Constructive Recognition of Finite Groups

Von der Fakultät für

Mathematik, Informatik und Naturwissenschaften

der Rheinisch-Westfälischen Technischen Hochschule Aachen

akzeptierte Habilitationsschrift

zur Erlangung der Venia legendi

von

Dr. rer. nat. Max Neunhöffer

geboren in Heidelberg

November 2009

Preface

The Goal

The mathematical area of the present book is computational group theory. The ultimate goal of this theory is to be able to do computations in and with groups. Traditionally, there are several ways to implement groups on computers. One way is to work with finitely presented groups, that is by specifying generators and relations. Another is to use group actions to actually store, multiply, invert and compare group elements in groups. Using actions on sets leads to the study of permutation groups, using linear actions leads to the study of matrix groups and using projective actions leads to the study of projective groups, by which we mean groups in which the elements are invertible matrices modulo scalars. Considering other actions to represent groups on a computer is possible (see for example [Koh08]) but up to now not studied much.

For finite permutation groups there are highly efficient algorithms to compute the order of a group, a composition series, centralisers of elements, stabilisers, Sylow p-subgroups, to find normal subgroups, maximal subgroups, group homomorphisms, to test membership of group elements and the like. Standard techniques use stabiliser chains, base and strong generating set methods, for a good account of the known methods see [Ser03]. These algorithms are implemented in computer algebra systems like GAP (see [GAP07]) and MAGMA (see [BC07]) and the status of this area can justifiably be called satisfactory.

A few interesting experiences have been made from the study of permutation group algorithms. One is that complexity theory, which is the study of the asymptotic behaviour of the runtime of algorithms as the problem size tends to infinity[1], has been a very important tool to devise efficient algorithms and to implement them. In particular, worst-case analysis of the complexity of permutation group algorithms has shown that it is in practice worthwhile to first exclude a few nasty cases — for which special methods apply — before applying standard methods, which would be disastrously slow in these special cases. Large base groups (see Section V.5.3) are an example for such nasty cases for stabiliser chain methods. Another interesting experience from the study of algorithms for finite permutation groups is that randomised algorithms, followed by some kind of deterministic verification of the results, have been very successful.

The situation for matrix groups over finite fields, which are subgroups of some general linear group $\mathrm{GL}(n, q)$ — the group of all invertible $n \times n$-matrices with entries in the finite field $\mathbb{F}_q$ with q elements with matrix multiplication as product — is less satisfactory. Roughly speaking, if one enters a few elements $g_1, \ldots, g_k$ of some $\mathrm{GL}(n, q)$ into a computer algebra system like GAP or

[1] For a brief discussion see Section I.1.

MAGMA, it is in general quite limited what these systems can tell you about the subgroup of $\mathrm{GL}(n, q)$ that is generated by the elements g_i.
So the ultimate goal in this subarea of computational group theory is to be able to do as much as possible with these matrix groups over finite fields, guided by what is already possible efficiently for finite permutation groups. The general idea is to use the complexity analysis of algorithms as a tool to come up with "efficient" algorithms. Generally, an algorithm is considered to be "good" if its runtime is bounded from above by a polynomial in the size of the input.
A first step in this direction is set out by the so-called matrix group recognition project (see [LG01] and [O'B06]). This project came to live after the seminal paper [NP92] by Peter Neumann and Cheryl Praeger in 1992, in which they present an algorithm which decides whether or not a group $G = \langle g_1, \ldots, g_k \rangle \leq \mathrm{GL}(n, q)$ given by a set of generators contains the special linear group $\mathrm{SL}(n, q)$. This algorithm answered a question posed by Joachim Neubüser at a workshop in Oberwolfach about computational group theory. The matrix group recognition project in the meantime has become an ongoing collaborative effort of many researchers, and its aim is to devise efficient algorithms to perform the constructive recognition of a matrix group $G \leq \mathrm{GL}(n, q)$ which is given by a set of generators. With the term "constructive recognition" we mean

- to find the group order $|G|$, and
- to set up a procedure to
  - test whether an element $g \in \mathrm{GL}(n, q)$ is contained in G, and if so,
  - express g explicitly as a product in the given generators.[2]

One good reason, why it is sensible to solve this problem for matrix groups first, is that the extremely successful method of stabiliser chains and bases and strong generating sets solves exactly this problem for permutation groups and is one of the fundamental algorithms for them. Therefore it is conceivable that an efficient solution to the constructive recognition problem for matrix groups will provide a similar fundament for the solution of further computational problems for matrix groups.
The basic approach for the constructive recognition of matrix groups is to compute a so-called composition tree for G. The nodes of this binary tree are groups together with a homomorphism to another group, the two descendants of a node are the kernel and image of this homomorphism. The leaves of the tree are simple groups, in which the constructive recognition problem has to be solved by other means, and the root of the tree is G itself. The infrastructure of the tree is organised such that a solution to the constructive recognition problem for G can be put together by recursively traversing the tree and finally using the solutions to the constructive recognition problems in the leaves. This description of the composition tree here is necessarily very rough, for details see [LG01] or [O'B06] or Section V.2.

This Book

The matrix group recognition project has produced many results but is not yet fully completed. This book gives an overview of the current state of the art and shows some contributions of the

[2]For more details on the constructive recognition problem see Section V.1.

author. The collaborative nature of the whole project lends itself to joint papers by more than one person. Therefore, some of the author's work in the context of this project, which can be found in Chapters III, V and VII, are in fact collaborations with other authors.

A second major part of the contributions of the author to the project lies in actual implementations. Together with Ákos Seress we have started an implementation of the best known algorithms for group recognition in the GAP computer algebra system (see [GAP07]). This implementation comes in the form of two GAP packages recogbase (see [NS08]) and recog (see [BLN$^+$08]) which are due to be published soon. The first provides a generic framework to implement composition trees in GAP for arbitrary types of groups. One interesting feature of the generic framework is that within a single composition tree there can be groups in different representations, that is, there can be a mixture of permutation groups, matrix groups and projective groups. The second package tries to collect the best known methods for group recognition for each of these types of groups. Thus, everybody who contributes implementations will be an author of the recog package.

This book describes a major part of the algorithms used in both packages. For some methods, mainly for the leaves of the composition tree, the reader is however referred to the literature.

Since the matrix group recognition project is not yet finished, the present work cannot give a complete description of a solution to the constructive recognition problem. In particular for the leaves of the composition tree a lot of work and improvement of algorithms still has to be done, as is explained in some detail in Chapter VIII.

We have intentionally left out a number of topics which are related, mostly because research on them is not completed or not even in a state for a satisfactory description. One is the whole field of algorithms for matrix groups that build on constructive recognition. Derek Holt and Mark Stather have recently published a paper [HS08] pointing in that direction and Mark Stather's PhD thesis [Sta06] and [Sta07] also lie in this area. Another topic left out is the constructive recognition of black box groups, because we wanted to concentrate on matrix groups and projective groups. We only cover the verification phase very briefly, although this phase is necessary to have computational proofs of the results. The reason for this is that in fact very little work has been done on the actual implementation of verification routines. What is needed for this is basically good presentations for the groups occurring in the leaves. Finally, a global analysis of the whole procedure of building a composition tree is still needed. This seems achievable using estimates on the length of a composition series but is not done yet.

Now we turn to the contents of this book and outline its structure.

Chapter I introduces briefly some concepts from computer science, which apply to computational group theory, and describes a way to produce nearly uniformly distributed random elements in a finite group. Chapter II introduces a new method to implement matrices over finite fields on a computer. Having an efficient implementation of the finite field arithmetic and linear algebra routines is obviously an indispensable foundation for implementing matrix group algorithms. The contents of this chapter are not yet published elsewhere.

Chapter III contains a new randomised method to compute the minimal polynomial of a square matrix over a finite field. It is basically a copy of the paper [NP08] jointly written by Cheryl Praeger and the author. Computing minimal polynomials of invertible matrices is an important ingredient to compute the order and projective order of such matrices, which are computations that are used throughout nearly all matrix group algorithms.

Chapter IV completes the description of the infrastructure for implementing matrix group algorithms by explaining how to compute the order and projective order of a matrix and how to perform higher level linear algebra computations like solving systems of linear equations and inverting matrices. In addition the two major obstacles for polynomial time algorithms, the discrete logarithm problem and integer factorisation, are introduced. This chapter is nothing new but is needed for the sake of completeness.

In Chapter V we explain the basic problem attacked by the matrix group recognition project, namely the constructive recognition problem. We give a gentle introduction starting with a rough formulation of the problem followed by two refinements. Then the fundamental approach using composition trees and a generic framework for group recognition are described. We explain in detail the idea and purpose of reduction homomorphisms. The contents of this chapter are a variation and extension of the paper [NS06] by Ákos Seress and the author, in which the composition tree approach is refined by allowing for a change in the generating set of the group to be recognised. This refinement can dramatically increase the performance, because the resulting straight line programs are much shorter than in the traditional version. Finally, the chapter closes with a description how the currently best known algorithms for the constructive recognition of permutation groups fit in nicely with the proposed framework.

The next Chapter VI explains a variant of Aschbacher's theorem on subgroups of classical groups (see [Asc84]) restricted to the general linear group case. A relatively short complete proof is given. Our variant changes the definition of the occurring classes of subgroups slightly to make it easier to devise algorithms for finding reduction homomorphisms for groups in some of these classes. This approach already seems to bear fruit in the last part of the chapter, where we present an overview of algorithms to find reduction homomorphisms for groups in the different classes. For our classes $\mathcal{D}_2$, $\mathcal{D}_4$ and $\mathcal{D}_7$ new ideas to tackle groups in these classes are given as well as references to the literature for the currently best known methods.

Chapter VII is a copy of the preprint [CNRD08] which is joint work of the author with Jon Carlson and Colva Roney-Dougal. It presents new completely analysed randomised algorithms to find a reduction for the case that $G \leq \mathrm{GL}(n, q)$ acts irreducibly on its natural module and lies in at least one of the semilinear or subfield Aschbacher classes $\mathcal{C}_3$ and $\mathcal{C}_5$.

Chapter VIII finally tries to summarise the state of the art for algorithms to do constructive recognition for the leaves of the composition tree, that is for groups in the Aschbacher classes $\mathcal{C}_8$ and $\mathcal{C}_9$. We do not try to explain any of the best known methods there, because the final word on them seems not to be spoken at the time of this writing. Rather, we explain the concepts of non-constructive recognition and standard generators and give references to the literature.

Acknowledgements

First and foremost I would like to thank Gerhard Hiß for a multitude of things. He was not only my PhD supervisor but also gave me the opportunity to be employed at the RWTH Aachen for 10 years in research and teaching positions. He holds the chair of Lehrstuhl D für Mathematik and thereby provided a superb environment for me to develop from being a student of mathematics to be a professional mathematician. I greatly enjoyed my time in Aachen and Gerhard Hiß has always been a great advisor and good friend. I also thank the RWTH Aachen and in particular all

the people at Lehrstuhl D for the wonderful time I had in Aachen.
I thank Alice Niemeyer and Cheryl Praeger for inviting me repeatedly to the University of Western Australia in Perth for research visits. During one of these visits I met Ákos Seress, whom I thank for motivating me to start working in the area of computational matrix group theory, for successfully collaborating with me over several years and for my trips to the Ohio State University in Columbus.
Furthermore I thank my coauthors Jon Carlson, Cheryl Praeger, Colva Roney-Dougal and Ákos Seress for the interesting and fruitful collaboration on a major part of the work that went into the present book.
I thank the University of St Andrews and in particular Steve Linton for employing me since 2007 and giving me the opportunity to complete the work on this book as part of my research job. I am grateful to all the people in both the Mathematics and Computer Science departments, because they integrated me very nicely and made it easy for me to feel well and start joint work.
A big thank-you also goes to Colva Roney-Dougal who helped a lot in proofreading, agreed to constantly correct my English and motivated me to apply for my position in St Andrews.
I wholeheartedly thank my parents for their permanent support and for making it possible for me to enjoy a broad and deep education in mathematics, which gave me the opportunity to get into my dream job.
Last but not least I would like to thank my wonderful wife Anja for her constant support and encouragement during the completion of this book. I also thank her for moving to Scotland, such that we can live together in this beautiful and relaxed country.

Contents

Chapter I

Introduction

This chapter covers a few fundamental concepts and algorithms which will be used in the rest of the book. We start talking about the complexity of algorithms, go on with the concept of straight line programs in groups and finish by mentioning the idea of randomisation in algorithms and a way to produce nearly uniformly distributed elements in a finite group.

I.1 Some notes on complexity theory

Already in this chapter, but all the more in the rest of the book, we talk about the analysis of algorithms. In this section we want to discuss briefly what we mean by this at all.

An algorithm is usually designed to solve a whole family of problems of different sizes. Obviously, for a single, particular instance of a computational problem one can simply store the answer and look it up when needed in nearly no time. But this is usually not what we intend to do when we develop an algorithm. Furthermore, it is clear that a small instance of a computational problem usually will need less time to solve than a big instance of the same problem. It takes for example longer to multiply two 10000×10000-matrices with entries in the finite field $\mathbb{F}_q$ than to multiply two 100×100-matrices with entries in the same finite field $\mathbb{F}_q$, even if these two situations are instances of the same computational problem "matrix multiplication over $\mathbb{F}_q$".

Therefore, when we talk about analysing an algorithm that solves a certain family of computational problems, we assign each instance of this problem a "size" and ask how the runtime of the algorithm grows in comparison to the size of different instances. This is in vague terms what is meant by "complexity of an algorithm", and "complexity theory" is the study of the complexity of algorithms. The size of an instance of the above mentioned matrix multiplication problem of two $n \times n$-matrices could for example be measured by the value n.

Of course, different computers are running at different speeds, so what we usually do is to count the number of steps in the algorithm needed to complete a particular instance of the computational problem as a function of the size of the problem instance. We try to keep the different steps comparable. We might for example count the number of elementary field operations (addition, subtraction, multiplication, inversion) during the execution of a matrix multiplication. The standard simple-minded approach to matrix multiplication would then need $(2n - 1) \cdot n^2$ elementary field operations since each entry of the result is a sum of n products of numbers in $\mathbb{F}_q$, that is, we need n products and $n - 1$ additions for each of the n^2 matrix entries. The complexity of this algorithm in terms of the size n of the input would then be $(2n - 1) \cdot n^2$ elementary field operations.

If the growth rate of the complexity of an algorithm A is slower than the one for another algorithm B solving the same problem, then we would consider A to be "better" than B. Note that it is possible that the complexity of B is in fact smaller than the one of A for small problem sizes. In that case we might want to implement both algorithms and use B for smaller instances and A for bigger instances. In this way complexity theory helps to come up with better implementations. If we had for example an algorithm B to multiply two $n \times n$-matrices over $\mathbb{F}_q$ using $n^4/20$ elementary field operations, it would be worthwhile to use it as opposed to the standard algorithm A for small matrices, since $n^4/20 \leq (2n - 1) \cdot n^2$ for $n \leq 39$. However, for big matrices A would outperform B dramatically.

Counting the steps in an algorithm can be very tedious and much more difficult if there are decisions and case distinctions during the execution. Randomisation as described in Section I.3 makes this even more difficult. Therefore we are usually happy to come up with an upper bound for the number of steps necessary. In addition, when we are only interested in the growth rate of the complexity, we are only interested in the type of function expressing this upper bound in terms of the problem size and are actually not interested in the constants.

So, to determine that the growth rate of the complexity of the example algorithm A above is smaller than the one of B, we do not need the exact number of steps $(2n - 1) \cdot n^2$ and $n^4/20$ respectively, but we only need to know that the first behaves "like n^3" and the second "like n^4". The higher exponent 4 in the second function eventually beats the smaller constant $1/20$. Note however that without knowing the constants we would in fact not know that the break-even point of where we should start using A in favour of B is at $n = 39$. Having noticed this, we would like to comment that knowing the constants in the complexity of algorithms can be very interesting for coming up with good implementations.

Different parts of this book go differently about this. Whenever possible we have tried to include constants in the complexity estimates. However, sometimes this would have been too tedious and would have complicated things dramatically. In these cases we are content with determining the growth rate. For this purpose, we adopt the standard big-O notations, which we will briefly repeat here. It is used to formulate statements about the asymptotic behaviour of functions.

I.1.1 Definition (Capital-O-notation)
Let $\mathbb{R}^+$ be the set of positive real numbers and $g : \mathbb{R}^+ \to \mathbb{R}$. We say that a function $f : \mathbb{R}^+ \to \mathbb{R}$ is $O(g)$ if there are two positive real constants C and D such that $|f(x)| \leq C \cdot |g(x)|$ for all $x > D$.

This will be used in the analysis of algorithms by saying for example: Algorithm A above has complexity $O(n^3)$ whereas algorithm B above has complexity $O(n^4)$. Obviously it has to be clear from the context, which family of computational problems the algorithm deals with and how the size n of an instance of the problem is measured.

There is a certain sloppiness in this usage of Definition I.1.1. Strictly speaking the following statement is true as well: "The function $n \mapsto (2n - 1) \cdot n^2$ is $O(n^5)$." This follows from the fact that the function $n \mapsto n^3$ is $O(n^5)$. However, this statement is not as strong as possible. Obviously, we are interested in the fact that the exponent 3 is the *least possible e* for which the

statement "the complexity of algorithm A is $O(n^e)$" is true. Sometimes we can prove that our statement is best possible and sometimes we cannot.
Up to now we have concentrated on the "time complexity" of algorithms, that is, the asymptotic behaviour of the runtime with growing problem sizes. Although this is usually the most interesting aspect, it is of course also necessary to keep an eye on other computational resources like memory usage. We speak of "space complexity" in this case and use a similar setup and language. This only plays a minor role in this book.
The point of view we adopt in this book is that the complexity analysis of algorithms is a tool to come up with good implementations. Complexity results tell us, which algorithms are suited best for which problem classes and possibly which problem sizes, and they tell us, what is considered to be a "satisfactory" algorithm for a problem class and what is considered to be "lacking". In general we would like to devise algorithms which have a polynomial as an asymptotic bound of their time complexity in terms of the problem size. We call these algorithms "polynomial-time algorithms".

I.2 Straight line programs

Let G be a group that is given by a tuple of generators $(g_1, \ldots, g_k)$. That is, every element in G can be expressed as a finite product of powers of the g_i and their inverses. However, since G is not necessarily abelian, the generators may occur more than once and the products can be quite long. We call such a product a *word in the generators* $(g_1, \ldots, g_k)$. For finite groups we do not need the inverses of the generators since they all have finite order and inverses can thus be expressed by positive powers.
Quite often in applications we want to encode rather long words in generators on a computer. One reason for this is that we want to store certain elements in a known group in terms of a generating tuple (see in particular Section VIII.3). Another reason is for example that to evaluate a group homomorphism $\varphi : G \to H$ on arbitrary elements $g \in G$, if we only know the images $\varphi(g_i)$ of a tuple of generators for G, we need to express g explicitly in terms of the generators $(g_1, \ldots, g_k)$. Finally, the problem of constructive recognition (see Problem V.2.1) involves sometimes rather long words in a tuple of generators, too.
To store and evaluate such words efficiently is the purpose of *straight line programs (SLP)*.
In general, straight line programs are programs that have no branches or loops, their execution follows a "straight line" under all circumstances. For the purpose of storing and evaluating words in generators in a group, we further restrict this to the following:

I.2.1 Definition (Straight Line Program (SLP))
A *straight line program* is a procedure that takes as input a k-tuple $(g_1, \ldots, g_k)$ of group elements in a group and consists of a finite sequence of steps, which are each one of the following:

- Compute the product of two stored elements, or
- compute the inverse of a stored group element.

An SLP starts with the elements $g_1, \ldots, g_k$, stores all intermediate results and returns one or more of the group elements collected during its execution. The number of steps in the SLP is called its *length*.

I.2.2 Remark
A straight line program of length l can encode words in its input of length up to 2^l. Obviously, it cannot encode all words of that length.

Implementations and data structures for straight line programs are available in the GAP (see [GAP07]) and MAGMA (see [BC07]) computer algebra systems. The WWW-Atlas of group representations uses straight line programs to store generators for maximal subgroups of groups. Note that the current implementations of straight line programs in these systems provide an even more compact storage by allowing arbitrary finite products of powers of previously stored group elements in each step.

I.3 Randomised algorithms

Traditionally, an algorithm is a completely deterministic procedure to achieve a certain goal. Whenever it is executed, it performs the same steps and thus behaves in the same way when called twice with the same input.
However, there is a certain limitation in this paradigm. In particular in situations, in which we want to find some result that can later be verified to be correct easily, randomised methods excel. By randomised methods we mean algorithms that involve a certain amount of random choices. That is, the sequence of steps performed by a randomised algorithm depends on certain random choices done during the algorithm. Of course, in practice we will usually use pseudo random numbers to do these random choices.
We do not want to go into too much detail here, but there are many examples in which randomised algorithms can greatly outperform deterministic algorithms. However, how do we measure or analyse the performance of a randomised algorithm, given that it does different things on different calls with the same input, and thus has different runtimes on different occasions?
One possibility for performance analysis is to look at the average or the expected runtime. Although this is a good and interesting thing to look at, this type of analysis often stays a bit unsatisfactory, since one never knows, how long the algorithm will take at most in a particular instance.
Therefore the most common approach for randomised algorithms is to do a worst-case analysis. However, clearly the absolutely worst case is that by incredible bad luck all random choices turn out to be wrong and the algorithm does not succeed even after a very long time. To get rid of this problem we have to allow our algorithms to fail in some way, most commonly simply by giving up with FAIL as answer after a certain time. Using this exit route, we can devise algorithms that are guaranteed to terminate after a certain number of steps or a given amount of time. To be useful in practice, we of course want to have a bound for the probability with which this failure occurs. Optimally, we want to prescribe an upper bound for the failure probability beforehand. The guaranteed upper bound for the runtime then might depend on the choice of the failure probability bound.
In general we distinguish between so-called "Monte Carlo" and "Las Vegas" algorithms as defined in the following.

I.3.1 Definition (Monte Carlo algorithm)
A *Monte Carlo algorithm with failure probability* ϵ is a randomised algorithm that is guaranteed to terminate after a finite amount of time with some result, such that the probability of returning a wrong result is bounded by ϵ.

A bit more satisfying is the following.

I.3.2 Definition (Las Vegas algorithm)
A *Las Vegas algorithm with failure probability* ϵ is a randomised algorithm that is guaranteed to terminate after a finite amount of time with either the correct result or FAIL indicating failure, such that the probability of failure is bounded by ϵ.

The two concepts are sometimes related by the following.

I.3.3 Remark (Upgrading Monte Carlo to Las Vegas by verification)
Assume that there is an efficient way to verify the correctness of the output of a Monte Carlo algorithm. Then we can immediately upgrade the algorithm to be of Las Vegas type by following it with a verification step that returns FAIL, if the result was incorrect in the first place. "Efficient" here means that the verification does not take much longer than the Monte Carlo computation in the first place.

We will use the terms "Monte Carlo algorithm" and "Las Vegas algorithm" in this sense throughout this book.

I.4 Random elements in finite groups

Randomised algorithms in group theory need random elements in groups. Moreover, to allow a proper analysis of the behaviour of such algorithms one needs to know quite a lot about the distribution of the random elements in the group. Usually, the best with respect to analysis is to have a source of uniformly distributed random elements.
For most applications we are content with pseudo randomness, that is, with a deterministic procedure which produces from some initial seed a sequence of elements with a good uniform distribution. Choosing different seeds (or maybe actually choosing the seed at random) then leads to a different behaviour of the algorithm in each call.
There are well-known methods to produce good uniformly distributed pseudo random numbers (see for example [Knu98, Chapter 3]). Building on these, there is a method to produce pseudo random elements in a finite group given by generators, which works astonishingly well in the sense that the distribution of the elements is very close to uniform. In the sequel we describe this method briefly but refer for proofs to the literature. After this we discuss some of the advantages and limitations of this method.

I.4.1 Definition (Random elements with RATTLE)
Let G be a group given by a tuple $(g_1, \ldots, g_k)$ of generators and $n, N \in \mathbb{N}$ with $n \geq k$. The RATTLE method to produce random elements in G is the following procedure:
It uses a variable $(a, (h_1, \ldots, h_n)) \in G \times G^n$ which is changed during the runtime by calls to Algorithm 1.
It first initialises $(a, (h_1, \ldots, h_n))$ by $a := \mathbf{1}_G$ and $h_i := g_i$ for $1 \leq i \leq k$ and $h_i := \mathbf{1}_G$ for $k < i \leq n$.
Then it calls Algorithm 1 repeatedly N times with $(a, (h_1, \ldots, h_n))$ as argument thereby changing it and finally returns the last value of a as a random element a_0 in G.

Algorithm 1 RATTLESTEP

```
Input: A pair $(a, (h_1, \dots, h_n))$ with $a \in G$ and $G = \langle h_1, \dots, h_n \rangle$.
Output: A modified pair $(a, (h_1, \dots, h_n))$ with $a \in G$ and $G = \langle h_1, \dots, h_n \rangle$.

$i := \text{RANDOM}(\{1, 2, \dots, n\})$
$j := \text{RANDOM}(\{1, 2, \dots, n-1\})$
$b := \text{RANDOM}(\{1, 2\})$
if $j = i$ then
    $j := j + 1$
end if
if $b = 1$ then
    $h_i := h_i \cdot h_j$
else
    $h_i := h_j \cdot h_i$
end if
$a := a \cdot h_i$
Return modified $(a, (h_1, \dots, h_n))$
```

After this initialisation phase it produces a sequence of random elements $(a_i)_{i \in \mathbb{N}}$ by calling Algorithm 1 repeatedly with $(a, (h_1, \dots, h_n))$ as argument thereby changing it and assigning the value of a to a_i after call number i.

I.4.2 Remark (Variant of product replacement)
The RATTLE method described above is a variant of the "product replacement algorithm", because the main part of a step replaces an element by a product of it with another one.

I.4.3 Remark (Comments on the implementation of RATTLE)
It is not completely clear how to choose the parameters n and N. In principal n could be chosen equal to k. However, what a good length N of the initialisation phase is depends on the group G and on the generating tuple $(g_1, \dots, g_k)$. In practice one chooses n slightly bigger than k and N around 100 unless k is very big. For large k the value of N has to be chosen bigger. It is clear that a constant value for N will not work well for $|G| \to \infty$.

I.4.4 Proposition (*RATTLE *converges to the uniform distribution)
The distribution of the element a_0 in the RATTLE *procedure (see Definition I.4.1) converges for $N \to \infty$ to the uniform distribution.*

Proof: See [LGM02, Section 4]. □

A brief discussion of RATTLE

Although it can be proved that for every finite group G and every generating tuple $(g_1, \dots, g_k)$ the distribution of the element a_0 for $N \to \infty$ tends to the uniform distribution (see Proposition I.4.4), there is not much known about the rate of convergence. So, picking the value for N in practice is difficult, in particular if we do not know anything about G.

If we use the sequence produced by the RATTLE method after initialisation as a sequence of random elements in G, every single element a_i is nearly uniformly distributed (as a_0 is by Proposition I.4.4). However, adjacent elements in the sequence are clearly not distributed independently. Obviously the next element in the sequence depends heavily on the previous state $(a, (h_1, \dots, h_n))$ and the previous element is equal to the a value of this state. On the other hand, the state contains of course more information than simply the value a.

Despite these obvious deficiencies, the algorithm works surprisingly well in practice. The computational cost for the initialisation is 200 multiplications and after that 2 more multiplications for every further element in the sequence. The memory requirements are minimal and the produced sequence of random elements is good enough for most purposes. Analysing algorithms that use RATTLE with the assumption that it produces uniformly distributed random elements in the group provides good predictions on how well these algorithms work in practice.

Throughout this book the RATTLE method is used to produce random elements in a group and the above assumption is made.

Chapter II

Matrices over finite fields

This chapter covers the implementation of the basic operations for matrices over finite fields. We begin with a description of a new concrete representation of such matrices on nowadays computers in Section II.1, both in main memory and on storage. Then we analyse the performance and complexity of matrix arithmetic for this new representation in Section II.2 and compare it to previous implementations.
We conclude our presentation of fundamental matrix algorithms in the next Chapter III in which we present a method to compute the characteristic polynomial of matrices over finite fields and a new method for the minimal polynomial, which is used to compute projective orders of matrices as will be shown in Chapter IV.

II.1 Representing vectors and matrices over finite fields

If you already know something about compact vectors and matrices over finite fields you can safely skip the next subsection and directly proceed to Section II.(1.2). First we explain the basic idea.

(1.1) The idea

If p is a prime then elements of the finite field $\mathbb{F}_p$ with p elements can be represented by the integers $0, 1, \dots, p-1$. Thus, storing one such element on a computer needs only $\lceil \log_2(p) \rceil$ bits. The finite extension field $\mathbb{F}_q$ with $q = p^k$ elements is built as quotient $\mathbb{F}_p[x]/c_k\mathbb{F}_p[x]$ using the Conway polynomial c_k (see [Nic88] or on the web at [Lüb01]). Since the Conway polynomials are monic by definition, an element of $\mathbb{F}_q$ can be represented by a polynomial over $\mathbb{F}_p$ of degree less than k and thus by storing k elements of $\mathbb{F}_p$ using $\lceil \log_2(p^k) \rceil$ bits.
Since linear algebra operations over finite fields can be performed rather efficiently by modern microprocessors, the limiting factor for such computations is the memory access. Therefore it is performance critical to store vectors and matrices using as little memory as possible and to choose the data structure in a way that allows for fast memory access. We call such memory efficient data structures “compact”.
This idea is quite old (see for example [Par84]) and has already been used extensively (see for example [Rin94] or [GAP07] or various other systems).

There is a fundamental difference between the characteristic $p = 2$ case and all other characteristics. The reason for this is that in characteristic 2 the addition of vectors can be implemented by using the XOR (exclusive or) operation available in every microprocessor instruction set. In odd characteristic, the available instructions do not fit so well to finite field arithmetic. Therefore previous implementations mostly rely on table lookups to perform arithmetic operations. Since the memory space available for lookup tables is limited, this method is limited to relatively small fields (usually up to fields with 256 elements) and has to use single byte accesses as opposed to processor word accesses, for which modern machines seem to be optimised. These limitations seem to become more serious as word lengths in standard microprocessors increase.

The main novelty in the approach presented here is to overcome this problem by choosing the data structures in a way that allows to use processor word operations for all arithmetic operations in all characteristics. A word operation is an arithmetical operation like an addition, which a microprocessor can perform on two integers each stored in a machine word in a single step.

Also this idea has been used before (see [FMR+95]), however, no implementation of this seems to be available any more and, more importantly, the approach still insisted on storing whole elements of $\mathbb{F}_q$ in one machine word. We will pack as many prime field elements as possible into a machine word and distribute the various prime field coefficients of a single element of $\mathbb{F}_q$ into distinct machine words to allow for better memory usage in the case of extension fields.

(1.2) Compact vectors

First we describe in detail how a vector of length n over the field $\mathbb{F}_q$ with $q = p^k$ elements is stored on a machine with word length 32 or 64 bits respectively. The ideas for the step from 32 to 64 bits can be applied analogously for future microprocessors with even larger word length. We ignore architectures with funny word lengths not being a multiple of 32 bits.

Let e be $\lceil \log_2(2p - 1) \rceil$ for $p > 2$ and 1 for $p = 2$. This is the number of bits necessary to store an integer in the range $0, 1, \dots, 2p - 2$ except for $p = 2$. This number e is the number of bits we reserve for every prime field element we want to store. For $p = 2$ this is evidently best possible, whereas for odd p we seem to waste one bit per prime field element, since we seem to need only to store numbers in the range $0, 1, \dots p - 1$. This additional bit in the data structure allows us to represent a sum of two numbers in the range $0, 1, \dots, p - 1$ using e bits in odd characteristic.

We start with the prime field case $q = p$.

We pack as many prime field elements as possible into a machine word, that is, $w := \lfloor 32/e \rfloor$ prime field elements per word on a machine with word length 32 bits. On machines with word length 64 bits we pack $w := 2 \cdot \lfloor 32/e \rfloor$ prime field elements into one word. Note that we do not use $\lfloor 64/e \rfloor$ elements which can be one more, since then the conversion between the different data formats for different word lengths becomes too expensive and awkward.

We always imagine the least significant bit in a machine word as being on the right hand side. To store a vector, we start filling machine words from right to left, always using e bits for one prime field element and packing w prime field elements into a word.

We illustrate this layout in the following example for $p = 5$ and thus $e = 4$ on a machine with 32-bit word length.

```
bit              3322|2222|2222|1111|1111|1100|0000|0000
number:          1098|7654|3210|9876|5432|1098|7654|3210
prime field          |    |    |    |    |    |    |
element number: 7777|6666|5555|4444|3333|2222|1111|0000
```

Within each block of e bits we represent prime field elements by the binary representation of a number in the range $0, 1, \dots, p-1$ with the least significant bit of this binary representation on the right hand side. Thus, in our example $1+1+1+1 \in \mathbb{F}_5$ would be represented by the bit sequence `0100` and $1+1+1$ by `0011`. Note that the most significant bit in this representation is always 0.

The first w prime field elements in a vector are stored in the first machine word of the vector, the next w in the next word and so on.

Now we proceed to the extension field case $q = p^k$. Let e and w be exactly the same values as above. Here we have to store k prime field elements for every element of $\mathbb{F}_q = \mathbb{F}_p[x]/c_k\mathbb{F}_p[x]$, namely the coefficients of the unique residue class representative of degree smaller than k, where c_k is the Conway polynomial used to construct the finite field extension $\mathbb{F}_q$ over $\mathbb{F}_p$ (for details see [Nic88] or on the web [Lüb01]). Namely, if $a \in \mathbb{F}_q$ is represented by the polynomial $\sum_{i=0}^{k-1} a_i x^i$, we have to store the prime field elements $a_0, a_1, \dots, a_{k-1}$.

