algorithms, for fields of even and certain odd characteristics respectively, which perform this task in polynomial time for bounded characteristic without use of such an oracle.

A chief series for G that refines the above characteristic series is particularly useful for carrying out structural computations in G, such as calculating its automorphism group, or its conjugacy classes of elements or subgroups. For details of such algorithms, see Holt et al. (2005, Chapter 10). While their implementations have used BSGS machinery to date, the algorithms are essentially black-box.

As a first step towards such a chief series, we construct a composition series that refines the characteristic series. We proceed up the composition series of G and, for each G_k, we do the following. If possible, we replace each $x_k \in X_k$ by $x_k g$ with $g \in G_{k-1}$, such that $x_k g \in L$, or (if this is not possible) $x_k g \in M$, or (if this is not possible) $x_k g \in K$. This does not change G_k, it only modifies X_k. After these changes, by taking the sets X_k in a different order – first those lying in L, second those lying in M, third those lying in K, and finally those not in K – we obtain a new composition series of G that passes through L, M, and K. We refer to this process as *rearranging* the series.

The algorithm to perform this task is described in Section 11.

4. Standard copies, generators, presentations, rewriting

We now discuss various technical issues that play a role in the algorithms.

4.1. A group pair

A quotient of a group by a cyclic central subgroup may occur as a node group in a composition tree. We define a *group pair* to be an ordered pair (G, N), where G and N are subgroups of some common overgroup U, and G centralises N. We call G and N respectively the *upper group* and the *null subgroup* of the pair, and the quotient group $GN/N \cong G/(G \cap N)$ is the *represented group* of the pair. For notational purposes, it is convenient to regard a group G as being a group pair with upper group G and trivial null subgroup.

We define a morphism $\theta : (G, N) \to (G_1, N_1)$ between group pairs to be a map $\theta : GN \to G_1N_1$ (note that we use θ for this map too) with $\theta(N) \leqslant N_1$, such that θ induces a group homomorphism $\phi : GN/N \to G_1N_1/N_1$. While $\theta : GN \to G_1N_1$ is *not* required to be a homomorphism, we nevertheless use $\operatorname{Ker}\theta$ to denote $\{g \in GN \mid \theta(g) = 1\}$. We call θ a monomorphism, epimorphism or isomorphism of group pairs if ϕ has this property (as a group homomorphism). Group pair morphisms $\theta_0 : (G_0, N_0) \to (G, N)$ and $\theta : (G, N) \to (G_1, N_1)$ form an *exact sequence* if $\operatorname{Im}\phi_0 = \operatorname{Ker}\phi$, which is true if and only if $\operatorname{Im}(\theta_0)N = \{g \in GN \mid \theta(g) \in N_1\}$.

We assign a group pair (G, N) to each node of a composition tree for G, and the group associated with the node is the represented group of the pair. If the node is not a leaf, then we define the associated exact sequences $\theta_0 : (G_0, N_0) \to (G, N)$ and $\theta : (G, N) \to (G_1, N_1)$, where (G_0, N_0) and (G_1, N_1) are the pairs assigned to the left- and right-hand child nodes. This induces the required exact sequence on the corresponding represented groups.

4.2. Standard copies

For each finite simple or quasisimple group G, we designate one *standard copy* S. To compute structural information in another copy H of G, we often first construct effective isomorphisms between H and the standard copy S, carry out the desired computations in S, and then use the isomorphisms to transfer the results back to H.

The standard copy S has a designated set of *standard generators*. One important component of a constructive recognition algorithm is to find generators X in an arbitrary representation H of G that correspond to the standard generators under an isomorphism, and to express the elements of X as SLPs in the input generators of H. A second is an algorithm to write elements of H as SLPs in X. These enable the construction of effective isomorphisms between H and S in both directions.

The standard copy of $\mathrm{Alt}(n)$ is on n points; its standard generators are $(1, 2, 3)$ and either of $(3, \dots, n)$ or $(1, 2)(3, \dots, n)$ according to the parity of n.

The standard copy of a quasisimple classical group G is the unique conjugate of its natural representation which preserves a specific form; if A is a central subgroup of G, then the standard copy of G/A is represented by the pair (G, A). Each classical group has at most eight standard generators (Leedham-Green and O'Brien, 2009; Dietrich et al., 2013).

For each simply connected finite exceptional group of Lie type, Howlett et al. (2001) provide defining matrices for a specific faithful irreducible representation of minimal dimension; we designate this representation as the standard copy. Each such group has a reduced *Curtis–Steinberg–Tits* presentation (Babai et al., 1997); those root elements which satisfy this presentation are the *standard generators* of the exceptional groups of Lie rank at least 2; other choices were made for the groups of rank 1 (Bäärnhielm, 2007, 2014; Bray and Bäärnhielm, 2009).

Wilson (1996) introduced standard generators for (coverings of) the sporadic groups: these are defined on the Atlas web site (Wilson et al.).

4.3. Verification: using presentations for simple groups

Our algorithm to construct a composition tree has a *verification* option, which guarantees the correctness of its output. Without this, there is a small probability (which, as we show in Proposition 5.1 below, can be made arbitrarily small by the user) that the group is larger than reported. This option is described in Section 5.3. It requires the computation of presentations of the node groups which, as we see in Section 4.5, reduces to finding presentations of the leaf groups.

For small and for soluble leaf groups, these can be computed using standard machinery. For other groups, the problem reduces to the construction of presentations for the finite simple groups on their standard generators. Hence a goal of both theoretical and practical interest is to obtain "short" presentations for the finite simple groups on *specific* generating sets.

Informally, the *length* of a presentation is the number of generators plus the sum of the lengths of the relators; see Guralnick et al. (2008) for one definition. Key to this work are short presentations for $\mathrm{Alt}(n)$ and $\mathrm{Sym}(n)$. Independently Bray et al. (2011) and Guralnick et al. (2008) proved that $\mathrm{Alt}(n)$ and $\mathrm{Sym}(n)$ have presentations with a bounded number of generators and relations, and length $\mathrm{O}(\log n)$.

In a major extension, Guralnick et al. (2008) proved that every non-abelian finite simple group of Lie rank n over $\mathbb{F}_q$, with the possible exception of ${}^2G_2(q)$, has a presentation with a bounded number of generators and relations and total length $\mathrm{O}(\log n + \log q)$. In all cases, the lengths are optimal, up to multiplication by a fixed constant. Unfortunately it does not seem possible to achieve this asymptotic bound with our standard generators, but we can, for example, find a presentation of length $\mathrm{O}(\log^2(n) + \log q)$ with $\mathrm{O}(\log n)$ relators on these generators, using Bray et al. (2011, Theorem 1.3(a)).

There remains the task of writing down explicit presentations, satisfying these limits, for the finite simple groups on their standard generators. Leedham-Green and O'Brien (2014) do this for the classical groups. Explicit presentations on reduced Steinberg generators for the exceptional groups of rank at least 2 appear in Liebeck and O'Brien (2014); see also Babai et al. (1997). Bray and Bäärnhielm (2009) provide a presentation for $\mathrm{Sz}(q)$ on its standard generators. Presentations on standard generators for sporadic groups are available at Wilson et al.

4.4. Generating sets

An upper group G stored in a node of a composition tree has an ordered list of *input generators* that generate G. For the root node, they are supplied by the user. If the node is a right child under a group pair morphism θ, then its input generators are the images of the input generators of G under θ.

The group G has a second list of *nice generators*. This idea was introduced by Neunhöffer and Seress (2006). In a non-leaf node with reduction epimorphism θ and kernel monomorphism θ_0, its nice generators are the images under θ_0 of the nice generators of the kernel node together with inverse images under θ of the nice generators of the image node. If G is a leaf and its non-abelian composition factor has standard copy S, then the nice generators of G are inverse images in G of the standard generators of S, together with a generator of $\mathrm{Z}(G)$.

We maintain the two lists for a number of reasons. A critical reason is that we can use the theoretical results described in Section 4.3 to write down a presentation for a leaf node on its nice generators; such is not feasible for an arbitrary generating set. This results in shorter presentations at all nodes than would be possible, in general, for an arbitrary generating set. Another is that we can more easily design rewriting algorithms that write an element of a leaf group in terms of its nice generators. We store SLPs that define the nice generators as words in the input generators, so words in the nice generators can be rewritten in terms of the input generators as required. As we show below, this facility allows us to rewrite a presentation on the nice generators to one on the input generators.

4.5. Presentations and rewriting for non-leaf nodes

Assume that presentations and rewriting algorithms are known for the child node groups G_0N_0/N_0 and G_1N_1/N_1 on their nice generators; we now describe how to provide them for GN/N. In the following descriptions, we ignore the null subgroups and work just with G_0, G_1 and G. The null subgroups introduce further technical complications; we defer discussing these until Section 8. To simplify notation, we also identify G_0 with its isomorphic image under θ_0.

4.5.1. Presentations

The algorithm to write down a presentation of G, using the fact that it is an extension of G_0 by G_1, is standard; see for example, Holt et al. (2005, Proposition 2.55) or Johnson (1990, §10.2).

Let $\{X_0 \mid R_0\}$ and $\{X_1 \mid R_1\}$ be presentations of G_0 and G_1. For each $x \in X_1$, choose $x' \in G$ with $\theta(x') = x$. Let $X_1' := \{x' \mid x \in X_1\}$ and $X := X_1' \cup X_0$. For each relator $r \in R_1$, we define a word r' in X_1' by replacing each generator x in r by the corresponding $x' \in X_1'$; so r' evaluates to an element of G_0. Let w_r be a word in X_0 that evaluates to the same element, and let $R_1' = \{r' w_r^{-1} \mid r \in R_1\}$. For each $x_0 \in X_0$ and $x \in X'$ observe that $x^{-1} x_0 x \in G_0$; so there is a word w_{xx_0} in X_0 that evaluates to this element. Let $S = \{x^{-1} x_0 x w_{xx_0}^{-1} \mid x \in X_1', x_0 \in X_0\}$. If $R := R_0 \cup R_1' \cup S$, then $\{X \mid R\}$ is a presentation of G.

We use this theory with X_0 and X_1 being the sets of nice generators of G_0 and G_1 respectively, and define $X := X_1' \cup X_0$ to be the set of nice generators of G. We use our rewriting algorithm in G_0 to calculate the words w_r and w_{xx_0}.

To rewrite this presentation on the input generators of G, we use the following result; see for example, Johnson (1990, §4.4, Remark 7). If $G = \langle X \rangle$, then two words in X are *G-equivalent* if they represent the same element of G.

Proposition 4.1. *Let $\{X \mid R\}$ be a presentation of a group G. Let Y be another generating set of G, and let Y_ρ and X_ρ be functions which respectively rewrite words in X to G-equivalent words in Y and words in Y to G-equivalent words in X. Let $S = \{Y_\rho(r) \mid r \in R\} \cup \{Y_\rho(X_\rho(y)) y^{-1} \mid y \in Y\}$. Then $\{Y \mid S\}$ is a presentation of G.*

4.5.2. Rewriting

The construction of the rewriting algorithm for G on its nice generators from those of G_0 and G_1 is straightforward. For $g \in G$, we rewrite $\theta(g) \in G_1$ as a word w in X_1 and define the corresponding word w' in X_1' as described above. We evaluate w' to obtain h in G, so $gh^{-1} \in G_0$, and write gh^{-1} as a word w_0 in X_0. Now g is the word $w_0 w$ in X.

We label as ρ the function that assigns the element gh^{-1} of G_0 to $g \in G$.

5. The composition tree algorithm

Recall that a composition tree for a group is stored as a full binary tree. Each node describes the represented group of a group pair (G, N), and has various data attached to it. Its right child, with associated group pair (G_1, N_1), represents the image of (G, N) under a group pair morphism θ, which arises from a reduction. Its left child, with associated group pair (G_0, N_0), represents the kernel of θ. More precisely, there is a group pair morphism θ_0 from the left child to (G, N) such that θ_0 and θ form an exact sequence. It is often – but not always – true that θ_0 is simply an inclusion map. This flexibility allows us to store the kernel in a different representation, which may be more efficient. For instance, if θ arises from a tensor decomposition of a matrix group, then we store both the image and the kernel as matrix groups of smaller dimension. We allow the represented group of the kernel or image to be trivial, provided that θ represents some kind of simplification, such as in the degree of the group or in the field size.

For the group pair (G, N) associated with the node, the null subgroup N is always cyclic and centralised by G. If G is a permutation group or elementary abelian, then N is trivial. If G is a matrix group, then N is either trivial, or consists of scalar matrices. It may also be non-trivial when G is a cyclic group. More details about the null subgroups, and their definitions for the various reductions, are given in Section 8. The null subgroup of the root node is trivial by default, but it may be assigned by the user subject to these constraints.

5.1. Data structures for a node

We list the important data components stored in a node with associated group pair (G, N). Some of these components are available only when the construction of the composition tree rooted at that node is complete.

(1) The input generators of G.

(2) A generator for the null subgroup N.
(3) The data structure required to execute the product replacement algorithm for G on its input generators.
(4) A list of algorithms that can be used to construct a reduction homomorphism for G, in order of priority. This priority may vary according to the reduction used to construct G.
(5) The nice generators of G.
(6) A list of SLPs of the nice generators in the input generators.
(7) A rewriting algorithm that expresses $g \in GN$ as an SLP in its nice generators that evaluates to an element in gN.
(8) A list of *mandarins*. These are random elements used to test the correctness of the subtree rooted at the node – see Section 5.2.
(9) A list of SLPs of relators for a presentation of G on its nice generators.
(10) A list of lists of SLPs of elements in the nice generators of G whose images (in the appropriate sections) generate each composition factor.
(11) The data associated with the composition series described in Section 3.2.

If the node is not a leaf, then there is a reduction epimorphism θ to its right child and a monomorphism θ_0 from its left child. The following additional data is stored.

(1) The reduction epimorphism θ.
(2) A list of SLPs of the images under θ_0 of the input generators of the left child in the input generators of the node.

If the node is a leaf, then the following additional data is stored.

(1) If G is abelian, then its isomorphism type; otherwise the name of its non-abelian chief factor.
(2) If G is quasisimple, then an effective epimorphism $\alpha : G \to S$ to the standard copy of the non-abelian composition factor of G, and effective inverse β.

5.2. *The main algorithm*

We now describe in more detail the composition tree algorithm. We apply this initially to the input group $G \leqslant \mathrm{GL}(d, \mathbb{F}_q)$ or $G \leqslant \mathrm{Sym}(n)$, but during the application of the algorithm it is applied recursively to the other node groups. We discuss the reductions as they apply to G, and defer until Section 8 the details of how we process the null subgroup and define the null subgroups of the node's children.

Before describing the algorithm, we discuss the important role played by the mandarins in establishing the correctness of the construction. When we start to process a node, the associated group has a generating set and a set of mandarins. In the root node, the mandarins are random elements of the input group. For every node, the expectation is that all of the mandarins lie in the associated group. For leaf nodes we test this immediately after processing the node. For non-leaf nodes, the successful construction of the subtree rooted at this node enables us to write the mandarins as SLPs in the nice generating set, which implies their containment in the group. A failure of a mandarin to lie in the group implies that something has gone wrong with our construction of the tree, and this provokes a *crisis*. Our method of correcting the mistake is described in Section 5.4.

We explain now how we construct the mandarins of the child nodes of a non-leaf node. The mandarins of the group G_1 of the right child are defined to be the images of those of G under the reduction epimorphism $\theta : G \to G_1$. If these images cannot be computed, then a crisis is provoked. If no problem is detected during the construction of the subtree rooted at G_1, then the mandarins of G_1 lie in G_1. The mandarins of the group G_0 of the left child are then obtained by applying the function ρ, defined in Section 4.5.2, to the mandarins of G.

In summary, the algorithm to construct the composition tree for the input group G is the following.

(1) Attempt to construct a reduction morphism $\theta : G \to G_1$. See Sections 6 and 7 for details.

(2) If no reduction is found, then the node is marked as a leaf.
   (a) Process the leaf as described in Section 9. This includes identifying the unique non-abelian composition factor of a non-abelian leaf group, and applying an appropriate constructive recognition algorithm to it.
   (b) If verification is required, then construct a presentation of the leaf group on its nice generators.
   (c) Test the mandarins of the node for membership in the group. If this fails, then a crisis is provoked. Otherwise, halt.

(3) Set up the data structure for the right child of the node, and try to define the mandarins for this child. If this fails, then a crisis is provoked. Otherwise recursively construct a composition tree for the right child.