In a compact vector of length n of elements of $\mathbb{F}_q$ we distribute those numbers in the following way: The first k machine words in the memory representation of the vector are used to store the first w elements of the vector. The first machine word holds all the coefficients a_0 of those $\mathbb{F}_q$ elements, the second the coefficients a_1 and so on. Since one machine word can hold up to w prime field elements, this fills the first k words rather satisfactorily. The second k machine words in the vector then hold the vector elements with indices $w+1, w+2, \dots, 2w$ in exactly the same way.

Note again that for machines with a word length of 64 bits we choose the value w only twice as big as for 32-bit machines, even if one more prime field element would fit into the machine word. The only natural limit of this implementation is that $\lceil \log_2(2p-1) \rceil$ must be smaller or equal to the word length of the microprocessor.

Now we can explain how we can add vectors in the above representation by doing only a series of word operations.

(1.3) Adding compact vectors

The basic addition formula for finite field elements is rather simple: For prime field elements we only have to add two numbers in the range from 0 to $p-1$ and subtract p, if the sum is greater or equal to p. The extension field elements are done componentwise. However, we have to solve the problem of doing this simple operation using word operations and thus doing this for w prime field elements at the same time. This is easy for $p = 2$, since we can use the standard XOR (exclusive or) operation.

The idea to overcome this problem for $p > 2$ is the following. Assume a and b are two integers in the range from 0 to $p-1$. By adding $2^{e-1} - p$ to the sum $a+b$ we get $t := a+b+2^{e-1}-p$, which has the property that $t \geq 2^{e-1}$ if and only if $a+b \geq p$ (remember $2p-1 \leq 2^e$ and thus $p-1 < 2^{e-1}$). That is, if the number t is represented using binary expansion with e bits, then the most significant bit is set if and only if $a+b \geq p$.

Now this idea is used for two words a and b, containing w prime field elements each. Every prime field element uses exactly e bits in its word and we call these sections of e adjacent bits in a word "components" for the moment.
We prepare an "offset" word o that contains in each component the number $2^{e-1} - p$ and a "mask" word m that contains in each component the number 2^{e-1} meaning, that in each component only the most significant bit is set and all others are zero. In addition we keep a word n containing the number p in each component.
The finite field sum c of a and b is now computed in the following way (note that from now on arithmetic operations like $+$ are processor word operations): First $s := a + b$ and $t := a + b + o$ are computed. Then we use an AND operation for words to extract exactly the most significant bits of those components, in which the sum was greater or equal to p. This is done by computing $r := t \,\&\, m$ (we use the & symbol to indicate bitwise AND operations). Bit-shifting the word r by $e - 1$ bits to the right (we use the notation $r \gg (e-1)$ for this) and subtracting the result from r now results in a word $u := r - (r \gg (e-1))$ having the number $2^{e-1} - 1$ in those components, in which the sum was greater or equal to p and 0 in the others. Finally, doing a bitwise AND operation of u with n results in exactly the right word to subtract from s to get the correct result. Thus, the complete formula is

$$a + b - \Big(\big(r - (r \gg (e-1)) \big) \,\&\, n \Big), \qquad \text{where} \quad r = (a + b + o) \,\&\, m.$$

We illustrate this by an example for $p = 3$ on a machine with 32-bit word length. In this case, $e = 3$ and $w = 10$. We want to add the words shown below in the rows depicted by `a` and `b`. We also show the prepared words o, m and n. The bits marked with `X` are not used and are all equal to zero.

```
bit             33|222|222|222|211|111|111|110|000|000|000
number:         10|987|654|321|098|765|432|109|876|543|210
                  |   |   |   |   |   |   |   |   |   |
a:              XX|000|010|001|000|010|001|000|010|001|000
b:              XX|000|010|010|010|001|001|001|000|000|000
o:              XX|001|001|001|001|001|001|001|001|001|001
m:              XX|100|100|100|100|100|100|100|100|100|100
n:              XX|011|011|011|011|011|011|011|011|011|011
```

In the following table we show some intermediate results and repeat the input values for easier verification.

```
a+b:            XX|000|100|011|010|011|010|001|010|001|000
a+b+o:          XX|000|101|100|011|100|011|010|011|010|001
r:              XX|000|100|100|000|100|000|000|000|000|000
u:              XX|000|011|011|000|011|000|000|000|000|000
u&n:            XX|000|011|011|000|011|000|000|000|000|000
result:         XX|000|001|000|010|000|010|001|010|001|000
a:              XX|000|010|001|000|010|001|000|010|001|000
b:              XX|000|010|010|010|001|001|001|000|000|000
```

Of course it is a coincidence here that u is equal to u & n, since $p = 2^{e-1} - 1$.
This means that the addition of two words containing w prime field elements can be done in 7 word operations for $p > 2$. For the $p = 2$ case we only need one XOR operation. Thus we have proved the following proposition.

II.1.1 Proposition (Addition of compact vectors)
We assume a machine with 32*-bit word length. For* $2 < p < 2^{31}$ *two compact vectors of length* n *over the field of* $q = p^k$ *elements can be added using* $7k \cdot \lceil n/w \rceil$ *word operations (plus memory fetches and stores), where* $w = \lfloor 32/e \rfloor$ *and* $e = \lceil \log_2(2p-1) \rceil$*.*
For $p = 2$ *only* $k \cdot \lceil n/32 \rceil$ *word operations are needed.*
For machines with 64*-bit word length the number of word operations is* $7k \cdot \lceil n/(2w) \rceil$ *and* $\lceil n/64 \rceil$ *respectively and we can work with primes* $p < 2^{63}$*.*

Proof: See above. □

II.1.2 Remark
Note that, since the amount of memory needed for a vector of length n is $k \cdot \lceil n/w \rceil$ for $p > 2$ and $\lceil n/32 \rceil$ for $p = 2$, this means that the addition needs 7 word operations for each word of a vector for $p > 2$ and 1 word operation for $p = 2$.
Assuming a microprocessor with a long enough instruction pipeline we can conclude that all these word operations together can be done as fast as accessing the main memory to fetch a and b and store the result somewhere. Therefore the number of 7 word operations does not hurt at all, since the bottleneck is not the actual computation but rather the memory access. Compare Section II.(1.6) and see Section II.3 for some additional comments on processor caches.

Next we consider multiplication of vectors by scalars.

(1.4) Multiplication by scalars

To explain the method, we restrict our attention to the case that one compact vector shall be multiplied by one scalar and the result shall be stored in some other memory location.
Let e and w be defined as in Section II.(1.2).
We start by discussing the prime field case $\mathbb{F}_p$. Since a scalar $s \in \mathbb{F}_p$ is represented by an integer in the range 0 to $p-1$ in its binary expansion, we can multiply a compact vector v by s by repeatedly adding vectors to themselves starting with v and adding up those multiples whose corresponding bits in the binary expansion of s are set. That is, if $s = \sum_{i=0}^{e-2} s_i 2^i$ with $s_i \in \{0, 1\}$ and again $e = \lceil \log_2(2p-1) \rceil$, we compute $s \cdot v$ by computing $v, 2^1 \cdot v, 2^2 \cdot v, \ldots, 2^{e-2} \cdot v$ and then summing $\sum_{i=0}^{e-2} s_i \cdot (2^i \cdot v)$. All this can be done with at most $2(e-2)$ vector additions.
We now proceed to the extension field case $\mathbb{F}_q = \mathbb{F}_p[x]/c_k\mathbb{F}_p[x]$ with the Conway polynomial c_k and $q = p^k$. Here a scalar $s \in \mathbb{F}_q$ is represented by an expansion $s = \sum_{i=0}^{k-1} s_i x^i + c_k \cdot \mathbb{F}_p[x]$ again. For the rest of this section we omit the "$+c_k \cdot \mathbb{F}_p[x]$" and denote cosets by their representing polynomials of degree less than k.
We reduce the problem to scalar multiplications of vectors with prime field elements. To this end, we have to be able to multiply a vector with the primitive root x.

Considering a single scalar $t = \sum_{i=0}^{k-1} t_i x^i \in \mathbb{F}_q$, we see that

$$xt = \sum_{i=1}^{k-1}(t_{i-1}x^i) + \sum_{i=0}^{k-1} t_{k-1} \cdot (x^k - c_k) \in \mathbb{F}_q$$

(remember that we compute in $\mathbb{F}_p[x]$ modulo c_k). Thus, the multiplication by x can be achieved by a shift, one multiplication of $x^k - c_k$ with the prime field element t_{k-1} and an addition.
Since we distribute the prime field elements belonging to a single extension field element in our vector into adjacent words, we can do the shift basically for free by accessing a shifted memory location. But we still have to deal with the fact that we have w possibly different highest coefficients t_{k-1} stored together in one word. However, since we have to multiply all of them with the same prime field element coming from the expansion of $x^k - c_k = \sum_{i=0}^{k-1} b_i x^i$, we can use the method described for the prime field case above to compute every word to be added to the shifted vector. That is, for each k adjacent words $(a_{jk}, a_{jk+1}, \ldots, a_{(j+1)k-1})$ in our vector we have to add the words shifted by one $(0, a_{jk}, a_{jk+1}, \ldots, a_{(j+1)k-2})$ to the words $(a_{(j+1)k-1} \cdot b_0, \ldots, a_{(j+1)k-1} \cdot b_{i-1})$. Therefore, the total cost is exactly the same as for one multiplication of a vector by a prime field scalar and one addition of vectors.
For the full computation of $s \cdot v$, we have to perform this multiplication by x altogether $k-1$ times, multiply each intermediate result by the prime field scalar s_i and sum everything up. Thus, the total cost of the multiplication $s \cdot v$ is at most $2k-1$ multiplications of a vector by a prime field scalar and $2k-1$ additions of vectors. Here we count one more addition to keep the formulas a bit simpler and to account for one copying operation. Thus we have proved the following proposition.

II.1.3 Proposition (Multiplication of vectors by scalars)
We assume a machine with 32-bit word length. Let p be a prime, $q = p^k$, and w and e as above: $w = \lfloor 32/e \rfloor$ and $e = \lceil \log_2(2p-1) \rceil$.
For $2 < p < 2^{31}$ the total cost of a multiplication of a compact vector v of length n over $\mathbb{F}_q$ by a scalar $s \in \mathbb{F}_p$ is at most $14k(e-2) \cdot \lceil n/w \rceil$ word operations (plus memory fetches and stores). For $p = 2$ the scalar can only be 0 or 1 and therefore no computation is necessary at all.
For $2 < p < 2^{31}$ the total cost of a multiplication of a compact vector v of length n over $\mathbb{F}_q$ by a scalar $s \in \mathbb{F}_q$ is at most $7k(2k-1)(2e-3) \cdot \lceil n/w \rceil$. For $p = 2$ the total cost is at most $2k(k-1) \cdot \lceil n/32 \rceil$ word operations.
For a machine with 64-bit word length the factors $\lceil n/w \rceil$ and $\lceil n/32 \rceil$ have to be changed to $\lceil n/(2w) \rceil$ and $\lceil n/64 \rceil$ respectively and we can work with primes $p < 2^{63}$.

Proof: See above and add up the number of word operations for up to $2k-1$ multiplications of a vector by a prime field scalar and $2k-1$ additions of vectors.
This gives the result $7k(2(e-2)+1)(2k-1) \cdot \lceil n/w \rceil$ for $p > 2$. Note for $p = 2$ that the vector n consists of $k \cdot \lceil n/32 \rceil$ words and that we have to count only one vector addition for the "multiplication with a prime field scalar", since the word $a_{(j+1)k-1}$ has to be XORed to those words whose corresponding bit is set in $x^k - c_k$. □

II.1.4 Remark
Since a vector of length n needs $k \cdot \lceil n/w \rceil$ words for $p > 2$, this means, that the number of word operations per word of the vector needed for one multiplication with a scalar is at most $7(2k-1)(2e-3)$ for $p > 2$.

(1.5) Memory throughput in real implementations

In this section we present timing results in real implementations. We compare two implementations: The first is an implementation of the ideas presented in this chapter in the GAP package cvec (see [Neu06]) by the author, and the second is the standard implementation of compact vectors in the GAP kernel written by Steve Linton (see [GAP07]). The latter uses byte-oriented table lookup for fields $\mathbb{F}_q$ with $3 \leq q \leq 256$. We compared vector additions over $\mathbb{F}_2$ and $\mathbb{F}_7$, and multiplications of vectors by scalars over $\mathbb{F}_7$ in both implementations. Finally, we tested multiplications of vectors by scalars over $\mathbb{F}_{3^k}$ for $1 \leq k \leq 5$ in the new implementation. Note that the GAP library does not offer direct access to the operation $z := v + w$ without memory allocation. Therefore this measurement is missing.

We used three relatively new machines with popular microprocessors, namely two different machines with an Intel Pentium 4 with 1024 kB second level cache running at 3.2 GHz and a machine with an AMD Athlon 64 X2 Dual Core Processor 3800+ with 512 kB second level cache running at 2.0 GHz. The two Pentium 4 machines were equipped with memory modules with the same specifications (PC 3200, CL3).

In all runs no other processes were consuming significant amounts of CPU time such that nothing should have interfered with the caches. Strangely enough, the two nearly identical machines show different performance. We cannot explain the differences in memory throughput. Therefore we have presented both results to show that such measurements have a certain fluctuation obviously involving parameters which cannot easily be determined. For a discussion of these results see below.

To demonstrate the effect of second level caches, we used different lengths of vectors. We used vectors using $32 \cdot 10^6$ bytes each in order to make sure that none of the data is in the second level cache, and vectors using 125000 and 62500 bytes each to make sure that most accesses should lead to cache hits in the second level cache. Of course all operations are repeated many times to get a high accuracy of the measured time for one operation by averaging.

We tested three different operations. The first, which we denote by "$z := v + w$", is adding two different vectors and writing the result to some other memory location. The time for memory allocation was not considered. The second operation, which we denote by "$v := v + w$", is also adding two different vectors, but the result is written into the same memory location as one of the summands. The third operation is denoted by "$w := sw$" and is multiplying a vector by a (non-zero) scalar in place, that is, the result overwrites the original vector. We chose the scalar $6 + 7\mathbb{Z} \in \mathbb{F}_7$, since its binary expansion is 110 and it is thus one of the most "expensive" scalars for multiplication. Note that for the case of table lookups the chosen scalar does not matter.

For vector times scalar computations over $\mathbb{F}_{3^k}$ we always chose as scalar the coset of a polynomial with no zero coefficients, which is the worst with respect to performance.

All results are memory throughput values in megabytes per second. That is, a result of 1000 means that altogether 1000 times 1024^2 bytes of memory have been accessed. "Altogether" means that for the operation $z := v + w$ both read operations (for v and w) and the write operation into z are counted, that is, for vectors of length 125000 one such addition needs to access 375000 bytes in memory, namely reading 250000 and writing 125000. The same holds for $v := v + w$, whereas the operation $w := sw$ only reads the memory once and writes it again, which means that one such operation has to access 250000 bytes of memory.

All our results are shown in Table II.1. The columns marked with “C” are results obtained using the cvec package and those marked with “L” are results obtained using the GAP library.
In the next section we discuss some aspects of these results.

Table II.1: Memory throughput for vector operations, C: cvec package, L: GAP library

Test	C, P4$_1$	C, P4$_2$	C, Ath	L, P4$_1$	L, P4$_2$	L, Ath
$\mathbb{F}_2$, vectors $32 \cdot 10^6 B$, $z := v + w$	2764	2684	1940	—	—	—
$\mathbb{F}_2$, vectors $32 \cdot 10^6 B$, $v := v + w$	3629	2965	2643	3577	2812	2677
$\mathbb{F}_2$, vectors $125\,000 B$, $z := v + w$	6839	4249	8499	—	—	—
$\mathbb{F}_2$, vectors $125\,000 B$, $v := v + w$	6167	3791	12076	4818	3298	12116
$\mathbb{F}_2$, vectors $62\,500 B$, $z := v + w$	5643	4379	7560	—	—	—
$\mathbb{F}_2$, vectors $62\,500 B$, $v := v + w$	6458	3969	11237	5109	3406	11635
$\mathbb{F}_7$, vectors $32 \cdot 10^6 B$, $z := v + w$	2380	1691	1902	—	—	—
$\mathbb{F}_7$, vectors $32 \cdot 10^6 B$, $v := v + w$	2624	1749	2515	1088	708	488
$\mathbb{F}_7$, vectors $32 \cdot 10^6 B$, $w := sw$	2289	1504	2267	1205	870	422
$\mathbb{F}_7$, vectors $125\,000 B$, $z := v + w$	2671	1761	5745	—	—	—
$\mathbb{F}_7$, vectors $125\,000 B$, $v := v + w$	2843	1816	5356	1102	713	496
$\mathbb{F}_7$, vectors $125\,000 B$, $w := sw$	2388	1490	4273	1375	876	427
$\mathbb{F}_7$, vectors $62\,500 B$, $z := v + w$	2765	1742	5472	—	—	—
$\mathbb{F}_7$, vectors $62\,500 B$, $v := v + w$	3080	1878	5529	1170	737	498
$\mathbb{F}_7$, vectors $62\,500 B$, $w := sw$	2386	1565	5138	1342	898	434
$\mathbb{F}_3$, vectors $32 \cdot 10^6 B$, $w := sw$	2297	1471	2219	—	—	—
$\mathbb{F}_9$, vectors $32 \cdot 10^6 B$, $w := sw$	153	98	336	—	—	—
$\mathbb{F}_{27}$, vectors $32 \cdot 10^6 B$, $w := sw$	99	68	232	—	—	—
$\mathbb{F}_{81}$, vectors $32 \cdot 10^6 B$, $w := sw$	88	58	204	—	—	—
$\mathbb{F}_{243}$, vectors $32 \cdot 10^6 B$, $w := sw$	70	47	170	1166	867	422

(1.6) Discussion of results

The results for the $\mathbb{F}_2$ case show the throughput for raw memory access since the XOR operations for addition within the processor registers cost nearly nothing.
First we look at the top $\mathbb{F}_2$ part of Table II.1.
We observe that both implementations perform very similarly, which is only to be expected, since they use exactly the same method. For main memory access we seem to get nearly identical throughput. For second level cache accesses we see some differences, which can be explained for the given architectures by looking at the C code and the produced assembler code, which would lead too far here. However, the general picture is the same.
One can clearly see in the $\mathbb{F}_2$ part of Table II.1 that the second level cache seems to help noticeably when the vectors involved are shorter. This is to be expected, but shows only a factor of between 1.5 and 3 for the Pentium 4 and up to a factor of 5 for the Athlon processor.

Theoretically from the technical specifications of main memory and second level cache one would expect a higher speedup factor, but modern processors and memory interfaces are highly optimised for linear memory accesses using so-called bursts. This means that the processor has a special method of accessing large amounts of consecutive memory words consecutively. All these effects together seem to lead to the behaviour we observe.

We turn now to the middle part of the table concerning the field $\mathbb{F}_7$. Here one can see two important things: The first is that the 7 word operations per word stated in Proposition II.1.1 can be done as fast as main memory access (compare main memory throughput for $p = 2$ and $p = 7$ on all architectures). If nearly all accesses lead to hits in the second level cache, then the 7 operations reduce the memory throughput, but only by a factor of about 2 to 3, depending on the processor architecture.

The second thing is that the byte oriented table lookup is noticeably slower than the word access. When working in the second level cache, this can cripple the memory throughput by a factor of up to 10, at least on the Athlon architecture. Note that the vector sizes are chosen small enough, such that not only the three vectors but also the lookup tables should fit completely in the second level cache together. The same observation holds for the multiplication with a scalar.

Of course, for other finite fields $\mathbb{F}_q$ with $q \leq 256$ the situation is different. Among these fields the number $7(2k-1)(2e-3)$ of word operations per vector word is at most 189. The experiments shown in the third part of Table II.1 concerning fields $\mathbb{F}_{3^k}$ indicate, that if this number is assumed, the performance of multiplication of a vector by a scalar is much slower than with table lookup.

However, in the light of the grease method explained in Section II.(2.2) computing too many vector times scalar operations can often be avoided, which renders the new method superior in many real life examples.

Finally note that due to the extra bit per prime field element that is necessary for doing word operations, vectors in the cvec implementation need a bit more memory than in the GAP library implementation (for characteristic not equal to 2). Thus, if the memory throughput is the same, the practical performance is a bit worse with word operations than with table lookup.

Considering all the aspects in this comparison we conclude that there is no method that is better in all situations.

II.2 Matrix arithmetic

Traditionally, matrices are implemented in the GAP system as lists of (row-) vectors. This means that the list object only holds references to the subobjects, which are the rows of the matrix. Therefore, one can for example permute rows very efficiently without actually touching the bulk of the data in the rows. Instead, one can simply redirect references. In addition, different matrices can share rows, which leads to the fact that the modification of one matrix can in fact modify the other matrix. Although this can give rise to nasty bugs, it can also help to increase efficiency both with respect to memory usage and with respect to performance. We call matrices implemented with this approach “row list matrices”.

An alternative way would be to actually embed the row data into the matrix object itself. A row would then no longer be a GAP object in its own right, but only a part of one. The advantage

here is that we need fewer memory allocations (only one per matrix) and can better control cache issues, since we know better, where in memory our row data is stored. We call matrices implemented with this approach "flat matrices".
When devising efficient algorithms dealing with matrices, one has to know which type of matrix one is using, because one has to know, when the system actually copies data and when it only passes references.
For the cvec package we chose the row list matrix approach, since it fits better into the existing GAP library and allows for easier code reusage. We also discuss this approach further in this section. We call a list of compact vectors of the same length over the same field a "compact matrix".
Note however that Beth Holmes and Richard Parker are currently working on an implementation of flat matrices in the GAP system.

(2.1) Addition and multiplication with scalars

This section is extremely short, since to add matrices or to multiply a matrix by a scalar simply means to add corresponding row vectors or multiply all rows by the scalar respectively. Thus we immediately get from Propositions II.1.1 and II.1.3 the following corollary.

II.2.1 Corollary (Matrix addition and multiplication of matrices by scalars)
We assume a machine with 32-bit word length. Let M and M' be two compact matrices with m rows and n columns over $\mathbb{F}_q$, where $q = p^k$.
For $2 < p < 2^{31}$ the matrices M and M' can be added using $7km \cdot \lceil n/w \rceil$ word operations (plus memory fetches and stores), where $w = \lfloor 32/e \rfloor$ and $e = \lceil \log_2(2p-1) \rceil$. For $p = 2$ only $mk \cdot \lceil n/32 \rceil$ word operations are needed.
For $2 < p < 2^{31}$ the total cost of a multiplication of M by $s \in \mathbb{F}_p$ is at most $14km(e-2) \cdot \lceil 32/w \rceil$ word operations (plus memory fetches and stores). For $p = 2$ the scalar can only be 0 or 1 and therefore no computation is necessary at all. For $2 < p < 2^{31}$ the total cost of a multiplication of M by $s \in \mathbb{F}_q$ is at most $7k(2k-1)m(2e-3) \cdot \lceil n/w \rceil$. For $p = 2$ the total cost is at most $2k(k-1)m \cdot \lceil n/32 \rceil$ word operations.
For a machine with 64-bit word length the factors $\lceil n/w \rceil$ and $\lceil n/32 \rceil$ have to be changed to $\lceil n/(2w) \rceil$ and $\lceil n/64 \rceil$ respectively and we can work with primes $p < 2^{63}$.

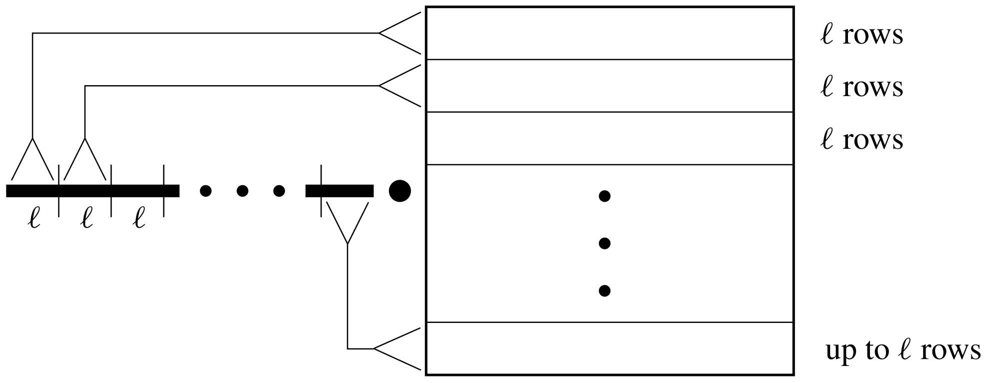
Proof: This is immediate from Propositions II.1.1 and II.1.3. □

(2.2) Vector-matrix multiplication

This section is about the multiplication of a row vector $v \in \mathbb{F}_q^{1\times m}$ by a matrix $M \in \mathbb{F}_q^{m\times n}$. The result is a vector $vM \in \mathbb{F}_q^{1\times n}$. Since by convention we are working mainly with row vectors, this is the only operation between vectors and matrices we have to consider. The left multiplication of a column vector by a matrix can be simulated using the transposed matrix.
The multiplication vM can be understood as taking a linear combination of the rows of M, the vector v contains the coefficients. This kind of reasoning already leads to the most efficient way to implement this operation: We multiply the i-th row of M by the i-th component v_i of v and

add the resulting vector to the final result for $1 \le i \le m$. Thus this operation can be done using at most m multiplications of a vector by a scalar plus $m - 1$ vector additions.
If we are considering only one vector-matrix multiplication, this is basically all one can say. However, in a situation in which we apply one given matrix repeatedly to lots of vectors (for example during matrix multiplication (see below) or when enumerating orbits), there is a trick called "grease", which was invented by Richard Parker and others (see for example [AHU75, Algorithm 6.2]).
The idea is that if the base field is small, there are not so many different linear combinations of a finite number of vectors. Thus we can distribute the rows of our matrix into small groups, say of ℓ adjacent rows each, and precompute all possible linear combinations of all vectors in each group. If we now want to multiply a vector v and the matrix, we extract the entries of v corresponding to each group, look up the linear combination and add it to the result. The number ℓ of elements in each group is called "grease level".

Figure II.1: Illustration for Vector-matrix multiplication and grease



This technique is illustrated pictorially in Figure II.1. There one can see a vector-matrix multiplication. The vector is divided into sections of length ℓ, beginning from the left, leaving a section of possibly less than ℓ elements at the right hand side. Each such section corresponds to ℓ (or less at the end) adjacent rows in the matrix; we call such a submatrix a "grease block". To do grease level ℓ, we compute all possible linear combinations of each block of ℓ rows and store them. If we have computed and stored all this data, we call the matrix "greased". Then a vector-matrix multiplication with this matrix only has to perform $\lceil m/\ell \rceil - 1$ vector additions of looked up vectors. More precisely, we have the following result.

II.2.2 Theorem (Grease: expected cost and gain)
Let M be a matrix with m rows and n columns over the field $\mathbb{F}_q$ for $q = p^k$, and let $\ell \in \mathbb{Z}_{>0}$. Then greasing the matrix M with grease level ℓ multiplies the amount of memory needed for M by at most q^ℓ. The precomputation of the linear combinations costs at most $q^\ell - k \cdot \ell - 1$ row vector additions and $\ell \cdot (k - 1)$ multiplications of a row vector with a primitive root of $\mathbb{F}_q$ per grease block. That is, for the complete matrix, we need at most $\lceil m/\ell \rceil \cdot (q^\ell - k \cdot \ell - 1)$ row vector

additions and $m \cdot (k-1)$ multiplications by the primitive root.
If M is greased, a vector v can be multiplied from the right by M using at most $\lceil m/\ell \rceil$ row vector additions plus the cost of extracting the finite field elements from the vector v, which is usually negligible.
The standard approach for a vector-matrix multiplication needs approximately $m(q-1)/q$ multiplications of a row vector by a non-zero scalar from $\mathbb{F}_q$ plus $(m-1)(q-1)/q$ row vector additions, assuming that about every q-th element of v is equal to zero.

Proof: First we discuss the precomputation step to compute the greased matrix. Since there are q^ℓ different linear combinations of ℓ vectors the statement about the needed memory is clear.
We have to do the following for every grease block and thus all costs are multiplied by $\lceil m/\ell \rceil$. After multiplying each vector in the block $k-1$ times by the primitive root of $\mathbb{F}_q$ represented by the polynomial $x \in \mathbb{F}_p[x]$ and storing the intermediate results, we can compute all $\mathbb{F}_q$-linear combinations of the vectors in the block by computing all $\mathbb{F}_p$-linear combinations of the vectors we already have. Thus, by Lemma II.2.4 below we can compute all those linear combinations with $p^{k\cdot\ell} - k \cdot \ell - 1 = q^\ell - k \cdot \ell - 1$ row vector additions, which is the cost in the theorem. Note that the scalar multiplications by the primitive root of $\mathbb{F}_q$ can be done more efficiently, since the element represented by the polynomial x is sparse, such that the scalar multiplication by it is faster.
After having greased the matrix, a multiplication of a vector by the matrix can be done by adding appropriate vectors. We have to add one for every grease block, leading to $\lceil m/\ell \rceil - 1$ additions. However, since usually the result has to be stored in a different location than the table of greased vectors, we have to copy the first summand somewhere. Thus we count one addition to account for this copying operation. □

II.2.3 Remarks

- The factor $(q-1)/q$ used for the standard approach is a good estimate for practical considerations when comparing to the grease approach. For larger values of q it is of course negligible, however for $q = 2$ it is equal to $1/2$.

- It is crucial to implement the extraction of entries from the vector v as efficient as possible. If this is not done right, the cost of extracting elements in one vector can cancel out the benefit of fewer scalar multiplications.

- The fact that we need very few scalar multiples in the precomputation phase and none in the vector times matrix phase is very useful with respect to our new implementation in the cvec package, since there an addition is sometimes much cheaper than a multiplication with a scalar.

The following lemma is used in the proof of Theorem II.2.2.

II.2.4 Lemma (Computing all $\mathbb{F}_p$-linear combinations)
Let $v_1, \ldots, v_j$ be $\mathbb{F}_p$-linearly independent vectors over a finite field $\mathbb{F}_q$ with $q = p^k$ for some $j \in \mathbb{Z}_{>0}$. If all $\mathbb{F}_p$-linear combinations of $v_1, \ldots, v_{j-1}$ are already computed, then all $\mathbb{F}_p$-linear combinations of $v_1, \ldots, v_j$ can be computed with $p^j - p^{j-1} - 1$ vector additions.

This implies in particular that all $\mathbb{F}_p$-scalar multiples of a vector can be computed with $p-2$ vector additions, and inductively that all $\mathbb{F}_p$-linear combinations of $v_1, \dots, v_j$ can be computed without assumption using $p^j - j - 1$ vector additions.

Proof: All non-zero $\mathbb{F}_p$-scalar multiples of v_j can be computed by successively adding v_j altogether $p-2$ times. Then every $\mathbb{F}_p$-linear combination of $v_1, \dots, v_j$ is either one of the already computed $\mathbb{F}_p$-linear combinations of $v_1, \dots, v_{j-1}$ or the non-zero multiples of v_j, or a sum of two such vectors. Thus we can compute all missing $p^j - p^{j-1} - (p-1)$ vectors with one vector addition each, which proves the first statement.
This includes the statement for $j = 1$, which handles the case of computing all $\mathbb{F}_p$-scalar multiples of one vector. Without assumptions, we can apply the first statement inductively for $i = 1, 2, \dots, j$ proving the last statement, since

$$\sum_{i=1}^{j} (p^i - p^{i-1} - 1) = p^j - j - 1.$$

□

II.2.5 Remark
Computing all $\mathbb{F}_p$-linear combinations can be done with one addition per vector that is not already given, assuming the zero vector and the input vectors are already there.

(2.3) Matrix multiplication

Let $M \in \mathbb{F}_q^{m \times n}$ and $N \in \mathbb{F}_q^{n \times s}$ with $q = p^k$. The matrix multiplication $M \cdot N$ can be computed by multiplying the rows of M from the right with the matrix N and putting the m resulting rows of length s into the resulting matrix in $\mathbb{F}_q^{m \times s}$.
There are at least three methods to improve the performance of this computation:
The first is using grease (see the previous section), which we will analyse in detail in this section. The second is arranging the same computations in a different order to achieve that more memory fetches and stores are handled by the second level cache. This approach is used in some implementations and can speed up computations. The third method is to use the methods of Strassen and Winograd to get a lower exponent than 3 in the complexity of matrix multiplication of square matrices. This latter method produces cache locality automatically, therefore we only discuss the second method briefly below. For some additional discussion of caches see Section II.3.
Now we discuss grease before we analyse for which sizes of matrices the Strassen/Winograd method is worthwhile doing.
When the matrices we want to multiply are big enough such that the number of rows of M is large in comparison to q^ℓ for a suitable chosen grease level ℓ, it is not necessary to grease the complete matrix N before beginning the multiplication. Rather, we can proceed with the whole multiplication “by grease blocks”, that is, we compute in a first step all possible linear combinations of the first ℓ rows of N, then run through all rows of M looking only at the first ℓ entries, look up the right linear combination of the first ℓ rows of N and add it to the corresponding row of the final result. After that we can forget our precomputed linear combinations and proceed to the next grease block in N and so on.

We have to determine the appropriate grease level (where 0 stands for not using grease at all). To this end we compare $(q^\ell/\ell - k - 1/\ell) + m/\ell$ row vector additions plus $k - 1$ row vector multiplications to $m \cdot (q-1)/q$ row vector multiplications and additions. By Theorem II.2.2 the former are the numbers of row vector operations necessary for the grease approach and the latter the number of row vector operations for the standard approach, both counted for one column of the left matrix. Obviously, grease becomes more interesting when m is large in comparison to q^ℓ. Furthermore, the advantage of grease grows with the matrix size.

To optimise this, a calibration phase is done once and for all: For all finite fields separately, a procedure determines, how long a multiplication of a vector by an arbitrary scalar and the primitive root respectively takes in comparison to an addition of two vectors. This depends on the particular machine architecture, especially on cache sizes and memory interfaces and thus has to be done for every machine separately. The result is a table advising which level of grease to use for which values of m. Since all the above mentioned cost functions are linear functions in m, it is easy to show that the optimal grease level depends monotonically increasing on m.

Note that this analysis shows that in particular for values of q for which multiplication of row vectors by scalars is expensive (see Section II.(2.1)) even grease level $\ell = 1$ is worthwhile doing.

When we use the optimal grease level determined as described, the cost of matrix multiplication depends on the size of the matrices in a relatively complicated way. In Figures II.2 and II.3 we show times t of matrix multiplications over $\mathbb{F}_2$ for different sizes n of square matrices using grease. The timings were obtained by using the cvec package in GAP on a machine with an AMD Athlon 64 X2 Dual Core processor 4600+ with 512 kB second level cache running at 2.4 GHz. The displayed behaviour is typical and holds also for other finite fields.

Figure II.2: Multiplication times over $\mathbb{F}_2$ against matrix dimension using grease

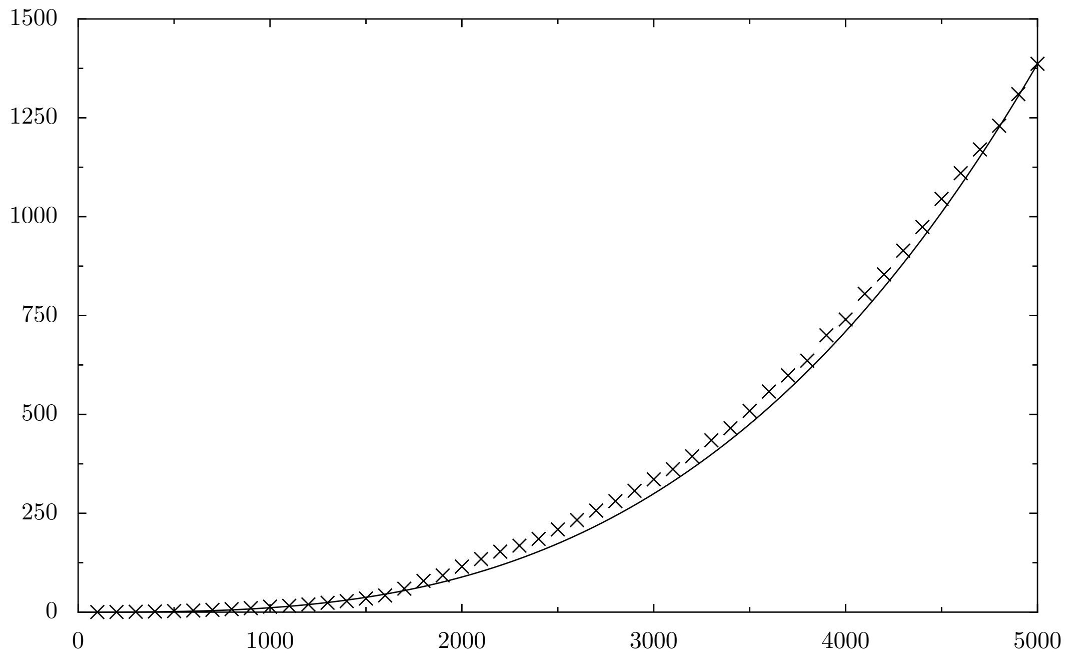
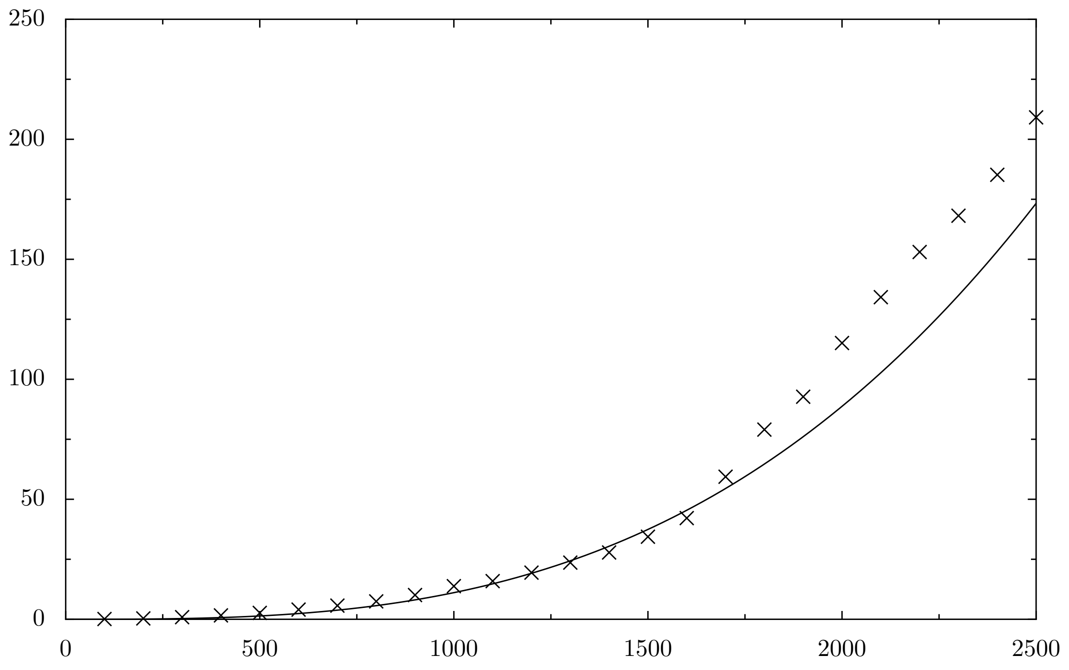


Figure II.3: Multiplication times over $\mathbb{F}_2$ against matrix dimension using grease (magnification)



The actual measurements are depicted by little $\times$ signs and the curve is a plot of the function $t = cn^3$ for some constant c, such that the curve passes through the actual measurement for $n = 5000$.

We see two effects. The first is that the actual measurements do not lie exactly on the theoretical n^3 curve. Rather, most measurements lie above that curve. This means that multiplying small matrices seems to take longer than expected by the needed number of operations in comparison to larger matrices. This effect can be explained on the one hand by grease, which is more effective the bigger the matrices become. On the other hand for larger matrices the actual row operations dominate the administrative tasks like loops and also memory accesses to large consecutive areas of memory seem to be more efficient.

The second effect is that for very small matrices (smaller than about 1500×1500 for the field $\mathbb{F}_2$) the multiplication is again faster than expected. This is due to the second level cache built into nowadays microprocessors. A (1500×1500)-matrix over $\mathbb{F}_2$ needs about 300 kB of memory. Thus, for smaller matrix multiplications most memory accesses are in fact cache hits.

For really large values of m and fixed ℓ we can neglect the cost for the precomputation step of greasing and thus the total number of row vector additions for the matrix multiplication of M by N is approximately $m \cdot \lceil n/\ell \rceil$. Note that we cannot neglect the cost for the precomputation for larger values of ℓ, since the precomputation cost contain a term q^ℓ. Thus in practice, the grease level always stays rather small.

Since the cost of a row vector addition of vectors of length s is proportional to s (compare Proposition II.1.1), we get an asymptotic upper bound for the total cost for matrix multiplication that is proportional to $mnsk$ (remember that we fixed $p < 2^{31}$ or $p < 2^{63}$ in this analysis). For square

$(n \times n)$-matrices this is the classical $O(n^3)$ complexity for the number of field operations.

We now discuss methods by Strassen and Winograd to reduce this complexity for large matrices. These methods apply in practice only to square matrices. The basic idea is that it is possible to multiply two (2×2)-matrices using only 7 multiplications instead of 8 in the standard approach. One pays for this with a few more additions. Using this idea for two $(2n \times 2n)$-matrices means cutting both into 4 blocks of $(n \times n)$-matrices each and computing the whole product by computing 7 matrix products of $(n \times n)$-matrices. For details of the procedures developed by Strassen and Winograd see [Knu98, 4.6.4]. With their help one can reduce the theoretical complexity for the multiplication of two $(n \times n)$-matrices from $O(n^3)$ to $O(n^{\log_2(7)}) \approx O(n^{2.807})$. A first version was published by Strassen and later slightly improved by Winograd. In the following we will concentrate on the Winograd variant. There are also more involved methods reducing the exponent further, but they seem to be of theoretical interest only (see also [Knu98, 4.6.4]).

Using the Winograd method recursively needs a certain amount of additional administration, mostly cutting bigger matrices into smaller ones and copying the smaller intermediate results back into the bigger result. Since the cost of all this administration is proportional to the area of the matrices involved (i.e. it is only quadratic in the size of the matrices), it is clear that this will be worthwhile for "large" matrices. For smaller matrices the administrative overhead might dominate the saving effect of the better complexity. In particular the method is only good if 7 multiplications of two $(n \times n)$-matrices plus the administrative overhead need in fact less time than one multiplication of two $(2n \times 2n)$-matrices.

The percentage of administrative work for the preparation of a matrix multiplication among the total work is higher for small matrices. This fact and the increasing effect of grease lead to the fact that for matrices over finite fields n needs to be rather large for the above condition to be fulfilled.

Therefore one has to test individually for each finite field and each machine type, from which limit for n on the Winograd method should be used. It seems that one can safely assume that provided it is worthwhile for some n_0, it is also useful for all $n > n_0$.

However, in a practical implementation things are not that easy. In the following Table II.2 we have collected empirical data about runtimes of matrix multiplication for different sizes of square matrices over the field $\mathbb{F}_2$. In the different columns we vary the number of times we use the Winograd trick. "Wino 2" for example means that the trick is used twice, that is, the matrices are cut once and the pieces are cut once again.

One can observe that it is not true that the bigger the matrices are the more often one should use the Winograd trick. For (4000×4000)-matrices it is for example better not to use the Winograd trick at all, whereas it helps to do it once for (3000×3000)-matrices. Even worse, if the best number of Winograd applications for $(n \times n)$-matrices is i, then it is not necessarily true that the best number of Winograd applications for $(2n \times 2n)$-matrices is $i + 1$ (see for example $n = 2000$ in Table II.2). The situation for the base field $\mathbb{F}_2$ is admittedly an extreme example, but it is typical for small fields as can be seen by comparison with the field $\mathbb{F}_3$ for which we show some analogous timings in Table II.3.

A good theoretical explanation of these findings is difficult, in particular since such measurements depend very much on the particular processor type and memory interface. As mentioned above, at least three factors seem to play a role: The first is that the saving effects of grease are greater for bigger matrices, the second is that administrative costs for the Winograd method play a bigger role

Table II.2: Matrix multiplication times over $\mathbb{F}_2$ using Winograd

n	Wino 0	Wino 1	Wino 2	Wino 3
1500	**36**	66	152	395
2000	125	**116**	242	600
2500	218	**190**	360	840
3000	350	**310**	505	1090
3500	**520**	600	670	1410
4000	**790**	960	870	1790
4500	**1070**	1270	1130	2180
5000	1480	1670	**1420**	2640
5500	1930	2070	**1720**	3130
6000	2540	2610	**2190**	3680
6500	**3090**	3170	3460	4240
7000	**3810**	3870	4670	4930
7500	4760	**4670**	5630	5580
8000	5700	**5600**	6800	6280
8500	**6700**	6720	8150	7160
9000	8050	**7910**	9130	8220
9500	9310	**9070**	10400	9130
10000	10800	11430	11920	**10170**

for smaller matrices, and finally the processor caches and specialities in main memory interfaces like burst accesses sometimes have nearly unpredictable effects on performance. For example, for the field $\mathbb{F}_2$ it seems that for square matrices in the range of $n = 3500$ to $n = 4500$ cutting them once does not make the pieces small enough to fit into the second level cache and doing the Winograd trick twice already involves too much administrative overhead, such that altogether it is best not to use the Winograd method at all. For smaller matrices in the range $n = 2000$ to $n = 3000$ on the other hand it seems worthwhile to use Winograd once, which makes the pieces fit into the second level cache. For $n = 5000$ in turn it seems to be the best to use Winograd twice to end up with matrices of size 1250 fitting into the second level cache. Thus it seems that this strange cache related behaviour in the range 2000–4500 is repeated in the range 5000–9000 and possibly above.

The cvec package contains an implementation of the Winograd method and uses a calibration phase to determine for every finite field with $q \leq 1024$ from which size the method is worth doing. In fact, there are two implementations of this method, one using more memory and less administrative overhead and one using less memory. The first is used for smaller matrices, since for them it is crucial to cut down administrative costs as much as possible. For really large matrices, the extra memory needed would hurt too much and the saving effects of the better complexity render the method useful even with the higher administrative overhead.

However, the question of when to use the Winograd method and how often in practical applications remains quite difficult and highly machine dependent.

Table II.3: Matrix multiplication times over $\mathbb{F}_3$ using Winograd

n	Wino 0	Wino 1	Wino 2	Wino 3
1500	425	**415**	510	800
2000	980	**940**	1020	1420
2500	**1810**	1880	1870	2400
3000	2970	3100	**2900**	3660
3500	4650	4760	**4420**	5350
4000	**6620**	6850	6690	7310
4500	**9260**	9820	9840	9970
5000	**12420**	12910	13250	13280
5500	**16340**	16740	17260	16770
6000	20800	20840	21780	**20760**
6500	26350	26370	27370	**25920**
7000	31990	32590	33470	**31210**
7500	39730	39350	41290	**39260**
8000	47430	**46610**	48350	47040
8500	57960	**55610**	57580	59040
9000	69450	**65310**	69100	69410
9500	78740	**75780**	79880	81380
10000	91130	**87290**	91120	93040

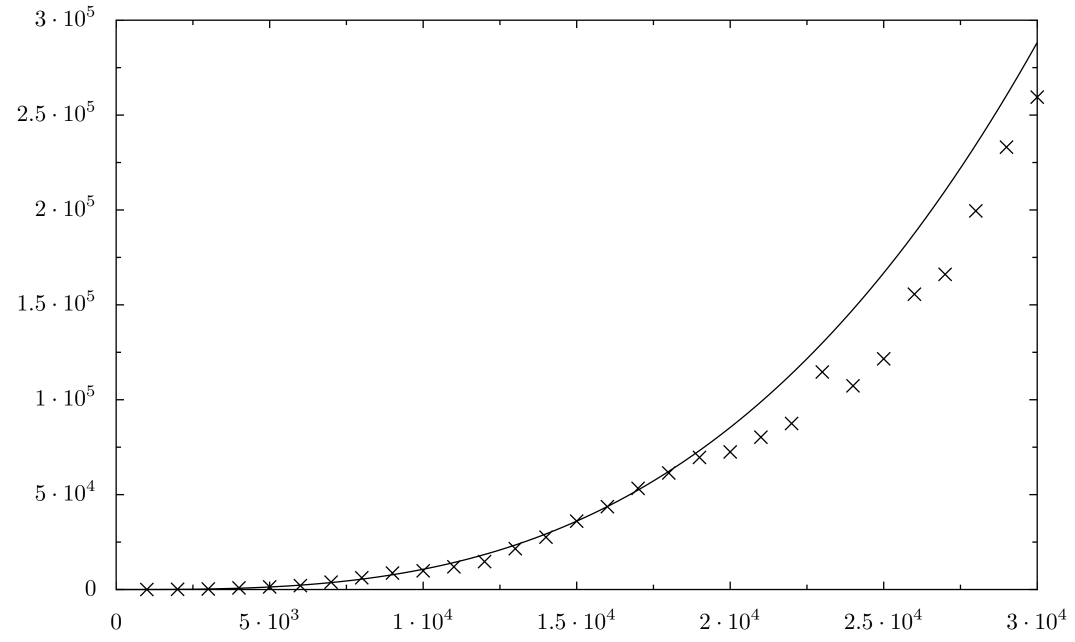
We conclude this section with Figures II.4 and II.5 showing the performance of matrix multiplication using grease together with the Winograd method. Here we used the Winograd method for matrices of size greater or equal to $n = 2500$ for $\mathbb{F}_2$ and ($n = 3000$ for $\mathbb{F}_3$), that is, from $n = 5000$ ($n = 6000$, respectively) on it is used twice and so on. The curve shown there is again the plot of the function $t = cn^3$ for some constant c, such that the curve passes through the actual measurement for $n = 15000$ ($n = 10000$, respectively). The better complexity begins to show with a few outliers following the pattern observed above.

II.3 Cache issues

All modern microprocessors have built-in caches, most of them more than one. For our purposes the second level cache is crucial. The potential of proper cache usage can be read off the data in Table II.1. Varying between different processor architectures, accesses into the cache are about 2 to 5 times faster than into the main memory. Of course, these factors apply only for the case of accessing larger areas of main memory consecutively, because in this case modern processors use burst accesses. However, most linear algebra operations over finite fields can be done using mostly such consecutive accesses. Note that these are empirical results, theoretically, cache accesses should be a bit faster, depending on the exact architecture.

We assume for the moment that cache accesses are 5 times faster than main memory accesses. If we manage to implement our linear algebra algorithms in a way such that 90% of all storage accesses are cache hits, then we can speed up 90% of the calculation by a factor of 5 (assuming

Figure II.4: Matrix multiplication times over $\mathbb{F}_2$ against matrix dimension



that memory access is the most time-critical part of the computation). Optimally, the other 10% remain unchanged. This means that the overall saving factor is between 3 and 4 because the needed time is $(90/5 + 10)\% = 28\%$ of the time without cache-aware methods.

However, reaping this saving factor is tricky. On one hand it proves increasingly difficult to understand exactly what modern processors, memory interfaces and caches actually do. On the other hand, for many algorithms it is quite difficult to rearrange the computations in a way, such that in fact 90% of memory accesses lead to cache hits. Quite often such rearrangements need a certain administrative effort, which can devour parts of the performance gain. Our observations displayed in Tables II.2 and II.3 provide a glance on the arising difficulties.

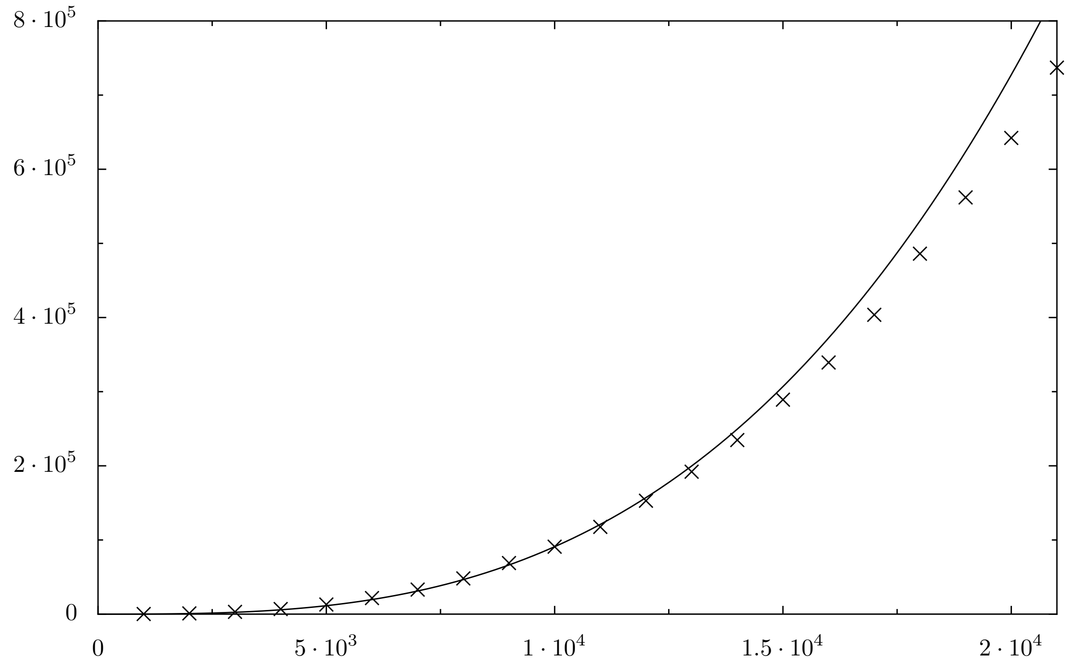
For the cvec package we have mostly ignored the potential gains through cache-aware programming since in most instances experiments indicated that the necessary effort did not promise enough benefit.

Contrary to these conclusions, Jon Thackray has reported successes in his stand-alone implementation of basic linear algebra routines using cache-aware programming. His performance gains seem to lie in the above mentioned expected range.

II.4 Raising matrices to powers

For the sake of completeness we conclude this chapter with the description of a well-known technique to raise an element to a power.

Figure II.5: Matrix multiplication times over $\mathbb{F}_3$ against matrix dimension



II.4.1 Lemma (Powering up)
If R is a monoid and $x \in R$, then the power x^m can be computed in at most $2\lceil \log_2(m) \rceil$ multiplications in R.

Proof: Let $l := \lfloor \log_2(m) \rfloor$ and write m in its binary expansion $m = \sum_{i=0}^{l} m_i 2^i$ with $m_i \in \{0, 1\}$. Then

$$x^m = \prod_{i=0, m_i \neq 0}^{l} x^{2^i}.$$

Computing the repeated squares x^{2^i} for $1 \leq i \leq l$ needs l multiplications in R and taking the final product needs at most another l multiplications. □

II.4.2 Remark
Note that there is an unpublished algorithm due to Charles Leedham-Green in the folklore that uses the minimal polynomial and polynomial arithmetic to raise a matrix over a finite field to an arbitrary power. Neglecting the time for integer arithmetic this algorithm needs for a given matrix only a constant number of operations to compute an arbitrary power.

Chapter III

Computing characteristic and minimal polynomials

This chapter is about the computation of characteristic and minimal polynomials of matrices over finite fields. The contents of this chapter are joint work with Cheryl Praeger and are already published as [NP08].

III.1 Introduction

Let $\mathbb{F}$ be a finite field and $M \in \mathbb{F}^{n\times n}$ a matrix. This chapter presents and analyses a Monte Carlo algorithm (see Section I.3) to compute the minimal polynomial of M, that is, the monic polynomial $\mu \in \mathbb{F}[x]$ of least degree, such that $\mu(M) = 0$. Determining the minimal polynomial is one of the fundamental computational problems for matrices and has a wide range of applications. As well as revealing information about the Frobenius normal form of M, the minimal polynomial also elucidates the structure of $\mathbb{F}^n$ viewed as $\mathbb{F}[x]$-module, where x acts by multiplication with M. In addition, the order of M modulo scalars is often found by first determining the minimal polynomial. Apart from these applications it has important practical utility, for example in the context of the matrix group recognition project [O'B06].

For these and other reasons a number of algorithms to determine the minimal polynomial may be found in the literature. We discuss some of them below. Our primary objective was to provide a simple and practical algorithm that could be implemented easily and would work well over small finite fields. In particular we did not want to produce matrices with entries in larger fields or polynomial rings as intermediate results, and we preferred to restrict ourselves to using only row operations (rather than a combination of row and column operations). In addition, we wished to use standard field and polynomial arithmetic, and we wanted to give an explicit worst-case upper bound for the number of elementary field operations needed, rather than only an asymptotic complexity statement. Our Monte Carlo algorithm adheres to these requirements for matrices over fields $\mathbb{F}_q$ of order q.

III.1.1 Theorem
For a given matrix $M \in \mathbb{F}_q^{n\times n}$ and a positive real number $\varepsilon < 1/2$, Algorithm 6 computes the minimal polynomial of M with probability at least $1 - \varepsilon$. For sufficiently large n and fixed ε,

the number of elementary field operations required is less than $7n^3$ plus the costs of factorising a degree n polynomial over $\mathbb{F}_q$ and constructing at most n random vectors in $\mathbb{F}^n$.

Our algorithm to compute the minimal polynomial first computes the characteristic polynomial in a standard way by spinning up and then factoring out cyclic subspaces. However, the novel aspect in this first phase is the introduction of randomisation. While not necessary for the computation of the characteristic polynomial it underpins our proof of the Monte Carlo nature of our minimal polynomial algorithm. In addition to the Monte Carlo minimal polynomial algorithm we present and analyse in Section III.8 a deterministic verification procedure to be run after Algorithm 6, that has a similar asymptotic complexity in many cases, but is $O(n^4)$ in the worst-case scenario. Our motivation for giving concrete upper bounds for the costs of various component procedures was that, in practical implementations, these assist us to compare different algorithms in order to decide which to use in different situations. At the end of this chapter we discuss a practical implementation and tests of the algorithms in the GAP system [GAP07].

(1.1) Other algorithms in the light of our requirements

There are several interesting and asymptotically efficient minimal polynomial algorithms for $n \times n$ matrices in the literature. The most asymptotically efficient deterministic algorithm is due to Storjohann (see [Sto01]). It is nearly optimal, 'requiring about the same number of field operations as required for matrix multiplication' (see [Sto01, Abstract, p368]). It involves a divide-and-conquer strategy that produces matrices with entries in polynomial rings as intermediate results. Changing the scalars to a larger field or polynomial ring is something we wished to avoid as it creates additional complications in practical applications within a computer algebra system used for group and matrix algebra computations.
Storjohann's earlier deterministic algorithm (see [Sto98]) uses classical field arithmetic and requires $O(n^3)$ field operations. It first reduces the matrix to 'zig-zag form', using a mix of row and column operations, then produces the Smith normal form of a matrix with polynomial entries and finally the Frobenius normal form. In systems such as GAP, matrices over small finite fields are stored in a compressed form (compare Chapter II) that makes row operations simple, but column operations difficult. Restricting to one of these types of operations was one of our criteria.
A Monte Carlo minimal polynomial algorithm of Giesbrecht (see [Gie95]) that runs in 'nearly optimal time' contains some features we find desirable for practical implementation, namely his algorithm first constructs a 'modular cyclic decomposition' using random vectors, similar to our characteristic polynomial computation in Section III.5. However, further steps include a modification of the 'divide-and-conquer' Keller-Gehrig algorithm [KG85] and lead to a Las Vegas algorithm that computes a Frobenius form over an extension field and then the minimal polynomial. The field size over which the given matrix is written is assumed to be greater than n^2, and if this is not the case, it is suggested that an embedding into a larger extension field be used. Several of these features were undesirable for us.
In [AC97, Section 4] Augot and Camion propose a deterministic algorithm to compute the minimal polynomial of a matrix, which is to some extent similar to our algorithm. It is deterministic with complexity $O(n^3 + m^2 \cdot n^2)$ field operations, where m is the number of blocks in the shift Hessenberg form. They prove that the complexity is $O(n^3)$ in the average case. However, in the worst case it is $O(n^4)$, and no constants are provided in the complexity estimates. Although

the principal approach of their algorithm is similar to ours, the details differ very much from our algorithm and analysis.

An interesting commentary on various algorithms, together with some new algorithms is given by Eberly [Ebe00]. Eberly (see Theorem 4.2 in [Ebe00]) gives in particular a randomised algorithm for matrices over small fields that produces output from which (amongst other things) the minimal polynomial can be computed, at a cost of $O(n^3)$. The papers [Ebe00, Gie95, Ste97, Sto98, Sto01] contain references to other minimal polynomial algorithms. In all of the algorithms mentioned the asymptotic complexity statements give no information about the constants involved.

On a practical note, the minimal polynomial algorithm implemented in the GAP library is the one in [Ste97] and (although we have been unable to confirm this) we assume that this is the algorithm implemented in MAGMA [BC07].

(1.2) Outline of the chapter

In Section III.2 we introduce our notation, in Section III.3 we cite a few complexity bounds for basic algorithms. The next Section III.4 introduces order polynomials and derives a few results about them. Then we turn to the computation of the characteristic polynomial in Section III.5, since this is the first step in our minimal polynomial algorithm, which is described and analysed in Section III.7. We explain and modify the well-known algorithm to compute characteristic polynomials by introducing some randomisation, because this is later needed in the analysis of our main Monte Carlo algorithm. In Section III.6 we give some probability estimates that are also used later in the analysis. Section III.8 covers the deterministic verification of the results of our Monte Carlo algorithm. We describe in detail cases in which this verification is efficient and when it has a worse complexity. Finally, in Section III.9 we report on the performance of an implementation of our algorithm, including runtimes in realistic applications. We compare these times with the current implementation for minimal polynomial computations in the GAP library (see [GAP07]), and as mentioned above, we believe that MAGMA and GAP are both using the algorithm in [Ste97]. We show that our algorithm performs much better in important cases, and that our bounds on the computing cost are reflected in practical experiments.

III.2 Notation

Throughout this chapter $\mathbb{F}$ will be a fixed field. Although we envisage $\mathbb{F}$ to be a finite field for our applications, this is not necessary for most of our results. However in the later sections we use some probability estimates from Section III.6 that are only valid for finite fields.

By an *elementary field operation* we mean addition, subtraction, multiplication or division of two field elements. In all our runtime bounds we will assume that one elementary field operation takes a fixed amount of time and we simply count the number of such operations occurring in our algorithms.

We denote the set of $(m \times n)$-matrices over $\mathbb{F}$ by $\mathbb{F}^{m\times n}$ and the set of row vectors of length m by $\mathbb{F}^m$. For a vector $v \in \mathbb{F}^m$ we write v_i for its i-th component and for a matrix $M \in \mathbb{F}^{m\times n}$ we denote its i-th row, which is a row vector of length n, by $M[i]$. We use "row vector times matrix" operations and in general right modules throughout. If V is a vector space over $\mathbb{F}$ and W is a

subspace, the quotient space is denoted by V/W and its cosets by $v + W$ for $v \in V$. The $\mathbb{F}$-linear span of the vectors $v^{(1)}, \dots, v^{(k)} \in V$ is denoted by $\langle v^{(1)}, \dots, v^{(k)} \rangle_{\mathbb{F}}$.
If $M \in \mathbb{F}^{n \times n}$ is a matrix and $V = \mathbb{F}^n$, we have a natural action of M as an endomorphism of V by right multiplication. The same holds for every M-invariant subspace $W < V$ and for the corresponding quotient space V/W. We describe such a situation by saying that "the matrix M induces an action on the $\mathbb{F}$-vector space" V, W, V/W respectively.
Throughout, $\mathbb{F}[x]$ denotes the polynomial ring over $\mathbb{F}$ in the indeterminate x. For a square matrix M and a polynomial $p \in \mathbb{F}[x]$ we denote the evaluation of p at M by $p(M)$.
Whenever a matrix M induces an action on a vector space U, we will view U as a right $\mathbb{F}[x]$-module by letting x act like M, that is $v \cdot x := v \cdot M$ in the above examples. We denote the characteristic polynomial of this action by $\chi_{M,U}$. That is, $\chi_{M,U}$ is the characteristic polynomial of the $(\dim_{\mathbb{F}}(U) \times \dim_{\mathbb{F}}(U))$-matrix given by choosing a basis of U and writing the action of M induced on U as a matrix with respect to that basis. We use the same convention analogously for the corresponding minimal polynomial $\mu_{M,U}$. Furthermore, we denote the $\mathbb{F}[x]$-submodule of U generated by the vectors $u^{(1)}, \dots, u^{(n)}$ by $\langle u^{(1)}, \dots, u^{(n)} \rangle_M$.
We use the two functions

$$s^{(1)}(a,b) := \sum_{i=b+1}^{a} i \quad \text{and} \quad s^{(2)}(a,b) := \sum_{i=b+1}^{a} i^2 \tag{III.1}$$

for complexity expressions. Note that for $a > b > c$ we have $s^{(j)}(a,c) = s^{(j)}(a,b) + s^{(j)}(b,c)$ for $j \in 1, 2$ and

$$s^{(1)}(n,0) = s^{(1)}(n,-1) = \frac{n(n+1)}{2} \quad \text{and} \tag{III.2}$$

$$s^{(2)}(n,0) = s^{(2)}(n,-1) = \frac{n(n+1)(2n+1)}{6}. \tag{III.3}$$

For later complexity estimates we note the following inequalities.

III.2.1 Lemma (Some upper bounds)
If $n = \sum_{i=1}^{k} d_i$ for some $d_i \in \mathbb{N} \setminus \{0\}$ and $s_j := \sum_{i=1}^{j} d_i$ we have

$$\sum_{j=1}^{k} s_j \le \frac{n(n+1)}{2} \quad \text{and} \quad \sum_{j=1}^{k} s_j(s_j+1) \le \frac{n(n+1)(n+2)}{3}.$$

Proof: We claim that for fixed n both expressions are maximal if and only if all d_i are equal to one. We leave it to the reader to check that both totals increase if we replace d_j in some sequence $(d_i)_{1 \le i \le k}$ by the two numbers a and $d_j - a$ resulting in the new sequence $(d'_1, \dots, d'_{k+1}) := (d_1, d_2, \dots, d_{j-1}, a, d_j - a, d_{j+1}, \dots, d_k)$ of length $k+1$. □

III.3 Complexity bounds for basic algorithms

In some algorithms presented in later sections we use greatest common divisors of univariate polynomials. To analyse these algorithms we use the following bounds which arise from standard polynomial computation. We take this approach because the standard algorithms for polynomials are good enough for our complexity estimates in applications and we do not need the asymptotically best algorithms, discussion of which may be found conveniently in [vzGG03].