(4) Construct a putative generating set for $\mathrm{Ker}\,\theta$, define the group G_0 of the left node and the isomorphism $\theta_0 : G_0 \to \mathrm{Ker}\,\theta$. See Section 5.3 for details. Construct the mandarins for this child.

(5) Recursively construct a composition tree for G_0. If we construct this subtree without provoking a crisis, then we have established that the mandarins associated with G_0 lie in G_0. It then follows from the definition of the mandarins of the child nodes that the mandarins of the node lie in G.

(6) Set up the nice generators and the rewriting algorithm for G as described in Section 4.4. Apply the rewriting algorithm to the input generators of G to check that they lie in the group generated by the nice generators. If not, then a crisis is provoked.

(7) If verification is required, then attempt to construct a presentation of G on its nice generators from presentations of its children. If we construct the presentation, then we have verified that the composition tree for G is correct. If the attempt to construct a presentation fails, then a crisis is provoked.

Proposition 5.1. *If the above algorithm completes successfully without using verification, and if the number of mandarins at each node is M, then the probability of an incorrect result being returned is at most $1-(1-2^{-M})^{\ell}$, where ℓ is the number of left child nodes in the resulting tree.*

Proof. The only source of possible error is when the left child of a node is created, and the generating set for this node is inadequate. If this is the case, then the deficiency will be detected unless the generating set generates a group that contains the mandarins. If the generating set generates a subgroup of index k, then the probability that all of the mandarins lie in this subgroup is $1/k^M$ since the construction of the generating set is independent of the mandarins. Since $k \geq 2$ the result follows. □

5.3. *Kernel generation*

A critical task is the construction of a generating set for $\mathrm{Ker}\,\theta$ in Step (4) of the main algorithm. The reductions described in Sections 7.4 and 7.6 provide these generating sets directly, but the others do not. We now describe two general methods to construct $\mathrm{Ker}\,\theta$: the *random element* method introduced in Leedham-Green (2001), and the *presentation* method introduced in Neunhöffer and Seress (2006).

If there are non-trivial null subgroups, then we must construct the kernel of $\phi : GN/N \to G_1N_1/N_1$ rather than of θ, but for the moment we ignore this problem and describe the computation of the kernel of $\theta : G \to G_1$. The presence of null subgroups introduces some technical complications that we discuss in Section 8. To simplify notation, we identify G_0 with $\mathrm{Ker}\,\theta$, whereas in reality $\mathrm{Ker}\,\theta$ is the isomorphic image of G_0 under θ_0. In all our reductions, images and inverse images under θ_0 are easily computed.

5.3.1. *The random element method*

This has two ingredients. The first is the construction of a set of random elements of G_0; this set, which may be augmented from time to time, will be the putative input generating set Y_0 of G_0. It can only be carried out when a composition tree for G_1 has been constructed. The second, *verification*, decides definitively whether Y_0 is adequate. It requires that we can construct a presentation for G_1 on its nice generators. In the absence of verification we have a Monte Carlo algorithm, with a bound to the probability of failure given by Proposition 5.1.

Constructing random elements of G_0 Given a rewriting algorithm for G_1, we define the function $\rho : G \to G_0$ of Section 4.5.2. Observe that $\rho(g) = gh^{-1}$, where h depends only on $\theta(g)$: namely, on the coset of $\operatorname{Ker}\theta$ that contains g, and not on the value of g within this coset. Thus if $U = \{g_1, \ldots, g_k\}$ is a set of (nearly uniformly distributed independent) random elements of G, then $\rho(U)$ and $\theta(U)$ are such sets for G_0 and G_1 respectively.

We use ρ to construct a putative generating set Y_0 of random elements of G_0. Various theoretical upper bounds to the size of Y_0 required to generate G_0 with high probability are known. Let $d(G)$ be the minimal size of a generating set of a finite group G. If $G \leqslant \mathrm{Sym}(n)$, then $d(G) \leqslant n/2$ for $n > 3$ by Cameron et al. (1989). If $G \leqslant \mathrm{GL}(d, \mathbb{F}_q)$, then $d(G) \leqslant 3d/2$ if G is completely reducible, and $d(G/O_p(G)) \leqslant 3d/2$ in general; these bounds are sharp (Kovács and Robinson, 1991). Improved bounds for completely reducible groups over specific fields appear in Holt and Roney-Dougal (2013): also if G is a subnormal subgroup of a primitive group, then $d(G) \leqslant 2\log_2 d$. Observe that $|O_p(G)| \leqslant q^{d(d-1)/2}$, so $d(G) = \mathrm{O}(d^2 \log q)$ in general; since there are p-groups $G < \mathrm{GL}(d, \mathbb{F}_q)$ with $d(G) = ed^2/4$, this is again best possible. Every finite group G is generated with high probability by $d(G) + \mathrm{O}(\log\log|G|)$ random elements of G; see, for example, Lubotzky (2002). Thus, to ensure that with provably high probability Y_0 generates $\operatorname{Ker}\theta$, the required value of $|Y_0|$ is $\mathrm{O}(d^2 \log q)$ in general, $\mathrm{O}(d)$ if $O_p(G) = 1$, or $\mathrm{O}(\log d)$ if G_0 is a subnormal subgroup of a primitive group (a condition that holds for many nodes).

As we show in Section 7.2, if the input group $G \leqslant \mathrm{GL}(d, \mathbb{F}_q)$, then $O_p(G)$ is always the group of the left child of the root node; so it and its descendants may need $\mathrm{O}(d^2 \log q)$ generators. On the other hand, the right child of the input group and its descendants almost always have $O_p(G) = 1$, and so need at most $3d/2$ generators. (The only possible exception is the right child of an extraspecial normaliser reduction.) If we know that G_0 is primitive, then we can use the $\mathrm{O}(\log d)$ bound.

Our experience suggests that using large numbers of kernel generators is detrimental to the performance of the algorithm, and it is preferable to impose an absolute upper bound on the initial value of $|Y_0|$. We use different bounds in the three cases: when $O_p(G) = 1$, when G_0 is primitive, and otherwise. If Y_0 does not generate G_0, then we double its size. Our implementation allows flexibility in the initial choice of $|Y_0|$.

Verification If the procedure to verify that $\langle Y_0 \rangle = G_0$ is carried out, then this is done at the same time as constructing the presentation for G from those of G_0 and G_1, as described in Section 4.5.1. In the notation of that section, if we succeed in rewriting the elements r' and $x^{-1}x_0x$ as words in the set X_0 of nice generators of G_0, then we simultaneously check that $\theta_0(\langle X_0 \rangle) = \operatorname{Ker}\theta$ and $\langle Y_0 \rangle = G_0$. (Recall $\langle X_0 \rangle \leq \langle Y_0 \rangle$ and $\langle Y_0 \rangle \leq \operatorname{Ker}\theta$.) Otherwise verification fails.

If $\{X_1 | R_1\}$ is the presentation of G_1, and X_1' is the set of nice generators of G corresponding to X_1, then verification is completed by checking that every $r \in R$, when evaluated on X_1', gives an element of $\langle X_0 \rangle$.

5.3.2. The presentation method

This can be used if a presentation of G_1 is known. Let R_1' be the set of inverse images in G of the relators R_1, as described in Section 4.5.1, so $G_0 = \langle R_1' \rangle^G$. Note that the relators R_1 are words in the nice generators X_1 of G_1, so they need to be rewritten as words in the input generators Y_1 in order to obtain the elements of R_1' as SLPs in Y. We use the Monte Carlo algorithm of Cooperman and Finkelstein (1993) to construct this normal closure. An upper bound to the number of generators needed to generate the normal closure is a function of the length, l_G, of the longest subgroup chain in G. If $G \leqslant \mathrm{GL}(d, \mathbb{F}_q)$ then $l_G = \mathrm{O}(d^2 \log q)$ (Solomon and Turull, 1991); if $G \leqslant \mathrm{Sym}(n)$ then $l_G \leqslant 3n/2$ (Cameron et al., 1989).

5.3.3. A comparison of the methods

Both methods construct random elements of the kernel. The random method relies on the use of the product replacement algorithm to construct random elements of G, and the function ρ to convert these into random elements of G_0. The presentation method requires a set of normal generators R_1' for G_0, as a subgroup of G, arising from a presentation for G_1 on X_1. It initialises Y_0 to R_1', and augments Y_0 by adding random normal subproducts (Cooperman and Finkelstein, 1993) of $\langle Y_0 \rangle$ in

$\langle Y \rangle$. While the resulting set generates G_0 with a provable probability, the elements of R_1' are not uniformly distributed in G_0, and so we may require a large generating set to construct G_0. Using the random method, we initialise Y_0 to contain some initial number of random elements of G_0 and add more as required; we expect that the resulting generating set for Y_0 is smaller. Verification using the presentation method is somewhat easier: since R_1' is already contained in Y_0, there is no need to check this during verification.

Our experience suggests that the random element method performs better than the presentation method provided that appropriate choices are made for the initial values of $|Y_0|$.

5.4. *Crisis management*

In practice, we use smaller bounds for the sizes of the generating sets than those demanded by theory. Hence both methods may fail to find the full kernel. This has consequences for the design of the composition tree algorithm. As part of the construction of the composition tree rooted at the kernel, we test the mandarins for membership in the kernel, which entails testing mandarins for membership of nodes that are descendants of the kernel. If a membership test for a mandarin fails, then some left child on the path from this node to the root was not calculated correctly; it need not be the node where the membership test fails. If this failure occurs, then it is a *crisis*, and we must discard a part of the tree and recompute it. If there are several kernel computations on the path from the node to the root, then we do not know which of them is incorrect. We therefore introduce the notion of a *safe* node defined as follows.

- 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 method guaranteed to construct the full kernel, in which case it is safe if and only if its parent is safe.

If a crisis occurs during rewriting in a node, then we backtrack along the path from the node to the root, until we reach a safe node. We discard the left subtree of this safe node, retaining only the known generators of its left child. We add more generators to this kernel and try again. The number of random elements selected as putative generators for a kernel of a node is always at least the number of generators for the node.

6. Reductions for permutation groups

This and the following section are devoted to Step (1) of the main algorithm summarised in Section 5.2. The individual algorithms that construct the various reduction morphisms are described elsewhere. Here, we explain what they do, and how they fit into the main algorithm.

We provide reductions from general permutation groups to small base groups, which can be then be studied using standard machinery. These reductions were introduced in Neunhöffer and Seress (2006).

6.1. *Intransitivity*

If $G \leqslant \mathrm{Sym}(n)$ is intransitive, then it induces an action on an orbit $\mathcal{O}$. We use standard machinery to construct $\theta : G \to \mathrm{Sym}(\mathcal{O})$.

6.2. *Imprimitivity*

A transitive subgroup of $\mathrm{Sym}(n)$ is imprimitive if it preserves a non-trivial partition of $\Omega = \{1, \dots, n\}$. We use standard machinery to set up the associated reduction.

6.3. Handling the giants

We use the Monte Carlo algorithm described in Seress (2003, §10.2) to determine whether G is $\mathrm{Alt}(n)$ or $\mathrm{Sym}(n)$. If so, it is processed as described in Section 9.5.

6.4. Large base primitive groups

The algorithm of Law et al. (2006) determines if a primitive subgroup G of $\mathrm{Sym}(n)$ is a subgroup of a wreath product of $\mathrm{Sym}(m)$ and $\mathrm{Sym}(r)$ in product action on k-element subsets of $\{1, \ldots, m\}$, containing $\mathrm{Alt}(m)^r$. If so, the algorithm constructs a monomorphism $\theta : G \to \mathrm{Sym}(mr)$ whose faithful image is imprimitive with r blocks of size m.

If none of these reductions applies, then G is a small base group. The leaves of the composition tree for a permutation group are either simple or soluble.

7. Reductions for matrix groups

The reductions that arise from our constructive version of Aschbacher's theorem are described in Leedham-Green (2001) and O'Brien (2006). Here we summarise both these and others. The order in which they are described is the default order in which they are applied; in particular situations they may be applied in a different order. For the most part, we defer discussion of the complications introduced by null subgroups until Section 8.

Neunhöffer (2009) reformulated the classes introduced by Aschbacher (1984) to facilitate easier membership problems; Neunhöffer and Seress (2006) adopt these reformulations.

7.1. Unipotent reductions

$G \leqslant \mathrm{GL}(d, \mathbb{F}_q)$ is unipotent if and only if every $g \in G$ has order a power of p, the characteristic of $\mathbb{F}_q$. Moreover, G is unipotent if and only if every composition factor of the $\mathbb{F}_q G$-module has dimension 1 and G acts trivially on every factor.

We first employ a fast negative test: if any generator of G has order not a power of p, then G is not unipotent. Otherwise we use the MeatAxe (Parker, 1984; Holt and Rees, 1994; Ivanyos and Lux, 2000) to decompose the $\mathbb{F}_q G$-module and test if its composition factors satisfy the criteria.

If G is unipotent, then the MeatAxe provides a change-of-basis matrix c such that G^c is lower unitriangular. The projection of G^c onto its first non-zero subdiagonal is a homomorphism. The image of this reduction is elementary abelian and is treated as a leaf. It has trivial null subgroup and is stored as a PC-group.

7.2. Submodule reductions

Let V be the natural $\mathbb{F}_q G$-module of $G \leqslant \mathrm{GL}(d, \mathbb{F}_q)$. First we find the indecomposable summands of V, so $V \cong V_1 \oplus \cdots \oplus V_k$. Next we find a composition series of each V_i. This also provides a change-of-basis matrix $c_i \in \mathrm{GL}(V_i)$ that exhibits this series. We use the change-of-basis matrix $c = \bigoplus_{i=1}^k c_i$ (diagonal join) to exhibit both the direct sum decomposition of V and the composition series of each V_i. The corresponding composition series of V is

$$0 = V_{1,0} < V_{1,1} < \cdots < V_1 < V_1 \oplus V_{2,1} < \cdots < V_1 \oplus V_2 < \cdots < V.$$

Now G^c is both block lower triangular, corresponding to the composition factors of V, and block diagonal, corresponding to the direct summands of V. We obtain a homomorphism by projecting onto the diagonal blocks corresponding to the composition factors. The kernel of this homomorphism is $O_p(G)$. If the input group G is not unipotent, then this is the first reduction in the composition tree, so the composition tree has the property that $O_p(G)$ is the first kernel. Thus the composition factors arising from $O_p(G)$ are at the bottom of the composition series determined by the tree; so we avoid later rearranging of the (potentially large number of) composition factors of $O_p(G)$. Furthermore, all node groups arising to the right of the root node of the tree (almost always) have trivial p-core and so have generating sets of cardinality $O(d)$.

The group G_1 of the right child is now contained in $\mathrm{GL}(V_1) \times \cdots \times \mathrm{GL}(V_k)$, embedded in $\mathrm{GL}(d, \mathbb{F}_q)$. From this node we obtain a homomorphism onto the first non-trivial summand block. The right child of this reduction is contained in $\mathrm{GL}(V_1)$. From this node we obtain a homomorphism $\theta_{1,1}$ to $\mathrm{GL}(V_{1,1})$, and then a homomorphism $\theta_{1,2}$ from $\mathrm{Ker}\,\theta_{1,1}$ to $\mathrm{GL}(V_{1,2}/V_{1,1})$, and so on. Continuing in this way, we obtain reductions to the groups acting on the composition factors of V.

Of course, we could also obtain a homomorphisms from G_1 onto its restriction to a composition factor of V. If V has n composition factors, then the number of kernel computations is $n-1$ in either case. One argument for proceeding through summands is that it leads to a reduction of dimension. If we map directly to factors, then the first kernel is embedded in $\mathrm{GL}(V/V_{1,1})$. If we first map to a summand, then $\mathrm{Ker}\,\theta_{1,1}$ is embedded in $\mathrm{GL}(V_1/V_{1,1})$ which may have much smaller dimension. Since matrix multiplication has cubic complexity in the dimension, this is desirable.

7.3. Absolute reducibility

Groups that act irreducibly but not absolutely irreducibly are in the *semilinear* Aschbacher class. We consider these separately from the general semilinear case, since we have a faster reduction, an isomorphism, in this special case. An irreducible group $G \leqslant \mathrm{GL}(d, \mathbb{F}_q)$ is not absolutely irreducible if it is reducible when embedded into $\mathrm{GL}(d, \mathbb{F}_{q^e})$ for some $e > 1$. The smallest e such that the constituents of the embedding into $\mathrm{GL}(d, \mathbb{F}_{q^e})$ are absolutely irreducible determines the *splitting field* $\mathbb{F}_{q^e}$ for G. Now G has a faithful representation $\theta : G \to \mathrm{GL}(d/e, \mathbb{F}_{q^e})$. Holt and Rees (1994) describe an extension of the MeatAxe that constructs θ.