III.3.1 Proposition (Complexity of standard greatest common divisor algorithm)
Let $f, g \in \mathbb{F}[x]$ with $n := \deg f \geq \deg g =: m$, and $f = qg + r$ with $q, r \in \mathbb{F}[x]$ such that $r = 0$ or $\deg r < \deg g$. Then there is an algorithm to compute q and r that needs less than $2(m+1)(n-m+1)$ elementary field operations.
Furthermore, there is an algorithm to compute $\gcd(f, g)$ that needs less than $2(m+1)(n+1)$ elementary field operations.

III.3.2 Remark
We intentionally give bounds here which are not best possible, since we want the bound for the gcd computation to be symmetrical in m and n.

Proof of Proposition III.3.1: Use polynomial division and the standard gcd algorithm and count. See [vzGG03, Section 2.4 and Section 3.3] for smaller bounds that imply our symmetric bounds. $\square$

Polynomial factorisation

Some of our algorithms return partially factorised polynomials which facilitate later factorisation into irreducible factors. However, since the extent of this partial factorisation is difficult to estimate, we use in our analyses the complexity of finding the complete factorisation of a polynomial over a finite field as a product of irreducibles. We need such factorisations in our main algorithm. In keeping with our other methods we make use of standard polynomial factorisation procedures. Details can be found in Knuth [Knu98, 4.6.2] of a deterministic polynomial factorisation algorithm inspired by an idea of Berlekamp. Its cost is polynomial in both the degree n and the field size $|\mathbb{F}| = q$, as it requires $O(q)$ computations of greatest common divisors. Thus it works well only for q small. There is available a randomised (Las Vegas) version of the procedure which (for arbitrary q) will always return accurately the number r of irreducible factors of $f(x) \in \mathbb{F}[x]$, but for which there is a small non-zero probability that it will fail to find all the irreducible factors. It involves the procedure RANDOMVECTOR, which is discussed further in Subsection III.(5.1), to produce independent uniformly distributed random elements of an n-dimensional vector space over $\mathbb{F}$ for which a basis is known. Throughout this chapter logarithms are always taken to base 2.

III.3.3 Remark (POLYNOMIALFACTORISATION)
Suppose $f(x) \in \mathbb{F}[x]$ of degree $n \geq 1$ with r irreducible factors (counting multiplicities) and, if q is large, suppose additionally that we are given a real number ε such that $0 < \varepsilon < 1/2$. The number $\mathrm{fact}(n, q)$ of elementary field operations required to find a complete set of irreducible factors of $f(x)$ is at most

$8n^3 + (3qr + 17\log q)n^2$	deterministic algorithm
$O\big((\log \varepsilon^{-1})(\log n)(\xi_n + n^2 \log^3 q) + n^3 \log^2 q\big)$	Las Vegas algorithm

where ξ_n is an upper bound for the cost of one run of RANDOMVECTOR on $\mathbb{F}^n$. The Las Vegas algorithm may fail, but with probability less than ε.

III.4 Order polynomials

Let M be a matrix in $\mathbb{F}^{n\times n}$ that induces an action on an $\mathbb{F}$-vector space V.
We briefly recall the definition of the term “order polynomial”:

III.4.1 Definition (Order polynomial $\mathrm{ord}_M(v)$ and relative order polynomial)
The *order polynomial* $\mathrm{ord}_M(v)$ of a vector $v \in V$ is the monic polynomial $p \in \mathbb{F}[x] \setminus \{0\}$ of smallest degree such that $v \cdot p(M) = 0 \in V$. In particular $\mathrm{ord}_M(0) = 1$.
For an M-invariant subspace $W < V$, the *relative order polynomial* $\mathrm{ord}_M(v + W)$ (of v relative to W) is the order polynomial of the element $v + W \in V/W$ with respect to the induced action of M on V/W.

III.4.2 Remark
If we consider V as an $\mathbb{F}[x]$-module as in Section III.2, then p is the monic generator of the annihilator $\mathrm{ann}_{\mathbb{F}[x]}(v)$ of v in $\mathbb{F}[x]$.

The following observation follows immediately from the definition above.

III.4.3 Lemma (Relative order polynomials)
For an M-invariant subspace $W < V$ and $v \in V$, $\mathrm{ord}_M(v + W)$ is the monic polynomial $p \in \mathbb{F}[x] \setminus \{0\}$ of smallest degree such that $v \cdot p(M) \in W$.

We now turn to the question of how one computes the order polynomial of a vector $v \in V$. The basic idea is to apply the matrix M to the vector repeatedly computing a sequence

$$v, vM, vM^2, \ldots, vM^d,$$

until vM^d is a linear combination

$$vM^d = \sum_{i=0}^{d-1} a_i vM^i,$$

with $a_i \in \mathbb{F}$, for $0 \le i < d$. If d is minimal such that this is possible, we have

$$\mathrm{ord}_M(v) = x^d - \sum_{i=0}^{d-1} a_i x^i.$$

Although this procedure is simple and well-known, we present it in order to make explicit the number of elementary field operations needed. To this end we describe in detail the computation of solutions for the systems of linear equations involved.

III.4.4 Definition (Row semi echelon form)
A non-zero matrix $S = (S_{i,j}) \in \mathbb{F}^{m\times n}$ is in *row semi echelon form* if there are positive integers $r \le m$ and $j_1, \ldots, j_r \le n$ such that, for each $i \le r$, $S_{i,j_i} = 1$ and $S_{k,j_i} = 0$ for all $k > i$, and also $S_{k,j} = 0$ whenever $k > r$. For $i \le r$, column j_i is called the *leading column of row i*, and we write $\mathrm{lc}(i) = j_i$. A sequence of vectors $u^{(1)}, \ldots, u^{(m)} \in F^n$ is said to be in semi echelon form if the matrix with rows $u^{(1)}, \ldots, u^{(m)}$ is in row semi echelon form.

Note that in Definition III.4.4 we do not assume $j_1 < j_2 < \cdots < j_r$ which is the usual condition for an echelon form.

III.4.5 Definition (Semi echelon data sequence)
Let $Y \in \mathbb{F}^{m\times n}$ be a matrix with $m \leq n$ and of rank m. A *semi echelon data sequence for* Y is a tuple $\mathcal{Y} = (Y, S, T, l)$, where $S \in \mathbb{F}^{m\times n}$ is in row semi echelon form with leading column indices $l = (\mathrm{lc}(1), \dots, \mathrm{lc}(m))$, and $T \in \mathrm{GL}(m, \mathbb{F})$ with $TY = S$. Further, T is a lower triangular matrix, that is, for $T = (T_{i,j})$ we have $T_{i,j} = 0$ for $i < j$. For a semi echelon data sequence $\mathcal{Y}$ we call the number m its *length*, sometimes denoted $\mathrm{length}(\mathcal{Y})$. A semi echelon data sequence $\mathcal{Y}' = (Y', S', T', l')$ is said to *extend* $\mathcal{Y}$ if $\mathrm{length}(\mathcal{Y}') > \mathrm{length}(\mathcal{Y})$, the first $\mathrm{length}(\mathcal{Y})$ rows of Y' and S' form the matrices Y and S respectively, and the first $\mathrm{length}(\mathcal{Y})$ entries of l' form the sequence l.

III.4.6 Remark
(a) The idea of this concept is that for a matrix $S \in \mathbb{F}^{m\times n}$ in row semi echelon form it is relatively cheap to decide whether a given vector $v \in \mathbb{F}^n$ lies in the row space of S, and if so, to write it as a linear combination of the rows of S, that is, to find a vector $a \in \mathbb{F}^m$ such that $v = aS = aTY$ (see Algorithm 2). Thus, the vector v is expressed as a linear combination of the rows of Y using the vector aT as coefficients.
(b) We call a semi echelon data sequence *trivial* if $m = 0$. In this case, by convention, we take the row spaces of the empty matrices Y and S to be the zero subspace of $\mathbb{F}^n$, we denote the empty sequence in $\mathbb{F}^0$ by 0, and we interpret aS as the zero vector of $\mathbb{F}^n$.

We now present Algorithm 2, which is one step in the computation of a semi echelon data sequence for a matrix Y. We denote by $S[i]$ the i-th row of the matrix S and by $\mathrm{RowSp}(S)$ the row space of S.

III.4.7 Proposition (Correctness and complexity of Alg. 2: CLEANANDEXTEND***)***
The output of Algorithm 2 satisfies the Output specifications. Moreover, Algorithm 2 requires at most $2mn$ *field operations if* $v \in \mathrm{RowSp}(Y)$ *and* $(2m+1)n + (m+1)^2 + 1$ *field operations otherwise.*

III.4.8 Remark
(a) Given a semi echelon data sequence (Y, S, T, l) with $Y, S \in \mathbb{F}^{m\times n}$ and a vector $v \in \mathbb{F}^n$, Algorithm 2 tries to write v as a linear combination of the rows of S. If this is not possible, it constructs an extended semi echelon data sequence.

(b) For the case of finite fields a simple and useful optimisation is to reduce, where possible, the number of operations for vectors and matrices, for example, where a vector is multiplied by the zero scalar and the result is added to some other vector. This can reduce the number of operations for sparse vectors and matrices. Our estimates for the numbers of field operations then become over-estimates.

Proof of Proposition III.4.7: The proof of the correctness of Algorithm 2 is left to the reader. The **for** loop needs $2mn$ field operations if we count both multiplications and additions. If $v \in \mathrm{RowSp}(Y)$ then the algorithm terminates after this loop. On the other hand, if $v \notin \mathrm{RowSp}(Y)$,

Algorithm 2 CLEANANDEXTEND

Input: A semi echelon data sequence $\mathcal{Y} = (Y, S, T, l)$ with $Y, S \in \mathbb{F}^{m\times n}$, $v \in \mathbb{F}^n$ (possibly $m = 0$).

Output: A triple $(c, \mathcal{Y}', a')$ where c is TRUE if $v \in \mathrm{RowSp}(Y)$ and FALSE otherwise, $\mathcal{Y}'$ equals or extends $\mathcal{Y}$ respectively with $\mathrm{length}(Y') \le \mathrm{length}(Y) + 1$, and $a' \in \mathbb{F}^{\mathrm{length}(\mathcal{Y}')}$, such that $v = a'S'$.

```
$w := v$
$a := 0 \in \mathbb{F}^m$ {note that $w = v - aS$}
for $i = 1$ to $m$ do
  $a_i := w_{l_i}$
  $w := w - a_i \cdot S[i]$
end for     {still $w = v - aS$}
if w = 0 then
  return (TRUE, $(Y, S, T, l), a$)
else
  $j :=$ index of first non-zero entry in $w$
  $a' := [a \;\; w_j]$,   $l' := [l \;\; j]$,
  $Y' := \begin{bmatrix} Y \\ v \end{bmatrix}$,   $S' := \begin{bmatrix} S \\ w_j^{-1} \cdot w \end{bmatrix}$,   $T' := \begin{bmatrix} T & 0 \\ -w_j^{-1} \cdot aT & w_j^{-1} \end{bmatrix}$
  return (FALSE, $(Y', S', T', l'), a'$)
end if
```

then Algorithm 2 needs one inversion of the scalar w_j plus $2 \cdot \sum_{i=1}^m i = m(m+1)$ field operations for the vector times matrix multiplication aT, because T is a lower triangular matrix. This is altogether $m(m+1)+1$ operations. Finally, the scalar negation of w_j^{-1} and the multiplication of aT by $-w_j^{-1}$ needs another $m+1$ field operations, and a further n operations are needed for the computation of $w_j^{-1}w$ in S'. Thus the total number of field operations is at most $2mn + (m+1)^2 + 1 + n$. □

Having Algorithm 2 at hand we can now present Algorithm 3, which computes relative order polynomials. Since a (non-relative) order polynomial may be regarded as a relative order polynomial with respect to the zero subspace, Algorithm 3 can also be used to compute order polynomials, starting with the trivial semi echelon data sequence, (see Remark III.4.6 (b)).

III.4.9 Proposition (Correctness and complexity of Alg. 3: RELATIVEORDPOLY***)***
Let $\mathcal{Y} = (Y, S, T, l)$ be a semi echelon data sequence with $Y, S \in \mathbb{F}^{m\times n}$ (possibly $m = 0$), $v \in \mathbb{F}^n$, and $M \in \mathbb{F}^{n\times n}$ such that $W := \mathrm{RowSp}(Y)$ is M-invariant. The output of Algorithm 3 satisfies the Output specifications, and moreover if $d > 0$, then the rows $m+1, \dots, m+d$ of Y' are equal to $v, vM, \dots, vM^{d-1}$ respectively. Algorithm 3 requires at most

$$\begin{aligned} 2dn^2 \;&+\; (n+2)d + 2(m+d)n + 2(n+1)s^{(1)}(m+d-1, m-1) + \\ &+\; s^{(2)}(m+d-1, m-1) + 2s^{(1)}(m+d, 0) \end{aligned}$$

elementary field operations, where $s^{(1)}$ and $s^{(2)}$ are the functions defined in (III.1).

Algorithm 3 RELATIVEORDPOLY

Input: A semi echelon data sequence $\mathcal{Y} = (Y, S, T, l)$ with $Y, S \in \mathbb{F}^{m\times n}$ (possibly $m = 0$), $v \in \mathbb{F}^n$, and $M \in \mathbb{F}^{n\times n}$ such that $W := \mathrm{RowSp}(Y)$ is M-invariant.

Output: A triple $(p, \mathcal{Y}', b)$ consisting of the relative order polynomial $p := \mathrm{ord}_M(v + W)$ of degree d, a semi echelon data sequence $\mathcal{Y}' = (Y', S', T', l')$ of length $m + d$ equal to or extending $\mathcal{Y}$, and a vector $b \in \mathbb{F}^{m+d}$ such that $vM^d = bY'$.

```
$(Y', S', T', l') := (Y, S, T, l)$   {the primed variables change during the algorithm}
$v' := v$
$m' := m$                            {can be zero!}
loop
  $(c, (Y', S', T', l'), a) :=$ CLEANANDEXTEND$((Y', S', T', l'), v')$
                                     $\{T'Y' = S', v' = aS'\}$
  if $c =$ TRUE then
    leave loop
  end if
  $v' := v' \cdot M$
  $m' := m' + 1$
end loop            {at this stage $c =$ TRUE, $v' = aS'$, $m' = \mathrm{length}(\mathcal{Y}')$}
$d := m' - m$
$b := a \cdot T'$
$p := x^d - \sum_{i=0}^{d-1} b_{m+1+i}x^i$
return $(p, (Y', S', T', l'), b)$
```

III.4.10 Remark

Note that, if $d = 0$ then $S' = S$, so $v = b'Y \in W$, and in this case $p = 1$. Algorithm 3 successively considers the vectors $v, vM, \dots, vM^d$ (those are the successive values of v') until $vM^d + W$ lies in the subspace of V/W generated by the vectors $v+W, vM+W, \dots, vM^{d-1}+W$. The given matrix S together with Algorithm 2 defines a direct sum decomposition of the $\mathbb{F}$-vector space $V = \mathbb{F}^n = W \oplus W'$, where W' is the subspace of vectors having 0 in all positions occurring in the list l. Since $W' \cong V/W$, Algorithm 3 effectively computes in V/W by always 'cleaning out' vectors using S first.

Proof of Proposition III.4.9: We again leave the proof of correctness of Algorithm 3 to the reader. Algorithm 3 calls Algorithm 2 (CLEANANDEXTEND) exactly $d + 1$ times with the lengths of the input semi echelon data sequences being $m, m + 1, \dots, m + d$. After each but the last call to Algorithm 2 the value of c returned is FALSE, and after the last call the value of c is TRUE. Thus, by Proposition III.4.7, the number of steps needed for the $d + 1$ runs of Algorithm 2 is at most

$$\left(\sum_{i=m}^{m+d-1} ((2i + 1)n + (i + 1)^2 + 1)\right) + 2(m + d)n$$

$$= s^{(2)}(m + d - 1, m - 1) + (2n + 2)s^{(1)}(m + d - 1, m - 1) + (n + 2)d + 2(m + d)n.$$

In addition, we have to do d multiplications of v' with M, which require $2n^2$ elementary field operations each, and finally the computation of b requires $2s^{(1)}(m+d,0)$ elementary field operations, again since T' is a lower triangular matrix. Summing up gives the expression in the statement. □

We conclude this section with two lemmas that are used to compute absolute order polynomials using relative ones. We again view V as an $\mathbb{F}[x]$-module by letting x act like M. For $\{v^{(1)},\dots,v^{(m)}\} \subseteq V$, we denote by $\langle v^{(1)},\dots,v^{(m)}\rangle_M$ the submodule of V generated by the vectors $\{v^{(1)},\dots,v^{(m)}\}$, that is, the smallest M-invariant subspace containing $\{v^{(1)},\dots,v^{(m)}\}$. If $m=1$ then $\langle v^{(1)}\rangle_M$ is the $\mathbb{F}$-span of the set $\{v^{(1)}, v^{(1)}M,\dots,v^{(1)}M^{n-1}\}$. We call $\langle v^{(1)}\rangle_M$ a *cyclic subspace* relative to M.

III.4.11 Lemma (Order polynomials in cyclic subspaces)
Let $v\in V$, $W=\langle v\rangle_M < V$, and $p := \mathrm{ord}_M(v)$ with $d:=\deg(p)$. Then for each $w\in W$, there is a unique polynomial $q\in\mathbb{F}[x]$ of degree less than d such that $w=vq(M)$. Moreover,

$$\mathrm{ord}_M(w) = \frac{p}{\gcd(p,q)}.$$

Proof: The cyclic module $v\mathbb{F}[x]$ is isomorphic to $\mathbb{F}[x]/(p\mathbb{F}[x])$ because $p\mathbb{F}[x] = \mathrm{ann}_{\mathbb{F}[x]}(v)$. Under this isomorphism w is mapped to $q+p\mathbb{F}[x]$. Thus $\mathrm{ord}_M(w)\mathbb{F}[x] = \mathrm{ann}_{\mathbb{F}[x]}(q+p\mathbb{F}[x])$. Since $\mathrm{lcm}(p,q) = qp/\gcd(p,q)$ in the unique factorisation domain $\mathbb{F}[x]$, the statement follows as all occurring polynomials here are monic. □

III.4.12 Lemma (Absolute and relative order polynomials)
Let W be an M-invariant subspace of V, $v\in V$ and $q:=\mathrm{ord}_M(v+W)\in\mathbb{F}[x]$. Then

$$\mathrm{ord}_M(v) = q\cdot \mathrm{ord}_M(vq(M)).$$

Proof: Let $p:=\mathrm{ord}_M(v)\in\mathbb{F}[x]$. Then $vp(M)\in W$ (since $vp(M)=0$), and hence by Lemma III.4.3, q divides p. If $q=p$ then $vq(M)=0$ so $\mathrm{ord}_M(vq(M))=1$ and the result follows. Otherwise $\deg(q)<\deg(p)$, $w:=vq(M)\in\langle v\rangle_M$, and so by Lemma III.4.11, $p=\mathrm{ord}_M(w)\cdot\gcd(p,q) = \mathrm{ord}_M(w)\cdot q$. On the other hand, p divides $q\cdot\mathrm{ord}_M(vq)$ since the latter is a polynomial annihilating v. Letting now $r\in\mathbb{F}[x]$ such that $p=qr$, we get that r divides $\mathrm{ord}_M(vq)$ and since $0=vqr$ that $\mathrm{ord}_M(vq)$ divides r. Since both polynomials are monic, it follows that they are equal thus proving the claim. □

III.5 Computing the characteristic polynomial

In this section we present a version of a standard algorithm for computing the characteristic polynomial of a matrix together with its analysis. It differs from the standard version in its use of randomisation.

(5.1) Random vectors

Our characteristic polynomial algorithm, and later ones, make use of the algorithms RANDOMVECTOR and RANDOMVECTOR* that produce independent uniformly distributed random vectors, and independent uniformly distributed random non-zero vectors, respectively, in a given finite vector space for which a basis is known. The algorithms are invoked for spaces $\mathbb{F}^s$, for $s \in \mathbb{N}$, and for subspaces of V of the form

$$V(l) = \{v \mid v_{l_i} = 0 \quad \text{for} \quad 1 \le i \le m\}, \quad \text{where} \quad l = (l_1, \dots, l_m).$$

If l is the empty sequence then $V(l) = V$. For a semi echelon data sequence $\mathcal{Y} = (Y, S, T, l)$, the vector space V is the sum $V = V(l) \oplus \mathrm{RowSp}(S)$. If $b = \textsc{RandomVector}(\mathbb{F}^{\mathrm{length}(\mathcal{Y})})$, then bS is a uniformly distributed random vector of $\mathrm{RowSp}(S)$. Moreover we assume that for the disjoint spaces $\mathbb{F}^{\mathrm{length}(\mathcal{Y})}$ and $V(l)$ the algorithms RandomVector and RandomVector* are applied independently so that in particular, if $a = \textsc{RandomVector}{*}(V(l))$ then the sum $a + bS$ is a uniformly distributed random vector of $V \setminus \mathrm{RowSp}(S)$.
RandomVector and, if we neglect the possibility of obtaining the zero vector, also RandomVector*, could proceed by selecting independent uniformly distributed random field elements as coefficients of the basis vectors. For the subspace $V(l)$, we could put zeros into the entries occurring in l and make random selections of elements from $\mathbb{F}$ for each entry not in l. For this reason we denote by ξ_r an upper bound for the cost of RandomVector or RandomVector* applied to an r-dimensional space for one of these cases. If $r < s$ then $\xi_r \le \xi_s$ and $\xi_{r_1} + \xi_{r_2} \le \xi_{r_1+r_2}$, and we would expect ξ_r to vary linearly with r. In practical implementations the cost is much less than the cost of the field operations involved in the algorithm below.

(5.2) Characteristic polynomial algorithm

The characteristic polynomial algorithm below would terminate successfully without making random selections of vectors. However, the use of randomisation is key to our application of this algorithm for finding minimal polynomials. As in previous sections, let M be a matrix in $F^{n\times n}$ acting naturally on $V := \mathbb{F}^n$.

III.5.1 Proposition (Correctness and complexity of Algorithm 4)
Algorithm 4 satisfies the Output specifications, and furthermore $\mathcal{Y} = (Y, S, T, l)$, where $Y \in \mathbb{F}^{n\times n}$ is invertible with rows

$$v^{(1)}, v^{(1)}M, \dots, v^{(1)}M^{d_1-1}, v^{(2)}, v^{(2)}M, \dots, v^{(2)}M^{d_2-1}, \dots, v^{(k)}, v^{(k)}M, \dots, v^{(k)}M^{d_k-1},$$

where $d_i := \deg(p^{(i)})$ for $1 \le i \le k$. Further, for $1 \le i \le k$, $v^{(i)}$ is a uniformly distributed random element of $V \setminus W_{i-1}$, $v^{(i)}M^{d_i} = b^{(i)}Y$ and $p^{(i)} = \mathrm{ord}_M(v^{(i)} + W_{i-1})$, where $W_{i-1} := \langle v^{(1)}, \dots, v^{(i-1)}\rangle_M$ (for $i > 1$), an M-invariant subspace of V of dimension $s_{i-1} := \sum_{j=1}^{i-1} d_j$, and $W_0 = 0$ of dimension $s_0 = 0$. Moreover, Algorithm 4 requires at most

$$\frac{33}{6}n^3 + 4n^2 + \frac{3}{2}n$$

elementary field operations, plus $k\xi_n$ for the k calls to RandomVector* *and* RandomVector. *Neglecting the latter cost this is less than $6n^3$ elementary field operations, for sufficiently large n.*

III.5.2 Remark
We denote the semi echelon data sequences $\mathcal{Y}^{(i)}$ in the algorithm using indices to enable us to speak more easily about the intermediate results. However in practice we have only one variable $\mathcal{Y} = (Y, S, T, l)$, the entries of which are growing during the execution of the algorithm.

Algorithm 4 CharPoly

```
Input: $M \in \mathbb{F}^{n\times n}$
Output: A tuple $(k, (p^{(j)})_{1\le j\le k}, \mathcal{Y}, (b^{(j)})_{1\le j\le k})$, where each $p^{(j)} \in \mathbb{F}[x]$ and $\prod_{i=1}^{k} p^{(i)} = \chi_{M,V}$ is the characteristic polynomial of $M$ in its action on $V$, each $b^{(j)} \in \mathbb{F}^n$ and $\mathcal{Y}$ is a semi echelon data sequence of length $n$ with the properties specified in Proposition III.5.1.

$i := 0$
$\mathcal{Y}^{(0)}$ := a trivial semi echelon data sequence
while length$(\mathcal{Y}^{(i)}) < n$ do
  $i := i + 1$
  $a :=$ RandomVector$(\mathbb{F}^{\mathrm{length}(\mathcal{Y}^{(i-1)})})$
  $c :=$ RandomVector*$(V(l^{(i-1)}))$
  $v^{(i)} := aS^{(i-1)} + c$
        { $v^{(i)} \notin \mathrm{RowSp}(S^{(i-1)})$, where $\mathcal{Y}^{(i-1)} = (Y^{(i-1)}, S^{(i-1)}, T^{(i-1)}, l^{(i-1)})$}
  $(p^{(i)}, \mathcal{Y}^{(i)}, b^{(i)}) :=$ RelativeOrdPoly$(\mathcal{Y}^{(i-1)}, v^{(i)}, M)$
        {$b^{(i)} \in \mathbb{F}^{\mathrm{length}(\mathcal{Y}^{(i)})}$; we add $n - \mathrm{length}(\mathcal{Y}^{(i)})$ zeros to make $b^{(i)} \in \mathbb{F}^n$}
end while
$k := i$
return $(k, (p^{(j)})_{1\le j\le k}, \mathcal{Y}^{(k)}, (b^{(j)})_{1\le j\le k})$
```

III.5.3 Remark
Note that we do not multiply together the factors of $\chi_{M,V}$ because in our application of Algorithm 4 we do not need the product itself.

Proof of Proposition III.5.1: Most statements in the proposition follow immediately from Proposition III.4.9: Note that, because of the conventions explained in Remark III.4.6(b), in the first run of the 'while' loop $v^{(1)} = c$ is a random non-zero vector of V and $p^{(1)} = \mathrm{ord}_M(v^{(1)})$, and more generally, in the i^{th} run of the 'while' loop, Algorithm 4 chooses a vector $v^{(i)}$ that is a uniformly distributed random element of $V \setminus \mathrm{RowSp}(S^{(i-1)})$ and applies Algorithm 3. This immediately establishes all statements about (Y, S, T, l) including the one about the invertibility and the rows of Y. Also it is clear that $p^{(i)} = \mathrm{ord}_M(v^{(i)} + W_{i-1})$.
Next we show that $\prod_{i=1}^{k} p^{(i)} = \chi_{M,V}$. This follows by considering the matrix YMY^{-1}, which has the same characteristic polynomial as M. Considering the action of M with respect to the ordered basis of $\mathbb{F}^n$ given by the rows of Y, it follows from the construction that YMY^{-1} (written with respect to the standard basis) is equal to

$$\begin{bmatrix} C_1 & 0 & \cdots & 0 \\ B_1^{(2)} & C_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ B_1^{(k)} & \cdots & B_{k-1}^{(k)} & C_k \end{bmatrix},$$

where the matrix C_i is the companion matrix of the polynomial $p^{(i)}$, and the $B_j^{(i)}$, for $2 \le i \le k$ and $1 \le j \le i-1$, are matrices in $\mathbb{F}^{d_i \times d_j}$ with one non-zero row at the bottom and all other rows zero. If $b^{(i)} = (b_1^{(i)}, \ldots, b_n^{(i)})$, then the bottom row of $B_j^{(i)}$ is $(b_{s_{j-1}+1}^{(i)}, \ldots, b_{s_j}^{(i)})$. With this format

at hand it is clear that the characteristic polynomial of YMY^{-1} is equal to the product $\prod_{i=1}^{k} p^{(i)}$ because the C_i are companion matrices.
Finally we derive the statement about the number of elementary field operations needed by Algorithm 4. In the i^{th} run of the **while** loop, the cost of constructing the random vectors a and c is at most

$$\xi_{n-\mathrm{length}(\mathcal{Y}^{(i-1)})} + \xi_{\mathrm{length}(\mathcal{Y}^{(i-1)})} \leq \xi_n$$

(see Subsection III.(5.1)). The cost to compute $v^{(i)}$ is at most $2s_{i-1}n$ elementary field operations, where $s_0 = 0$, and for $i \geq 1$, $s_i = \sum_{j=1}^{i} d_j$ with $d_j = \deg p^{(j)}$. The cost of applying Algorithm RELATIVEORDPOLY is, by Proposition III.4.9, at most

$$2d_i n^2 + (n+2)d_i + 2s_i n + 2(n+1)s^{(1)}(s_i - 1, s_{i-1} - 1) + s^{(2)}(s_i - 1, s_{i-1} - 1) + 2s^{(1)}(s_i, 0)$$

elementary field operations, noting that the value of 'd' is d_i, the value of 'm' is s_{i-1}, $s_{i-1}+d_i = s_i$, and $s^{(1)}$, $s^{(2)}$ are the functions defined in (III.1).
We consider the different terms one by one, summing each over i from 1 to k. The total cost of constructing the random vectors is at most $k\xi_n$. Summing the terms $2s_{i-1}n$ gives $2n\sum_{i=1}^{k} s_{i-1}$, and summing the terms $2d_i n^2$ gives $2n^3$ since $\sum_{i=1}^{k} d_i = n$. Similarly, summing the terms $(n+2)d_i$ gives $(n+2)n$. From the terms $2s_i n$ we get a contribution of $2n\sum_{i=1}^{k} s_i$. The next two expressions involving the functions $s^{(1)}$ and $s^{(2)}$ sum to $2(n+1)s^{(1)}(n-1, 0) = n(n+1)(n-1)$ and $s^{(2)}(n-1, 0) = \frac{(n-1)n(2n-1)}{6}$ respectively, using (III.2) and (III.3) and the properties noted above them. Finally, the terms $2s^{(1)}(s_i, 0)$ sum to $2\sum_{i=1}^{k} s^{(1)}(s_i, 0) = \sum_{i=1}^{k} s_i(s_i + 1)$. Thus in total we obtain $k\xi_n$ plus

$$\begin{aligned} 2n^3 &+ n(n+1)(n-1) + \frac{(n-1)n(2n-1)}{6} + n(n+2) + 2n\sum_{i=1}^{k} s_{i-1} \\ &+ 2n\sum_{i=1}^{k} s_i + \sum_{i=1}^{k} s_i(s_i+1) \end{aligned}$$

elementary field operations. The first four of these terms sum to $\frac{10}{3}n^3 + \frac{3}{2}n^2 + \frac{7}{6}n$. Using Lemma III.2.1,

$$2n\sum_{i=1}^{k} s_{i-1} + 2n\sum_{i=1}^{k} s_i + \sum_{i=1}^{k} s_i(s_i+1) \leq 2n^2(n+1) + \frac{n(n+1)(n+2)}{6}$$

so the total cost is at most

$$\frac{33}{6}n^3 + 4n^2 + \frac{3}{2}n + k\xi_n.$$

For sufficiently large n this is less than $6n^3 + k\xi_n$. □

III.6 Probability estimates using the structure theory for modules

The basic idea of our minimal polynomial Algorithm 6 is to compute the order polynomials of a few random vectors under the action of a given matrix M and to prove that, with high probability,

their least common multiple is equal to the minimal polynomial of M. The purpose of this section is to use the structure theory of $V = \mathbb{F}^n$ as an $\mathbb{F}[x]$-module to derive probability estimates to be used in that proof.
First suppose that the characteristic polynomial of M is written as a product $\chi_{M,V} = \prod_{i=1}^t q_i^{e_i}$ with pairwise distinct irreducible polynomials $q_i \in \mathbb{F}[x]$ and positive integer multiplicities e_i.
Using [Jac85, Theorem 3.12] we can then write the $\mathbb{F}[x]$-module V as a direct sum of primary cyclic modules

$$V \cong \bigoplus_{i=1}^{t} \bigoplus_{j=1}^{m_i} w_{i,j}\mathbb{F}[x], \tag{III.4}$$

such that $\mathrm{ord}_M(w_{i,j}) = q_i^{f_{i,j}}$ with $e_i \geq f_{i,1} \geq f_{i,2} \geq \cdots \geq f_{i,m_i} \geq 1$ and $\sum_{j=1}^{m_i} f_{i,j} = e_i$ for $1 \leq i \leq t$.
The minimal polynomial $\mu_{M,V}$ is the least common multiple of the order polynomials of the vectors $(w_{i,j})_{1\leq i\leq t, 1\leq j\leq m_i}$, and hence is $\mu_{M,V} = \prod_{i=1}^t (q_i)^{f_{i,1}}$.
We use this structural description to derive the first probability bound for the case where $\mathbb{F} = \mathbb{F}_q$ is a finite field with q elements.

III.6.1 Proposition (Probability that a q_i has equal mult. in $\mu_{M,V}$ and $\mathrm{ord}_M(v)$)
Let $\mathbb{F} = \mathbb{F}_q$ be a finite field with q elements, let $V = \mathbb{F}^n$, let U be a (possibly zero) M-invariant subspace such that the multiplicity of q_i in $\mu_{M,U}$ is strictly smaller than in $\mu_{M,V}$, and let v be a uniformly distributed random element of $V \setminus U$. Then the multiplicity of q_i is the same in $\mathrm{ord}_M(v)$ and in $\mu_{M,V}$ with probability greater than $1 - q^{-\deg q_i}$.