7.4. Semilinearity

If $G \leqslant \mathrm{GL}(d, \mathbb{F}_q)$ acts absolutely irreducibly and semilinearly, then there exists $K \lhd G$ that acts irreducibly but not absolutely irreducibly, and so has a splitting field $\mathbb{F}_{q^e}$. Moreover, there is a non-scalar $C \in Z(K)$ such that for every $g \in G$ there exists $i_g \in \{1, \ldots, e\}$ where $Cg = gC^{q^{i_g}}$.

The Smash algorithm of Holt et al. (1996a) returns e and C; so we obtain a homomorphism $\theta : G \to C_e$, defined by $g \mapsto i_g$, with kernel that is not absolutely irreducible.

For this reduction, we obtain generators for the kernel G_0 as follows. Let $G = \langle x_1, \ldots, x_m \rangle$ and let the image $G_1 = \langle y_1, \ldots, y_m \rangle \leqslant C_e$.

(1) Use the extended Euclidean algorithm to obtain $n = \gcd(y_1, \ldots, y_m, e)$ and an expression $n = a_0 e + \sum_{i=1}^{m} a_i y_i$ with $a_i \in \mathbb{Z}$.
(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 $\theta(x) = n$, so each $h_i \in G_0$. Let $k_0 = x^{e/n}$. Then $k_0 \in G_0$, and $\{k_0, h_1, \ldots, h_m\}$ is a normal generating set for G_0.
(3) Hence $G_0 = \langle \{k_0\} \cup \{h_i^{x^j} : 1 \leqslant i \leqslant m, 0 \leqslant j \leqslant e-1\} \rangle$.

7.5. Imprimitivity

$G \leqslant \mathrm{GL}(d, \mathbb{F}_q)$ is imprimitive if it permutes a non-trivial direct sum decomposition of $V = \mathbb{F}_q^d$. Hence $V \cong V_1 \oplus \cdots \oplus V_k$, where all V_i have the same dimension. The algorithm of Holt et al. (1996b) constructs the homomorphism $G \to \mathrm{Sym}(k)$.

The kernel G_0 consists of those elements that preserve the decomposition of V, so it is reducible. The algorithm provides a change-of-basis matrix that exhibits the block diagonal structure of G_0. When we process G_0, we first apply the submodule reductions.

7.6. Extraspecial and symplectic normalisers

$G \leqslant \mathrm{GL}(d, \mathbb{F}_q)$ is in 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 $r | q-1$. If $r > 2$ then R is extraspecial; if $r = 2$ then R is either extraspecial, or a central product of an extraspecial 2-group and a cyclic group of order 4.

If $m = 1$ then the algorithm of Niemeyer (2005) constructs this homomorphism $G \to \mathrm{Sp}(2, \mathbb{F}_r)$. If $m > 1$ then the algorithm of Brooksbank et al. (2006) constructs a homomorphism $G \to \mathrm{GL}(2n, \mathbb{F}_r)$ or $G \to \mathrm{Sym}(r^n)$, where $1 \leqslant n \leqslant m$. (The latter is an action on blocks of imprimitivity.) The algorithms also construct generators of the kernel of the homomorphism.

7.7. *Smaller fields modulo scalars*

$G \leqslant \mathrm{GL}(d, \mathbb{F}_q)$ is in this Aschbacher class if there exists $c \in \mathrm{GL}(d, \mathbb{F}_q)$ such that $H \leqslant \mathrm{GL}(d, \mathbb{F}_s)Z$ with $H^c = G$, where $Z = \mathrm{Z}(\mathrm{GL}(d, \mathbb{F}_q))$ and $\mathbb{F}_s < \mathbb{F}_q$. We use the Las Vegas polynomial-time algorithm of Glasby et al. (2006) to compute the change-of-basis matrix c. If $H \leqslant \mathrm{GL}(d, \mathbb{F}_s)$, then this conjugation is an isomorphism $\theta : G \to H$, in which case H becomes the group of the right child node, whereas the group of the left child node is trivial.

Otherwise each $g \in G$ satisfies $g = h^c\lambda_g$ with $h \in H$ and λ_g scalar. This factorisation is not unique: we can replace h^c and λ_g by $h^c\lambda_s$ and $\lambda_s^{-1}\lambda_g$ respectively for any scalar matrix $\lambda_s \in \mathrm{GL}(d, \mathbb{F}_s)$. We define the upper group G_1 of the image to be the subgroup of Z generated by the elements λ_g for each generator g of G, and the null subgroup N_1 of the image to be the subgroup of Z generated by the null subgroup N of the node and $\mathrm{Z}(\mathrm{GL}(d, \mathbb{F}_s))$. These are stored as cyclic groups. While the map $\theta : GN \to G_1N_1$ defined by $g \mapsto \lambda_g$ for $g \in G$ and $n \mapsto n$ for $n \in N$ is not a homomorphism (or even well-defined), it induces a homomorphism from GN to G_1N_1/N_1. The upper group of the left child is a subgroup of $HN \cap \mathrm{GL}(d, \mathbb{F}_s)$, with $\theta_0 : h \mapsto h^c$, and its null subgroup is $N \cap \mathrm{GL}(d, \mathbb{F}_s)$.

It is this reduction and that of Section 7.8 that obliges us to introduce the null subgroup. Further details are presented in Section 8.

The algorithm of Glasby et al. (2006) requires that G', the derived group of G, acts absolutely irreducibly. Carlson et al. (2009) present a Las Vegas polynomial-time algorithm to find a reduction of an irreducible matrix group G that satisfies at least one of the following properties:

(1) G acts semilinearly;
(2) G can be written modulo scalars over a smaller field;
(3) G' does not act absolutely irreducibly.

In particular, the algorithm either (i) finds a reduction of G of type semilinearity, imprimitivity, smaller field modulo scalars, or tensor product; or (ii) constructs a non-trivial homomorphism from G to $\mathbb{F}_q^\times$.

7.8. *Tensor products*

$G \leqslant \mathrm{GL}(d, \mathbb{F}_q)$ lies in this Aschbacher class if its natural module $V \cong U \otimes W$ and G respects this decomposition. It follows that G is isomorphic to a subgroup of $H_0 \circ H_1$ where $H_0 \leqslant \mathrm{GL}(U)$ and $H_1 \leqslant \mathrm{GL}(W)$.

We use the algorithm of Leedham-Green and O'Brien (1997) to obtain a change-of-basis matrix $c \in \mathrm{GL}(d, \mathbb{F}_q)$ such that G^c is an explicit Kronecker product. For $g \in G$, it is now straightforward to compute $g_0 \in \mathrm{GL}(U)$ and $g_1 \in \mathrm{GL}(W)$ with $g = g_0 \otimes g_1$. This factorisation is not unique: we can replace g_0, g_1 by $\lambda g_0, \lambda^{-1} g_1$ respectively for any scalar matrix $\lambda \in \mathrm{GL}(d, \mathbb{F}_q)$. We therefore define the full group of scalars in $\mathrm{GL}(W)$ to be the null subgroup N_1 of G_1 and define $\theta : g \mapsto g_1$. While θ is not necessarily a homomorphism, it induces a homomorphism from G to $\mathrm{PGL}(W)$.

7.9. *Tensor induction*

$G \leqslant \mathrm{GL}(d, \mathbb{F}_q)$ is tensor induced if its natural module $V \cong U_1 \otimes U_2 \otimes \cdots \otimes U_k$, where all U_i have the same dimension and are permuted by G. We use the algorithm of Leedham-Green and O'Brien (2002) to compute the homomorphism $G \to \mathrm{Sym}(k)$.

The kernel G_0 consists of those elements that preserve the tensor decomposition of V. When we process G_0, we first apply the tensor product reduction.

7.10. Nearly simple reductions

We use the algorithms of Niemeyer and Praeger (1998, 1999) to test if G contains and normalises a classical group in its natural representation. If not, then we assume that G is in the Aschbacher class $\mathcal{S}$; this consists of other irreducible matrix groups that are almost simple modulo scalars and are not semilinear or defined modulo scalars over a smaller field. In either case G is *nearly simple*: it has structure $Z.S.E$ where $Z = \mathrm{Z}(G)$ is the scalar subgroup of G, and S is a non-abelian simple group, and $S.E \leqslant \mathrm{Aut}(S)$.

We apply additional reductions to G. The first three are homomorphisms θ to cyclic groups. If G normalises a classical group in its natural representation, then these three reductions construct a quasisimple group $Z'.S$ with $Z' \leq Z$. If G is in class $\mathcal{S}$, then a fourth is required. In both cases, we treat $Z'.S$ as a leaf.

7.10.1. Determinant

The determinant map for $G \leqslant \mathrm{GL}(d, \mathbb{F}_q)$ is $g \mapsto \det(g) \in \mathbb{F}_q^\times$. Converting $\det(g) \in \mathbb{F}_q^\times$ to a power of a generator of $\mathrm{Im}(\theta)$ requires an invocation of a discrete log algorithm in $\mathbb{F}_q$.

7.10.2. Form action

If $G \leqslant \mathrm{GL}(d, \mathbb{F}_q)$ normalises a classical group H which is not $\mathrm{SL}(d, \mathbb{F}_q)$, then H preserves a non-degenerate bilinear or sesquilinear form with matrix F. Then $hF\bar{h}^T = F$ for every $h \in H$, where $\bar{h} = h$ unless H is a unitary group, in which case $\bar{h} = h^{\sqrt{q}}$ (recall q is a square in this case). Elements of G preserve F up to a scalar, so for each $g \in G$ there exists $\lambda_g \in \mathbb{F}_q^\times$ such that $gF\bar{g}^T = \lambda_g F$. Hence we obtain a homomorphism $g \mapsto \lambda_g$, with cyclic image. Again, converting λ_g to a power of a generator of $\mathrm{Im}(\theta)$ requires a discrete log algorithm in $\mathbb{F}_q$. To determine whether G preserves a classical form modulo scalars, we use the MeatAxe (Holt et al., 2005, §7.5.4).

7.10.3. Spinor norm

If $G \leqslant \mathrm{GL}(d, \mathbb{F}_q)$ is an orthogonal group, then the determinant and form action reductions ensure that $G \leqslant \mathrm{SO}^\epsilon(d, \mathbb{F}_q)$. But $\mathrm{SO}^\epsilon(d, \mathbb{F}_q)$ is not perfect, and we obtain a homomorphism $G \to C_2$ where $g \in G$ maps to its spinor norm. The kernel of this map is $\Omega^\epsilon(d, \mathbb{F}_q)$ which is simple modulo scalars for $d \geq 6$. To calculate spinor norms, we use the algorithms of Murray and Roney-Dougal (2011).

7.10.4. Naming the non-abelian composition factor

We assume for the remainder of this section that G is in class $\mathcal{S}$. The next step is to identify the isomorphism type of its non-abelian composition factor. We first calculate the *stable derivative* $D := G^{(\infty)}$ of G, by repeatedly computing commutators, taking normal closures, and testing whether the resulting group is perfect (O'Brien, 2006, §2). Since G/Z is almost simple, its third derived group is stable.

Since $G < \mathrm{SL}(d, \mathbb{F}_q)$ and $\mathrm{Z}(G)$ is the scalar subgroup of G, we use the bound $|\mathrm{Z}(G)| \leq \gcd(d, q-1)$ to calculate $|\mathrm{Z}(G)|$ using the Monte Carlo algorithm of Babai and Shalev (2001, Theorem 4.15).

The procedure to name S has four steps.

(1) Decide from $|\mathrm{Z}(D)|$ and the (central) orders of a sample of elements whether D could be isomorphic to either $\mathrm{Alt}(n)$ or $2.\mathrm{Alt}(n)$ for some n. If so, attempt to verify the identification by applying the constructive recognition algorithms of Bratus and Pak (2000) and Jambor et al. (2013) to D. If the verification succeeds, then halt.

(2) Investigate the (central) orders of a sample of elements to decide if D could be isomorphic to (a covering group of) a sporadic group. If so, attempt to verify the identification by using the black-box algorithms of Wilson et al. to construct its (central) standard generators. If the verification succeeds, then halt.

(3) We may now assume that S is a group of Lie type. Find the characteristic of S using the algorithm of Liebeck and O'Brien (2007). (An alternative algorithm for absolutely irreducible matrix groups appears in Kantor and Seress, 2009.)

(4) Find the name and defining field of S using the black-box Monte Carlo polynomial-time algorithm of Babai et al. (2002); this assumes that the characteristic is known.

If the naming procedure fails, then we attempt no further reductions on this node; see Section 9.6 for further details.

7.10.5. Coset action

The final reduction is a mapping to a (small degree) permutation group. After applying the previous three reductions, we obtain a nearly simple group with structure $Z'.S.E'$, where Z' and E' are isomorphic to subgroups of the original Z and E. We revert to the notation $G := Z.S.E$ for this group. The determinant reduction ensures that $|Z|$ divides $\gcd(d, q-1)$.

We now construct a homomorphism from G to a permutation group isomorphic to E. We use the method described in Holt and Stather (2008). Observe D is simple modulo scalars, and the scalars in D are restricted to those in the Schur multiplier of S. Let $G_0 := \langle Z, D\rangle$. We construct the permutation action of G on the cosets of G_0; the image of this reduction is isomorphic to E. We test two cosets for equality using the rule $G_0x_1 = G_0x_2$ if and only if $x_1x_2^{-1} \in G_0$. We use the algorithm of Leedham-Green and O'Brien (2002) to decide membership in a normal subgroup.

Since we have identified the isomorphism type of S, we can use our knowledge of $\mathrm{Aut}(S)$ to obtain an upper bound to $|E|$ – this provides a useful termination condition for the reduction. If S is an alternating or sporadic group, then $|E| \leqslant 2$. Otherwise S is a known group of Lie type. If S is in defining characteristic p, then $|G : D| \leqslant dg \log_p q$ where d and g are listed in Conway et al. (1985, Table 5, p. xvi). Since $|G : D| \geqslant |E|$ this provides a useful upper bound to $|E|$. (We thank Frank Lübeck for discussion on this point.)

8. The null subgroup

This section discusses the technicalities involved in handling the null subgroups of nodes, and is probably principally of interest to implementors of the main algorithm. In the GAP implementations (Neunhöffer and Seress, Neunhöffer et al.), the null subgroup is either trivial or the full group of scalars. Our approach offers greater generality, and potentially fewer discrete log calculations.

In Section 7 we discuss the types of reduction epimorphisms θ and kernel monomorphisms θ_0 computed for the non-leaf nodes representing matrix groups. There we usually ignore the fact that the group represented by the node is not just a matrix group, but is the quotient GN/N for a group pair (G, N). Recall that, if (G_0, N_0) and (G_1, N_1) are the group pairs of the left and right children of the node, then we need to define maps $\theta_0 : G_0N_0 \to GN$ and $\theta : GN \to G_1N_1$, with $\theta_0(N_0) \leqslant N$ and $\theta(N) \leqslant N_1$, that induce a monomorphism $\phi_0 : G_0N_0/N_0 \to GN/N$ and an epimorphism $\phi : GN/N \to G_1N_1/N_1$ such that $\mathrm{Ker}\,\phi = \mathrm{Im}\,\phi_0$ or, equivalently, $\mathrm{Im}(\theta_0)N = \{g \in GN \mid \theta(g) \in N_1\} = \theta^{-1}(N_1)$.

Let θ be a reduction map defined on a node, and let X and X_1 be the nice generators of G and of G_1, so $X_1 = \{\theta(x) : x \in X\} \setminus \{1\}$. In Section 4.5.2 we define the map $\rho : G \to \mathrm{Ker}\,\theta$ by $\rho(g) = gw^{-1}$, where w is the result of writing $\theta(g)$ as an SLP in X_1 and evaluating that SLP in X.

To carry out the required computations involving the null subgroup, we need to extend two of our existing techniques:

(i) Our rewriting algorithm at a node must be capable of writing $x \in GN$ as an SLP in X that evaluates to an element of xN.
(ii) We need to extend ρ to a function $\rho : GN \to \theta^{-1}(N_1)$.

Assume that we can solve (i) in the image node. To achieve (ii), we apply our rewriting algorithm in G_1N_1 to $\theta(x)$ for $x \in GN$, and define $\rho(x) = xw^{-1}$, where w is the result of evaluating the SLP for $\theta(x)$ in X.