Proof: By assumption the multiplicity of q_i in $\mu_{M,U}$ is less than its multiplicity $f := f_{i,1}$ in $\mu_{M,V}$. Let $w := w_{i,1}$, with $w_{i,1}$ as in (III.4), so that $V = X \oplus Y$ with X, Y invariant under M and $X = \langle w \rangle_M$. Then $\mu_{M,X} = q_i^f$. We may identify the primary cyclic $\mathbb{F}[x]$-module X with $w\mathbb{F}[x]$, which is isomorphic to the module $\mathbb{F}[x]/(q_i^f\mathbb{F}[x])$ via the map $wh \mapsto h + q_i^f\mathbb{F}[x]$ for $h \in \mathbb{F}[x]$. This module is uniserial with composition series

$$0 < \frac{q_i^{f-1}\mathbb{F}[x]}{q_i^f\mathbb{F}[x]} < \frac{q_i^{f-2}\mathbb{F}[x]}{q_i^f\mathbb{F}[x]} < \cdots < \frac{q_i\mathbb{F}[x]}{q_i^f\mathbb{F}[x]} < \frac{\mathbb{F}[x]}{q_i^f\mathbb{F}[x]}.$$

Thus, X has a unique maximal $\mathbb{F}[x]$-submodule, namely $X' := \langle wq_i(M)\rangle_M$, and X' has codimension $r := \deg(q_i)$ in X.
As discussed above, each vector $v \in V$ has a unique expression as $v = x + y$ with $x \in X$, $y \in Y$. Moreover $\mathrm{ord}_M(v)$ is the least common multiple of $\mathrm{ord}_M(x)$ and $\mathrm{ord}_M(y)$. In particular, if $x \notin X'$, then $\mathrm{ord}_M(x) = q_i^f$ and hence the multiplicity of q_i in $\mathrm{ord}_M(v)$ and $\mu_{M,V}$ is the same. The number of vectors $v = x + y$ with $x \notin X'$ is

$$|X \setminus X'| \cdot |Y| = (1 - \frac{1}{q^r})|X| \cdot |Y| = (1 - \frac{1}{q^r})q^n.$$

Each of these vectors v lies in $V \setminus U$ since the multiplicity of q_i in $\mu_{M,U}$ is less than f. Thus the probability, for a uniformly distributed random $v \in V \setminus U$, that the multiplicity of q_i in $\mathrm{ord}_M(v)$ and $\mu_{M,V}$ is the same is at least

$$(1 - \frac{1}{q^r})\frac{q^n}{|V \setminus U|} > 1 - \frac{1}{q^r}.$$

$\square$

III.6.2 Remark
If for some irreducible factor q_i we have $m_i > 1$ and $f_{i,1} = f_{i,2}$, then the above probability is even higher, because we can apply the above argument independently to two or more summands $w_{i,1}\mathbb{F}[x]$ and $w_{i,2}\mathbb{F}[x]$.

We now give a second probability bound which will be crucial in our Monte Carlo algorithm to compute the minimal polynomial. In that algorithm we choose a sequence of vectors $v^{(1)}, \dots, v^{(u)}$ such that $v^{(1)}$ is a uniformly distributed random element of $V \setminus \{0\}$, and for $i \geq 2$ we choose $v^{(i)}$ as a uniformly distributed random element of $V \setminus U$, where $U = \langle v^{(1)}, \dots, v^{(i-1)} \rangle_M$. We hope to find $\mu_{M,V}$ as the least common multiple of the orders of these vectors.

III.6.3 Proposition (Probability that an lcm of order polynomials equals $\mu_{M,V}$)
Let $\mathbb{F} = \mathbb{F}_q$ be a finite field with q elements. Suppose a sequence of vectors $v^{(1)}, \dots, v^{(u)} \in V$ is chosen as follows: $v^{(1)}$ is a uniformly distributed random element of $V \setminus \{0\}$, and for $i > 1$, $v^{(i)}$ is a uniformly distributed random element of $V \setminus \langle v^{(1)}, \dots, v^{(i-1)} \rangle_M$. Let

$$f := \operatorname{lcm}(\operatorname{ord}_M(v^{(1)}), \operatorname{ord}_M(v^{(2)}), \dots, \operatorname{ord}_M(v^{(u)})).$$

Then the probability that $f = \mu_{M,V}$ is greater than

$$1 - \sum_{i=1}^{t} q^{-u \deg q_i}.$$

Proof: Consider the random experiment described in the statement. We first examine one irreducible factor q_i. Let E_i denote the event that the multiplicity of q_i in f is strictly smaller than the multiplicity $f_{i,1}$ of q_i in $\mu_{M,V}$. Furthermore, for $1 \leq j \leq u$, let F_j be the event that the multiplicity of q_i in $\operatorname{ord}_M(v^{(j)})$ is strictly smaller than $f_{i,1}$.
Note that the F_j are not stochastically independent since we choose $v^{(j)}$ outside of the space $\langle v^{(1)}, \dots, v^{(j-1)} \rangle_M$. However, $E_i = F_1 \cap F_2 \cap \cdots \cap F_u$, because f is the least common multiple of the order polynomials of the $v^{(j)}$. By Proposition III.6.1 applied with $U = \{0\}$, the probability $\operatorname{Prob}(F_1)$ is less than $q^{-\deg q_i}$. Moreover, in the situation that $F_1 \cap \cdots \cap F_j$ holds and $j < u$, we apply Proposition III.6.1 with the subspace $U := \langle v^{(1)}, \dots, v^{(j)} \rangle_M$ to conclude that the conditional probability $\operatorname{Prob}(F_{j+1} | F_1 \cap \cdots \cap F_j)$ is less than $q^{-\deg q_i}$. Thus we have

$$\begin{aligned}\operatorname{Prob}(E_i) = \operatorname{Prob}(F_1) \quad \cdot \quad & \operatorname{Prob}(F_2|F_1) \cdot \operatorname{Prob}(F_3|F_1 \cap F_2) \cdot \cdots \\ & \cdots \cdot \operatorname{Prob}(F_u|F_1 \cap \cdots \cap F_{u-1}) < q^{-u \deg q_i}.\end{aligned}$$

Finally we consider all the different irreducible factors q_i.
Even though the events $E_1, \dots, E_t$ may not be stochastically independent, we have

$$\operatorname{Prob}(E_1 \cup \cdots \cup E_t) \leq \sum_{i=1}^{t} \operatorname{Prob}(E_i) < \sum_{i=1}^{t} q^{-u \deg q_i}$$

as claimed. □

Figure III.1: Overview of the matrix YMY^{-1}

$$\left[\begin{array}{cccccc|cccccc|c|cccccc}
0&1&&&& & &&&&& & & &&&&&\\
&0&1&&0& & &&&&& & & &&&&&\\
&&\ddots&&\ddots& & &&&&& & & &&&&&\\
&0&&0&1& & &&&&& & & &&&&&\\
&&&&0&1 & &&&&& & & &&&&&\\
\ast&\ast&\ast&\ast&\ast&\ast & &&&&& & & &&&&&\\
\hline
&&&&& & 0&1&&&& & & &&&&&\\
&&&&& & &0&1&&0& & & &&&&&\\
&&&&& & &&\ddots&&\ddots& & & &&&&&\\
&&&&& & &0&&0&1& & & &&&&&\\
&&&&& & &&&&0&1 & & &&&&&\\
\ast&\ast&\ast&\ast&\ast&\ast & \ast&\ast&\ast&\ast&\ast&\ast & & &&&&&\\
\hline
&&&\vdots&& & &&&\vdots&& & \ddots & &&&&&\\
\hline
&&&&& & &&&&& & & 0&1&&&&\\
&&&&& & &&&&& & & &0&1&&0&\\
&&&&& & &&&&& & & &&\ddots&&\ddots&\\
&&&&& & &&&&& & & &0&&0&1&\\
&&&&& & &&&&& & & &&&&0&1\\
\ast&\ast&\ast&\ast&\ast&\ast & \ast&\ast&\ast&\ast&\ast&\ast & \ast & \ast&\ast&\ast&\ast&\ast&\ast
\end{array}\right]$$

III.7 Computing minimal polynomials

Our minimal polynomial algorithm runs Algorithm 4 as its first step. So assume, from now on, that we have already run Algorithm 4 and obtained all the output it produces, in particular the basis given by the rows of the matrix Y (as in Proposition III.5.1)

$$(v^{(1)}, v^{(1)}M, \ldots, v^{(1)}M^{d_1-1}, \ldots, v^{(k)}, v^{(k)}M, \ldots, v^{(k)}M^{d_k-1}),$$

the relative order polynomials $p^{(i)} = \mathrm{ord}_M(v^{(i)} + W_{i-1})$, and the vectors $b^{(i)}$ for $1 \leq i \leq k$. Also assume that we have factorised all the polynomials $p^{(i)}$ as products $p^{(i)} = \prod_{j=1}^{t} q_j^{e_{i,j}}$ of irreducible polynomials $(q_j)_{1\leq j\leq t}$.
The matrices M and YMY^{-1} have the same characteristic and minimal polynomials. Also the order polynomials $\mathrm{ord}_M(v)$ and $\mathrm{ord}_{YMY^{-1}}(vY^{-1})$ are equal for every $v \in V$ and thus also the order polynomials $\mathrm{ord}_M(vY)$ and $\mathrm{ord}_{YMY^{-1}}(v)$ are equal for every $v \in V$.
For the convenience of the reader we display the matrix YMY^{-1} in Figure III.1. Note in particular that the matrix is sparse, provided that the degrees d_i are not too small. Due to the special form of YMY^{-1} it is much more efficient to compute the images of vectors under YMY^{-1} than under M. This is crucial in the analysis of our algorithms. Therefore we will from now on do all computations of order polynomials with respect to YMY^{-1}.

Set $M' := YMY^{-1}$ and $W_i' := W_iY^{-1}$ for $1 \le i \le k$. Note that we have $v^{(i)} = e^{(s_{i-1}+1)}Y$ for $1 \le i \le k$, where $e^{(1)}, \dots, e^{(n)}$ is the standard basis of $\mathbb{F}^n$. (That is, $e^{(i)}$ contains exactly one 1 in position i and otherwise zeros. Recall that $s_i = \sum_{j=1}^{i} d_j$ with $s_0 = 0$.) Furthermore, for $1 \le i \le k$, the space $W_i = \langle v^{(1)}, \dots, v^{(i)} \rangle_M$ is equal to the space $\{vY \mid v \in \mathbb{F}^n \text{ with } v_j = 0 \text{ for } j > s_i\}$. Thus, the space W_i' is the $\mathbb{F}$-linear span $\langle e^{(1)}, e^{(2)}, \dots, e^{(s_i)} \rangle_{\mathbb{F}}$ and we have a filtration

$$0 = W_0' < W_1' < W_2' < \cdots < W_k' = V$$

such that each quotient W_i'/W_{i-1}' is an M'-cyclic space generated by the coset represented by the standard basis vector $e^{(s_{i-1}+1)}$.
We begin by presenting Algorithm 5 which computes the absolute order polynomial of a vector with respect to the matrix YMY^{-1}, using all the data acquired during Algorithm 4. We will apply this later in the minimal polynomial algorithm to the first few of the vectors $e^{(s_{i-1}+1)}$ produced during a run of Algorithm 4. Note that for the analysis it is crucial that a number z such that the vector v lies in W_z' is given as input to the algorithm.

Algorithm 5 ORDPOLY

Input: $M, k, (Y, S, T, l), (p^{(j)})_{1\le j\le k}, (b^{(j)})_{1\le j\le k}$ as returned by CHARPOLY, an integer z with $1 \le z \le k$, $v \in W_z'$, and the factorisation $p^{(j)} = \prod_{r=1}^{t} q_r^{e_{j,r}}$ for all $j \le k$
Output: A list of factorised polynomials, the product of which is $\mathrm{ord}_{YMY^{-1}}(v)$

```
$i := z$          {will run down to 1}
$f := [\,]$          {empty list}
repeat
  $h := \sum_{j=1}^{d_i} v_{s_{i-1}+j}x^{j-1}$ {this is the $q$ from Lemma III.4.11 for computing $\mathrm{ord}_{M'}(v + W_{i-1}')$}
  if $h \ne 0$ then
    $\hat{g} := p^{(i)}/\gcd(h, p^{(i)})$          {factorised}
    add $\hat{g}$ to list $f$
    compute product $g$ of factors in $\hat{g}$ {$g = \hat{g}$, but the latter only exists in factorised form}
    if $i > 1$ then
      $v := v \cdot g(YMY^{-1})$          {see Proposition III.7.1 for this computation}
    end if
  end if
  $i := i - 1$
until $i = 0$
return $f$
```

***III.7.1 Proposition (Correctness and complexity of Algorithm 5:** ORDPOLY*)*
Let $\mathbb{F} = \mathbb{F}_q$ be a field with q elements. The output of Algorithm 5 satisfies the Output specifications. Moreover, Algorithm 5 requires at most

$$\sum_{j=1}^{z}\left(4d_j^2 + 3d_js_j + 2d_j\sum_{r=1}^{j}s_r\right) \le (\frac{z}{2}+9)s_z^2$$

elementary field operations, where $d_j = \deg p^{(j)}$, $s_j = \sum_{r=1}^{j} d_j$ for $j \geq 1$ and $s_0 = 0$; and this is less than n^3 for n sufficiently large.

Proof: Since we are computing an order polynomial with respect to the matrix $M' = YMY^{-1}$ we can always use the form of this matrix as displayed in Figure III.1.
The basic idea of Algorithm 5 is to use Lemmas III.4.11 and III.4.12 applied to the filtration

$$0 = W'_0 < W'_1 < W'_2 < \cdots < W'_k = V.$$

Starting with $i := z$ and the original v lying in the space W'_z, the variable i runs downwards until 1. In each step, Algorithm 5 computes the relative order polynomial $g := \mathrm{ord}_{M'}(v + W'_{i-1})$ for the then current value of $v \in W'_i$. This assertion follows from Lemma III.4.11 noting that, by our discussion above, $p^{(i)} = \mathrm{ord}_M(v^{(i)} + W_i) = \mathrm{ord}_{M'}(e^{(s_{i-1}+1)} + W'_{i-1})$. Next $vg(M')$ is evaluated, which lies in W'_{i-1} by Lemma III.4.3, and the induction can go on with i replaced by $i-1$. The product of the polynomials in the list f returned is the product of all the relative order polynomials computed in the repeat loop, and this is equal to $\mathrm{ord}_{M'}(v)$, by Lemma III.4.12.
To count the number of elementary field operations is a bit complicated here. Note first that by assumption we already know a factorisation of all the $p^{(i)} = \prod_{j=1}^{t} q_j^{e_{i,j}}$ into irreducible factors. Now $\gcd(h, p^{(i)})$ is equal to the product of the greatest common divisors $\gcd(h, q_j^{e_{i,j}})$, for $j \leq t$. Since the degrees of the polynomials $q_j^{e_{i,j}}$ sum up to the degree d_i of $p^{(i)}$, finding these gcd's, by Proposition III.3.1, requires at most

$$2(\deg(h) + 1) \cdot \sum_{j=1}^{t} \big(\deg(q_j)e_{i,j} + 1\big)$$

field operations, which is at most $4d_i^2$ since $\deg(q_j)e_{i,j} + 1 \leq 2\deg(q_j)e_{i,j}$. Note that this is a rather crude estimate. At this stage we know all multiplicities of the q_j in $\gcd(h, p^{(i)})$ and thus in $g := p^{(i)}/\gcd(h, p^{(i)})$. Thus we have computed g in factorised form, which is denoted by $\hat{g}$ in Algorithm 5.
Now we discuss the number of operations needed to evaluate $vg(M')$. Due to the sparseness of M', a multiplication of a vector w of W'_i from the right by M' needs only a shift (which we neglect here) and an addition of a multiple of the non-zero part of $b^{(r)}$ for $1 \leq r \leq i$ requiring $2s_r$ operations for each r. Thus computing wM' requires at most $\sum_{r=1}^{i} 2s_r$ elementary operations. Note that wM' lies in W'_i still. If $f(x) \in \mathbb{F}_q[x]$ of degree d, say $f(x) = \sum_{r=0}^{d} c_r x^r$, then we can compute $wf(M') = \sum_{r=0}^{d} c_r w(M')^r$ by first computing $w(M')^r \in W'_i$ for $1 \leq r \leq d$ at a cost of at most $2d \sum_{r=1}^{i} s_r$, next computing $c_r w(M')^r$ for $0 \leq r \leq d$ at a cost of at most $(d+1)s_i$, and then adding these vectors at a further cost of at most ds_i, making a total cost to compute $wf(M')$ of at most $(2d+1)s_i + 2d\sum_{r=1}^{i} s_r$ elementary field operations.
The polynomial $g(x)$ is available in factorised form, say $g(x) = \prod_{s=1}^{u} f_s(x)$ with $\deg f_s = m_s$, where $\sum_{s=1}^{u} m_s = \deg g = d_i$. From the previous paragraph we see that $vg(M')$ can be computed at a cost of at most

$$\sum_{s=1}^{u} \left((2m_s + 1)s_i + 2m_s \sum_{r=1}^{i} s_r \right) = (2d_i + u)s_i + 2d_i \sum_{r=1}^{i} s_r \leq 3d_i s_i + 2d_i \sum_{r=1}^{i} s_r$$

elementary field operations.

Subsequent runs of the repeat loop require similar numbers of elementary operations, with i replaced by j, where $i-1 \geq j \geq 1$. Thus Algorithm 4 needs at most

$$\sum_{j=1}^{z} \left(4d_j^2 + 3d_j s_j + 2d_j \sum_{r=1}^{j} s_r \right)$$

elementary field operations, as claimed in the proposition.
To find a simpler upper bound we look at the terms one by one. The last and most important term can be bounded by

$$\begin{aligned} 2\sum_{j=1}^{z}\left(d_j \sum_{r=1}^{j} s_r\right) &= 2\sum_{r=1}^{z} s_r \left(\sum_{j=r}^{z} d_j\right) = 2\sum_{r=1}^{z} s_r(s_z - s_{r-1}) \\ &= 2\sum_{r=1}^{z} s_r(s_z - s_r) + 2\sum_{r=1}^{z} s_r d_r \leq (\frac{z}{2}+2)\cdot s_z^2, \end{aligned}$$

since the function $t(s_z - t)$ has maximum value $s_z^2/4$ for t in the interval $[0, s_z]$.
The second term $3\sum_{j=1}^{z} d_j s_j$ is at most $3s_z^2$, and the term $4\sum_{j=1}^{z} d_j^2$ is at most $4s_z \sum_{j=1}^{z} d_j = 4s_z^2$. Altogether this amounts to a bound of $(\frac{z}{2}+9)s_z^2$ as claimed. Asymptotically, this is bounded above by n^3 in the worst case as $n \to \infty$. □

Now we present our main procedure, Algorithm 6.

Algorithm 6 MINPOLYMC

```
Input: $M \in \mathbb{F}_q^{n\times n}$, $\varepsilon$ with $0 < \varepsilon < 1/2$.
Output: A tuple $(b, f)$, where $b$ is either TRUE or UNCERTAIN and $f \in \mathbb{F}_q[x]$
        (see Proposition III.7.2 for details).
$((p^{(j)})_{1\leq j\leq k}, (Y, S, T, l), (b^{(j)})_{1\leq j\leq k})$ := CHARPOLY($M$)
Factorise all $p^{(j)} = \prod_{r=1}^{t} q_r^{e_{j,r}}$
Determine the least $u \in \mathbb{N}$ such that $\sum_{r=1}^{t} q^{-u \deg q_r} \leq \varepsilon$
$u := \min\{u, k\}$
$f := \mathrm{lcm}(p^{(1)}, \dots, p^{(k)})$
for $i = 2$ to $u$ do
  $f$ := lcm($f$, ORDPOLY($M, k, (Y, S, T, l), (p^{(j)})_{1\leq j\leq k}, (b^{(j)})_{1\leq j\leq k}, i, e^{(s_{i-1}+1)}$))
end for
if $u = k$ or $\deg f = n$ then
  return (TRUE, $f$)
else
  return (UNCERTAIN, $f$)
end if
```

III.7.2 Proposition (Correctness and complexity of Algorithm 6: MINPOLYMC***)***
Given a matrix $M \in \mathbb{F}_q^{n\times n}$ and a number ε with $0 < \varepsilon < 1/2$, Algorithm 6 returns a tuple (b, f), where b is either TRUE *or* UNCERTAIN *and $f \in \mathbb{F}_q[x]$ is a polynomial. With probability at least*

$1-\varepsilon$ the polynomial $f=\mu_{M,\mathbb{F}_q^n}$, and if $b=$ TRUE, then $f=\mu_{M,\mathbb{F}_q^n}$ is guaranteed. Moreover, if $f\neq\mu_{M,\mathbb{F}_q^n}$, then f is a proper divisor of $\mu_{M,\mathbb{F}_q^n}$ and every irreducible factor of $\mu_{M,\mathbb{F}_q^n}$ divides f. The number of elementary field operations needed by Algorithm 6 is bounded above by

$$\mathrm{char}(n,q)+\mathrm{fact}(n,q)+\sum_{i=1}^{u}(\frac{i}{2}+9)s_i^2,$$

where $\mathrm{char}(n,q)$ is an upper bound for the number of elementary field operations needed to compute the characteristic polynomial (see Proposition III.5.1), $\mathrm{fact}(n,q)$ is an upper bound for the number of elementary field operations needed to factorise each of a set of polynomials over $\mathbb{F}_q$ whose degrees sum to n (see Subsection III.3). Moreover either $u=k$, or $u<k$ and $\sum_{j=1}^{t}q^{-u\deg q_j}\leq\varepsilon$.
For n sufficiently large and fixed ε, the number of elementary field operations needed is less than

$$6n^3+\mathrm{fact}(n,q)+\frac{1}{3}\lceil\frac{\log n-\log\varepsilon}{\log q}\rceil^2\cdot n^2,$$

which is less than $7n^3+\mathrm{fact}(n,q)$ (plus the cost of computing at most n random vectors in Algorithm 4).

III.7.3 Remark (Overall complexity)
Note that if we use a randomised polynomial factorisation algorithm (necessary for large q), then the algorithm can be modified to allow for a possible failure of factorisation of the 'Factorise' step (line 2). Thus Theorem III.1.1 follows from Proposition 6. An upper bound for the term $\mathrm{fact}(n,q)$ in the complexity bound is given in Remark III.3.3, and this yields an upper bound in Proposition III.7.2 of $O(n^3\log^3 q)$ for n sufficiently large and fixed ε.

III.7.4 Remark (Modification to get a deterministic version)
Algorithm 6 can be changed into a deterministic one by running the 'i-loop' for i up to k, instead of u. An upper bound of the cost is then given by replacing u by k in the formula for the cost in Proposition III.7.2.
If $k>u$, then these additional $k-u$ runs of the 'i-loop' may be viewed as a 'verification algorithm'. By Proposition 6, the additional cost of these extra runs is

$$\sum_{i=u+1}^{k}(\frac{i}{2}+9)s_i^2\leq s_k^2\left(\frac{(k+u+1)(k-u)}{4}+9(k-u)\right)=s_k^2\frac{(k-u)(k+u+37)}{4},$$

and for sufficiently large n this cost is less than $n^4/4$ field operations.

Proof of Proposition III.7.2: Algorithm 6 first computes the characteristic polynomial of M in its action on $\mathbb{F}_q^n$ and its factorisation. This computation provides firstly the irreducible factors q_j of the minimal polynomial that allow us to determine u, and secondly the input needed for running Algorithm 5 to compute the order polynomials of $v^{(2)},\ldots,v^{(u)}$. Thirdly, it also yields a nice base change matrix Y such that these order polynomials with respect to the matrix M can in fact be determined using Algorithm 3 for the vectors $e^{(s_1+1)},\ldots,e^{(s_{u-1}+1)}$ since we have $\mathrm{ord}_M(v^{(i)})=\mathrm{ord}_{YMY^{-1}}(e^{(s_{i-1}+1)})$ for $1\leq i\leq k$. Note that $p^{(1)}=\mathrm{ord}_{YMY^{-1}}(e^{(1)})=\mathrm{ord}_M(v^{(1)})$.

By Proposition III.5.1, the vector $v^{(j)}$ is a uniformly distributed random element of $V\setminus\{0\}$ if $j = 1$, or $V \setminus \langle v^{(1)}, \dots, v^{(j-1)}\rangle_M$ if $j > 1$. Hence, by Propositions III.6.3 and III.7.1, the probability that f after termination of Algorithm 6 is equal to $\mu_{M,\mathbb{F}_q^n}$ is at least $1 - \varepsilon$.
From the discussion at the beginning of Section III.7, $\mu_{M,\mathbb{F}_q^n}$ is the least common multiple of the k polynomials $\mathrm{ord}_M(v^{(1)}), \dots, \mathrm{ord}_M(v^{(k)})$, and hence if $u = k$ then $f = \mu_{M,\mathbb{F}_q^n}$. This also implies, since the initial value of f is $\mathrm{lcm}(p^{(1)}, \dots, p^{(k)})$, that the returned polynomial f divides $\mu_{M,\mathbb{F}_q^n}$ and every irreducible factor of $\mu_{M,\mathbb{F}_q^n}$ divides f. In particular, if $\deg f = n$ then we must have $f = \chi_{M,\mathbb{F}_q^n} = \mu_{M,\mathbb{F}_q^n}$. Thus, if (TRUE, f) is returned then $f = \mu_{M,\mathbb{F}_q^n}$ is guaranteed.
The number of elementary field operations needed follows from Propositions III.5.1 and III.7.1 and summing. Note that, after the factorisations computed in line 2 of the algorithm, we neglect the forming of least common multiples and the products here, because all results from Algorithm 5 come already factorised into irreducible factors. We can thus compute the least common multiples by taking maxima of multiplicities. Hence the first displayed upper bound is proved.
For the asymptotic complexity bound we have to consider the initial value of the number u, namely the least integer u such that $\sum_{j=1}^{t} q^{-u \deg q_j} \le \varepsilon$. The largest value of this sum occurs when all the q_j have degree 1, and as there are then at most n such polynomials, $\sum_{j=1}^{t} q^{-u \deg q_j} \le nq^{-u}$. Thus u is at most the least integer such that $nq^{-u} \le \varepsilon$, namely

$$u_0 := \left\lceil \frac{\log n + \log(\varepsilon^{-1})}{\log q} \right\rceil$$

and the value of u used in the algorithm is at most $\min\{u_0, k\} \le u_0$. By Proposition III.5.1, the asymptotic value of $\mathrm{char}(n, q)$ is less than $6n^3$ for n sufficiently large, (plus the cost $k\xi_n$ of making k random selections of vectors). By Proposition III.7.1, the number of elementary field operations used for the computation of the $u \le u_0$ order polynomials is at most

$$\sum_{i=1}^{u} (\frac{i}{2} + 9)s_i^2 \le \frac{u(u+1)}{4} s_u^2 + 9us_u^2 \le u_0 \left(\frac{u_0 + 37}{4}\right) n^2,$$

which, for sufficiently large n and fixed ε, is less than $\frac{1}{3}u_0^2 n^2 < n^3$. □

III.8 Deterministic verification

In this section we explain how the probabilistic result of our Monte Carlo algorithm can be verified deterministically. We begin by discussing cases that can be handled rather cheaply, before we present several general verification procedures, all of which, unfortunately, have a worst-case cost of $O(n^4)$ field operations.
All notation from previous sections remains in force. The first result follows immediately from Proposition III.7.2.

III.8.1 Proposition (Cases, in which the result is already proven to be correct)
If the output polynomial of Algorithm 6 is equal to $\chi_{M,\mathbb{F}_q^n}$, then the output is (TRUE, $\chi_{M,\mathbb{F}_q^n}$) *and correct.*

For the next result observe that, if Algorithm 6 is modified so that the **for** loop is run k times, then the resulting polynomial f is guaranteed to be the minimal polynomial, giving a deterministic algorithm with proven result. (Proof of correctness is given in the proof of Proposition III.7.2.)

III.8.2 Proposition (Case of few random vectors chosen during computation of $\chi_{M,\mathbb{F}_q^n}$)
If $k \leq \sqrt{n}$, and the **for** *loop in Algorithm 6 is run k times, then the output polynomial is $\mu_{M,\mathbb{F}_q^n}$. The overall cost of this modification of Algorithm 6 is at most*

$$\operatorname{char}(n,q) + \operatorname{fact}(n,q) + \frac{1}{4}n^3 + \frac{37}{4}n^{5/2}$$

elementary field operations.

Proof: The only change to the complexity estimate is for the number of elementary field operations in the second last line of the proof of Proposition III.7.2:

$$\sum_{i=1}^{k}(\frac{i}{2}+9)s_i^2 \leq \frac{k(k+1)}{4}s_k^2 + 9ks_k^2 \leq \frac{k(k+1)}{4}n^2 + 9kn^2 \leq \frac{1}{4}n^3 + \frac{37}{4}n^{5/2}.$$

The rest follows from Proposition III.7.2. □

For the case of large k, we may use the procedure suggested in Remark III.7.4 as a verification algorithm, at a cost of $O(k^2n^2)$ field operations. Two alternative verification procedures are given below. The first involving evaluation on vectors is given in Proposition III.8.3 and the second using null space computations in Proposition III.8.7.

III.8.3 Proposition (Verification by evaluation on vectors)
For the output (UNCERTAIN, f) *of Algorithm 6 one can verify $f = \mu_{M,\mathbb{F}_q^n}$ using at most $dn(k-u)(k+u+4)$ elementary field operations, where u and k are as in Proposition III.7.2 and $d = \deg f$.*

Proof: The idea here is to check whether $e^{(s_{i-1}+1)}f(YMY^{-1})$ is equal to zero, for $u+1 \leq i \leq k$, by direct evaluation using the techniques described in the proof of Proposition III.7.1. Recall first that the result f comes in factorised form. Since $e^{(s_{i-1}+1)}$ lies in W_i', the arguments in the proof of Proposition III.7.1 show that this evaluation can be done using at most $3ds_i + 2d\sum_{r=1}^{i} s_r$ elementary field operations. Thus, an upper bound for the total cost for all these evaluations is

$$3d\sum_{i=u+1}^{k} s_i + 2d\sum_{i=u+1}^{k}\sum_{r=1}^{i} s_r.$$

The first term is bounded above by $3dn(k-u)$. As to the second term, for $1 \leq j \leq u+1$, the value s_j occurs in this expression with coefficient $2d(k-u)$, while for $u+2 \leq j \leq k$, it occurs with coefficient $2d(k-j+1)$. Thus the second term is at most

$$2dn(u+1)(k-u) + 2dn(k-u)(k-u-1)/2 = dn(k-u)(k+u+1).$$

Adding this to the upper bound for the first term we get at most $dn(k-u)(k+u+4)$ as claimed in the proposition. □

For the following discussion we need a lemma:

III.8.4 Lemma (Cost of evaluation of a polynomial at a matrix)
Let $M \in \mathbb{F}^{n\times n}$ be a matrix and $f \in \mathbb{F}[x]$ a polynomial with degree $d < n$. Then the evaluation $f(M)$ can be computed using at most $2dn^3$ elementary field operations.

Proof: We take $2n^3$ elementary field operations as an upper bound for a matrix multiplication. The computation of the powers $M^2, M^3, \dots, M^d$ needs at most $2(d-1)n^3$ elementary field operations. The multiplication, for each $i = 1, \dots, d$, of M^i by a coefficient of f and addition of the result to the already computed matrix (the sum of previous terms) needs another $2dn^2$ elementary field operations. Finally, the constant term of f has to be added along the diagonal, which is yet another n elementary field operations. Since $d + 1 \le n \le 2n^2$, this is altogether at most $2dn^3$ as claimed. □

Of course, this immediately implies:

III.8.5 Corollary (Small degree minimal polynomial)
If $\deg \mu_{M,\mathbb{F}_q^n} < n$, then the output of Algorithm 6 can be verified by evaluation using at most

$$2 \cdot n^3 \cdot \deg \mu_{M,\mathbb{F}_q^n}$$

elementary field operations.

III.8.6 Remark
Note that using [AC97, Theorem 2] we could lower the complexity in Lemma III.8.4 to $O(\sqrt{d}n^3)$ provided we stored $O(\sqrt{d})$ matrices in memory at the same time. However, since storing a matrix in $\mathbb{F}^{n\times n}$ needs $O(n^2)$ of memory, this approach would often become impractical before a concrete problem would become intractable because of time constraints. We use our estimates in Lemma III.8.4 because of these practical considerations. However, in some practical situations, an improved polynomial evaluation algorithm using more memory may be suitable.

We now present Algorithm 7 that can be run after Algorithm 6 to verify the correctness of the resulting polynomial deterministically.

III.8.7 Proposition (Deterministic minimal polynomial verification)
If Algorithm 7 is called with candidate minimal polynomial $\prod_{i=1}^{t} q_i^{f_i}$ from Algorithm 6, then it either returns TRUE *or a positive integer j. In the former case, $\mu_{M,\mathbb{F}_q^n} = \prod_{i=1}^{t} q_i^{f_i}$, while in the latter case the multiplicity of q_j in $\mu_{M,\mathbb{F}_q^n}$ is greater than f_j. The number of elementary field operations required by Algorithm 7 is at most*

$$n^3 \cdot \sum_{i=1}^{t} (2 \deg q_i + 2\lceil \log f_i \rceil + 1)\,.$$

Proof: Let $r_i := \deg q_i$ for $i = 1, \dots, t$. We again view $\mathbb{F}^n$ as $\mathbb{F}[x]$-module as in Section III.6 by letting x act as right multiplication by M. By [Jac85, Theorem 3.12], it is isomorphic to a direct sum of primary cyclic $\mathbb{F}[x]$-modules

$$\mathbb{F}^n \cong \bigoplus_{i=1}^{t} \bigoplus_{j=1}^{m_i} w_{i,j}\mathbb{F}[x],$$

Algorithm 7 MINPOLY VERIFICATION

Input: $M \in \mathbb{F}^{n\times n}$, $\chi_{M,V} = \prod_{i=1}^{t} q_i^{e_i}$ (factorised), and a candidate $\prod_{i=1}^{t} q_i^{f_i}$ for $\mu_{M,\mathbb{F}_q^n}$ (factorised), all data from Algorithm 6
Output: TRUE or a positive number j (see Proposition III.8.7 for details).

```
for $i = 1$ to $t$ do
  if $f_i < e_i$ then
    $M' := q_i(YMY^{-1})^{f_i}$
    $d := \dim_{\mathbb{F}}(\ker(M'))$
    if $d < \deg(q_i) \cdot e_i$ then
      return $i$
    end if
  end if
end for
return TRUE
```

such that $\mathrm{ord}_M(w_{i,j}) = q_i^{f_{i,j}}$ with $e_i \geq f_{i,1} \geq f_{i,2} \geq \cdots \geq f_{i,m_i} \geq 1$ and $\sum_{j=1}^{m_i} f_{i,j} = e_i$. Thus, for each i, q_i occurs in $\mu_{M,\mathbb{F}_q^n}$ with multiplicity $f_{i,1}$, and so in particular $f_i \leq f_{i,1}$. The element $q_i^{f_i}$ acts invertibly on all direct summands $w_{i',j}\mathbb{F}[x]$ with $i' \neq i$ since q_i is irreducible and every order polynomial of a non-zero vector in such a direct summand is a power of $q_{i'}$, by Lemma III.4.11. For $i' = i$ however, the dimension of the kernel of the action of $q_i^{f_i}$ on $w_{i,j}\mathbb{F}[x]$ is $r_i \cdot \min\{f_i, f_{i,j}\}$. Thus the dimension of the kernel of the action of $q_i^{f_i}$ on the whole of $\mathbb{F}^n$ is equal to

$$r_i \sum_{j=1}^{m_i} \min\{f_i, f_{i,j}\} \leq r_i \sum_{j=1}^{m_i} f_{i,j} = r_i e_i$$

with equality if and only if $f_i \geq f_{i,1}$. Since $f_i \leq f_{i,1}$, equality holds above if and only if f_i is equal to the multiplicity $f_{i,1}$ of q_i in $\mu_{M,\mathbb{F}_q^n}$. Therefore, Algorithm 7 always returns the result as stated in the proposition.

As to the cost, Algorithm 7 evaluates q_i at YMY^{-1}, which needs at most $2r_i n^3$ elementary field operations by Lemma III.8.4. It then takes the result to the f_i^{th} power, which can be done by repeated squaring with at most $2n^3\lceil \log f_i \rceil$ elementary field operations, and finally computes the dimension of a null space, which can be done with at most n^3 elementary field operations (compute a semi echelon basis of the row space of the matrix). Note that we are not using the sparseness of YMY^{-1} here. □

III.8.8 Remark (Performance in practice)

The cost in Proposition III.8.7 is much smaller than n^4 in many cases. One of the worst cases occurs when $\chi_{M,\mathbb{F}^n}$ contains lots of different factors of degree 1 each occurring with multiplicity 3, and all the f_i are equal to 2. Then Algorithm 7 has to square about $n/3$ matrices and compute the null spaces of the results. This amounts to about $2n^4/3$ elementary field operations, which is only about twice as fast as directly evaluating the minimal polynomial at M. Note that even in this case only about every sixth entry of YMY^{-1} is different from zero.

III.8.9 Remark (Further optimisations)
As in each of our procedures there are some simplifications we could make in practice which do not reduce the worst case complexity estimates. For example, in Algorithm 7, there is no need to compute the kernel of $q_i(YMY^{-1})^{f_i}$ if the irreducible q_i does not divide any of the relative order polynomials $p^{(j)}$ for $u < j \leq k$.

III.9 Performance in practice

In this section we give some experimental evidence concerning the performance of Algorithm 6 in comparison with that of algorithms currently implemented in the GAP library (see [GAP07]). All computations were done on a machine with an Intel Core 2 Quad CPU Q6600 running at 2.40 GHz with 8 GB of main memory and two times 4 MB of second level cache.
We were unable to confirm that MAGMA [BC07] uses an algorithm based on the canonical forms algorithm of Alan Steel presented in [Ste97] for computing minimal polynomials, although this is indicated in [Ste97, Abstract] and on the web
(see `http://magma.maths.usyd.edu.au/magma/htmlhelp/text347.htm`).
Our colleague Colva Roney-Dougal kindly ran the Baby Monster example matrix M_2 in the MAGMA system and the resulting times were roughly equivalent to the timing in the column "Lib" of Figure III.2, suggesting that this is indeed the case. Since the minimal polynomial algorithm in the GAP library is also based on the algorithm in [Ste97], we did not conduct extensive comparison tests of our algorithm on MAGMA.

(9.1) Guide to the test data

The timing results are in Figure III.2, all times are in seconds. The column marked "n" contains the dimension of the matrix, the column marked "q" the number of elements of the base field. The columns marked "Lib" and "AS" contain the times needed for one run of the minimal polynomial algorithm based on [Ste97] as implemented in the GAP library, and as implemented (by the author) in the GAP language, respectively. The column "MC" contains the total time for our Monte Carlo algorithm as presented in Algorithm 6. The next three columns marked "Spin", "Fact" and "OrdP" contain the times for the three phases of this algorithm respectively, namely the first phase to compute the characteristic polynomial via relative order polynomials, the second phase to factor all factors of the characteristic polynomial and count multiplicities, and the third phase to compute some absolute order polynomials to guess the minimal polynomial. Finally, the last column marked "Ver." contains the time for the deterministic verification via Algorithm 7. The maximal error probability for our Monte Carlo algorithm was $\varepsilon = 1/100$ for all runs.

(9.2) The test matrices

Next, we describe the matrices $M_1, \ldots, M_{10}$ we used.
(a) The matrices M_1 and M_1' were purely random matrices from $\mathbb{F}_3^{1000\times 1000}$ with all entries chosen with uniform distribution from the field $\mathbb{F}_3$. Such matrices are with very high probability cyclic, that is, their characteristic and minimal polynomials are equal. Usually, Algorithm 4 only has to pick very few random vectors for such matrices. The **for** loop of Algorithm 7 quickly

checks whether the least common multiple of the relative order polynomials (which is the input candidate polynomial) already has degree n. It turned out that M_1' was cyclic but not M_1, and this explains the big differences in the runtimes for these matrices.

(b) The matrix M_2 is one coming from actual applications. Namely, it is the matrix $a+b+ab$, where the two matrices $a, b \in \mathbb{F}_2^{4370\times 4370}$ describe the action of two standard generators of the Baby monster sporadic simple group on its smallest faithful simple module over $\mathbb{F}_2$. The matrices a and b were downloaded from the WWW-Atlas of group representations (see [WWT+99]). The matrix $a+b+ab$ is interesting because it is one of the algebra words that is used in the MeatAxe (see [Par84] and [HR94]) to compute composition series of modules and we could very well imagine using the minimal polynomial instead of the characteristic polynomial in some places in the MeatAxe.

The reason why the standard algorithm for the minimal polynomial performed rather badly on this matrix is that its characteristic polynomial has irreducible factors of degrees 1, 1, 2, 4, 6, 88, 197, 854 and 934 with respective multiplicities 2, 2277, 4, 1, 1, 1, 1, 1 and 1. Therefore the standard algorithm spins up large subspaces many times.

(c) The matrices M_3 to M_7 were constructed in the following way: In the language of $\mathbb{F}[x]$-modules we chose the order polynomials of the generators of their primary cyclic submodules, that is we chose the minimal polynomials on the primary cyclic submodules. For irreducible factors of degree one this amounts to choosing the sizes and numbers of the Jordan blocks occurring in the Jordan normal form of the matrix. After writing down the corresponding normal form of the matrix we conjugated it with a random element of the general linear group to get a dense matrix with the same normal form.

For $M_3 \in \mathbb{F}_5^{600\times 600}$ we chose one cyclic summand with minimal polynomial $(x-\zeta_5)^{300}$ plus 300 summands with minimal polynomial $x-\zeta_5$, where $\zeta_5 \in \mathbb{F}_5$ is a primitive root. This is a typical case in which our Monte Carlo algorithm and the deterministic verification both perform very well in comparison with older techniques. The reason for this is that the high dimensional cyclic subspace is spun up many times in the standard minimal polynomial algorithm as for the matrix M_2.

For $M_4 \in \mathbb{F}_3^{1200\times 1200}$ we chose 400 cyclic summands with minimal polynomial $(x+1)^2$ plus 400 cyclic summands with minimal polynomial $(x+1)$. In contrast with the matrix M_3, our algorithms performed very well in this case but they were not much faster than the older techniques, since the standard algorithm run on M_4 does not spin up many large cyclic subspaces.

For $M_5 \in \mathbb{F}_{251}^{600\times 600}$ we chose 200 different linear factors $x-\alpha$ and for each added one cyclic space with minimal polynomial $(x-\alpha)^2$ and one with $x-\alpha$. This example originally was a worst case scenario for our deterministic verification. However, since $k=2$ is quite small, a deterministic verification can be done relatively cheaply as described in Proposition III.8.2 and Remark III.7.4, even though the integer u in Algorithm 6 is only 1. The deterministic verification Algorithm 7 ran very slowly (more than 300 seconds) in this example.

For $M_6 \in \mathbb{F}_2^{2391\times 2391}$ we chose the irreducible polynomial $f(x) = x^3+x^2+1 \in \mathbb{F}_2[x]$ of degree 3 and added cyclic spaces with respective minimal polynomials f^{400}, f^{200}, f^{100}, f^{50}, f^{25}, f^{12}, f^6, f^3 and f.

For $M_7 \in \mathbb{F}_{3^4}^{220\times 220}$ we chose an irreducible polynomial $f(x) \in \mathbb{F}_{10}[x]$ of degree 10 and added cyclic spaces with respective minimal polynomials f^{10}, f, f^2, f^3, f, f^2 and f^3.

Figure III.2: Timings for minimal polynomial computation

M	q	n	Lib	AS	MC	Spin	Fact	OrdP	Ver.	k
M_1	3	1000	1.95*	0.65	13.7	0.33	13.3	0.05	0	2
M_1'	3	1000	1.31*	0.68	0.32	0.32	0	0	0	2
M_2	2	4370	12975	3098	5.74	3.80	1.10	0.83	3.02	2212
M_3	5	600	59.5	21.0	0.33	0.16	0.08	0.08	0.19	301
M_4	3	1200	2.00*	0.45	0.44	0.38	0.06	0.01	0.06	800
M_5	251	600	2.9	3.3	3.26	2.82	0.55	0.04	0	2
M_6	2	2391	14.6	3.3	2.25	0.91	0.18	1.15	1.02	9
M_7	243	220	0.77	0.88	0.36	0.34	0.01	0.01	0.21	7
M_8	17	400	0.46	0.20	0.048	0.032	0.012	0.004	0.00	399
M_9	17	400	0.26	0.20	0.23	0.23	0	0	0	1
M_{10}	17	400	0.26	0.19	0.22	0.22	0	0	0	1

* averaged over 10 runs

(d) The matrices M_8 and M_9 were standard generators of GL(400, 17), conjugated by the pseudo-random element M_{10} of the same group. Note that M_8 had order 16 while M_9 and M_{10} had very high order and were cyclic matrices. We chose these examples because they may be typical of difficult cases in an application of the minimal polynomial algorithm for computing the projective order of a matrix.

Our algorithm very quickly discovered that the least common multiple of the relative order polynomials was already equal to the characteristic polynomial.

Chapter IV

Further linear algebra algorithms

In this chapter we collect for the sake of completeness some linear algebra algorithms together with their complexity analysis that are used in later chapters. At the end we comment on two major open problems, namely the discrete logarithm problem in finite fields and integer factorisation, which come up in the analysis of matrix group algorithms.

IV.1 Order and projective order of a matrix

This section is about the computation of the order and the projective order of a matrix $M \in \mathbb{F}_q^{n\times n}$. After defining these terms we describe the ideas of the method in [CLG97] in particular to show two things: First that computing the minimal polynomial of a matrix is a crucial step in the computation of its order, and second that integer factorisations of some of the numbers $q^i - 1$ for $1 \le i \le n$ can be needed in the process (see Section IV.5).

IV.1.1 Definition/Proposition (Order and projective order)
Let $\mathbb{F}$ be a field and $M \in \mathbb{F}^{n\times n}$ an invertible matrix. The *order* of M is the least natural number o such that M^o is equal to the identity matrix $\mathbf{1}$. The *projective order* of M is the least natural number p such that M^p is a scalar matrix, that is, a scalar multiple of the identity matrix. It follows immediately by division with remainder in the exponent that o divides every natural number k for which M^k is equal to $\mathbf{1}$ and that p divides every natural number k for which M^k is a scalar multiple of $\mathbf{1}$. Thus p divides in particular o and it follows immediately that, if $M^p = \lambda \cdot \mathbf{1}$ with $\lambda \in \mathbb{F}$, then o is p times the order of the scalar λ. □

For a monic polynomial $f \in \mathbb{F}[X]$ with non-zero constant term we define the *order* (*projective order*, respectively) of $X + f\mathbb{F}[X]$ in $\mathbb{F}[X]/f\mathbb{F}[X]$ as the least natural number p such that X^p is congruent to 1 (a scalar, respectively) modulo f or equivalently, that $X^p - 1$ ($X^p - \lambda$, respectively) is divisible by f (for some $\lambda \in \mathbb{F}$). The order and projective order of a matrix M as above and that of X modulo the minimal polynomial of M are linked by the following lemma.

IV.1.2 Lemma (Orders and projective orders)
Let $\mathbb{F}$ be a field and $M \in \mathbb{F}^{n\times n}$ an invertible matrix. Then the (projective) order of M is equal to the (projective) order of $X + \mu_M\mathbb{F}[X]$ modulo the minimal polynomial μ_M of M.

Proof: The polynomial $X^p - \lambda$ is divisible by μ_M if and only if $M^p = \lambda\mathbf{1}$ for all $\lambda \in \mathbb{F}$. □

In the following we use this lemma to switch between matrices and polynomials as seems appropriate for the argumentation.
Now we want to discuss the computation of both the order and the projective order of a matrix or its minimal polynomial respectively.
Let $f \in \mathbb{F}[X]$ be a monic polynomial with non-zero constant term. By the Chinese Remainder theorem the factor ring $\mathbb{F}[X]/f\mathbb{F}[X]$ is isomorphic to

$$\mathbb{F}[X]/f\mathbb{F}[X] \cong \prod_{i=1}^{k} \mathbb{F}[X]/f_i^{e_i}\mathbb{F}[X],$$

where $f = \prod_{i=1}^{k} f_i^{e_i}$ is the factorisation of f into its pairwise distinct irreducible monic factors f_i. Using this isomorphism the (projective) order of $X + f\mathbb{F}[X]$ is equal to the least common multiple of the (projective) orders of the $X + f_i^{e_i}\mathbb{F}[X]$ in $\mathbb{F}[X]/f_i^{e_i}\mathbb{F}[X]$. Thus, as a first step we factorise f completely and now determine the (projective) order of $X + g^e\mathbb{F}[X]$ for an irreducible monic polynomial g of degree d with non-zero constant term.
From now on we switch back to matrices. Let $C \in \mathbb{F}^{d\times d}$ be the companion matrix of g and N the $(de \times de)$-block matrix with C along the main block diagonal, $(d \times d)$-identity matrices along the block diagonal directly above the main diagonal and zero blocks elsewhere:

$$N = \begin{bmatrix} C & \mathbf{1} & 0 & \cdots & 0 \\ 0 & C & \mathbf{1} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & C & \mathbf{1} \\ 0 & \cdots & \cdots & 0 & C \end{bmatrix}.$$

The minimal polynomial μ_N of N over $\mathbb{F}$ is g^e which can be seen as follows: Let K be the splitting field of g over $\mathbb{F}$. Since g is irreducible and thus has no multiple roots, the matrix C is similar to a diagonal matrix over K, that is, there is an invertible $T \in K^{d\times d}$ such that TCT^{-1} is diagonal with pairwise disjoint eigenvalues. Thus, conjugating N with the block matrix having T along the main block diagonal and permuting rows and columns suitably shows that the Jordan normal form of N over K consists of d blocks of size e, thus every eigenvalue of C occurs in the minimal polynomial of N with multiplicity e. Thus the minimal polynomial of N over K is g^e and thus also $\mu_N = g^e$.
We assume from now on that $\mathbb{F}$ is the finite field $\mathbb{F}_q$ with q elements of characteristic p. Writing $N = \tilde{C} + \tilde{\mathbf{1}}$, where $\tilde{C}$ is the matrix with only C along the main block diagonal and $\tilde{\mathbf{1}}$ accordingly we have $\tilde{C} \cdot \tilde{\mathbf{1}} = \tilde{\mathbf{1}} \cdot \tilde{C}$ and thus $N^k = \sum_{i=0}^{k} \binom{k}{i} \tilde{C}^{k-i} \tilde{\mathbf{1}}^i$. This immediately implies that the $(d \times d)$-block of N^k in position $(j, j+i)$ is $\binom{k}{i} C^{k-i}$ for all $1 \leq j \leq d - i$.
Therefore the (projective) order of N can be determined in the following way: The number $\lceil \log_p(e) \rceil$ is the number of factors p that have to occur in k, such that all binomial coefficients $\binom{k}{i} \equiv 0 \pmod{p}$ for $1 \leq i < e$ (where we set $\binom{k}{i} = 0$ for $i > k$). Let l be the (projective) order of $X + g\mathbb{F}[X]$ in the field $\mathbb{F}[X]/g\mathbb{F}[X]$. Then the (projective) order of N is the product $p^{\lceil \log_p(e) \rceil} \cdot l$. Note that determining the number l involves computing in the field extension $\mathbb{F}[X]/g\mathbb{F}[X]$.
In practice one takes $q^d - 1$ (respectively $\frac{q^d-1}{q-1}$) as an upper bound for the (projective) order of $X + g\mathbb{F}[X]$ and then uses the "bounded order algorithm" described in [CLG97, Section 2] to get

the actual (projective) order. However, this algorithm depends on the integer factorisation (see Section IV.5) of $q^d - 1$ since it uses a divide-and-conquer approach using this factorisation. Altogether we can conclude the following result.

IV.1.3 Proposition (Computing the (projective) order of a matrix)
Let M be a matrix in $\mathbb{F}_q^{n\times n}$. Assume that the minimal polynomial of M is known and completely factorised into irreducible factors, and that all integer factorisations of $q^i - 1$ for $1 \leq i \leq n$ are known (see Section IV.5).
Then the (projective) order of M can be computed in $O(n^3 \cdot \log_2(q) \cdot \log_2(n \log_2(q)))$ field operations.

Proof: Use the same arguments as in the proof of [CLG97, ORDER ALGORITHM]. □

IV.1.4 Remark
This shows that every improvement of the algorithm to compute the minimal polynomial also helps to compute the (projective) order.

IV.2 Solving systems of linear equations

This section describes and analyses the problem of solving a system of linear equations. This can be formulated as follows:
Let $\mathbb{F}$ be a field, $Y \in \mathbb{F}^{m\times n}$ and $b \in \mathbb{F}^{1\times n}$. Find all $x \in \mathbb{F}^{1\times m}$ with $xY = b$.
Sometimes the problem has to be solved for more than one b and sometimes it is enough to find one solution x. In particular the homogeneous case $b = 0$ is important. Here the set of solutions is called the *nullspace* of M.
The general approach to this problem is standard, thus we are mostly interested in the complexity. The basic procedure is Algorithm 8: By running all rows of Y successively through Algorithm 2 we find three things in the process: First a subsequence $t = (t_1, t_2, \ldots, t_r)$ of $(1, 2, \ldots, m)$ with $1 \leq t_1 < t_2 < \cdots < t_r \leq m$ such that the matrix $Y' \in \mathbb{F}^{r\times n}$ consisting of the rows with numbers $t_1, t_2, \ldots, t_r$ of Y has rank r. Secondly we get a semi echelon data sequence $\mathcal{Y} = (Y', S, T, l)$ for Y' as defined in Definition III.4.5. Thirdly, for every row of Y that is a linear combination of the rows above it in Y, we find a linear relation of the rows of Y and all these relations span the nullspace of Y.
We leave the details and the proof that this procedure does what we claim to the reader. The following proposition analyses the cost of Algorithm 8.

IV.2.1 Proposition (Complexity of Algorithm 8)
If r is the rank of M, Algorithm 8 needs at most

$$\frac{r(r+1)(2r+1)}{6} + nr^2 + 2rn(m-r) + r$$

elementary field operations. This is $O(nm^2)$.

Algorithm 8 SEMIECHELONISE

```
Input: A matrix $Y \in \mathbb{F}^{m \times n}$.
Output: Indices $1 \leq t_1 < t_2 < \cdots < t_r$, a matrix $Y' \in \mathbb{F}^{r \times n}$ with semi echelon data sequence
        $\mathcal{Y} = (Y', S, T, l)$ and a matrix $N \in \mathbb{F}^{(m-r) \times n}$ whose rows span the nullspace of $Y$
$\mathcal{Y}$ := empty semi echelon sequence for rows with length $n$
$N$ := empty matrix for rows with length $n$
$t$ := empty sequence
for $i = 1$ to $m$ do
  $(f, \mathcal{Y}, a) :=$ CLEANANDEXTEND($\mathcal{Y}, Y[i]$)
  if $f =$ TRUE then
    Append a row to $N$ using $a$
  else
    Append $i$ to sequence $t$
  end if
end for
Return: $(t, \mathcal{Y}, N)$
```

Proof: We add up the maximal number of elementary field operations needed by the calls to CLEANANDEXTEND using Proposition III.4.7. We start with the calls that extend the semi echelon data sequence:

$$\begin{aligned}\sum_{i=0}^{r-1} \big((2i+1)n + (i+1)^2 + 1\big) &= \frac{r(r-1)}{2} \cdot 2n + rn + \frac{r(r+1)(2r+1)}{6} + r \\ &= nr^2 + \frac{r(r+1)(2r+1)}{6} + r,\end{aligned}$$

where we use Formulas III.2 and III.3. In the worst case that the first r rows of Y are linearly independent the number of operations used to clean the rows that do not extend the semi echelon data sequence is at most $(m-r) \cdot 2rn$. For the asymptotic statement we use $r \leq \min\{m, n\}$. □

After Algorithm 8 has completed, the rows of the matrix N span the solution space of the system of linear equations $xY = 0$. For a given $0 \neq b \in \mathbb{F}^{1 \times n}$ we have to run the "cleaning" part of Algorithm 2 once. The result is either that the system has no solutions or a vector $a \in \mathbb{F}^{1 \times r}$ such that $b = aS = aTY'$ for the matrix S in $\mathcal{Y}$ that is in row semi echelon form. Since Y' consists of a selection of the rows of Y, the vector aT contains all the necessary information to put together a solution x with $b = xY$. The set of all solutions is then equal to $\{x + n \mid n \in \mathrm{RowSp}(N)\}$.

IV.2.2 Corollary (Solving a system of linear equations)

The set of solutions of the system of linear equations $xY = 0$ can be read off from the output of Algorithm 8. For the system $xY = b$ it can be derived using $2rn + r(r+1)$ elementary field operations.

Thus such a system can be solved in $O(nm^2)$ elementary field operations.

Proof: The above method needs one call to CLEANANDEXTEND using at most $2rn$ elementary field operations and the vector matrix multiplication aT with the lower triangular matrix T needs at most $2 \cdot \frac{r(r+1)}{2}$ elementary field operations. □

IV.3 Inverting matrices

Inverting a matrix $M \in \mathbb{F}^{n\times n}$ is the same as solving the system of linear equations $xM = e_i$ for all the standard basis vectors $e_i \in \mathbb{F}^{1\times n}$. By Proposition IV.2.1 and Corollary IV.2.2 we get the following result.

IV.3.1 Proposition (Inverting a matrix)
Let $M \in \mathbb{F}^{n\times n}$ be an invertible matrix. Then the inverse can be computed using at most

$$\frac{n(n+1)(2n+1)}{6} + 4n^3 + n^2 + n$$

elementary field operations, which is asymptotically less than $5n^3$ and this is $O(n^3)$.

Proof: We first run Algorithm 8 using at most

$$\frac{n(n+1)(2n+1)}{6} + n^3 + n$$

elementary field operations by Proposition IV.2.1. Then we solve $xM = e_i$ for $1 \le i \le n$ using a total of

$$n(2n^2 + n(n+1)) = 3n^3 + n^2$$

elementary field operations. Summing up gives the claim. □

IV.4 The discrete logarithm problem

IV.4.1 Problem (The discrete logarithm problem in finite fields)
Let $q = p^k$ be a power of a prime p. Then the multiplicative group of the finite field $\mathbb{F}_q$ is cyclic, say $\mathbb{F}_q^\times = \langle \zeta \rangle$ for a fixed element ζ of order $q-1$.
The discrete logarithm problem (DLP) in the finite field $\mathbb{F}_q$ is the following:

$$\boxed{\text{Given } y \in \mathbb{F}_q^\times, \text{ find } a \in \mathbb{Z} \text{ such that } y = \zeta^a.}$$

Computing discrete logarithms

Obviously this problem can be solved easily using an algorithm that simply tries all powers of ζ. However, this simplistic approach has complexity $O(q)$ in the worst case, which very quickly becomes impractical as q grows.
Better methods are the Shanks and Pollard methods (see [Odl00, Section 3]) with complexity $O(q^{1/2})$, the Index Calculus Method due to Kraitchik (see [McC90]) and the more recent Number Field Sieve methods (see [Odl00, Section 4]).

An overview of the current state of the art to solve this problem including lots of references is given in [Odl00]. For some values of q (in particular for "small p") there are known algorithms with an asymptotic complexity of

$$\exp((C + o(1))(\log q)^{1/3}(\log\log q)^{2/3})$$

for some constant C, but it is still an active area of research to come up with methods with this complexity for all values of q.
Currently there seems to be no hope for algorithms that have polynomial complexity in $\log q$ on classical computers. The development of quantum computers might change this in the future.
For practical purposes in matrix group algorithms we can safely assume that the discrete logarithm problem does not pose any significant difficulties as long as q is smaller or equal to 2^{32}.

IV.5 Integer factorisation

IV.5.1 Problem (The integer factorisation problem)
Let $n \in \mathbb{N}$ be an integer. The integer factorisation problem is the following:

$$\boxed{\text{Find the unique expression of } n \text{ as a product of prime powers } n = \prod_{i=1}^{k} p_i^{e_i}.}$$

Computing integer factorisations

Obviously this problem can be solved by trial division in time $O(n^{1/2}\log^2 n)$, however, like for the discrete logarithm problem, there is currently no known algorithm to solve this problem with complexity that is polynomial in $\log n$.
An overview of the current state of the art in integer factorisation is given in [Neb00].
For numbers n of the special form $a^k \pm 1$, which includes the cardinalities $p^k - 1$ of the multiplicative groups of finite fields, extensive tables for "small" values of a and k have been collected. They can be found on the site [Bre] and are built into computer algebra systems to speed up integer factorisation. In particular the factorisations of all numbers $a^k \pm 1$ which are smaller than 10^{255} are provided there.
For practical purposes in matrix group algorithms we can safely assume that the integer factorisation problem for $n = q^i - 1$ does not pose any significant difficulties as long as q^i is smaller than 10^{255}.

Chapter V

Composition trees

After lots of preparations in the previous chapters the main part of the whole book begins in this chapter. We start to talk about the recognition of matrix groups. We start by formulating the concept of "constructive recognition" and explain the reasoning behind it in Section V.1. In this section we develop a preliminary formulation of the main problem which will be refined in the following Section V.2. There we explain the fundamental approach to achieve constructive recognition by means of a "composition tree". The refinement of the formulation of the main problem there is necessary to allow for an efficient recursive solution. We chose to "develop" the problem formulation in this way in the hope to provide both a gentle introduction as well as a precise final formulation with good reasons for its complex structure.

In the following sections we develop a framework for group recognition that is not only suitable for matrix groups and projective groups, but also helps to implement the asymptotically best algorithms to handle permutation groups. In this framework we can switch between different representations of groups within one composition tree, which allows for example to use permutation group methods during matrix group recognition, provided we find some set our matrix group is acting upon.

The general approach to build a composition tree is not new and is already described in [LG01]. What is new in our approach is first the abstraction to allow for different representations intermixed in the same composition tree and second the fact that we change the generating set in each node of the composition tree which dramatically decreases the length of the resulting straight line programs.

The contents of this chapter stem from joint work with Ákos Seress and are an elaboration on the article [NS06].

V.1 Constructive recognition

There are at least two fundamentally different ways to represent groups on a computer. The first uses a presentation of the group and then expresses group elements as words in a free group representing cosets of the normal subgroup generated by the relations. The second approach uses an ambient group whose elements can be represented and multiplied directly on the computer, and the group is defined by giving a list of generators. As ambient groups one can use symmetric

groups, general linear groups or projective groups, since we can store and manipulate permutations and matrices efficiently on a computer.
In this book we concentrate on the second approach. To formalise our problem we first write down our assumptions about the ambient group and then formulate the fundamental problem.

V.1.1 Hypothesis (Ambient group)
When we speak about an *ambient group* we mean a finite group that can be represented on a computer such that we can perform the following tasks:

- Store and compare group elements using a bounded amount of memory per element.
- Multiply group elements.
- Invert group elements.
- Compute the order of a group element.

V.1.2 Remark
All finite symmetric groups fulfil the hypotheses in Section V.1.1. We call subgroups of finite symmetric groups *permutation groups*.
For a prime power q the groups $\mathrm{GL}(n,q)$ and $\mathrm{PGL}(n,q)$ also fulfil the hypotheses. We call subgroups of $\mathrm{GL}(n,q)$ *matrix groups* and subgroups of $\mathrm{PGL}(n,q)$ *projective groups*.

V.1.3 Problem (Constructive recognition — first formulation)
Let $\mathcal{G}$ be an ambient group in the sense of Hypothesis V.1.1 and assume that we are given a generating tuple $(g_1,\ldots,g_k) \in \mathcal{G}^k$ for a group $G := \langle g_1,\ldots,g_k\rangle \leq \mathcal{G}$.
We say that we have *recognised G constructively* if we have computed $|G|$ and prepared a procedure that does the following: Given $g \in \mathcal{G}$, decide whether $g \in G$ and if so, express g as a straight line program in $(g_1,\ldots,g_k)$. This latter procedure is called *constructive membership test*. □

Note that this formulation will be refined in Sections V.1.6 and V.2.1.

The problem as posed in Problem V.1.3 can be solved easily by just enumerating the finite group G completely. However, this is neither a sensible approach nor very interesting. The crucial point missing in Problem V.1.3 is that we want to do constructive recognition *efficiently*. To express what we mean by this we use complexity theory (see Section I.1). The class of problems is given by all possible tuples of generators of subgroups of our ambient groups. We have to specify what we mean by "input size".

V.1.4 Definition (Input size parameters)
Let $\mathcal{G}$ be an ambient group in the sense of Hypothesis V.1.1 and $(g_1,\ldots,g_k) \in \mathcal{G}^k$. If $\mathcal{G}$ is either $\mathrm{GL}(n,q)$ or $\mathrm{PGL}(n,q)$, then the input size of our problem is measured by n, k and $\log_2(q)$. If $\mathcal{G}$ is the symmetric group on n points, then the input size is measured by n and k. We silently set $q := 1$ in this case such that we can uniformly speak of the input size parameters n, k and $\log_2(q)$ in all cases.

V.1.5 Remark
Note that we use $\log_2(q)$ as one of the parameters of the input size instead of q itself. The reason for this is that one needs $\log_2(q)$ bits of data to store one finite field element of $\mathbb{F}_q$. This means

that the amount of storage needed for the input matrices is proportional to $kn^2 \log_2(q)$. Of course, the fact that the discrete logarithm problem in $\mathbb{F}_q$ prevents us from finding a real polynomial time algorithm for constructive recognition stems from this decision (compare Section IV.4). □

There is another reason why the formulation in Problem V.1.3 is not practical. Namely, the straight line programs for elements $g \in G$ written in terms of the original generators can be extremely long. In addition, for most generating tuples of G it can be tedious, if not impossible, to find a method to express group elements in G as straight programs in these generators. Usually, a recognition procedure first finds another "nice" generating tuple for G for which a good constructive membership testing procedure is available. To solve the original problem, such recognition procedures remember how they got the nice generating tuple from the original generating tuple by means of a single, rather long straight line program.
With the above notion of efficiency and these arguments in mind we can now formulate another preliminary version of the constructive recognition problem.