If we can solve (i) in the leaf nodes, then we use the method described in Section 4.5.2, together with the extended definition of ρ, to solve (i) in non-leaf nodes. Only nodes that represent cyclic or matrix groups may have non-trivial null subgroups. We explain how to solve (i) in cyclic leaf nodes in Section 8.1, and in non-abelian leaf nodes in Section 9.

8.1. Null subgroups of cyclic group nodes

Nodes with cyclic groups are leaves so there are no further reductions. We use a single nice generator for the cyclic upper group G of the node. As explained above, we need the ability to rewrite $x \in GN$ as an SLP in the nice generators of G that evaluates to an element of N. To achieve this, we first find $n \in N$ with $xn \in G$; we then write xn as a power of the nice generator of G.

As we show in Section 8.2, cyclic node groups with non-trivial null subgroups occur only as subgroups of $\mathbb{F}_q^\times$, so both $|G|$ and $|N|$ divide $q-1$. We apply the following proposition (used again in Section 8.2) with $C = G$ and $D = N$ to find the required $n \in N$ with $xn \in G$, and so avoid a discrete log calculation in N.

Proposition 8.1. *Let C and D be subgroups of C_m and let $x \in CD$. Given a list of the primes dividing m, in polynomial time we can find $z \in D$ such that $xz \in C$. (We assume that all elements, including single generators of C and D, are given as powers of a generator of C_m.)*

Proof. Let

$$|C| = \prod_{i=1}^{n} p_i^{a_i}, \qquad |x| = \prod_{i=1}^{n} p_i^{b_i}, \qquad |D| = \prod_{i=1}^{n} p_i^{c_i}$$

where each p_i is prime, $a_i, b_i, c_i \geqslant 0$, $a_i + c_i > 0$ and $b_i \leqslant \max(a_i, c_i)$. From knowledge of the primes dividing these orders and the fact that the exponents a_i, b_i, c_i are polynomially bounded, we can compute a_i, b_i, c_i in polynomial time. Let $q_i = |x|/p_i^{b_i}$ for $i = 1, \dots, n$, and observe that $\gcd(q_1, \dots, q_n) = 1$. Using the extended Euclidean algorithm, we find integers m_i such that $1 = \sum_{i=1}^{n} m_i q_i$.

Let $z_i = x^{m_i q_i}$ for $i = 1, \dots, n$. 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 $|x^{q_i}| = p_i^{b_i}$. Since $z_i = (x^{q_i})^{m_i}$ and $p_i \nmid m_i$, we deduce that $|z_i| = p_i^{b_i}$.

Now define $z := (\prod_{b_i > a_i} z_i)^{-1}$ and observe that if $b_i > a_i$ then $b_i \leqslant c_i$. This implies that $|z|$ divides $|D|$ and hence $z \in D$. Finally, $|xz| = \prod_{b_i \leqslant a_i} z_i$, and since $|z_i| = p_i^{b_i}$ it follows that $xz \in C$. $\square$

8.2. Null subgroups for matrix group reductions

We now define the null subgroups for the various reductions, and discuss ensuing complications in the definitions of θ_0 and θ. For reductions of type `TensorProduct` and `SmallerFieldModScalars`, we introduce a non-trivial null subgroup N_1 into the image node, even if N is trivial, so we cannot avoid the use of null subgroups.

In Table 1, we record how the null subgroups N_0 and N_1 of the kernel and image nodes are defined for each of the reductions applied to $G \leqslant \mathrm{GL}(d, \mathbb{F}_q)$. The notation used there is the following. The null subgroup of a matrix group node always consists of scalar matrices. An entry λ in the table denotes an element of $\mathbb{F}_q$ such that the corresponding null subgroup is $\langle \lambda I_d \rangle$, so $N_0 = \langle \lambda_0 I_{d_0} \rangle$ and $N_1 = \langle \lambda_1 I_{d_1} \rangle$. If G_1 is a cyclic group, it is convenient (for notational purposes) to identify G_1 and N_1 with subgroups of $\mathbb{F}_q^\times$ and define λ_1 to be a generator of N_1. For the `FormAction` reduction, $t = 2$ except in the unitary case, when $t = \sqrt{q} + 1$. The symbol ω_q denotes a primitive element of $\mathbb{F}_q$. The smaller field in the `SmallerField` and `SmallerFieldModScalars` reductions is $\mathbb{F}_s < \mathbb{F}_q$, so $|\omega_s| = s - 1$ and $|\omega_q| = q - 1$.

We now discuss each reduction in greater detail, listing the more straightforward cases first.

`Unipotent`. If G is unipotent, then $G \cap N = 1$, so $GN/N \cong G$ and the null subgroup can be ignored. We put $N_0 = N_1 = 1$.

Table 1
Definitions of null subgroups for children of a node.

Reduction	λ_0	λ_1
`Unipotent`	1	1
`SubmoduleReduction`	1	λ
`AbsoluteReducibility`	1	λ
`Semilinearity`	λ	1
`Imprimitivity`	λ	1
`ExtraspecialNormaliser`	λ	1
`SmallerField`	1	$\lambda^{\lvert\lambda\rvert/\gcd(\lvert\lambda\rvert,\lvert\omega_s\rvert)}$
`SmallerFieldModScalars`	$\lambda^{\lvert\lambda\rvert/\gcd(\lvert\lambda\rvert,\lvert\omega_s\rvert)}$	$\omega_q^{\lvert\omega_q\rvert/\mathrm{lcm}(\lvert\lambda\rvert,\lvert\omega_s\rvert)}$
`TensorProduct`	λ	ω_q
`TensorInduction`	λ	1
`Determinant`	$\lambda^{\lvert\lambda\rvert/\gcd(\lvert\lambda\rvert,d)}$	λ^d
`FormAction`	$\lambda^{\lvert\lambda\rvert/\gcd(\lvert\lambda\rvert,t)}$	λ^t
`SpinorNorm`	λ or λ^2	$\theta(\lambda I_d) = \pm 1$
`CosetAction`	λ	1

`AbsoluteReducibility`. Here θ is an isomorphism, so both G_0 and N_0 are trivial. We put $N_1 = N$.

`SmallerField`. Again θ is an isomorphism, so both G_0 and N_0 are trivial. We define N_1 to be the intersection of N with $\mathrm{GL}(d, \mathbb{F}_s)$.

`Imprimitivity`, `Semilinearity`, `TensorInduction`, `ExtraspecialNormaliser` and `CosetAction`. In each case, θ is a homomorphism with $N < \mathrm{Ker}\,\theta$, so we set $N_1 = 1$ and $N_0 = N$, define $G_0 := \{\rho(g) \mid g \in G\}$, and θ_0 is the identity map.

`TensorProduct`. Recall from Section 7.8 that G respects a decomposition $V \cong U \otimes W$. Let $d = d_0 d_1$, where $d_0 = \dim(U)$ and $d_1 = \dim(W)$. The projection $\theta : GN \to \mathrm{GL}(d_1, \mathbb{F}_q)$ is not generally a homomorphism (it is not even well-defined), but it induces a homomorphism $\phi : GN \to \mathrm{PGL}(d_1, \mathbb{F}_q)$ with $N < \mathrm{Ker}\,\phi$. We define N_1 to be the full scalar subgroup of $\mathrm{GL}(d_1, \mathbb{F}_q)$. Every element of $\mathrm{Ker}\,\phi$ can be written uniquely as $g_0 \otimes I_{d_1}$ with $g_0 \in \mathrm{GL}(d_0, \mathbb{F}_q)$. We define G_0 to be the subgroup of $\mathrm{GL}(d_0, \mathbb{F}_q)$ generated by the elements g_0 arising in this way from a generating set of $\mathrm{Ker}\,\phi$. Recall that $N = \langle \lambda I_d \rangle$. We define $N_0 := \langle \lambda I_{d_0} \rangle$, so $N = N_0 \otimes I_{d_1}$. The map $\theta_0 : G_0 N_0 \to GN$ is defined by $\theta(x) = x \otimes I_{d_1}$. The inverse of θ_0 is computed by writing elements in $\mathrm{Ker}\,\phi$ as $x \otimes I_{d_1}$.

`SubmoduleReduction`. Here $\theta : GN \to \mathrm{GL}(d_1, \mathbb{F}_q)$ is a homomorphism that restricts to a monomorphism on N, and we define $N_1 := \theta(N) = \langle \lambda I_{d_1} \rangle$ and $N_0 = 1$. The elements $\rho(x)$ with $x \in GN$ satisfy $\theta(\rho(x)) = \mu_x I_{d_1} \in N_1$ for some $\mu_x \in \mathbb{F}_q$. We define $G_0 := \{\rho(x)\mu_x^{-1} \mid x \in GN\}$ (so $G_0 \leqslant \mathrm{Ker}\,\theta$), and θ_0 is the identity map.

`Determinant` and `FormAction`. In both cases $\theta : GN \to \mathbb{F}_q^\times$ is a homomorphism and we define $N_1 = \theta(N)$ and $N_0 = \mathrm{Ker}\,\theta|_N$. If $x \in GN$ then $\theta(\rho(x)) = \mu_x \in N_1$. To avoid the possibility of infinite recursion, we must ensure that $G_0 \leqslant \mathrm{Ker}\,\theta$ (where θ_0 is the identity map in these reductions), so we need to find $\nu_x \in N$ with $\theta(\nu_x I_d) = \mu_x$. We define $G_0 := \{\rho(x)\nu_x^{-1} \mid x \in GN\}$.

For `Determinant` reductions, $\theta(\nu_x I_d) = \nu_x^d$, so we need to find ν_x with $\nu_x^d = \mu_x$. We find the roots in $\mathbb{F}_q$ of the polynomial $x^d - \mu_x$ in polynomial time (Geddes et al., 1992, Theorem 8.12). For each root, $\nu_x I_d \in N$ if and only if $|\nu_x|$ divides $|N|$, so we use this test to find ν_x.

Now consider `FormAction` reductions. In the non-unitary case $\theta(\nu_x I_d) = \nu_x^2$, and we proceed 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 $\theta(\nu_x I_d) = \mathrm{N}(\nu_x)$, so we must find $\nu_x \in \mathbb{F}_q^\times$ such that $\mathrm{N}(\nu_x) = \mu_x$. This is a norm equation, which we solve using Murray and Roney-Dougal (2011, Proposition 2.2) to obtain a solution ν_0. Again, we need a solution ν_x such that $\nu_x I_d \in N = \langle \lambda I_d \rangle$. Each solution has the form $\nu_0 \nu_1$ with $\mathrm{N}(\nu_1) = 1$. But $\mathrm{N}(\nu_1) = \nu_1^{\sqrt{q}+1} = 1$ if and only if ν_1 is in the subgroup of $\mathbb{F}_q^\times$ of order $\sqrt{q}+1$, so we can find ν_1 with $\nu_0 \nu_1 \in \langle \lambda \rangle$ by Proposition 8.1.

`SpinorNorm`. This is similar to the previous two reductions, but since $\mathrm{Im}\,\theta$ has order 1 or 2 in this case, it is straightforward to write down generators of $\mathrm{Ker}\,\theta$ directly (as we do in Section 7.4 for semilinear reductions).

`SmallerFieldModScalars`. Recall from Section 7.7 that $H \leqslant \mathrm{GL}(d, \mathbb{F}_s)Z$ with $H^c = G$, where $Z = Z(\mathrm{GL}(d, \mathbb{F}_q))$ and $\mathbb{F}_s < \mathbb{F}_q$, and $\theta(x) = \lambda_x$, where $x = h^c\lambda_x$ and $x \in GN$. While $\theta : GN \to \mathbb{F}_q^\times$ is not generally a homomorphism (it is not even well-defined), it induces a homomorphism $\bar{\theta} : GN \to \mathbb{F}_q^\times/\mathbb{F}_s^\times$. We define $N_1 = \langle \mathbb{F}_s^\times, \theta(N)\rangle$ and $N_0 = N \cap \mathrm{Ker}\,\bar{\theta} = N \cap \mathrm{GL}(d, \mathbb{F}_s)$.

We want G_0 to be a subgroup of $\mathrm{GL}(d, \mathbb{F}_s)$ and to define $\theta_0 : G_0N_0 \to GN$ by $\theta_0(h) = h^c$. To find a generator of G_0, we first choose random $x \in GN$; now $\theta(\rho(x)) = \mu_x \in N_1$. Using Proposition 8.1, we find $\nu_x \in F_q$ satisfying $\mu_x\nu_x^{-1} \in \mathbb{F}_s^\times$. Observe that $(\rho(x)\nu_x^{-1})^{c^{-1}} \in \mathrm{GL}(d, \mathbb{F}_s)$; it is used as a generator of G_0.

9. Processing the leaves

It remains to describe Step (2) of the main algorithm summarised in Section 5.2. For the various types of leaf groups, we employ algorithms to construct nice generators, to solve the rewriting problem, and (possibly) compute a presentation for the represented groups of each leaf in the composition tree of G. The individual algorithms are described elsewhere. Here, we describe what they do, and how they fit into the main algorithm. We also describe how to compute composition series of the leaf groups which, as we saw in Section 3.2, is necessary if we want to carry out further structural computations on the input group G.

As usual, we denote the upper group of a leaf node by G and its null subgroup by N. Recall that the rewriting algorithm must write $g \in GN$ as an SLP in the nice generators of G that evaluates to an element of gN, and the presentation must be on the nice generators of G and define GN/N.

9.1. Cyclic groups

A single generator of a cyclic group is chosen as its nice generator. A presentation is straightforward: a single relation is required. The represented group $H := GN/N$ has presentation $\{x \mid x^{|H|}\}$. The composition factors are straightforward to obtain using standard machinery. Proposition 8.1 provides the rewriting algorithm.

9.2. Elementary abelian groups

These arise from unipotent reductions and we store them as PC-groups. We designate a group's PC-generators as its nice generators, the standard collection process as the rewriting algorithm, and use the corresponding PC-presentation. (Recall that $N = 1$ in this case.) The composition factors are straightforward to obtain using standard machinery.

9.3. Permutation groups

These leaves are simple or soluble primitive permutation groups. If $G \cong \mathrm{Alt}(n)$, then we use the black-box algorithms of Bratus and Pak (2000) and Jambor et al. (2013) to solve the constructive recognition and rewriting problems.

Otherwise, depending on the isomorphism type of G, we either use BSGS machinery or one of the black-box algorithms discussed in Section 9.5.2. If a presentation is not known, then it is constructed using the algorithm of Cannon (1973).

9.4. Classical groups in their natural representation

Since GN is in the kernel of the determinant, form action and spinor norm maps, it is quasisimple and so $N < G$. The leaf groups lie in the families $\mathrm{SL}(d, \mathbb{F}_q)$, $\mathrm{Sp}(d, \mathbb{F}_q)$, $\mathrm{SU}(d, \mathbb{F}_q)$ and $\Omega^\epsilon(d, \mathbb{F}_q)$ for $\epsilon \in \{\pm, \circ\}$.

The standard copy of the simple group $G/\mathrm{Z}(G)$ is represented by the pair $(H, \mathrm{Z}(H))$ where $H^c = G$ for some $c \in \mathrm{GL}(d, \mathbb{F}_q)$, and c is readily computed as a change-of-basis matrix that transforms the form fixed by H to that fixed by G. The set of nice generators of G is X^c, where X is the set of standard generators of H. The algorithms of Leedham-Green and O'Brien (2009) and Dietrich et al. (2013) are used to construct the nice generators of G as SLPs in its input generators. The algorithms of Costi (2009) solve the rewriting problem for H on X. A presentation on the standard generators is available from Leedham-Green and O'Brien (2014).

The composition factors consist of the simple group together with those of $\mathrm{Z}(G)$ modulo N. Since G is a quasisimple classical group in its natural representation, we can write down an explicit scalar matrix that generates $\mathrm{Z}(G)$. We obtain this matrix as an SLP in X^c using Costi (2009). We construct a PC-presentation for the centre, compute the quotient by its null subgroup, and then use standard machinery to obtain its composition factors.

9.5. *Groups in class $\mathcal{S}$*

If $G \leqslant \mathrm{GL}(d, \mathbb{F}_q)$ is a leaf group in class $\mathcal{S}$, then $G = Z.S$, where S is a finite simple group and $Z = \mathrm{Z}(G)$ is the (cyclic) scalar subgroup of G. The extension $Z.S$ may not be perfect, so Z need not lie in the Schur multiplier of S. Since the determinant reduction has been applied, $GN \leqslant \mathrm{SL}(d, \mathbb{F}_q)$, so both $|Z|$ and $|N|$ divide $\gcd(d, q-1)$.

9.5.1. *Calculating the centre*

To obtain a generating set for Z, we use the Monte Carlo algorithm of Babai and Shalev (2001, Theorem 4.15) and the upper bound $\gcd(d, q-1)$ to its order. We then obtain a single generator u of Z as an SLP in the generators of G.

9.5.2. *Constructive recognition and rewriting*

Recall that we have named S using the algorithm of Section 7.10.4. The constructive recognition algorithm for S provides an effective homomorphism $\alpha : GN \to H$ with kernel ZN, where H is the standard copy of S. Let Y be the set of input generators of G. By definition of a constructive recognition algorithm, H has a designated set $\overline{X}$ of standard generators, and there are algorithms to find these generators as SLPs in $\alpha(Y)$, and to rewrite elements in H as SLPs in $\overline{X}$. These algorithms enable the construction of an effective inverse map $\beta : H \to G$ that satisfies $\beta(\alpha(g)) \in gZN$ for all $g \in GN$. We define the nice generators X of G to be the images under β of the standard generators $\overline{X}$ of H, together with the generator u for Z constructed in Section 9.5.1.

We can now describe the algorithm that rewrites $x \in GN$ to an SLP in X that evaluates to an element of xN. For $x \in GN$, we compute $z := x\beta(\alpha(x^{-1})) \in ZN$. We obtain an SLP in X for $\beta(\alpha(x^{-1}))$ by taking the image under β of an SLP in $\overline{X}$ for $\alpha(x^{-1})$, so it remains to obtain an SLP in X for some element of xN. Using Proposition 8.1, we find $n \in N$ with $zn \in Z$, and then we complete the rewriting process by expressing zn as an SLP in u. Since $|u| \leqslant \gcd(d, q-1)$, this is easy.

Since a presentation for S on its standard generators is available, by applying the map β, we obtain a presentation $\{\mathcal{X} \mid \mathcal{R}\}$ for the isomorphic group G/Z on its generating set $\{xZ \mid x \in X \setminus \{u\}\}$. We then obtain a presentation for the represented group GN/N on X as follows. Each relator $r \in \mathcal{R}$ is regarded as a word in X, and evaluates in G to $u_r \in ZN$. After multiplying by a suitable element of N (using Proposition 8.1), we express each u_r as an SLP w_r in u, as above. The relators for the presentation of GN/N are then

$$\left\{rw_r^{-1} : r \in \mathcal{R}\right\} \cup \left\{[x, u] : x \in X\right\} \cup \left\{u^{|u|/\gcd(|u|,|N|)}\right\}.$$

To obtain composition factors for a group in class $\mathcal{S}$ is easy, since we know its centre.

We now list the constructive recognition algorithms available for the various types of simple groups. Some are also applied to permutation group leaves.

Alternating groups We use the black-box algorithms of Bratus and Pak (2000) and Jambor et al. (2013). (In fact we have already done this in verifying our identification of S.)

Sporadic groups We have already found standard generators of S during the naming procedure. As a rewriting algorithm, we use either the "reduction" algorithm of Holmes et al. (2008) or BSGS machinery. If we use the latter, then, as described in O'Brien (2006, §7.6), we choose a base targeted to the given representation of S. This allows us to define the maps α and β.

Classical groups in defining characteristic If $S \cong \mathrm{PSL}(2,q)$, then we use the algorithm of Conder et al. (2006).

If $S \cong \mathrm{PSL}(3,q)$, then we use the algorithm of Lübeck et al. (2007) to obtain the maps α and β. This algorithm applies to GN only if ZN is contained in the Schur multiplier of S. Instead, we apply it to the stable derivative D of GN, which we computed earlier, so obtaining $\overline{X}$ as SLPs in the generators of D, and hence also in Y. This allows us to define the nice generators as above. Note that the domain of α is D, not GN. This poses no problem, since for rewriting in GN we use the algorithm of Costi (2009) which only requires GN and X as input.

If S is a projective representation of degree at most n^2 for a classical group of degree at most n, then we use the algorithms of Magaard et al. (2008) and Corr (2013) to identify the related homomorphism. These algorithms do not apply to "small" n: for example $n \geq 4$ for linear groups, and $n \geq 8$ for orthogonal groups. Otherwise, we use the constructive recognition algorithms of Dietrich et al. (2014). In both cases we use the rewriting algorithms of Costi (2009).

Black-box algorithms for classical groups We use the algorithms of Dietrich et al. (2014) to obtain the maps α and β, and Schneider's implementation of the black-box rewriting algorithm of Ambrose et al. (2011).

Other available black-box algorithms include those of Kantor and Seress (2001) for $\mathrm{PSL}(d,q)$; Brooksbank (2008) for $\mathrm{PSp}(d,q)$; Brooksbank (2003) for $\mathrm{PSU}(d,q)$; and Brooksbank and Kantor (2006) for $\Omega^{\epsilon}(d,q)$.

Exceptional groups We use the algorithms of Bäärnhielm (2006, 2007, 2014) and Bray and Bäärnhielm (2009) to recognise constructively the Suzuki and Ree groups. The Schur multiplier of ${}^2B_2(q)$ is 2^2 for $q=8$ and trivial otherwise. The representations of ${}^2B_2(q)$ in (defining) characteristic 2 have dimension 4^n for some $n \geqslant 1$; since $|Z|$ divides $\gcd(4^n, 2^k-1)$, we deduce that $Z=1$. In odd characteristic, the black-box recognition algorithm for ${}^2B_2(q)$ handles the case when $|Z|=2$ and $Z \leqslant [G,G]$.

The algorithms for ${}^2F_4(q)$ constructively recognise only the smallest representation of dimension 26, with defining field of size $q=2^{2m+1}$ for some $m>0$. Since $\gcd(26, 2^{2m+1}-1)=1$, we deduce that $Z=1$. (By exploiting condensation, we can also work effectively with the 246-dimensional representation.)

The algorithms for ${}^2G_2(q)$ constructively recognise representations in defining characteristic with field size $q=3^{2m+1}$ for some $m>0$. Such representations have dimension $7^n 3^{3k}$ where $n \geqslant 0$, $k \geqslant 0$ and $n+k>0$. Since $\gcd(21, 3^{2m+1}-1)=1$, we deduce that $Z=1$.

Let G be an absolutely irreducible representation in defining characteristic of a finite exceptional groups of Lie type and of rank at least 2. We use the Las Vegas polynomial-time algorithms of Liebeck and O'Brien (2014) to construct standard generators in G; and the *generalised row and column reduction* algorithm of Cohen et al. (2004) to solve the rewriting problem for such a representation. The algorithms of Liebeck and O'Brien (2014) also apply to black box representations.

Kantor and Magaard (2013) present alternative black-box Las Vegas algorithms to recognise constructively the exceptional simple groups of Lie type and rank at least 2, other than ${}^2F_4(q)$, defined over a field of known size.

Default methods If (an implementation of) a constructive recognition algorithm is not available for the simple composition factor of G, then we can use the black-box algorithm of Holmes et al. (2008), which reduces the constructive membership problem to three instances of the same problem for involution centralisers in G. Let X be the set of inverse images in G of the standard generators of GZ/Z. For membership testing, we apply Holmes et al. (2008) to GZ; membership testing in the involution centralisers is done using recursive applications of `CompositionTree`. We obtain an SLP for the element in $X \cup \{\lambda\}$ where $N = \langle\lambda\rangle$, and rewrite this to an SLP in X by setting $\lambda = 1$.

As we observed in Section 3.2, we need a constructive recognition algorithm to compute efficiently a composition series of G; in its absence, we use BSGS machinery. A presentation is found using the algorithm of Cannon (1973).

9.6. *Potential failures*

We identify various components of the main algorithm which may fail, or return incorrect answers.

(i) One of the membership algorithms for Aschbacher classes fails to deduce that a group lies in this class.
(ii) The Monte Carlo naming algorithm described in Section 7.10.4 returns an incorrect answer.
(iii) The constructive recognition algorithm fails.

In practice, these happen infrequently. We do not detect (i), but this rarely leads to failure, since the group is often also in Aschbacher class $\mathcal{S}$ and is processed as such. If (ii) occurs, it will lead to a failure of type (iii). If (iii) occurs, then we compute a BSGS of GN using the random Schreier–Sims algorithm (Holt et al., 2005, §4.4.5), and (optionally) verify it using the Todd–Coxeter–Schreier–Sims algorithm (Holt et al., 2005, §6.2.2). This allows constructive membership testing in G. As above, we rewrite an SLP for an element to obtain it in Y. The *strong generators* of G are chosen as its nice generators, and a presentation is obtained on these. Composition factors are found using BSGS machinery.

10. Identifying automorphisms of finite simple groups

As explained in Section 3.2, `CompositionTree` can compute a composition series $1 = G_0 \lhd G_1 \lhd G_2 \lhd \cdots \lhd G_m = G$ of G. For $1 \leqslant k \leqslant m$, it also computes effective maps $\tau_k : G_k \to S_k$, $\phi_k : S_k \to G_k$, where S_k is the standard copy of the simple group G_k/G_{k-1}, and τ_k is an epimorphism with kernel G_{k-1}, and, for $g \in S_k$, $\phi_k(g)$ is an element of G_k with $\tau_k\phi_k(g) = g$.

As we shall see in the next section, to apply the rearrangement algorithm of Section 3.3 to this series, we must decide whether an automorphism of a non-abelian simple factor S_k that normalises $A \leqslant \mathrm{Aut}(S_k)$ lies in A. If so, then we must identify it as an element of A. More precisely, we solve the following problem.

Problem 10.1. Assume, for some k, that $g_1, g_2, \ldots, g_t \in G$ normalise G_k and G_{k-1}. For $1 \leqslant i \leqslant t$, let α_i be the automorphism of S_k induced by conjugation by g_i; namely, $\alpha_i : x \mapsto \tau_k(g_i^{-1}\phi_k(x)g_i)$.

Let $A_i = \langle \mathrm{Inn}(S_k), \alpha_1, \ldots, \alpha_i\rangle \leqslant \mathrm{Aut}(S_k)$ for $1 \leqslant i \leqslant t$ where $A_0 = \mathrm{Inn}(S_k)$. For $0 \leqslant i < t$, assume that $A_i \lhd A_{i+1}$ and A_{i+1}/A_i has prime order.

Let $g \in N_G(G_k) \cap N_G(G_{k-1})$ be such that the automorphism α of S_k induced by conjugation by g is guaranteed to normalise A_t and either to lie in A_t or to generate a subgroup $A_{t+1} := \langle A_t, \alpha\rangle$ such that A_{t+1}/A_t has prime order.

Decide whether $\alpha \in A_t$. If so, then compute $x \in S_k$ and integers e_i such that $\alpha = c_x\alpha_1^{e_1}\alpha_2^{e_2}\cdots\alpha_t^{e_t}$, where $c_x \in \mathrm{Inn}(S_k)$ is conjugation by x. If $\alpha \notin A_t$, then set $g_{t+1} := g$ and $\alpha_{t+1} := \alpha$.

We describe an efficient algorithm to solve Problem 10.1 when S_k is a classical group in its natural representation. For all other S_k, we currently solve this problem using standard automorphism group algorithms (Holt et al., 2005, Chapter 10).

10.1. *Classical groups in their natural representation*

To ease exposition, we postpone discussion of the triality automorphism of $\Omega^+(8, \mathbb{F}_q)$ to Section 10.1.5. The notation used is local to this section, and may not be consistent with that used in the remainder of the paper.

10.1.1. Automorphism groups of classical groups

Let $\Omega = S_k$ be a non-abelian simple group isomorphic to one of $\mathrm{PSL}(d,q)$ $(d \geq 2)$, $\mathrm{PSp}(d,q)$ $(d \geq 4)$, $\mathrm{PSU}(d,\sqrt{q})$ $(d \geq 3)$, or $\mathrm{P}\Omega^{\epsilon}(d,q)$ $(d \geq 7)$. Here Ω is represented by the group pair $(\widetilde{\Omega}, \mathrm{Z}(\widetilde{\Omega}))$ with $\widetilde{\Omega} = \mathrm{SL}(d,\mathbb{F}_q)$, $\mathrm{Sp}(d,\mathbb{F}_q)$, $\mathrm{SU}(d,\mathbb{F}_q)$, or $\Omega^{\epsilon}(d,\mathbb{F}_q)$, in its natural representation as a subgroup of $\mathrm{GL}(d,\mathbb{F}_q)$. If $\Omega \neq \mathrm{PSL}(d,q)$, then let F be the bilinear, sesquilinear or quadratic form preserved by $\widetilde{\Omega}$, and used to define $\widetilde{\Omega}$.

Denote $\mathrm{Aut}(\Omega)$ by A. Following Kleidman and Liebeck (1990, Chapter 2), observe that A has a chain of normal subgroups

$$1 < \Omega \cong \mathrm{Inn}(\Omega) \leqslant S \leqslant I \leqslant \Delta \leqslant \Gamma \leqslant A,$$

which are defined as follows.

(i) $S = \mathrm{Inn}(\Omega)$ except when $\Omega = \mathrm{P}\Omega^{\epsilon}(d,q)$, in which case S is the group of automorphisms induced by conjugation by elements of $\mathrm{SO}^{\epsilon}(d,\mathbb{F}_q)$.
(ii) I consists of automorphisms induced by conjugation by all elements of $\mathrm{GL}(d,\mathbb{F}_q)$ when $\Omega = \mathrm{PSL}(d,q)$, and by the elements of $\mathrm{GL}(d,\mathbb{F}_q)$ that preserve F in the other cases.
(iii) $\Delta = I$ when $\Omega = \mathrm{PSL}(d,q)$, and consists of automorphisms induced by conjugation by elements of $\mathrm{GL}(d,\mathbb{F}_q)$ that preserve F modulo scalars in the other cases. Equivalently, Δ consists of automorphisms induced by the elements of the normaliser of $\widetilde{\Omega}$ in $\mathrm{GL}(d,\mathbb{F}_q)$.
(iv) Γ is the subgroup of A generated by Δ and the field automorphisms.
(v) A is generated by Γ and a graph automorphism of order 2 when $\Omega = \mathrm{PSL}(d,q)$ with $d \geq 3$, or $\Omega = \mathrm{PSp}(4,2^n)$. Otherwise $A = \Gamma$.

Let $\mathcal{L} := (S/\Omega, I/S, \Delta/I, \Gamma/\Delta, A/\Gamma)$. All members of $\mathcal{L}$ are cyclic and three of them, S/Ω, Δ/I, and A/Γ, have order at most 2.

10.1.2. Data structures for the algorithm

The algorithm to solve Problem 10.1 performs membership testing in a subgroup $A_t = \langle \mathrm{Inn}(\Omega), \alpha_1, \ldots, \alpha_t\rangle$ of A, which is initialised to $A_0 = \mathrm{Inn}(\Omega)$, where the α_i are represented by $g_i \in G$ that induce α_i by conjugation.

For each cyclic quotient $X/Y \in \mathcal{L}$, we store the projection $(A_t \cap X)/(A_t \cap Y)$. If $\alpha_i \in X \setminus Y$, then we record this. For those α_i that lie in Δ, we also store $x_i \in \mathrm{GL}(d,\mathbb{F}_q)$ that induces α_i in its conjugation action on $\widetilde{\Omega}$.

For those X/Y that may have order greater than 2, namely I/S and Γ/Δ, we store a cyclic group isomorphic to $(A_t \cap X)/(A_t \cap Y)$ together with the elements of this cyclic group that correspond to those $\alpha_i \in X \setminus Y$. This enables us to write a new automorphism $\alpha \in X$ with $\alpha Y \in (A_t \cap X)Y$ as a word in these α_i that evaluates to an element of αY. It is trivial to do this when $|X/Y| \leqslant 2$.

To perform membership testing of $\alpha \in A$ in A_t, we consider the five quotients X/Y in reverse order. For each non-trivial quotient, we test membership of α in $(A_t \cap X)Y$. Assume for the moment that we can do this.

If $\alpha \in (A_t \cap X)Y$, then we multiply α by suitable elements of $A_t \cap X$ to get $\alpha \in Y$ (more precisely, we modify the $g \in G$ that induces α), and then proceed to the next quotient X/Y. If this process succeeds for each X/Y, then the modified α lies in $\mathrm{Inn}(\Omega)$, and hence $\alpha \in A_t$. We also need to identify $x \in \Omega$ where α is induced by conjugation by x but, as we see below, x is already calculated during the membership testing in the quotients X/Y.