V.1.6 Problem (Constructive recognition — preliminary formulation)
Let $\mathcal{G}$ be an ambient group in the sense of Hypothesis V.1.1 and assume we are given a generating tuple $(g_1, \ldots, g_k) \in \mathcal{G}^k$ for a group $G := \langle g_1, \ldots, g_k \rangle \leq \mathcal{G}$.
We say that we have *recognised G constructively* if we have computed $|G|$ and a generating tuple $(g'_1, \ldots, g'_l)$ for G (the nice generators), for which we have prepared a procedure that does the following: Given $g \in \mathcal{G}$, decide whether $g \in G$ and if so, express g as a straight line program in $(g'_1, \ldots, g'_l)$. The new generating tuple may or may not be the same as the original one.
We assume that we have factorised all integers $q^i - 1$ for $1 \leq i \leq n$ and that we can solve the discrete logarithm problem in all fields $\mathbb{F}_{q^i}$ for $1 \leq i \leq n$ efficiently.
We call the first phase until G is recognised constructively the *recognition phase* and the second phase in which we decide $g \in G$ the *(constructive) membership test phase*.
The aim is that the algorithms for both phases have a runtime that is proved to be bounded by a fixed polynomial in the input size parameters n, k and $\log_2(q)$ in the sense of Definition V.1.4. □

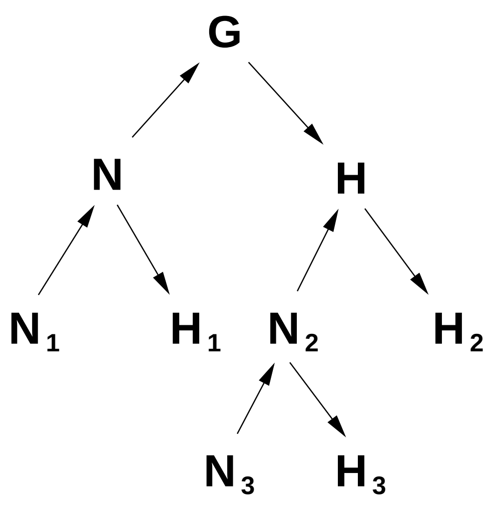
Note that this formulation will be refined in Problem V.2.1.

V.1.7 Remark (Randomised algorithm)
We are content with using Las Vegas or Monte Carlo algorithms (see Section I.3) provided that we can later on verify our results deterministically, even if this takes longer than the randomised recognition step. However, also the verification phase should have a runtime that is proved to be bounded by a fixed polynomial in the input size parameters n, k and $\log_2(q)$.

V.2 A recursive approach — reductions

Traditionally, the constructive recognition problem for permutation groups is solved by computing a stabiliser chain and a set of strong generators using the Schreier-Sims method (see [Sim70] and [Ser03]).
Although an analogous method can be used for matrix and projective groups, this becomes infeasible already for very small values of n since the occurring orbit lengths can be very large. Also for large base permutation groups (for a formal definition see Section V.5) like the symmetric and

Figure V.1: A composition tree



alternating groups in their natural representation there are asymptotically better methods because the stabiliser chains are very long.

Therefore one seeks *reductions* in these cases. A reduction is a surjective group homomorphism $\varphi : G \to H$ that can be computed explicitly and for which H is a subgroup of a possibly different ambient group. In addition the homomorphism φ must either have a non-trivial kernel or the group H must be "easier to handle" in some sense. This can for example mean that one of the values n, k and q is smaller for H than for G while the others remain unchanged, or that the expression $kn^2 \log_2(q)$ becomes smaller such that we need less memory to store the generators for H than those for G. Another possibility is that for a matrix group G the homomorphic image H is a permutation group.

Having found a reduction $\varphi : G \to H$ one first tries to recognise H constructively, then computes generators for the kernel N of φ, recognises N constructively and then puts everything together to achieve a constructive recognition of G from all other intermediate results. This procedure can be repeated recursively for the image H and the kernel N such that we end up with a tree (see Figure V.1), where the nodes correspond to subfactors of the original G and where we have to recognise the groups in the leaf nodes constructively by other means without a further reduction.

To make this recursion work we have to say quite carefully what has to happen at a non-leaf node G such that we actually have solved the constructive recognition problem for the group G at this node, provided we have solved the corresponding problem for the homomorphic image $H = \varphi(G)$ and the kernel $N = \ker(\varphi)$. Note that the problem in our first formulation in Problem V.1.3 could be solved recursively in this way. However, for our formulation in Problem V.1.6 this is not possible since we can only express group elements in the nice generators and not in the original ones. This is the reason why we have to refine the formulation once more. This time we present the final formulation.

V.2.1 Problem (Constructive recognition — final formulation)
Let $\mathcal{G}$ be an ambient group in the sense of Hypothesis V.1.1 and assume that we are given a generating tuple $(g_1, \ldots, g_k) \in \mathcal{G}^k$ for a group $G := \langle g_1, \ldots, g_k \rangle \leq \mathcal{G}$.
We say that we have *recognised G constructively* if we have

- computed $|G|$ and
- a generating tuple $(g_1', \ldots, g_l')$ for G (the nice generators), for which we have
- prepared a procedure that does the following: Given $g \in \mathcal{G}$, decide whether $g \in G$ and if so, express g as a straight line program in $(g_1', \ldots, g_l')$. Furthermore, we have
- prepared a procedure that computes, given preimages $(p_1, \ldots, p_k)$ of $(g_1, \ldots, g_k)$ under some (surjective) homomorphism $\psi : \hat{G} \to G$ for some group $\hat{G}$, preimages under ψ of the nice generators $(g_1', \ldots, g_l')$.

The new generating tuple may or may not be the same as the original one. The last point can for example be achieved by storing a straight line program that expresses $(g_1', \ldots, g_l')$ in terms of $(g_1, \ldots, g_k)$.
We assume that we have factorised all integers $q^i - 1$ for $1 \leq i \leq n$ and that we can solve the discrete logarithm problem in all fields $\mathbb{F}_{q^i}$ for $1 \leq i \leq n$ efficiently.
We call the first phase until G is recognised constructively the *recognition phase* and the second phase in which we decide $g \in G$ the *(constructive) membership test phase*.
The aim is that the algorithms for both phases have a runtime that is proved to be bounded by a fixed polynomial in the input size parameters n, k and $\log_2(q)$ in the sense of Definition V.1.4.
In this book we refer to this problem as the *constructive recognition problem (CRP)*. □

We claim that this final formulation of the constructive recognition problem has the property that if $\varphi : G \to H$ is a reduction, then having solved the constructive recognition problem for H and $N := \ker(\varphi)$ in fact suffices to solve the constructive recognition problem for G. To this end we have to clearly say what the original generators and the new, "nice" generators of our groups G, H and N are. This is the purpose of the following definition.

V.2.2 Definition (Reduction node)
Let $G = \langle g_1, \ldots, g_k \rangle \leq \mathcal{G}$ be as in Problem V.2.1. We say that we have set up a *reduction node* for G if we have performed the following steps:

(1) Finding a surjective group homomorphism $\varphi : G \to H$ for which we have a procedure to compute $\varphi(g)$ for all $g \in G$. If this procedure is given an element $g \in \mathcal{G} \setminus G$ then it returns either FAIL or an arbitrary element in the ambient group of H.

(2) Solving Problem V.2.1 for $H = \langle \varphi(g_1), \ldots, \varphi(g_k) \rangle$ with nice generators $(h_1, \ldots, h_s)$.

(3) Computing preimages $(h_1', \ldots, h_s') \in G^s$ under φ of the nice generators $(h_1, \ldots, h_s)$ of H using the solution of (2).

(4) Computing generators $(n_1, \ldots, n_t) \in N^t$ with $N := \ker(\varphi)$ together with a straight line program expressing $(n_1, \ldots, n_t)$ in terms of $(g_1, \ldots, g_k)$.

(5) Solving Problem V.2.1 for $N = \langle n_1, \dots, n_t \rangle$ with nice generators $(n'_1, \dots, n'_u) \in N^u$.

For a reduction node we call the tuple $(h'_1, \dots, h'_s, n'_1, \dots, n'_u)$ its nice generators.

V.2.3 Proposition (The recursion works)
Let $G = \langle g_1, \dots, g_k \rangle \le \mathcal{G}$ be as in Problem V.2.1. If we have set up a reduction node for G (in the sense of Definition V.2.2), then we have solved Problem V.2.1 for G.

Proof: The group order $|G|$ is equal to the product $|H| \cdot |N|$ since φ is surjective and $N = \ker(\varphi)$. Recall that we call the tuple $(h'_1, \dots, h'_s, n'_1, \dots, n'_u)$ the nice generators for G. Since

$$(\varphi(h'_1), \dots, \varphi(h'_s)) = (h_1, \dots, h_s)$$

generates the epimorphic image H of G and $(n'_1, \dots, n'_u)$ generates the kernel N the tuple of nice generators is a generating tuple for G.
Given $g \in \mathcal{G}$, we can test membership in G constructively as follows: First we try to map g with φ. If this fails, the element g does not lie in G by our assumptions in V.2.2.(1). Otherwise denote the result of the procedure assumed in V.2.2.(1) by h, it lies in the ambient group of H. Note that in this case it can still be that g is not contained in G, regardless whether h lies in H or not!
Then we try to express h as a straight line program s_1 in the nice generators $(h_1, \dots, h_s)$ of H. If this fails, h is not contained in H and thus g is not contained in G. Otherwise we evaluate s_1 with inputs $(h'_1, \dots, h'_s)$, which are preimages of the nice generators of H under φ. If g is contained in G, the result h' is a preimage under φ of $h = \varphi(g)$. Thus $m := h'^{-1} \cdot g$ is contained in the kernel N of φ. We try to express m as a straight line program s_2 in $(n'_1, \dots, n'_u)$. If this fails, m is not contained in N and thus g is not contained in G. Otherwise, we put together the straight line programs s_1 and s_2 to form one big straight line program s that reaches $h'm$ from the input $(h'_1, \dots, h'_s, n'_1, \dots, n'_u)$. In this case we have proved that g lies in G. □

We conclude this section with a short overview of the rest of the chapter and an outlook onto the following ones: We have now found our final formulation of the constructive recognition problem and we have shown a general approach for its solution by means of reduction. In the next Section V.3 we explain how to find the kernel N of a homomorphism as in Definition V.2.2 once we have constructively recognised the epimorphic image H. After that, in Section V.4, we describe a generic framework to organise the finding of reductions using a database of different methods. Finally we explain how this whole framework can be used to implement the asymptotically best algorithms for arbitrary permutation groups. The following chapters cover the question how to find reductions and solve the constructive recognition problem for matrix groups and projective groups.

V.3 Finding the kernel of a homomorphism

In this section we assume that we are in the situation described in Problem V.2.1 and want to construct a reduction node in the sense of Definition V.2.2. We assume that we can produce evenly distributed random elements in G (see Section I.4).
Assume that we have already found a homomorphism $\varphi : G \to \mathcal{H}$ into some ambient group $\mathcal{H}$ in the sense of Hypothesis V.1.1, such that we can map arbitrary group elements $g \in G$ with φ.

Then we set $H := \langle \varphi(g_1), \ldots, \varphi(g_k) \rangle$ and constructively recognise H which produces among other things nice generators $(h_1, \ldots, h_s)$ of H. Another result of the recognition is that we can compute preimages $(h'_1, \ldots, h'_s) \in G^s$ of the $(h_1, \ldots, h_s)$ under φ.
Equipped with all this data and methods we can now do the following for an arbitrary element $g \in G$: Map it with φ into H and call $h := \varphi(g)$. Then find a straight line program s reaching h from $(h_1, \ldots, h_s)$ and evaluate it on $(h'_1, \ldots, h'_s)$ to get an element $h' \in G$ with the property that $\varphi(g) = \varphi(h') = h$, thus $h'^{-1} \cdot g$ lies in the kernel N of φ. For this construction we have the following result.

V.3.1 Proposition (Even distribution of kernel elements)
If $x_1, x_2, \ldots, x_m$ are evenly distributed random elements in G, the above procedure produces evenly distributed random elements in N.

Proof: Since $H \cong G/N$, the elements of H correspond to cosets of N in G. Our constructive membership test in H produces a straight line program for every element $h \in H$ and the evaluation of this program on the preimages $(h'_1, \ldots, h'_s)$ thus chooses exactly one element of the coset G of N that is mapped to h. Since the x_i are evenly distributed in the whole of G, their images under φ are distributed evenly in H. Multiplying x_i with the inverse of the chosen coset representative in the coset $x_i N$ amounts to mapping $x_i N$ bijectively onto N. Again by the even distribution of the x_i it follows that the elements y_i are evenly distributed in N. $\square$

Proposition V.3.1 now allows to produce evenly distributed elements in N. To find a generating tuple of N we simply produce a certain amount of random elements. Since every proper subgroup of N has index at least 2 in N, the probability to increase the subgroup that is generated by the already produced elements is at least $1/2$ for every new element. Thus the probability to find a generating set for the whole of N is very high, provided we produce enough random elements. Then we can compute the normal closure using the methods in [Ser03, Chapter 2], in particular [Ser03, Theorem 2.3.9].

Sometimes we not only recognise H constructively in the sense of Problem V.2.1, but also find a presentation in terms of the nice generators. This happens mostly, but not exclusively, if H is an almost simple group. In this case we have the following result.

V.3.2 Proposition (Kernel if a presentation of H is known)
Assume that in the above situation H is isomorphic to the finitely presented group with generators $\bar{f}_1, \ldots, \bar{f}_s$ subject to relations $r_1, \ldots, r_m$ given as a straight line program t in terms of generators of the free group $F = \langle f_1, \ldots, f_s \rangle$. Let $(r'_1, \ldots, r'_m)$ be the result of t evaluated on $(h'_1, \ldots, h'_s)$, that is, the $(r'_1, \ldots, r'_m)$ are all preimages of the identity under φ. Let further x_i for $1 \leq i \leq k$ be elements in $\langle h'_1, \ldots, h'_s \rangle$ with $\varphi(x_i) = \varphi(g_i)$ obtained by the procedure described directly before Proposition V.3.1, and set $y_i := x_i^{-1} \cdot g_i \in N$. Then the kernel N of φ is the normal closure in G of the group generated by $(y_1, \ldots, y_k, r'_1, \ldots, r'_m)$.

Proof: Clearly, all the elements y_i and r'_i lie in N. Since N is a normal subgroup of G, the normal closure $\tilde{N}$ of the group generated by $(y_1, \ldots, y_k, r'_1, \ldots, r'_m)$ is contained in N.
On the other hand every element w of N is a product of the generators g_i. Thus, setting $w = g_{i_1} \cdots g_{i_t}$ for some numbers $i_j \in \{1, \ldots, k\}$, we get

$$w = x_{i_1} x_{i_1}^{-1} g_{i_1} \cdots x_{i_t} x_{i_t}^{-1} g_{i_t} = x_{i_1} y_{i_1} \cdots x_{i_t} y_{i_t} = \tilde{w} \cdot x,$$

where $\tilde{w}$ is a product of some G-conjugates of the y_{i_j} and x is a product of the h'_i that lies in N. Since the h'_i are preimages of the generators h_i of H and r'_i are constructed by the relations r_i the element x lies in the normal closure of the subgroup generated by the r'_i and thus $\tilde{N} = N$. □

V.3.3 Remark (Computing the kernel with a given presentation of *H*)
Using the results of the constructive recognition of H we can explicitly compute preimages r'_i of the relations r_i since the latter are given as straight line programs in terms of the nice generators of H. Thus we can explicitly compute all the generators in Proposition V.3.2. The normal closure can be computed using the methods in [Ser03, Chapter 2], in particular [Ser03, Theorem 2.3.9].

In this Section V.3 we have shown that to set up a reduction node we only need an explicitly computable homomorphism φ from G into some ambient group $\mathcal{H}$. Given this, one can constructively recognise the image, compute generators for the kernel and constructively recognise it, thereby fulfilling all requirements for a reduction node.

V.4 Finding homomorphisms and nice generators

During the recursive recognition procedure we have to solve Problem V.2.1 at every node of the composition tree, either by finding a reduction and setting up a reduction node or by solving the constructive recognition problem directly. In this section we describe a generic framework to organise an algorithm to achieve this.
We have to explore every group G as in Problem V.2.1 occurring in our tree to find out whether we can solve the constructive recognition problem or what kind of homomorphism can be applied to it. To this end, the framework holds a collection of so-called "find homomorphism" methods in stock. A find homomorphism method's objective is either to find a homomorphism $\varphi : G \to H$ onto some subgroup H, thereby setting up a reduction node and creating a new non-leaf node, or to solve the constructive recognition problem directly, which can but does not always have to find an isomorphism to some known group.
For each type of groups (permutation groups, matrix groups, projective groups and black-box groups), the system has a database of find homomorphism methods. We call the procedure that decides which methods to try and in which order the "method selection". For this purpose we define a very simple and yet versatile algorithm which we will describe now. It is usable independently from group recognition, but we will explain it here in the context of our generic recognition procedure. Note that this new method selection procedure is not to be confused with the GAP (see [GAP07]) method selection.
The group recognition procedure for a node just calls the generic method selection procedure with the database of find homomorphism methods corresponding to the type of the group.
The methods in each database are ranked, thereby defining a total order. The method selection procedure calls the methods one after another, starting with high ranks. A find homomorphism method reports back to the generic procedure by returning one of the four values in Table V.1.
The first case is the only one that terminates the recognition procedure for the current node in the composition tree. All other cases make it necessary to try other methods. The difference between these latter three cases lies in the fact, how the generic procedure chooses the next method called. If a method returns `NotApplicable`, then the method selection just calls the next method in

Table V.1: Possible results of a find homomorphism method

`true`:	The method was successful and has either set up a leaf or a non-leaf node. For details see below.
`fail`:	The method has failed to find a homomorphism or to solve the constructive recognition problem, at least temporarily.
`false`:	The method has failed and will do so always for the group G in question, such that there is no point in trying this method again for the group G.
`NotApplicable`:	The method is currently not applicable but it might become applicable later, provided new knowledge is found out about the group G.

the database. In the other two cases `false` and `fail`, the method selection again starts with the highest ranked method, but skips all methods that have already been tried and have failed by returning either `false` or `fail`.

When all available methods either have declared themselves `NotApplicable` or have failed, the method selection starts all over again, now calling methods again that have returned `fail` once, but of course skipping methods that have returned `false`. This whole process is repeated until each method has failed a configurable number of times, when the method selection finally gives up.

We hope that this design is simple enough to keep an overview of what is tried in which order to prove the whole algorithm to work correctly, and yet versatile enough to implement a wide range of different algorithms, in which "trying different methods" is involved. Now we explain the rationale behind some of the features of this procedure.

The idea behind the fact that after a method having returned `fail` or `false` the method selection starts again with the highest ranked method is that even a failed method might have found out new information about the group, thereby making higher ranked methods that have refused to work earlier by returning `NotApplicable` newly applicable.

In the GAP system, information about a group G is collected in so-called *attributes* (for example, a permutation group object may store as an attribute whether it is transitive or not), but we may also acquire the information that G is simple or that it is solvable, or we may know $|G|$, etc. Note that at any given time the value of an attribute for a given group object can already be computed or not, and the group object can learn new information about itself during its lifetime. This feature is already part of the GAP system library.

Further attributes may be computed when the method selection tries to apply different find homomorphism methods while processing the current group G. Therefore a find homomorphism method can just look whether or not a certain attribute is already known and then decide if it starts to work or declares itself `NotApplicable`. With new attributes being computed, this decision might be changed. By convention, a method should never use much computation time to find out that it is not applicable.

The idea behind a method returning `fail` is that often randomised algorithms are used that have the potential to fail, but still may succeed when tried again. The ranking and the failure probabilities of course have to be tuned carefully to assemble a sensible recognition system.

Once a method returns `true` it reports whether it has found a reduction or has solved the constructive recognition problem by other means. In the first case the generic machinery sets up a recognition node as described in Sections V.2 and V.3. In the other case a leaf in the composition tree is built and the find homomorphism method that returned `true` is responsible for the data and methods to do constructive membership testing in the group of the node. In this way a composition tree is built recursively using the database of find homomorphism methods together with the generic code organising the recursive framework.
Another important feature of this framework is the following. Quite often a find homomorphism method derives new information about the group in question that can be used further down in the composition tree most notably in the homomorphic image or the kernel of the group homomorphism found. For example it might already know that it has set up an isomorphism such that the kernel is trivial or it might know that the homomorphic image of a matrix group is a permutation group. To communicate such information to the methods used further down in the tree there is an infrastructure to hand down so-called "hints". These hints might for example consist of additional knowledge about the group such that certain find homomorphism methods are particularly well suited. Having this type of information available further down in the tree helps choosing the best find homomorphism method first. Furthermore, even a failed method can store already obtained data about the group in the data structure of the node it tried to work on. In this way methods that are called later in the method selection process can profit from this additional information. In the chapters to come in this book we describe find homomorphism methods and mention such hints that can help other methods along with the description of the methods.
We illustrate the generic method selection described in this section in the next Section V.5, where we explain how it is used to glue together the asymptotically best algorithms currently known for the handling of permutation groups to set up a constructive recognition algorithm for the case that the ambient group is a symmetric group.

V.5 Asymptotically best algorithms for permutation groups

Traditionally, the constructive recognition problem for permutation groups is solved by computing a stabiliser chain and a set of strong generators using the Schreier-Sims method (see [Sim70] and [Ser03]).
However, this method does not work very efficiently for so-called "large base groups". We start by developing the necessary language from the complexity theory of permutation groups.

V.5.1 Definition (Base of a permutation group)
Let $G \leq S_n$ be a subgroup of the symmetric group on $\{1, \ldots, n\}$. A tuple $B \in \{1, \ldots, n\}^l$ is called a *base of G with length l*, if only the identity of G fixes every point in B:

$$\{g \in G \mid b_i^g = b_i \text{ for all } 1 \leq i \leq l\} = \{\mathrm{id}\}.$$

The *stabiliser chain* belonging to a base B is the chain of subgroups

$$G \geq G_{b_1} \geq G_{b_1,b_2} \geq \cdots \geq G_{b_1,\ldots,b_k} = \{\mathrm{id}\},$$

where $G_{b_1,\ldots,b_j} = \{g \in G \mid b_i^g = b_i \text{ for all } 1 \leq i \leq j\}$.

Obviously, the length of every stabiliser chain for a permutation group G is at least as big as the length of a shortest base of G. This is the reason why computing stabiliser chains is not efficient for groups without a "reasonably short" base. Of course, the Schreier-Sims method does not necessarily find a shortest possible base. However, one can implement the method such that in the resulting stabiliser chain all inclusions are proper. Therefore we consider not the shortest possible base length but an upper bound for the length of a base.

V.5.2 Proposition (Maximal base length)
If $G \leq S_n$ then every base of G, for which all inclusions in its stabiliser chain are proper, has length at most $\lceil \log_2(|G|) \rceil$.

Proof: The index of one stabiliser in the previous one in the stabiliser chain is at least 2. □
To formulate precise complexity statements we have to talk about families of permutation groups.

V.5.3 Definition (Families of small and large base groups)
Let $\mathcal{F}$ be a family of permutation groups, all embedded into possibly different symmetric groups S_n and given by generating tuples. For a $G \in \mathcal{F}$ we denote by $n(G)$ the n of the symmetric group S_n into which G is embedded and by $k(G)$ the number of generators by which G is defined.
The family $\mathcal{F}$ is called *a family of small base groups* if there is a positive constant c such that $\log_2(|G|) \leq \log_2^c(n(G))$ for all $G \in \mathcal{F}$. If there is no such constant then $\mathcal{F}$ is called *a family of large base groups*.

V.5.4 Remark (A single permutation group)
Note that the terms "large base group" and "small base group" are *not defined* for a single permutation group G although one is sometimes tempted to call a member of a family of large base groups a "large base group". However, complexity statements about algorithms only make sense for families of inputs and thus it is only sensible to talk about "small" or "large" bases in the context of families of permutation groups.

The motivation for the terms "family of large/small base permutation groups" is the following result.

V.5.5 Theorem (Complexity of the Schreier-Sims method)
Let $\mathcal{F}$ be a family of small base groups. Then a randomised version of the Schreier-Sims method computes a base and strong generators for any $G \in \mathcal{F}$ in nearly-linear time in the input size $k(G) \cdot n(G)$, that is the runtime is bounded by a function in $O(k(G)n(G) \cdot \log_2^c(n(G)))$ for some constant c.

Proof: See [BCFS91] or [Ser03, Theorem 4.5.5]. □

If a permutation group $G \leq S_n$ is given by a list of generators, it is at first glance not clear, whether it has a short base. Of course, we want to avoid to compute a base by an application of the Schreier-Sims method if there is no short base.
To solve Problem V.2.1 for a permutation group we use the method selection procedure described in Section V.4. In the following we describe the methods that are used in this process. An overview of these methods is given in Table V.2. The basic idea is to recognise and handle alternating composition factors without computing a stabiliser chain.

Table V.2: Find homomorphism methods for permutation groups

Rank	Name	Action	Hom/Leaf
50	`NonTransitive`	Restrict to orbit	Hom
40	`Giant` A_n/S_n	Find standard generators	Leaf
30	`Imprimitive`	Find block system	Hom
20	`Jellyfish`	Find standard generators	Leaf
10	`StabilizerChain`	Compute a base and strong generators	Leaf

The method with the highest rank 50 is the reduction method called `NonTransitive` for intransitive groups. It first tests whether G is transitive and if not, constructs an explicitly computable homomorphism by restricting the action to one of the orbits in the permutation domain. If G turns out to be transitive, the `NonTransitive` method immediately returns `false` indicating that it can never be successful for the group G.
The next method `Giant` with rank 40 tries to decide, whether the group $G \leq S_n$ is a "giant", that is, it is either equal to A_n or to S_n itself. In these two cases, there are better methods than computing a stabiliser chain to solve the constructive recognition problem. These methods are described in detail in [Ser03, Section 10.2], see in particular [Ser03, Section 10.2.4]. The algorithm described there is one-sided Las Vegas in the following sense: If it recognises A_n or S_n, the result is guaranteed to be correct. If G is not a giant, it fails very quickly. With a small error probability it can fail even if G is equal to A_n or S_n. Thus, the method `Giant` returns `fail` in case of failure meaning that it could be called later on to try again. However, in the setup described here this will not happen, since later methods will always succeed, even if they take a very long time.
The third method `Imprimitive` with rank 30 tries to find a block system using the algorithm described in [Ser03, Section 5.5.1]. If it finds one, it constructs a homomorphism onto the action on the set of blocks. Since the algorithm is deterministic, it returns `false` if no block system is found and the group G is proved to be primitive.
Note that in the case that the `Imprimitive` method finds a block system, it hands the information about the blocks down to the node that is set up for the kernel. This allows firstly to use a different method to create generators for the kernel since one needs often much more elements to generate a subgroup of a repeated direct product than usual. Secondly, the system can use a special find homomorphism method for the kernel that produces a balanced composition tree, which is more efficient both during the recognition phase and the membership testing phase. Here the infrastructure in our framework to hand down hints to kernel and homomorphic image of a homomorphism is used extensively.
The fourth find homomorphism method `Jellyfish` with rank 20 handles another special case in which the Schreier-Sims method does not work efficiently. This case is that an alternating or symmetric group on $\{1, \ldots, m\}$ acts on $n = \binom{m}{k}^r$ r-tuples of k-subsets of $\{1, \ldots, m\}$. Of course, the action is not given in this way but simply as an action on the set $\{1, \ldots, n\}$ and the problem is to identify each number in $\{1, \ldots, n\}$ with an r-tuple of k-subsets of $\{1, \ldots, m\}$. In fact, the algorithm does not solve this latter problem but tries to guess m, k and r and directly find standard generators for A_m or S_m. A deterministic algorithm to solve this problem is described in [BLS97, Section 4] and a faster randomised one in [LNPS06].

Finally, if everything that has previously been tried has failed, we compute a base and strong generators using the Schreier-Sims method. This is done by the method called `StabilizerChain` with rank 10.
This setup tries the right things in the right order to handle groups with small as well as with large bases and thus implements a framework to handle all permutation groups with the asymptotically best algorithms known.

Chapter VI

Finding homomorphisms

In the previous Chapter V we formulated the problem of constructive recognition of groups in Problem V.2.1 and explained what a reduction is. This chapter describes how to find reductions for matrix groups and projective groups and how to prove that a certain collection of methods will for any given matrix group or projective group either find a reduction or show that the constructive recognition problem can be solved efficiently by other means. Together with the methods described in the next Chapter VIII and using the arguments in Chapter V this solves Problem V.2.1 whenever the ambient group is $\mathrm{GL}(n, q)$ or $\mathrm{PGL}(n, q)$.

We use two fundamental theoretical tools. The first is a variant of Aschbacher's theorem about the subgroup structure of the classical groups, which we describe in detail for the case of $\mathrm{GL}(n, q)$ in Section VI.1, and the second is the classification of finite simple groups together with the known results about their absolutely irreducible modular matrix representations. We are content with the statements in Aschbacher's theorem about the general linear group since they suffice for the purposes of constructive recognition and the statements and arguments are simpler than for the other classical groups. For details see [Asc84].

Roughly speaking, Aschbacher's theorem states that every subgroup of $\mathrm{GL}(n, q)$ is either a subgroup of a member of at least one of 7 concretely given classes $\mathcal{C}_1$ to $\mathcal{C}_7$ of subgroups, or it contains a classical group in its natural representation, or it is an almost simple group modulo scalars.

For the classes $\mathcal{C}_1$ to $\mathcal{C}_7$ the geometric way in which they are defined (compare Sections VI.(1.1) to VI.(1.7)) promises some kind of reduction. The two other cases are covered by two further classes $\mathcal{C}_8$ and $\mathcal{C}_9$, which are described in Sections VI.(1.8) and VI.(1.9). For members of the latter two classes one will usually have to solve the constructive recognition problem without further reduction.

Rather than presenting the original version of Aschbacher's theorem for $\mathrm{GL}(n, q)$, we present a variant, for which we define slightly different classes $\mathcal{D}_1$ to $\mathcal{D}_9$ of subgroups of $\mathrm{GL}(n, q)$ (see Sections VI.(1.1) to VI.(1.9)). The statement then says that every subgroup of $\mathrm{GL}(n, q)$ is contained in at least one of the classes $\mathcal{D}_1$ to $\mathcal{D}_9$.

The general idea is to provide efficient algorithms for all the classes $\mathcal{D}_1$ to $\mathcal{D}_7$ to recognise whether a given matrix group lies in the class, and if so, to find a reduction using this information. If none of these algorithms succeeds, our variant of Aschbacher's theorem shows that the group must be a member of $\mathcal{D}_8$ or $\mathcal{D}_9$. In this case the constructive recognition problem has to be solved by

different means, usually by first finding out which classical or almost simple group it is and then by using this information to do the constructive recognition in a special case, for example using standard generators (see Sections VIII.4 and VIII.5).
Our classes $\mathcal{D}_1$ to $\mathcal{D}_9$ are similar to the classes $\mathcal{C}_1$ to $\mathcal{C}_9$ but slightly different. First of all we include many but not all subgroups of class $\mathcal{C}_i$ into $\mathcal{D}_i$ such that we can replace the formulation "G is a subgroup of a member of $\mathcal{C}_i$" by "G is contained in $\mathcal{D}_i$". Many authors have used this formulation for the Aschbacher classes $\mathcal{C}_1$ to $\mathcal{C}_9$ anyway, thereby more or less implicitly changing the definition. But we have also added further conditions on the groups in our classes $\mathcal{D}_1$ to $\mathcal{D}_9$ for three reasons. The first is that we wanted those classes that were difficult to handle algorithmically to be smaller, in order to make it easier to devise algorithms for the new classes. The second is that we wanted the classes to be "more disjoint", so that fewer groups lie in more than one class. The third reason is that some more conditions on the groups in the classes immediately come out of the proof in the $\mathrm{GL}(n, q)$ case.
The purpose of this chapter is to explain the statement of our variant of Aschbacher's theorem for $\mathrm{GL}(n, q)$ in detail, to give a proof and to give an overview of the known methods to deal with the different classes together with references to the literature. Furthermore we present new ideas to handle the classes $\mathcal{D}_2$, $\mathcal{D}_4$ and $\mathcal{D}_7$, which seem to work very well in practice, but are not yet fully analysed with respect to their applicability and complexity. It seems that our changed definition of these classes helped to derive better algorithms than was previously possible for the more general classes $\mathcal{C}_2$, $\mathcal{C}_4$ and $\mathcal{C}_7$. An algorithm to recognise and handle the classes $\mathcal{D}_3$ and $\mathcal{D}_5$ provided that the group does not lie in class $\mathcal{D}_1$ is described in detail in Chapter VII. An overview of the handling of groups in classes $\mathcal{D}_8$ and $\mathcal{D}_9$ is given in the next Chapter VIII.

VI.1 A variant of Aschbacher's theorem

Note again that we restrict ourselves to the general linear group throughout, which is only a special case of Aschbacher's theorem.

VI.1.1 Notation

For this section we fix $n \in \mathbb{N}$ and $q = p^e$ for a prime p. We denote the vector space $\mathbb{F}_q^{1\times n}$ by V and note that $\mathrm{GL}(n, q)$ acts from the right on V by vector-matrix multiplication.