If $\alpha \notin (A_t \cap X)Y$ for one of X/Y, then we define $g_{t+1} := g$, $\alpha_{t+1} := \alpha$ and $A_{t+1} := \langle A_t, \alpha\rangle$. We also record that $\alpha \in X \setminus Y$. If X/Y is Γ/Δ or I/S, then we modify the stored cyclic group that represents this quotient to make it isomorphic to $(A_{t+1} \cap X)/(A_{t+1} \cap Y)$. Since we assume that $\langle A_t, \alpha\rangle/A_t$ is either trivial or has prime order, at most one of the intersections $A_t \cap X$ changes.

10.1.3. Lifting automorphisms from Ω to $\widetilde{\Omega}$

It remains to describe how to perform membership testing of automorphisms α in $(A_t \cap X)Y$ in the quotients $X/Y \in \mathcal{L}$. Before doing this, we discuss an additional technicality.

The automorphism α of Ω is defined by $g \in G$ that induces α in its conjugation action on G_k/G_{k-1}. The maps τ_k and ϕ_k enable us to calculate the image of elements of $\Omega = S_k$ under α. In all cases under consideration, α lifts uniquely to an automorphism $\tilde{\alpha}$ of $\widetilde{\Omega}$. (This is not true for the triality automorphism of $\Omega^+(8, \mathbb{F}_q)$ when q is odd.) As we shall see later, for the membership tests to work, we need to be able to calculate the action of $\tilde{\alpha}$ on elements of $\widetilde{\Omega}$.

To achieve this, we proceed as follows. Observe $\Omega = \widetilde{\Omega}/Z(\widetilde{\Omega})$. Thus, if $x, y \in \widetilde{\Omega}$, then $\tilde{\alpha}([x, y]) = [a, b]$ for all inverse images a, b of $\alpha(xZ), \alpha(yZ)$ in $\widetilde{\Omega}$. Hence we can calculate the action of $\tilde{\alpha}$ on commutators. Since $\widetilde{\Omega}$ is perfect, it is generated by commutators, so we choose a set of random commutators that generate $\widetilde{\Omega}$; that these generate can be verified using the Monte Carlo algorithm of Niemeyer and Praeger (1998). Since we know the action of $\tilde{\alpha}$ on a generating set, we can compute random elements of $\widetilde{\Omega}$ as SLPs in these generators and then calculate their images under $\tilde{\alpha}$.

10.1.4. The membership tests

We now describe the membership tests for α in each of the five quotients $X/Y \in \mathcal{L}$.

Membership testing in A/Γ and Γ/Δ These tests use characteristic polynomials. Fix $x \in \widetilde{\Omega}$, and suppose that the characteristic polynomial of x is

$$X^d + c_{d-1}X^{d-1} + \cdots + c_1X + c_0.$$

(i) If γ is the graph automorphism of $\mathrm{SL}(d, \mathbb{F}_q)$ $(d \geq 3)$ induced by the inverse–transpose map, then $\gamma(x)$ has characteristic polynomial $X^d + c_1X^{d-1}/c_0 + \cdots + c_{d-1}X/c_0 + 1/c_0$.

(ii) If $\tilde{\phi}$ is the field automorphism of $\widetilde{\Omega}$ induced by $\phi \in \mathrm{Aut}(K)$, where $K = \mathbb{F}_q$, then $\tilde{\phi}(x)$ has characteristic polynomial $X^d + \phi(c_{d-1})X^{d-1} + \cdots + \phi(c_1)X + \phi(c_0)$.

We call $x \in \widetilde{\Omega}$ *full* if the coefficients of its characteristic polynomial generate K. Assume for the moment that $\widetilde{\Omega} \neq \mathrm{Sp}(4, \mathbb{F}_q)$ with q even. By finding a small number of full $x \in \widetilde{\Omega}$ and calculating the characteristic polynomials of both x and $\tilde{\alpha}(x)$, the two properties listed above enable us quickly to decide whether $\tilde{\alpha} \in \Gamma$; if so, we use the isomorphism between Γ/Δ and $\mathrm{Aut}(K)$ to identify the coset of Δ in which it lies. Hence we can perform membership testing in A/Γ and in Γ/Δ.

Now consider the case $\widetilde{\Omega} = \mathrm{Sp}(4, \mathbb{F}_q)$ with q even. If, after calculating the characteristic polynomials of both x and $\tilde{\alpha}(x)$ for a small number of full elements x, we fail to identify α as an element of Γ, then we conclude that $\alpha \in A \setminus \Gamma$. If we conclude incorrectly that $\alpha \in \Gamma$, then the module isomorphism test described below will fail, and we again conclude that $\alpha \in A \setminus \Gamma$.

Identifying a conjugating element We now assume that $\alpha \in \Delta$; thus, $\tilde{\alpha}$ is induced by conjugation by some $x \in \mathrm{GL}(d, \mathbb{F}_q)$. We perform a module isomorphism test between the natural module V for $\widetilde{\Omega}$ and the module $V^{\tilde{\alpha}}$ defined by applying $\tilde{\alpha}$ to the action matrices of V, and obtain x as the matrix describing the explicit isomorphism. This test can be carried out efficiently using MEATAXE methods (Holt et al., 2005, §7.5.3). Since $\widetilde{\Omega}$ is absolutely irreducible, x is determined up to multiplication by a scalar.

Membership testing in Δ/I If $\Omega = \mathrm{PSL}(d, q)$, then $I = \Delta$. Otherwise x preserves the form F modulo scalars: it transforms F to λF for some $\lambda \in K$. If the result of the membership test is positive, then we ensure that the (possibly modified) element x fixes F.

If $\Omega = \mathrm{PSU}(d, \sqrt{q})$, $\mathrm{PSp}(d, q)$ (q even) or $\mathrm{P}\Omega^{\pm}(d, q)$ (d odd or q even), then $\Delta = I$, so $\alpha \in I$, but we still need to multiply x by a scalar to make it fix F. If $\Omega = \mathrm{PSU}(d, \sqrt{q})$, then λ must lie in the subfield of K of order q, and we solve a norm equation to find $\mu \in K$ with $\mu^{1+q} = \lambda$; now $\mu^{-1}x$ preserves F, and we replace x by $\mu^{-1}x$. In the other cases, λ must be a square in K, and we find a square root μ of λ in F and replace x by $\mu^{-1}x$.

It remains to consider the cases $\Omega = \mathrm{PSp}(d, q)$ or $\mathrm{P}\Omega^{\pm}(d, q)$ for even d and odd q, when $|\Delta/I| = 2$. We first test whether λ has a square root $\mu \in K$. If so, then $\alpha \in I$, and we replace x by $\mu^{-1}x$ to make x preserve F. If λ is a non-square in K, then $\alpha \in \Delta \setminus I$. If there is already an automorphism $\alpha_i \in A_t$ $(i \leqslant t)$ with $\alpha_i \in \Delta \setminus I$, then we replace α by $\alpha\alpha_i$ (or rather g by gg_i) and x by xx_i, where x_i is the stored element of $\mathrm{GL}(d, \mathbb{F}_q)$ associated with α_i. We then multiply x by a scalar to make it preserve F. If λ is a non-square in K and there is no existing $\alpha_i \in A_t$ with $\alpha_i \in \Delta \setminus I$, then the result of the membership test is negative.

Membership testing in I/S and S/Ω Now $\alpha \in I$ and x preserves F. The determinant of x determines the coset of S in which α lies, so the membership test in I/S is straightforward. If we find that $\alpha \in A_t S$, then we multiply α by a suitable word in the stored α_i and x by the same word in the associated x_i to get $\alpha \in S$. We can then multiply x by a scalar to get $\det(x) = 1$.

In the orthogonal case, the membership test in S/Ω is carried out by calculating the spinor norm of x, using Murray and Roney-Dougal (2011).

This concludes the description of the membership tests in the quotients $X/Y \in \mathcal{L}$. As explained earlier, if the result of all of these tests is positive, then the modified α lies in $\mathrm{Inn}(\Omega)$, and we have computed $x \in \widetilde{\Omega}$ where α is induced by conjugation by x.

10.1.5. $\mathrm{P}\Omega^+(8, q)$

The general algorithm does not address the triality automorphism τ of $\Omega := \mathrm{P}\Omega^+(8, q)$. In this case, we define the subgroups $\Omega, S, I, \Delta, \Gamma = A$ of $\mathrm{Aut}(\Omega)$ as for the other classical groups. Now $\mathrm{Aut}(\Omega) = \langle A, \tau \rangle$ and $|\mathrm{Aut}(\Omega) : A| = 3$. If we encounter an automorphism α that is not in A, then we store it, and view it as a coset representative of A in $\mathrm{Aut}(\Omega)$. We can test whether automorphisms lie in A or in the same coset as α and, if not, then we store a second automorphism representing the other non-trivial coset. This is similar to the method employed for the graph automorphism of $\mathrm{PSp}(4, 2^e)$.

11. The rearranging algorithm

The method of identifying automorphisms of classical groups presented in Section 10 is one of the major components of the rearranging algorithm which we now describe in detail.

The non-abelian S_k may be represented either as permutation groups, or as absolutely irreducible matrix groups over finite fields. In the latter case, S_k may be represented by a group pair $(\tilde{S}_k, Z)$, where the null subgroup Z is the scalar subgroup of $\tilde{S}_k$.

`CompositionTree` returns generating sets X_k of S_k that are guaranteed not to contain the identity element, together with functions ψ_k that express elements of S_k as SLPs in X_k. If S_k is cyclic, then $|X_k| = 1$. `CompositionTree` also returns a function δ that rewrites elements of G as SLPs in the nice generators of G.

We start the rearranging process by using the maps ϕ_k to compute and store sets W_k of inverse images in G_k of the elements of X_k. Since $1 \notin X_k$, we know that $g \notin G_{k-1}$ for all $g \in W_k$. We use δ to compute sets $\overline{W}_k$ of SLPs in the nice generators of G that represent the elements of W_k.

We define the *height* $h(g)$ of $g \in G$ to be the smallest k such that $g \in G_k$. The maps τ_k, ϕ_k and ψ_k allow us to compute both $h(g)$ and an SLP for g in $\bigcup_{k=1}^{h(g)} W_k$.

The algorithm proceeds by considering each set W_k, for $k = 1, 2, \ldots, m$ in turn and, when appropriate, replacing each $g \in W_k$ by gh for some suitable $h \in G_{k-1}$, while also making the corresponding changes to the elements of $\overline{W}_k$. These changes do not affect the property that τ_k maps W_k to X_k. After completing the process, each term of the characteristic series $1 \leqslant L \leqslant M \leqslant K \leqslant G$ discussed in Section 3.3 is generated by the union of some of the adjusted sets W_k.

More precisely, for $g \in W_k$, we first attempt to find $h \in G_{k-1}$ with $gh \in K$. If we succeed, then we replace g by gh and attempt to find a new $h \in G_{k-1}$ with $gh \in M$. If that also succeeds, then we attempt to find $h \in G_{k-1}$ with $gh \in L$.

It is straightforward to show that $G_k K = G_{k-1} K$ is equivalent to the following condition: for each $g \in W_k$, there exists $h \in G_{k-1}$ with $gh \in K$. Since $G_{k-1} K$ is a normal subgroup of $G_k K$ and G_k/G_{k-1} is simple, $G_k \cap G_{k-1} K$ must equal either G_k or G_{k-1}. Since we assume that $g \notin G_{k-1}$ for all $g \in W_k$, the elements h exist either for all $g \in W_k$ or for none of them. Hence, to decide this question, we need only consider one $g \in W_k$. Of course, if h exists, then we must compute such for every $g \in W_k$.

Since the result of a positive outcome of these tests is to replace the elements of W_k by elements that lie in K, M, or L, we frequently refer (with considerable abuse of language) to these processes as testing the composition factors for membership of K, M and L.

After their completion, K is generated by those W_k for which we successfully replaced the $g \in W_k$ by $gh \in K$, and similarly for M and L. Thus we can rearrange the composition series so that it passes through L, M and K.

If we test a non-abelian composition factor G_k/G_{k-1} for membership of M, then, since K/M is soluble, we know *a priori* that the answer must be positive. In this situation, we replace the $g \in W_k$ by elements gh that generate one of the simple direct factors of M/L.

11.1. Auxiliary data required

Before explaining how we test for the existence of suitable h with $gh \in K$, M or L, we describe some extra data maintained during the rearranging process.

When we consider the composition factor G_k/G_{k-1}, the (adjusted) generators W_j of some of the non-abelian factors G_j/G_{j-1} with $j < k$ that we have already considered may generate simple factors of M/L. Let these factors (if any) be indexed by $j_1, j_2, \dots, j_r$, where $0 \leqslant r < k$ and $j_1 < j_2 < \cdots < j_r < k$, and let $\Delta = \{j_1, j_2, \dots, j_r\}$.

Let $P \leqslant \mathrm{Sym}(\Delta)$ be generated by the conjugation action of the generators W_i of those factors G_i/G_{i-1} with $i < k$ that do not lie in K on the simple factors $\langle W_j\rangle L/L$ of M/L with $j \in \Delta$. We keep track of the correspondence between the generators of P and the corresponding elements of the W_i.

Initially, $r = 0$ and P is the trivial group acting on the empty set Δ. The factor G_k/G_{k-1} under consideration may be a new simple factor of M/L, in which case we adjoin k to Δ. This new factor is fixed under conjugation by the elements of W_i for all $i < k$ in the induced permutation action of G on the complete set of simple factors of M/L. Thus the only change needed to P when we adjoin k to Δ is to extend the definition of its generators by making them fix the new point k.

On completion of the algorithm, Δ can be identified with the set of all simple factors of M/L (also named Δ in Section 3.3), and $P = \mathrm{Im}\,\phi$, where $\phi : G \to \mathrm{Sym}(\Delta)$ is the homomorphism with kernel K.

During the course of the algorithm, we carry out membership testing in P, and rewrite elements as words in its generators, using standard permutation group algorithms.

For each $j \in \Delta$, we maintain a subgroup A_j of $\mathrm{Aut}(S_j)$. It is generated by automorphisms induced by conjugation by the generators W_i of certain composition factors G_i/G_{i-1} that lie in K but not in M. As with P, we keep track of the correspondence between the generators of A_j and the corresponding sets W_i. More precisely, immediately before we consider the composition factor G_k/G_{k-1}, A_j is the group of automorphisms of S_j induced by those elements of $G_{k-1} \cap K$ that centralise all of the simple factors $\langle W_{j'}\rangle L$ of M/L with $j < j' \in \Delta$.

When G_j/G_{j-1} is first identified as a direct factor of M/L, we initialise A_j to $\mathrm{Inn}(S_j)$. Suppose that the factor G_k/G_{k-1} with $k > j$ under consideration turns out to lie in K but not in M. Since G_k/G_{k-1} must be cyclic, $W_k = \{g\}$ for some g. As we explain in more detail in Section 11.2.2, we now adjust g such that, for some specific $j \in \Delta$, it centralises all factors $\langle W_{j'}\rangle L$ with $j < j' \in \Delta$, but the automorphism α of S_j induced by conjugation by g does not lie in A_j. We then append α to A_j. The computations required in A_j are described in Section 10.

As was the case with P, this setup is not disturbed by later composition factors G_k/G_{k-1} that are simple factors of M/L. The elements in earlier factors that induce the generators of the groups A_j automatically induce the identity automorphism on all such later factors.

11.2. The rearrangement algorithm in detail

Observe that non-abelian composition factors cannot belong to K/M, and abelian composition factors cannot belong to M/L. We first list the three main steps of the algorithm and then describe them in greater detail.

Initialise Δ to {} and P to $\mathrm{Sym}(\Delta)$.
For $k = 1, 2, \dots, m$, do the following, with $W_k = \{g_1, g_2, \dots, g_s\}$, or $W_k = \{g\}$ if S_k is cyclic.

(i) Decide whether there exist $h_i \in G_{k-1}$ $(1 \leqslant i \leqslant s)$ with $g_i h_i \in K$. If so, replace g_i by $g_i h_i$. If not, then adjoin the permutations of Δ induced by conjugation by the elements of W_k as new generators of P, and proceed to the next value of k.

(ii) If S_k is cyclic, then decide whether there exists $h \in G_{k-1}$ with $gh \in L$. If so, replace g by gh. If not, then there exists $h \in G_{k-1}$ with the following property: there exists $j \in \Delta$ such that gh centralises $\langle W_{j'} \rangle L/L$ for all $j' \in \Delta$ with $j' > j$, but the automorphism α of S_j induced by gh does not lie in A_j. Replace g by gh and adjoin α to the set of generators of A_j.

(iii) If S_k is insoluble, then find $h_i \in G_{k-1}$ such that, for $1 \leqslant i \leqslant s$, each $g_i h_i$ centralises each factor G_j/G_{j-1} with $j \in \Delta$, and replace each g_i by $g_i h_i$. (The adjusted elements g_i generate a new simple factor of M/L.) Adjoin k to Δ and initialise A_k to $\mathrm{Inn}(S_k)$.

11.2.1. *Does G_k/G_{k-1} lie in K?*

If $r \leqslant 1$, then there is nothing to do in Step (i), since G_k/G_{k-1} must already lie in M. Suppose that $r > 1$. We consider the first generator g_1 in W_k, and calculate its permutation action σ_1 on $\Delta = \{j_1, j_2, \ldots, j_r\}$. To do this, we let x_i be the first generator in W_{j_i} for $1 \leqslant i \leqslant r$, and calculate $x_i' := x_i^{g_1}$ and its height $h_i := h(x_i')$. Since x_i and hence also x_i' lie in M, each composition factor G_{h_i}/G_{h_i-1} must lie in M. If S_{h_i} is non-abelian then, since conjugation by g_1 permutes the simple factors of M/L, it follows that $h_i \in \Delta$ and h_i is the required image $j_i^{\sigma_1}$ that we are attempting to calculate. If S_{h_i} is cyclic, then G_{h_i}/G_{h_i-1} must lie in L. In that case, we replace x_i' by $x_i'\phi_i(\tau_i(x_i'))^{-1}$, which reduces the height of x_i' without changing $x_i'L$, and try again. Eventually, x_i' must lie in a non-abelian composition factor, which enables us to compute the required image $j_i^{\sigma_1}$.

Thus we compute σ_1, and it is easily seen that the composition factor G_k/G_{k-1} lies in K if and only if $\sigma_1 \in P$, which we can test. Irrespective of the result of this test, we then proceed to calculate σ_i for $2 \leqslant i \leqslant j_r$. As explained above, $\sigma_1 \in P$ if and only if $\sigma_i \in P$ for all i. If so, then we rewrite each σ_i as a word in the generators of P_i. Since these generators correspond to elements of W_i for $i < k$, we then multiply g_i (for $1 \leqslant i \leqslant t$) by the inverse of the corresponding elements of G_i to yield elements $g_i h_i$ that induce the identity permutation on Δ, and hence lie in K, as required.

If $\sigma_i \notin P$, then we define $P^* := \langle P, \sigma_i \rangle$. Since G_k/G_{k-1} is simple, P^*/P is simple and isomorphic to G_k/G_{k-1}. We now replace P by P^*. When the algorithm completes, we know a composition series for P.

11.2.2. *Does cyclic G_k/G_{k-1} lie in L?*

Suppose that G_k/G_{k-1} is cyclic and lies in K, so we must test it for membership of M. Since M/L has no abelian composition factors, this is equivalent to membership of L. Since G_k/G_{k-1} is cyclic, $W_k = \{g\}$ for some g.

Since g is in K, it normalises each simple factor $\langle W_j \rangle L/L$ of M/L with $j \in \Delta$. For each such j, let α_j be the automorphism of S_j induced by conjugation by g; that is, $x^{\alpha_j} = \tau(g^{-1}\phi(x)g)$ for $x \in S_j$.

We proceed as follows. We consider $\langle W_j \rangle L/L$ in turn for $j = j_r, j_{r-1}, \ldots, j_1$. We first use the methods described in Section 10 to test whether $\alpha_j \in A_j$ and, if so, express α_j as a product of an inner automorphism of S_j and a word in the generators of $\mathrm{Aut}(S_j)$ that have been adjoined to A_j when considering earlier composition factors of G.

If $\alpha_j \in A_j$, then we use the calculated word to find $h \in G_{k-1}$ such that the automorphism induced by gh centralises $\langle W_j \rangle L/L$. We then replace g by gh, recalculate the $\alpha_{j'}$ for $j' \in \Delta$ with $j' < j$, and proceed to the next value of j.

If $\alpha_j \in A_j$ for all $j \in \Delta$, then the automorphism α induced by conjugation by the new element g centralises M/L, so $g \in L$, and we have achieved our objective.

On the other hand, if $\alpha_j \notin A_j$ for some j, then we adjoin α_j as a new generator of A_j and proceed to the next value of k.

We consider the factors $\langle W_j \rangle L/L$ in reverse order because the property that the elements g that induce the automorphisms that generate A_j centralise all factors $\langle W_{j'} \rangle L/L$ with $j' > j$ is not affected by new factors of M/L discovered later.

As we observed for P, if $A^* = \langle A_j, \alpha_j \rangle$, then $A_j^*/A_j \cong G_k/G_{k-1}$, which is cyclic of prime order. The algorithm described in Section 10 assumes that this property holds.

11.2.3. *Processing insoluble G_k/G_{k-1} in K*

Finally, suppose that G_k/G_{k-1} is insoluble and lies in K. Since K/M is soluble, G_k/G_{k-1} must lie in M, and we wish to replace the elements of W_k by elements that generate a simple direct factor of M/L.

For $g_i \in W_k$ and $j \in \Delta$, let α_{ij} be the automorphism of S_j induced by conjugation by g_i. As above, we consider the $j \in \Delta$ in decreasing order. Since we know already that G_k/G_{k-1} lies in M, it is guaranteed that $\alpha_{ij} \in A_j$ at each step. For each j in turn, we replace g_i by suitable g_ih_i with $h_i \in G_{k-1}$ such that g_ih_i centralises $\langle W_j\rangle L/L$, recalculate the $\alpha_{ij'}$ for $j' \in \Delta$ with $j' < j$, and then proceed to the next j. After doing this for all $j \in \Delta$, the resulting elements of W_j centralise all of the factors $\langle W_j\rangle L/L$, and hence generate a simple direct factor of M/L as required.

12. Algorithms exploiting the Soluble Radical Model

After rearranging the composition series to pass through L, M and K, we compute representations of the soluble layers L and K/M as PC-groups, together with effective maps that enable us to move between L or K/M and its PC-representation. We do this also for the unipotent radical $O_p(G)$ of G which, from the design of `CompositionTree`, is guaranteed to be at the bottom of the composition series. Doing this is straightforward using the available functions for rewriting elements of the simple factors S_k as SLPs in their generators. For each cyclic factor, the SLPs are in a single generator.

We now briefly discuss examples of computations that can be carried out in $G \leqslant \mathrm{GL}(d, \mathbb{F}_q)$ by exploiting `CompositionTree` and the Soluble Radical Model.

12.1. *A chief series*

Computing a chief series that refines the characteristic series can be subdivided into producing those factors that lie in G/K, K/M, M/L and L. Since G/K is represented as a permutation group P, we use standard permutation group machinery to find a chief series. For M/L there is essentially nothing further to do: the chief factors within this layer correspond to the orbits of P.

This leaves the two soluble factors K/M and L. We use standard PC-group machinery to find chains of characteristic subgroups of K/M and L with elementary abelian layers, set up the layers as $\mathbb{F}_rG$-modules for the appropriate primes r, and calculate composition series for these modules. The terms in these series correspond to terms in a chief series of G.

12.2. *The centre*

First we compute $Z(L)$ (which contains $Z(G)$) in the PC-representation of L. Next we calculate the conjugation action of the generators of G on $Z(L)$. Now $Z(G)$ is the subgroup of $Z(L)$ fixed under this action. We set up $Z(L)$ under the action of G as a $\mathbb{Z}G$-module and find the fixed subgroup via a matrix nullspace computation.

12.3. *Sylow subgroups*

Let r be a prime. Our method for computing a Sylow r-subgroup of G assumes that we can find Sylow r-subgroups of the simple direct factors of M/L, and also that we can solve the conjugacy problem for two such Sylow r-subgroups. For classical groups in their natural representation, algorithms (with MAGMA implementations) to solve both problems are described in Stather (2007). For other types of simple groups, we use BSGS machinery.

Here is a brief outline of our algorithm to construct $R \in \mathrm{Syl}_r(G)$. If $G = L$ then we use its PC-representation to find R, so assume not, in which case M/L is non-trivial. First we find Sylow r-subgroups R_1, R_2, R_3 and R_4 of L, M/L, K/M and G/K. We do this using the PC-representations for L and for K/M, the permutation representation of G/K, and the methods of Stather (2007) for M/L. If R_2 is trivial, then we also find a Sylow 2-subgroup T of M/L; as we see below, the fact that T is non-trivial reduces the problem to a computation in a smaller group.

By solving the conjugacy problem for Sylow r-subgroups of K/M, we find inverse images in G of the generators of R_4 which (modulo M) normalise R_3 and have order a power of r. We combine these with generators of R_3 to produce generators of a subgroup R_5 of G/M which normalises R_3.

Similarly, by solving the conjugacy problem for Sylow r-subgroups of M/L (or for Sylow 2-subgroups if $R_2 = 1$), we find inverse images in G of the generators of R_5 which (modulo L) normalise R_2 (or T if $R_2 = 1$) and have order a power of r. We combine these with generators of R_2 to produce generators of a subgroup R_6 of G/L which normalises R_2 (or T if $R_2 = 1$).

Finally, by solving the conjugacy problem for Sylow r-subgroups of L, we find inverse images in G of the generators of R_6 which normalise R_1 and have order a power of r. We combine these with generators of R_1 to produce generators of a subgroup R of G which normalises R_1. Since RL/L normalises R_2 or T, it is a proper subgroup of G, and contains a Sylow r-subgroup of G. We compute its order (either using `CompositionTree` or BSGS machinery); in practice, usually $R \in \mathrm{Syl}_r(G)$ but, if not, then we apply the algorithm recursively to R.

As we demonstrate in Example 5 below, a Sylow 2-subgroup can often be found as a subgroup of an appropriate involution centraliser.

12.4. Other algorithms

We have also implemented algorithms which exploit both `CompositionTree` and the Soluble Radical Model to perform the following computations in $G \leqslant \mathrm{GL}(d, \mathbb{F}_q)$: test for its nilpotency and solubility; determine its derived group; determine the normal closure in G of a subgroup; determine the unipotent and soluble radicals and Fitting subgroup of G.

Hulpke (2013) describes algorithms that use a combination of BSGS methods and the GAP `recog` package to compute centralisers and conjugacy classes in matrix groups; we have implemented these using `CompositionTree` in Magma.

13. Implementation and performance

We briefly discuss our implementation of the algorithms in Magma and report their performance on some examples.

13.1. Implementation

The `CompositionTree` intrinsic in Magma has a large number of options. Most are technical, we describe just a few of the significant ones. The kernel of a reduction can be constructed using the random element or presentation method. Short presentations for the classical groups on standard generators are incorporated. The user can verify the correctness of the tree using presentations at each node during the construction of the tree. Alternatively, the complete tree can be verified after its construction. The initial number of kernel generators used in the random element method of Section 5.3.1 can significantly affect the performance. It can be specified as a function, depending on factors such as the degree, the field, and reduction type being used. The number of mandarins can be set by the user.

Our package for structural computations in matrix groups over finite fields decides initially whether to use `CompositionTree` or BSGS machinery on a given group G. To do so, it carries out irreducibility and imprimitivity tests on G, as described in Section 7, and then makes a base change to reflect the submodule or block structure of the associated module. This assists the search for a base with short basic orbits. If the BSGS algorithm encounters a basic orbit of length exceeding some (user-supplied) upper limit, then we use `CompositionTree`. The resulting decision is not always optimal, and the user can opt to apply `CompositionTree` to G.

The package uses a large body of code, developed both as part of this work and independently. These contributions are noted in the online documentation.

13.2. Examples

There are some situations in which the rearranging algorithm described in Section 11 does not perform well. For example, we can construct soluble matrix groups of moderately small degree and a large number of composition factors by starting with Sym(4) as a subgroup of $\mathrm{GL}(2, \mathbb{F}_p)$ for some prime p, and then repeatedly taking wreath products with Sym(4). Usually `CompositionTree` produces a composition series that requires much rearranging to obtain a chief series. The implementations of Holt and Stather (2008) and Stather (2006) perform better on such examples. Wreath products of small degree matrix groups with transitive permutation groups of moderately high degree are also demanding. Such examples of degree up to about 50 can typically be processed easily using BSGS machinery (see Example 3 below).

We now discuss some illustrative examples. All computations were carried out on a Dell Latitude E6510 with 4 GB of RAM using the implementation publicly available as part of Magma 2.20-4. Since most of the algorithms are randomised, individual timings may vary. Lübeck and Müller, and Parker and Wilson have provided challenge problems. Our machinery can identify the composition factors of each challenge group.

(1) One such challenge is the sporadic simple group J_4 given as a 2-generator subgroup of $\mathrm{GL}(112, 4)$. `CompositionTree` completed for this group in 50 s; the first reduction writes the group over a smaller field.
(2) A second challenge is an irreducible 4-generator subgroup of $\mathrm{GL}(168, 61^2)$. `CompositionTree` completed for this group in 1100 s. It is a tensor product of 12- and 14-dimensional groups. The composition factors, obtained in 50 s, are $\mathrm{PSL}(12, 61^2)$, $\Omega^-(14, 61)$, and cyclic groups of order 2 (six times), 3 (twice), 5 and 31. A chief series was computed in 270 s and a Sylow 3-subgroup in 165 s.
(3) A third challenge is a 2-generator subgroup of $\mathrm{GL}(84, 7)$: an imprimitive wreath product of a 4-dimensional matrix representation of $\mathrm{SL}(2, 7)$ with a permutation representation of $\mathrm{PSL}(2, 7)$ of degree 21. `CompositionTree` completed in 5 s, and the chief factors were obtained in 45 s. This example illustrates well the cost of the rearrangement algorithm. By contrast, the BSGS machinery for the same tasks completed in 10 s, following an initial change to a basis compatible with the 21 blocks.
(4) One priority is to provide a wide range of functionality for carrying out computations reasonably quickly in matrix groups of moderately small degree. We have extensively tested maximal subgroups of classical groups, which can be obtained using the Magma intrinsic `Classical-Maximals`. For example, $\mathrm{SU}(12, 5)$ has 26 classes of maximal subgroups (up to conjugation in its automorphism group), of which eleven are reducible (with unipotent radical of order dividing 5^{48}); six imprimitive (block sizes $6, 6, 4, 3, 2, 1$); one semilinear; two tensor products (2×6 and 3×4); two defined over proper subfields modulo scalars (orthogonal and symplectic groups over $\mathbb{F}_5$); and three in the Aschbacher class $\mathcal{S}$ (extensions of $\mathrm{SL}(2, 23)$, 6.Suz and $6.A_7$). Chief series and all non-trivial Sylow p-subgroups were found for all of these subgroups in a total of 1760 s.
(5) As one example of a problem that arose naturally in some research work, we were asked to find a Sylow 2-subgroup of the simple exceptional Lie type group $G := F_4(3)$. The constructive membership problem for G can be solved using the technique described in Holmes et al. (2008), and `CompositionTree` successfully completed in 35 s, so confirming that the order of its Sylow 2-subgroup is 2^{15}. We cannot yet construct the Sylow subgroups directly. Instead, we used our Magma intrinsic `CentraliserOfInvolution`, based on the Monte Carlo algorithm of Bray (2000), to compute $C_G(t)$ for an involution $t \in G$. We did this for random involutions t until we found $C_G(t)$ with order divisible by 2^{15}. (This may occur with $C_G(t) \cong 2.\mathrm{O}_9(3)$ or $2.\mathrm{O}_8^-(3)$.) It was now a 1-second computation, using the method described in Section 12.3, to find the Sylow 2-subgroup of G as a subgroup of $C_G(t)$.

14. Future directions

14.1. Aschbacher reductions

Algorithms with better theoretical and practical performance are desirable for certain of the reductions based on Aschbacher's theorem. The existing algorithms sometimes have polynomial time complexity, subject to certain conditions not always satisfied. While usually their implementations run efficiently, they sometimes fail to complete.

14.2. Processing $O_p(G)$

It seems likely that any algorithm for processing $O_p(G)$ of a matrix group G defined over a field $\mathbb{F}_q$ of characteristic p has complexity $O(d^7 \log_p^3 q)$ in field operations over $\mathbb{F}_p$. Constructing a generating set for $O_p(G)$ remains a bottleneck, since its rank may be large. It may be possible to improve our treatment of $O_p(G)$ by first constructing a normal series for this group, where the corresponding factors consist of rectangular blocks arising from a composition series for the natural module for G, and then applying representation theory to these blocks to improve the way in which they are processed.

14.3. Subgroups and factor groups

Further developments will improve the functionality of the package. For example, subgroups, normal subgroups and quotient groups should be built into the package. If $H \leqslant G$ is defined by a generating set, and its relationship with G is to be ignored, then there is nothing to say. But in general this relationship should be respected. In this case the composition tree for H could be constructed by refining the composition tree for G. The reorganisation of the composition tree for G to refine a chief series passing through the soluble radical and other characteristic subgroups of G would induce a rearranged composition tree for H that would not, in general, pass through the soluble radical or other characteristic subgroups of H. A normal subgroup N of G could be similarly processed. Then N would cover certain chief factors of G, and avoid others. It would be possible to use the resultant data structure to compute in G/N. To combine the twin goals of having, on the one hand, a computational model for a finite group G as being defined by a chief series that passes through the soluble radical and other characteristic subgroups of G, and on the other hand having a model that passes readily to subgroups and quotient groups, would require complex data structures. However, the fact that we have a platform from which one can consider computing in quotients of a matrix group may be of interest.

References

Ambrose, S., Murray, S., Praeger, C.E., Schneider, C., 2011. Constructive membership testing in black-box classical groups. In: Proceedings of the Third International Congress on Mathematical Software. In: Lect. Notes Comput. Sci., vol. 6327, pp. 54–57.

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

Bäärnhielm, Henrik, 2006. Recognising the Suzuki groups in their natural representations. J. Algebra 300 (1), 171–198.

Bäärnhielm, Henrik, 2007. Algorithmic problems in twisted groups of Lie type. PhD thesis. Queen Mary University of London.

Bäärnhielm, Henrik, 2014. Recognising the Ree groups in their natural representations. J. Algebra 416, 139–166.

Babai, László, 1991. Local expansion of vertex-transitive graphs and random generation in finite groups. In: STOC '91: Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing. ACM Press, New York, NY, USA, pp. 164–174.

Babai, László, Beals, Robert, 1999. A polynomial-time theory of black box groups. I. In: Groups St. Andrews 1997 in Bath, I. In: Lond. Math. Soc. Lect. Note Ser., vol. 260. Cambridge Univ. Press, Cambridge, pp. 30–64.

Babai, László, Shalev, Aner, 2001. Recognizing simplicity of black-box groups and the frequency of p-singular elements in affine groups. In: Groups and Computation, III. Columbus, OH, 1999. In: Ohio State Univ. Math. Res. Inst. Publ., vol. 8. de Gruyter, Berlin, pp. 39–62.

Babai, László, Szemerédi, Endre, 1984. On the complexity of matrix group problems, I. In: Proc. 25th IEEE Sympos. Foundations Comp. Sci., pp. 229–240.

Babai, L., Goodman, A.J., Kantor, W.M., Luks, E.M., Pálfy, P.P., 1997. Short presentations for finite groups. J. Algebra 194, 79–112.

Babai, László, Kantor, William M., Pálfy, Péter P., Seress, Ákos, 2002. Black-box recognition of finite simple groups of Lie type by statistics of element orders. J. Group Theory 5 (4), 383–401.

Babai, László, Beals, Robert, Seress, Ákos, 2009. Polynomial-time theory of matrix groups. In: Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, pp. 55–64.

Borovik, Alexandre, Yalçınkaya, S., 2013. Fifty shades of black. http://arxiv.org/abs/1308.2487.

Bosma, Wieb, Cannon, John, Playoust, Catherine, 1997. The Magma algebra system. I. The user language. In: Computational Algebra and Number Theory. London, 1993, J. Symb. Comput. 24 (3–4), 235–265.

Bratus, Sergey, Pak, Igor, 2000. Fast constructive recognition of a black box group isomorphic to S_n or A_n using Goldbach's conjecture. J. Symb. Comput. 29 (1), 33–57.

Bray, John N., 2000. An improved method for generating the centralizer of an involution. Arch. Math. (Basel) 74 (4), 241–245.

Bray, John N., Bäärnhielm, Henrik, 2009. Black-box constructive recognition for the Suzuki groups. Preprint.

Bray, John, Conder, M.D.E., Leedham-Green, C.R., O'Brien, E.A., 2011. Short presentations for alternating and symmetric groups. Trans. Am. Math. Soc. 363, 3277–3285.

Brooksbank, Peter A., 2003. Fast constructive recognition of black-box unitary groups. LMS J. Comput. Math. 6, 162–197 (electronic).

Brooksbank, Peter A., 2008. Fast constructive recognition of black box symplectic groups. J. Algebra 320 (2), 885–909.

Brooksbank, Peter A., Kantor, William M., 2006. Fast constructive recognition of black box orthogonal groups. J. Algebra 300 (1), 256–288.

Brooksbank, Peter, Niemeyer, Alice C., Seress, Ákos, 2006. A reduction algorithm for matrix groups with an extraspecial normal subgroup. In: Finite Geometries, Groups, and Computation. Walter de Gruyter, Berlin, pp. 1–16.

Cameron, Peter J., Solomon, Ron, Turull, Alexandre, 1989. Chains of subgroups in symmetric groups. J. Algebra 127, 340–352.

Cannon, John J., 1973. Construction of defining relators for finite groups. Discrete Math. 5, 105–129.

Carlson, Jon F., Neunhöffer, Max, Roney-Dougal, Colva M., 2009. A polynomial-time reduction algorithm for groups of semilinear or subfield class. J. Algebra 322, 613–637.

Celler, Frank, Leedham-Green, C.R., 1997. Calculating the order of an invertible matrix. In: Groups and Computation, II. New Brunswick, NJ, 1995. In: DIMACS Ser. Discret. Math. Theor. Comput. Sci., vol. 28. Amer. Math. Soc., Providence, RI, pp. 55–60.

Celler, Frank, Leedham-Green, Charles R., Murray, Scott H., Niemeyer, Alice C., O'Brien, E.A., 1995. Generating random elements of a finite group. Commun. Algebra 23 (13), 4931–4948.

Cohen, Arjeh M., Murray, Scott H., Taylor, D.E., 2004. Computing in groups of Lie type. Math. Comput. 73 (247), 1477–1498.

Conder, M.D.E., Leedham-Green, C.R., O'Brien, E.A., 2006. Constructive recognition of $\mathrm{PSL}(2, q)$. Trans. Am. Math. Soc. 358 (3), 1203–1221.

Conway, J.H., Curtis, R.T., Norton, S.P., Parker, R.A., Wilson, R.A., 1985. Atlas of Finite Groups. Oxford University Press.

Cooperman, Gene, Finkelstein, Larry, 1993. Combinatorial tools for computational group theory. In: Groups and Computation. New Brunswick, NJ, 1991. In: DIMACS Ser. Discret. Math. Theor. Comput. Sci., vol. 11. Amer. Math. Soc., Providence, RI, pp. 53–86.

Corr, Brian, 2013. Estimation and computation with matrices over finite fields. PhD thesis. University of Western Australia.

Costi, Elliot, 2009. Constructive membership testing in classical groups. PhD thesis. Queen Mary, University of London.

Detinko, A.S., Flannery, D.L., O'Brien, E.A., 2011. Algorithms for the Tits alternative and related problems. J. Algebra 344, 397–406.

Detinko, A.S., Flannery, D.L., O'Brien, E.A., 2013. Recognizing finite matrix groups over infinite fields. J. Symb. Comput. 50, 100–109.

Dietrich, Heiko, Leedham-Green, C.R., Lübeck, Frank, O'Brien, E.A., 2013. Constructive recognition of classical groups in even characteristic. J. Algebra 391, 227–255.

Dietrich, Heiko, Leedham-Green, C.R., O'Brien, E.A., 2014. Effective black-box constructive recognition of classical groups. Preprint.

GAP Group 2014. GAP – Groups, Algorithms, and Programming, Version 4.7.4. http://www.gap-system.org.

Geddes, K.O., Czapor, S.R., Labahn, G., 1992. Algorithms for Computer Algebra. Kluwer Academic Publishers.

Glasby, S.P., Leedham-Green, C.R., O'Brien, E.A., 2006. Writing projective representations over subfields. J. Algebra 295 (1), 51–61.

Guralnick, R.M., Kantor, W.M., Kassabov, M., Lubotzky, A., 2008. Presentations of finite simple groups: a quantitative approach. J. Am. Math. Soc. 21 (3), 711–774.

Holmes, P.E., Linton, S.A., O'Brien, E.A., Ryba, A.J.E., Wilson, R.A., 2008. Constructive membership in black-box groups. J. Group Theory 11 (6), 747–763.

Holt, Derek F., Rees, Sarah, 1994. Testing modules for irreducibility. J. Aust. Math. Soc. A 57 (1), 1–16.

Holt, Derek F., Roney-Dougal, Colva M., 2013. Minimal and random generation of permutation and matrix groups. J. Algebra 387, 195–214.

Holt, Derek F., Stather, Mark J., 2008. Computing a chief series and the soluble radical of a matrix group over a finite field. LMS J. Comput. Math. 11, 223–251.

Holt, Derek F., Leedham-Green, C.R., O'Brien, E.A., Rees, Sarah, 1996a. Computing matrix group decompositions with respect to a normal subgroup. J. Algebra 184 (3), 818–838.

Holt, Derek F., Leedham-Green, C.R., O'Brien, E.A., Rees, Sarah, 1996b. Testing matrix groups for imprimitivity. J. Algebra 184, 795–817.

Holt, Derek F., Eick, Bettina, O'Brien, Eamonn A., 2005. Handbook of Computational Group Theory. Discrete Mathematics and its Applications. Chapman & Hall/CRC, Boca Raton, FL.

Howlett, R.B., Rylands, L.J., Taylor, D.E., 2001. Matrix generators for exceptional groups of Lie type. J. Symb. Comput. 31 (4), 429–445.

Hulpke, A., 2013. Computing conjugacy classes of elements in matrix groups. J. Algebra 387, 268–286.

Ivanyos, Gábor, Lux, Klaus, 2000. Treating the exceptional cases of the MeatAxe. Exp. Math. 9 (3), 373–381.

Jambor, Sebastian, Leuner, Martin, Niemeyer, Alice C., Plesken, Wilhelm, 2013. Fast recognition of alternating groups of unknown degree. J. Algebra 392, 315–335.

Johnson, D.L., 1990. Presentations of Groups. Lond. Math. Soc. Stud. Texts, vol. 15. Cambridge University Press, Cambridge.
Kantor, W.M., Kassabov, M., 2014. Black box groups isomorphic to $\mathrm{PGL}(2, 2^e)$. J. Algebra. In press.
Kantor, W.M., Magaard, K., 2013. Black box exceptional groups of Lie type. Trans. Am. Math. Soc. 365 (9), 4895–4931.
Kantor, William M., Seress, Ákos, 2001. Black box classical groups. Mem. Am. Math. Soc. 149 (708), viii+168.
Kantor, William M., Seress, Ákos, 2009. Large element orders and the characteristic of Lie-type simple groups. J. Algebra 322, 802–832.
Kleidman, Peter, Liebeck, Martin, 1990. The Subgroup Structure of the Finite Classical Groups. Lond. Math. Soc. Lect. Note Ser., vol. 129. Cambridge University Press, Cambridge.
Kovács, L.G., Robinson, Geoffrey R., 1991. Generating finite completely reducible linear groups. Proc. Am. Math. Soc. 112, 357–364.
Law, Maska, Niemeyer, Alice C., Praeger, Cheryl E., Seress, Ákos, 2006. A reduction algorithm for large-base primitive permutation groups. LMS J. Comput. Math. 9, 159–173 (electronic).
Leedham-Green, Charles R., 2001. The computational matrix group project. In: Groups and Computation, III. Columbus, OH, 1999. In: Ohio State Univ. Math. Res. Inst. Publ., vol. 8. de Gruyter, Berlin, pp. 229–247.
Leedham-Green, C.R., O'Brien, E.A., 1997. Recognising tensor products of matrix groups. Int. J. Algebra Comput. 7 (5), 541–559.
Leedham-Green, C.R., O'Brien, E.A., 2002. Recognising tensor-induced matrix groups. J. Algebra 253 (1), 14–30.
Leedham-Green, C.R., O'Brien, E.A., 2009. Constructive recognition of classical groups in odd characteristic. J. Algebra 322, 833–881.
Leedham-Green, C.R., O'Brien, E.A., 2014. Short presentation for the classical groups on their standard generators. Preprint.
Liebeck, Martin W., O'Brien, E.A., 2007. Finding the characteristic of a group of Lie type. J. Lond. Math. Soc. (2) 75 (3), 741–754.
Liebeck, Martin W., O'Brien, E.A., 2014. Recognition of finite exceptional groups of Lie type. Trans. Am. Math. Soc.
Lübeck, Frank, Müller, Jürgen. Challenge problems. www.math.rwth-aachen.de:8001/~Frank.Luebeck/data/MatrixChallenges/index.html.
Lübeck, F., Magaard, K., O'Brien, E.A., 2007. Constructive recognition of $\mathrm{SL}_3(q)$. J. Algebra 316 (2), 619–633.
Lubotzky, Alexander, 2002. The expected number of random elements to generate a finite group. J. Algebra 257, 452–459.
Magaard, Kay, O'Brien, E.A., Seress, Ákos, 2008. Recognition of small dimensional representations of general linear groups. J. Aust. Math. Soc. 85, 229–250.
Murray, Scott H., Roney-Dougal, Colva M., 2011. Constructive homomorphisms for classical groups. J. Symb. Comput. 46, 371–384.
Neunhöffer, Max, 2009. Constructive recognition of finite groups. Habilitationsschrift, RWTH Aachen.
Neunhöffer, Max, Praeger, Cheryl E., 2008. Computing minimal polynomials of matrices. LMS J. Comput. Math. 11, 252–279.
Neunhöffer, Max, Seress, Ákos, 2006. A data structure for a uniform approach to computations with finite groups. In: ISSAC 2006. ACM, New York, pp. 254–261.
Neunhöffer, Max, Seress, Ákos. recogbase – a framework for constructive recognition. neunhoef.github.io/recogbase/.
Neunhöffer, Max, Seress, Ákos, et al. recog – methods for constructive recognition. neunhoef.github.io/recog/.
Niemeyer, Alice C., 2005. Constructive recognition of normalizers of small extra-special matrix groups. Int. J. Algebra Comput. 15 (2), 367–394.
Niemeyer, Alice C., Praeger, Cheryl E., 1998. A recognition algorithm for classical groups over finite fields. Proc. Lond. Math. Soc. (3) 77 (1), 117–169.
Niemeyer, Alice C., Praeger, Cheryl E., 1999. A recognition algorithm for non-generic classical groups over finite fields. J. Aust. Math. Soc. A 67 (2), 223–253.
O'Brien, E.A., 2006. Towards effective algorithms for linear groups. In: Finite Geometries, Groups, and Computation. Walter de Gruyter, Berlin, pp. 163–190.
O'Brien, E.A., 2011. Algorithms for matrix groups. In: Quick, Martyn, Roney-Dougal, Colva (Eds.), Groups. St. Andrews. 2009. In: Lecture Notes Lond. Math. Soc., vol. 388. Cambridge University Press, pp. 297–323.
Pak, Igor, 2000. The product replacement algorithm is polynomial. In: FOCS '00: Proceedings of the 41st Annual Symposium on Foundations of Computer Science. IEEE Computer Society, Washington, DC, USA, pp. 476–485.
Parker, R.A., 1984. The computer calculation of modular characters (the meat-axe). In: Computational Group Theory. Durham, 1982. Academic Press, London, pp. 267–274.
Seress, Ákos, 2003. Permutation Group Algorithms. Camb. Tracts Math., vol. 152. Cambridge University Press, Cambridge.
Shparlinski, Igor E., 1999. Finite Fields: Theory and Computation. The Meeting Point of Number Theory, Computer Science, Coding Theory and Cryptography. Math. Appl., vol. 477. Kluwer Academic Publishers, Dordrecht.
Solomon, Ron, Turull, Alexandre, 1991. Chains of subgroups in groups of Lie type. III. J. Lond. Math. Soc. 44, 437–444.
Stather, Mark James, 2006. Algorithms for computing with finite matrix groups. PhD thesis. University of Warwick.
Stather, Mark, 2007. Constructive Sylow theorems for the classical groups. J. Algebra 316, 536–559.
Wilson, Robert A., 1996. Standard generators for sporadic simple groups. J. Algebra 184 (2), 505–515.
Wilson, R.A., et al. Atlas of Finite Group Representations. brauer.maths.qmul.ac.uk/Atlas.