For the original formulation of his theorem in [Asc84] Aschbacher defines 8 classes of subgroups of $\mathrm{GL}(n, q)$ and proves that every subgroup G is either a subgroup of some member of one of these 8 classes or has a certain list of properties, the latter groups are usually denoted to lie in a class $\mathcal{C}_9$. We change this formulation in the following way: Instead of the original 8 Aschbacher class $\mathcal{C}_1$ to $\mathcal{C}_8$ we define different classes $\mathcal{D}_1$ to $\mathcal{D}_8$ (for details see the descriptions in Sections VI.(1.1) to VI.(1.8)). Furthermore we collect the subgroups $G \le \mathrm{GL}(n, q)$ that fulfil the properties in the statement of Aschbacher's theorem in the class $\mathcal{D}_9$. The slight modifications to the class definitions on the one hand stem from our proof of the theorem, on the other hand they are motivated in the following way: We try to increase the number of subgroups contained in classes that can be handled algorithmically well and try to decrease the number of subgroups contained in classes for which the known algorithms are not yet completely satisfying, mostly with respect to their complexity analysis. In addition we try to reduce the overlap between the classes.
We formulate a variant of Aschbacher's theorem in the following way.

VI.1.2 Theorem (Variant of Aschbacher's theorem, specialised to $\mathbf{GL}(n,q)$***)***
Let G be a subgroup of $\mathrm{GL}(n,q)$ with $n \geq 2$. Then G is contained in at least one of the classes $\mathcal{D}_1$ to $\mathcal{D}_9$ of subgroups described in Sections VI.(1.1) to VI.(1.9).

Proof: Compare [Wil83, Appendix 2, Theorem 1], [Asc84], [KL90] and [HLGOR96b, Theorem 1]. For a proof see Section VI.2. □

We proceed with our definition of the subgroup classes $\mathcal{D}_1$ to $\mathcal{D}_9$. Throughout we denote for a subgroup $G \leq \mathrm{GL}(n,q)$ its subgroup of scalar matrices by Z, that is, $Z := Z(\mathrm{GL}(n,q)) \cap G$. For each class we either give an alternative, more structural definition, or give the structure of some "typical" example groups in that class.

(1.1) Description of class $\mathcal{D}_1$: "reducible"

A group $G \leq \mathrm{GL}(n,q)$ is a member of $\mathcal{D}_1$ if there is a subspace $0 < W < V$ that is stabilised by G, that is, $Wg = W$ for all $g \in G$.

Differences to Aschbacher's class $\mathcal{C}_1$: Our class $\mathcal{D}_1$ consists of all members of $\mathcal{C}_1$ and all their subgroups.

Alternative definition:
A group lies in $\mathcal{D}_1$ if and only if it is conjugate in $\mathrm{GL}(n,q)$ to a subgroup of a group

$$P_m := \left\{ \begin{bmatrix} A & 0 \\ C & D \end{bmatrix} \mid A \in \mathrm{GL}(m,q),\, D \in \mathrm{GL}(n-m,q) \text{ and } C \in \mathbb{F}_q^{(n-m)\times m} \right\}$$

for some $0 < m < n$. The group P_m is called a *maximal parabolic subgroup* and is a semidirect product of the normal p-subgroup

$$U_m := \left\{ \begin{bmatrix} \mathbf{1}_m & 0 \\ C & \mathbf{1}_{n-m} \end{bmatrix} \mid C \in \mathbb{F}_q^{(n-m)\times m} \right\}$$

and $\mathrm{GL}(m,q) \times \mathrm{GL}(n-m,q)$, the factors being embedded on the diagonal blocks. Here $\mathbf{1}_m$ is the $(m \times m)$-identity matrix and $\mathbf{1}_{n-m}$ is the $(n-m) \times (n-m)$-identity matrix.

(1.2) Description of class $\mathcal{D}_2$: "imprimitive"

A group $G \leq \mathrm{GL}(n,q)$ is a member of $\mathcal{D}_2$ if the natural module V is absolutely irreducible and G has a normal subgroup N containing Z such that $V|_N$ is isomorphic as an $\mathbb{F}_q N$-module to a direct sum of irreducible modules that are not all isomorphic.

Note that by Clifford theory this immediately implies that there is a decomposition of V as a direct sum of m-dimensional subspaces $V = V_1 \oplus \cdots \oplus V_t$ such that the summands are permuted transitively by G. That is, for every $1 \leq i \leq t$ and every $g \in G$ there is a j with $V_i g = V_j$ and the action on the summands is transitive. The V_i are the homogeneous components of $V|_N$.

Differences to Aschbacher's class $\mathcal{C}_2$: We include in $\mathcal{D}_2$ the subgroups of the members of $\mathcal{C}_2$ that permute the homogeneous components of $V|_N$ transitively, but we exclude all groups acting not absolutely irreducibly. We also exclude groups in $\mathcal{C}_2$ for which the kernel of the permutation action is contained in the centre of G.

Example members:
The groups $\mathrm{GL}(m,q) \wr S_t$, where the $\mathrm{GL}(m,q)$ factors act on the direct summands V_i and the symmetric group on top permutes the subspaces, all lie in $\mathcal{D}_2$. Subgroups of these groups belong to $\mathcal{D}_2$ if they act absolutely irreducibly.

(1.3) Description of class $\mathcal{D}_3$: "semilinear"

A group $G \le \mathrm{GL}(n,q)$ lies in $\mathcal{D}_3$ if the natural module V is irreducible and there is a finite field extension $\mathbb{F}_{q^s}$ of $\mathbb{F}_q$, for which we can extend the $\mathbb{F}_q$-vector space structure of V to an $\mathbb{F}_{q^s}$-vector space structure of dimension n/s, such that for every $g \in G$ there exists an automorphism α_g of $\mathbb{F}_{q^s}$ with

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

for all $v, w \in V$ and all $\lambda \in \mathbb{F}_{q^s}$. This means that we can interpret V as an $\mathbb{F}_{q^s}$-vector space for which the natural action of G is $\mathbb{F}_{q^s}$-semilinear, so G is a subgroup of $\Gamma\mathrm{L}(n/s, q^s)$.

Note that the automorphisms of $\mathbb{F}_{q^s}$ occurring in the semilinear actions of group elements will automatically fix every element of $\mathbb{F}_q$, since the original action is $\mathbb{F}_q$-linear. Therefore they are elements of the Galois group $\mathrm{Gal}(\mathbb{F}_{q^s}/\mathbb{F}_q)$. Note further that the group G lies in $\mathcal{D}_3$ with trivial automorphisms α_g for all G if and only if V is irreducible but not absolutely irreducible (see [CR62, (29.13)]).

Differences to Aschbacher's class $\mathcal{C}_3$: We include in $\mathcal{D}_3$ subgroups of the members of $\mathcal{C}_3$ excluding all groups acting reducibly.

Alternative definition:
A group lies in $\mathcal{D}_3$ if and only if it acts irreducibly on the natural module and is conjugate in $\mathrm{GL}(n,q)$ to a subgroup of $\Gamma\mathrm{L}(n/s, q^s)$ for some $s \mid n$, realised as $(n \times n)$-matrices over $\mathbb{F}_q$ by choosing an $\mathbb{F}_q$-basis of $\mathbb{F}_{q^s}$.

(1.4) Description of class $\mathcal{D}_4$: "tensor decomposable"

A group $G \le \mathrm{GL}(n,q)$ lies in class $\mathcal{D}_4$ if the natural module V is absolutely irreducible and G has a normal subgroup N such that $V|_N$ is isomorphic as an $\mathbb{F}_qN$-module to a direct sum of $k \ge 2$ modules which are all isomorphic to a single absolutely irreducible $\mathbb{F}_qN$-module W with $\dim_{\mathbb{F}_q}(W) > 1$.

Differences to Aschbacher's class $\mathcal{C}_4$: We define $\mathcal{D}_4$ including more conditions on the structure of the group and its natural module than Aschbacher. In particular, we omit simple groups acting on an irreducible tensor product of two irreducible modules. The original definition of $\mathcal{C}_4$ uses a tensor product decomposition of V which is invariant under G. This follows from our conditions, see Proposition VI.2.1. Contrary to $\mathcal{C}_4$, we allow for $\mathcal{D}_4$ the dimensions of the tensor factors to be equal.

Example members:
The group $\mathrm{GL}(d_1, q) \circ \mathrm{GL}(d_2, q)$ is the central product of $\mathrm{GL}(d_1, q)$ and $\mathrm{GL}(d_2, q)$ for $d_1 \cdot d_2 = n$ and thus is contained in $\mathrm{GL}(n, q)$. It is the set of Kronecker products of a matrix in $\mathrm{GL}(d_1, q)$ and one in $\mathrm{GL}(d_2, q)$. These groups are members in $\mathcal{D}_4$ if $1 < d_1 < n$.

(1.5) Description of class $\mathcal{D}_5$: "subfield"

A group $G \le \mathrm{GL}(n, q)$ lies in $\mathcal{D}_5$ if the natural module V is absolutely irreducible and there exists a proper subfield $\mathbb{F}_{q_0}$ of $\mathbb{F}_q$, a matrix $T \in \mathrm{GL}(n, q)$, and scalars $(\beta_g)_{g \in G}$ with $\beta_g \in \mathbb{F}_q$ such that $\beta_g \cdot T^{-1} g T \in \mathrm{GL}(n, q_0)$ for all $g \in G$.

Differences to Aschbacher's class $\mathcal{C}_5$: We include in $\mathcal{D}_5$ subgroups of the members of $\mathcal{C}_5$ excluding all groups acting not absolutely irreducibly.

Alternative definition:
A group lies in $\mathcal{D}_5$ if and only if it is conjugate in $\mathrm{GL}(n, q)$ to a subgroup of $\mathrm{GL}(n, q_0) \cdot \mathbb{F}_q^*$, where $\mathbb{F}_{q_0}$ is a proper subfield of $\mathbb{F}_q$.

(1.6) Description of class $\mathcal{D}_6$: "extraspecial"

A group $G \le \mathrm{GL}(n, q)$ lies in $\mathcal{D}_6$ if the natural module V is absolutely irreducible, $n = r^m$ for a prime r and

- **either** r is odd and G has a normal subgroup E that is an extraspecial r-group of order r^{1+2m} and exponent r,
- **or** $r = 2$ and G has a normal subgroup E that is either extraspecial of order 2^{1+2m} or a central product of a cyclic group of order 4 with an extraspecial group of order 2^{1+2m},

and in both cases the linear action of G on the $\mathbb{F}_r$-vector space $E/Z(E)$ of dimension $2m$ is irreducible.

Differences to Aschbacher's class $\mathcal{C}_6$: We have added the condition about the irreducible $\mathbb{F}_r$-linear action of G on $E/Z(E)$ because it is an easy by-product of our proof and might help to devise algorithms to find a reduction for groups in this class. On the other hand we include subgroups of members of $\mathcal{C}_6$ if they fulfil this condition.

Example members:
The following subgroups of $\mathrm{GL}(n, q)$ for suitable (n, q) lie in $\mathcal{D}_6$:
$r^{1+2m}.\,\mathrm{Sp}(2m, r)$ and $2_+^{1+2m}.\mathrm{O}^+(2m, 2)$ and $2_-^{1+2m}.\mathrm{O}^-(2m, 2)$ and $(4 \circ 2^{1+2m}).\,\mathrm{Sp}(2m, 2)$.

(1.7) Description of class $\mathcal{D}_7$: "tensor induced"

A group $G \leq \mathrm{GL}(n,q)$ lies in $\mathcal{D}_7$ if it acts absolutely irreducibly on the natural module V and, for some k, it has a normal subgroup N containing Z that is isomorphic to a central product $T \circ \cdots \circ T$ of k copies of a group T which is a central extension of a non-abelian simple group $\bar{T}$ by Z, such that:
The restricted module $V|_N$ is isomorphic to an outer tensor product $W_1 \otimes_{\mathbb{F}_q} \cdots \otimes_{\mathbb{F}_q} W_k$ of k absolutely irreducible $\mathbb{F}_q T$-modules of the same dimension on which Z acts as scalars, and G/N acts on the set of central factors of N by permuting them transitively.

By the term "outer tensor product" we mean that every factor T in the central product N acts on exactly one of the tensor factors W_i. Note that the $\mathbb{F}_q T$-modules W_i need not necessarily be isomorphic to each other.

Differences to Aschbacher's class $\mathcal{C}_7$: We define $\mathcal{D}_7$ including more conditions on the structure of the group and its natural representation than Aschbacher. On the other hand we have added some subgroups of the members of $\mathcal{C}_7$.

Note: If G is a group in $\mathcal{D}_7$, then its natural projective representation is tensor induced, see Proposition VI.2.3. However, the natural representation of G needs not be tensor induced as is shown in Remark VI.2.4 by an example.

Example members:
The following subgroups of $\mathrm{GL}(n,q)$ lie in $\mathcal{D}_7$:
$\mathrm{GL}(r,q)^{\otimes k}: S_k$, where $n = r^k$ and $\mathrm{GL}(r,q)^{\otimes k}$ denotes the central product of k copies of $\mathrm{GL}(r,q)$.

(1.8) Description of class $\mathcal{D}_8$: "classical"

A group $G \leq \mathrm{GL}(n,q)$ lies in $\mathcal{D}_8$ if G/Z contains a classical simple group in its natural representation in one of the following ways:

- G/Z contains $\mathrm{PSL}(n,q)$ and $(n,q) \notin \{(2,2),(2,3)\}$,
- n is even, G is contained in $N_{\mathrm{GL}(n,q)}(\mathrm{Sp}(n,q))$ for some non-singular symplectic form, G/Z contains $\mathrm{PSp}(n,q)$ and $(n,q) \notin \{(2,2),(2,3),(4,2)\}$,
- q is a square, G is contained in $N_{\mathrm{GL}(n,q)}(\mathrm{SU}(n,q^{1/2}))$ for some non-singular Hermitian form, G/Z contains $\mathrm{PSU}(n,q^{1/2})$ and $(n,q^{1/2}) \notin \{(2,2),(2,3),(3,2)\}$,
- G is contained in $N_{\mathrm{GL}(n,q)}(\Omega^\epsilon(n,q))$, the corresponding $\mathrm{P}\Omega^\epsilon(n,q)$ is simple and contained in G/Z. The group $\mathrm{P}\Omega^\epsilon(n,q)$ is simple if and only if
  - $n \geq 3$, and
  - q is odd if n is odd, and
  - ϵ is – if $n = 4$, and
  - $(n,q) \notin \{(3,3),(4,2)\}$.

Note: For the orthogonal groups the given restrictions on (n,q) are necessary for $\mathrm{P}\Omega^\epsilon(n,q)$ to be simple, see [CCN+85, Section 2.4] for details.

Differences to Aschbacher's class $\mathcal{C}_8$: We only include groups in $\mathcal{D}_8$ that modulo scalars contain a simple classical group in its natural representation defined over $\mathbb{F}_q$.

(1.9) Description of class $\mathcal{D}_9$: "almost simple"

A group $G \leq \mathrm{GL}(n, q)$ lies in $\mathcal{D}_9$, if it is not in $\mathcal{D}_8$ and there is a non-abelian simple group $\bar{N}$ and a group T with $\bar{N} \leq T \leq \mathrm{Aut}(\bar{N})$ such that G/Z is isomorphic to T (in this case G/Z is called "almost simple") and the natural module V gives rise to an absolutely irreducible projective representation for T that is not realisable over a proper subfield of $\mathbb{F}_q$.

Note: The existence of the projective representation given by V limits the possibilities for $\bar{N}$ and T for given (n, q) and thus provides an interesting application for the representation theory of finite simple groups, their Schur covers and automorphism groups.

Differences to Aschbacher's class $\mathcal{C}_9$: Aschbacher does not define a class $\mathcal{C}_9$ but in the literature many authors have called the groups which fulfill the conditions in his main theorem the "Aschbacher class $\mathcal{C}_9$". For the definition of $\mathcal{D}_9$ we have left out the condition that the simple group leaves a form invariant.

Example members:
The absolutely irreducible projective representations of the quasi-simple groups provide examples of groups in $\mathcal{D}_9$.

VI.2 A proof of the GL-version of Aschbacher's theorem

The contents of this section are a variation on "The Theory behind SMASH" from [HLGOR96a, Section 2] and provide a proof for Theorem VI.1.2. The part about the semilinear action is taken from Section VII.(6.4) and thus from [CNRD08, Section 6.4].
Let G be any subgroup of $\mathrm{GL}(n, q)$ acting from the right on the natural module $V := \mathbb{F}_q^{1\times n}$.
If the natural module V is reducible, then G is contained in class $\mathcal{D}_1$ as defined in Section VI.(1.1).
From now on we assume that V is irreducible.
If V is not absolutely irreducible, then by [CR62, (29.13)] and the usual Wedderburn theorems the endomorphism ring $\mathrm{End}_{\mathbb{F}_q G}(V)$ is a proper finite extension field $\mathbb{F}_{q^s}$ of $\mathbb{F}_q$. This automatically extends the $\mathbb{F}_q$-vector space structure of V to an $\mathbb{F}_{q^s}$-vector space structure such that the G-action is $\mathbb{F}_{q^s}$-linear. Thus G lies in class $\mathcal{D}_3$ (see VI.(1.3)) with trivial Galois automorphisms.
From now on we assume that V is absolutely irreducible. Let Z be the subgroup of scalar matrices contained in G, i.e. $Z := G \cap Z(\mathrm{GL}(n, q))$. Since V is absolutely irreducible it follows that Z is the centre of G.
At this stage of the proof we mention that obviously G can lie in $\mathcal{D}_5$ (see VI.(1.5)).
From now on we assume that G does not lie in $\mathcal{D}_5$. In the sequel this assumption will be used to rule out classical groups defined over smaller fields and achieve the corresponding statement in class $\mathcal{D}_9$ (see VI.(1.9)).
We assume first that G/Z is a simple group. If G/Z were cyclic of prime order, then G would be abelian and V would not be absolutely irreducible, contrary to our assumptions.
There are two possibilities: The first is that G/Z is one of the classical simple groups in its natural representation defined over $\mathbb{F}_q$ and G lies in $\mathcal{D}_8$ (see VI.(1.8)). Note that by our assumption of G not lying in $\mathcal{D}_5$ we can exclude classical simple groups defined over a proper subfield. Otherwise

G lies in $\mathcal{D}_9$ (see VI.(1.9)) with $\bar{N} = T \cong G/Z$ since V gives rise to an absolutely irreducible projective representation of the finite simple group G/Z that is by assumption not realisable over a proper subfield of $\mathbb{F}_q$.
We assume from now on that G/Z is not simple, let $\bar{N}$ be a minimal normal subgroup of G/Z and N be the corresponding non-scalar normal subgroup of G with $Z < N \trianglelefteq G$.
Next we use Clifford theory applied to the natural module V. By Clifford's theorem (see [CR62, (49.2) and (49.7)]) the restricted $\mathbb{F}_q N$-module $V|_N$ is a direct sum of irreducible $\mathbb{F}_q N$-modules which are all G-conjugates of one irreducible $\mathbb{F}_q N$-submodule $W \leq V|_N$. In particular all these irreducible summands have the same dimension.
If there is more than one homogeneous component (i.e. not all conjugates of W are isomorphic to W as $\mathbb{F}_q N$-modules), then G lies in $\mathcal{D}_2$ (see VI.(1.2)), since the homogeneous components provide a direct sum decomposition that is preserved by G and the summands are permuted transitively.
We assume from now on additionally that there is only one homogeneous component, that is, all Wg are isomorphic to W as $\mathbb{F}_q N$-modules. Then $\dim W > 1$ because N is not scalar.
First we assume that W is a proper subspace of V. If W is not absolutely irreducible then the first part of Proposition VI.2.1 shows that G is semilinear and contained in $\mathcal{D}_3$ (see VI.(1.3)). Otherwise we have shown that G lies in $\mathcal{D}_4$ (see VI.(1.4)).
We proceed to the case that $W = V|_N$, that is, the restriction $V|_N$ is irreducible.
We first assume that $V|_N$ is not absolutely irreducible. Then $\mathrm{End}_{\mathbb{F}_q N}(V|_N)$ is a proper finite extension field of $\mathbb{F}_q$, say $\mathbb{F}_{q^s}$, and $V|_N$ can be considered as an $\mathbb{F}_{q^s}$-vector space of smaller dimension such that the action of N on it is $\mathbb{F}_{q^s}$-linear. We can embed $\mathbb{F}_{q^s} = \mathrm{End}_{\mathbb{F}_q N}(V) \leq \mathrm{GL}(n, q)$.
We claim that the action of G is $\mathbb{F}_{q^s}$-semilinear proving that G lies in $\mathcal{D}_3$ (see VI.(1.3)). Let $c \in \mathrm{GL}(n, q)$ generate the multiplicative group of $\mathbb{F}_{q^s}$. Then, for all $h \in N$ and $g \in G$, we have $hc = ch$ by definition and thus $h^g c^g = c^g h^g = h' c^g = c^g h'$, for some $h' \in N$. As h varies over N, the element h' takes every value in N, therefore $\langle c \rangle = \langle c^g \rangle$ and so $c^g = c^k$ for some k. Suppose that $c^i + c^j = c^l$, then $(c^i)^g + (c^j)^g = (c^l)^g$ so g acts as field automorphism on $\mathbb{F}_{q^s}$. We have thus proved that G lies in $\mathcal{D}_3$ (see VI.(1.3)).
From now on we assume that W is absolutely irreducible and distinguish several cases.
Recall that by assumption $\bar{N}$ is a minimal normal subgroup of G/Z. As such, by [DM96, Theorem 4.3A.(iii)], it is a direct product $\bar{T}_1 \times \cdots \times \bar{T}_k$ of copies of a simple group $\bar{T}$ which are all conjugate under G/Z. Thus N is a central product of the corresponding preimages $T_1, \ldots, T_k$ under the natural map $G \to G/Z$. We distinguish three cases:

(i) $\bar{T}$ is cyclic of prime order r,
(ii) $\bar{T}$ is non-abelian simple with $k > 1$ and
(iii) $\bar{T} \cong N/Z$ is non-abelian simple.

We now consider case (i) that $\bar{T}$ is cyclic of prime order r, then $\bar{N} = N/Z$ is an elementary-abelian r-group of order r^k. Since N is not abelian (W is an absolutely irreducible $\mathbb{F}_q N$-module with dimension greater than 1), we have $k > 1$. The derived subgroup N' of N is contained in Z and we recall that Z is the centre of N since $V|_N = W$ is absolutely irreducible. So N is nilpotent and thus the direct product of its Sylow subgroups. Let R be the Sylow r-subgroup of N. All other Sylow subgroups of N consist of scalar matrices since N/Z is elementary abelian.

Thus the module $V|_R$ is absolutely irreducible and R is not abelian. For $x, y \in R$, we have $1 = [x^r, y] = [x, y]^r$ since x^r and $[x, y]$ lie in $Z(R) = R \cap Z$:

$$x^{-r}y^{-1}x^r y = x^{-r}y^{-1}x^{r-1}yx \cdot [x, y]^1 = \cdots = x^{-r}y^{-1}yx^r \cdot [x, y]^r.$$

Therefore $R' = N'$ has exponent r and because it is contained in the cyclic group $Z(R) \leq Z$ we have $|R'| = r = |N'|$.

Assume first that r is odd. For $x, y \in R$ we have $(xy)^r = x^r y^r [y, x]^{r(r-1)/2} = x^r y^r$ since R' has order r. Thus the elements of R whose order divides r form a characteristic subgroup of N and thus a normal subgroup E of G. Since for r odd, an r-group containing a unique subgroup of order r is cyclic, E is not contained in Z and thus by the minimality of N/Z we have $ZE = N$. It immediately follows that $V|_E$ is absolutely irreducible, E is not abelian, $E \cap Z = Z(E) = E' = R' = N'$ has r elements and $N/Z = ZE/Z \cong E/(E \cap Z) = E/Z(E)$. Thus the Frattini subgroup $\Phi(E)$ of E is equal to $Z(E)$ and E is shown to be extraspecial of order r^{1+k} and exponent r. It follows using [Hup67, V.16.14] that k is even and $n = r^{k/2}$ because this holds for the faithful absolutely irreducible representations of an extraspecial group. The linear action of G on the $\mathbb{F}_r$-vector space $E/Z(E)$ is irreducible because every minimal normal subgroup of $N/Z \cong E/Z(E)$ is conjugated to a full basis by G. Hence G lies in $\mathcal{D}_6$ with odd r.

Consider now $r = 2$. Recall $R' \leq Z$ with $|R'| = 2$ and let E be the set of elements $x \in R$ with $x^2 \in R'$. For $x, y \in E$ we have $(xy)^2 = x^2y^2[y, x] \in R'$ proving that E is a characteristic subgroup of R and thus a normal subgroup of G. We claim that E is not contained in Z: If 4 does not divide $|Z(R)|$ then this is trivial. If 4 divides $|Z(R)|$ and $xZ(R)$ and $yZ(R)$ are different elements of $R/Z(R)$, then at least one of x^2, y^2 and $(xy)^2$ is a square in the cyclic group $Z(R)$ and thus one of x, y and xy can be multiplied by an element of $Z(R)$ to get an involution. Therefore R and thus E both contain a non-central element whose square is contained in R'. By the minimality of N/Z we have $ZE = N$ as above. We immediately get that $V|_E$ is absolutely irreducible, E is not abelian, $E \cap Z = Z(E)$ is cyclic with either 2 or 4 elements and $E' = R' = N'$ has 2 elements and is contained in $Z(E)$. Furthermore, $N/Z = ZE/Z \cong E/(E \cap Z) = E/Z(E)$. Thus the Frattini subgroup $\Phi(E)$ of E is equal to E' and E is either extraspecial of order 2^{1+k} or a central product of a cyclic group of order 4 consisting of scalar matrices and an extraspecial group of order 2^{1+k}. In both cases it follows using [Hup67, V.16.14] again that k is even and $n = 2^{k/2}$. As above the linear action of G on the $\mathbb{F}_2$-vector space $E/Z(E)$ is irreducible and thus G lies in $\mathcal{D}_6$ with $r = 2$.

This concludes case (i) that $\bar{T}$ is cyclic of prime order r.

We now consider case (ii) that $\bar{N}$ is a direct product of more than one copies of a non-abelian simple group $\bar{T}$. The conjugation action of G on N fixes Z elementwise and permutes the direct factors of N/Z and therefore also the central factors of N. Thus N is a central product of the groups $T_1, \ldots, T_k$, each of which is isomorphic to a single group T, which is a central extension of $\bar{T}$ by Z. The absolutely irreducible representations of N in which Z acts as scalars are just tensor products of absolutely irreducible representations of T in which $Z \leq T$ acts as scalars. Furthermore, since $T_1, \ldots, T_k$ are all conjugate under G, the natural module $V|_N$ must be isomorphic to an outer tensor product $W_1 \otimes_{\mathbb{F}_q} \cdots \otimes_{\mathbb{F}_q} W_k$ of k absolutely irreducible $\mathbb{F}_q T$-modules W_i on which Z acts as scalars. Since G acts on N by permuting the factors transitively, all modules W_i have the same dimension. So G lies in $\mathcal{D}_7$ concluding case (ii).

We finally consider case (iii) that $\bar{N} = N/Z$ is a non-abelian finite simple group. Since in this case the centraliser $C_{G/Z}(\bar{N})$ is trivial (because $W = V|_N$ is absolutely irreducible), G/Z is contained in the automorphism group of $\bar{N}$. Thus either N is a classical simple group in its natural representation defined over $\mathbb{F}_q$ and G lies in $\mathcal{D}_8$, or G lies in $\mathcal{D}_9$. Note that the assumptions we picked up during this proof allow us to conclude that V gives rise to an absolutely irreducible projective representation for G which is not realisable over a subfield (since G is not $\mathcal{D}_5$). □

In the rest of this section we present results relating our class definitions to those of Aschbacher and to other concepts of group representations. The following proposition is also used in the previous proof of Theorem VI.1.2.

VI.2.1 Proposition ($V|_N$ is homogeneous but N is not scalar)
Let $G \leq \mathrm{GL}(n,q)$ act absolutely irreducibly on its natural module $V = \mathbb{F}_q^{1\times n}$ and $Z = Z(G)$ be the set of scalar matrices in G. Let $Z < N \lhd G$ be a normal subgroup such that $V|_N$ is reducible and homogeneous, that is, $V|_N$ is isomorphic to a direct sum

$$V|_N = \bigoplus_{i=1}^{k} W_i$$

of at least 2 irreducible, pairwise isomorphic $\mathbb{F}_q N$-modules W_i.
If one and thus all W_i are not absolutely irreducible, then G lies in class $\mathcal{D}_3$.
If all W_i are absolutely irreducible, then G is tensor decomposable, by which we mean the following: There is a decomposition of $V = V_1 \otimes V_2$ into a tensor product with $1 < d_1 := \dim_{\mathbb{F}_q}(V_1) < n$ and $d_2 := \dim_{\mathbb{F}_q}(V_2)$ that is preserved by G, that is, for every $g \in G$ there are elements $g_1 \in \mathrm{End}_{\mathbb{F}_q}(V_1)$ and $g_2 \in \mathrm{End}_{\mathbb{F}_q}(V_2)$ such that $(v_1 \otimes v_2)g = v_1 g_1 \otimes v_2 g_2$ for all $v_1 \in V_1$ and $v_2 \in V_2$.

Proof: This proof is an extended version of the one in Section VII.(6.6), which is originally from [CNRD08, Section 6.6].

By Clifford theory, all W_i have the same dimension and for every i there is an element $g_i \in G$ such that $W_i = W_1 \cdot g_i$. Because N does not consist of scalar matrices by hypothesis, we have $\dim_{\mathbb{F}_q}(W_i) > 1$. Since the W_i are pairwise isomorphic, all have isomorphic endomorphism rings. Let $W := W_1$ and $d := \dim_{\mathbb{F}_q}(W)$.

We first do a base change such that every element of N is represented by a block diagonal matrix in which all diagonal blocks are identical of size d. Let now $E := \mathrm{End}_{\mathbb{F}_q N}(W)$ and $s := \dim_{\mathbb{F}_q}(E)$. Note that s is equal to 1 if and only if all W_i are absolutely irreducible.

Since W is irreducible and $\mathbb{F}_q$ is finite, we get by Schur's Lemma and Wedderburn's Theorem about finite division rings that E is a field and $E \cong \mathbb{F}_{q^s}$. By our special choice of basis we have $E = \mathrm{End}_{\mathbb{F}_q N}(W_i) \subseteq \mathbb{F}_q^{d\times d}$ for all i.

As $N \trianglelefteq G$, for all $h \in N$ and $g \in G$, the product $g^{-1}hg \in N$ and thus $g^{-1}hg$ is also a block diagonal matrix in which all $d \times d$-blocks along the diagonal are identical. Fixing g, we conclude that $g \cdot (g^{-1}hg) = hg$ for all $h \in N$. If we now cut g into $d \times d$-blocks, we get:

$$g \cdot (g^{-1}hg) = \left[\begin{array}{c|c|c|c} g_{1,1} & g_{1,2} & \cdots & g_{1,n/d} \\ \hline g_{2,1} & g_{2,2} & \cdots & g_{2,n/d} \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline g_{n/d,1} & g_{n/d,2} & \cdots & g_{n/d,n/d} \end{array}\right] \cdot \left[\begin{array}{c|c|c|c} D^{g^{-1}}(h) & 0 & \cdots & 0 \\ \hline 0 & D^{g^{-1}}(h) & \cdots & 0 \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline 0 & 0 & \cdots & D^{g^{-1}}(h) \end{array}\right]$$

$$= \left[\begin{array}{c|c|c|c} D(h) & 0 & \cdots & 0 \\ \hline 0 & D(h) & \cdots & 0 \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline 0 & 0 & \cdots & D(h) \end{array}\right] \cdot \left[\begin{array}{c|c|c|c} g_{1,1} & g_{1,2} & \cdots & g_{1,n/d} \\ \hline g_{2,1} & g_{2,2} & \cdots & g_{2,n/d} \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline g_{n/d,1} & g_{n/d,2} & \cdots & g_{n/d,n/d} \end{array}\right] = hg,$$

where the $g_{i,j}$ are $d \times d$-matrices, $D(h)$ is a matrix representing h on the module W and $D^{g^{-1}}(h) = D(g^{-1}hg)$ is the same representation twisted by the element g^{-1}. By the block diagonal structure of the matrices in N we get $g_{i,j} \cdot D^{g^{-1}}(h) = D(h) \cdot g_{i,j}$ for all i and j and all $h \in N$.
By hypothesis, the matrix representations D and $D^{g^{-1}}$ of N are isomorphic. Thus there is an invertible matrix $T \in \mathbb{F}_q^{d \times d}$ with $T \cdot D^{g^{-1}}(h) = D(h) \cdot T$ for all $h \in N$. By Schur's lemma and since the representation D is irreducible, the matrix T is invertible and unique up to left multiplication by an element of $C_{\mathrm{GL}(d,q)}(D(N))$, which is isomorphic as a group to the group of units of the extension field $E = \mathrm{End}_{\mathbb{F}_q N}(W) \cong \mathbb{F}_{q^s}$.
This shows that for every pair $(i, j) \in \{1, \ldots, n/d\} \times \{1, \ldots, n/d\}$ there is a unique element $e_{i,j} \in E$ (possibly 0) with $g_{i,j} = e_{i,j} \cdot T$. Recall that we can view E as being embedded into $\mathbb{F}_q^{d \times d}$.
We first consider the non-absolutely irreducible case that $s > 1$. Let $c \in E \subseteq \mathbb{F}_q^{d \times d}$ be a generator of the multiplicative group of E and let

$$C := \left[\begin{array}{c|c|c|c} c & 0 & \cdots & 0 \\ \hline 0 & c & \cdots & 0 \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline 0 & 0 & \cdots & c \end{array}\right]$$

be the block-diagonal matrix having c in all blocks on the diagonal. We denote elements of this form by $\mathbf{1} \otimes c$ in the rest of this proof.
We now compute $g^{-1}Cg$ where g is an arbitrary element of G, which we cut into $d \times d$ blocks as above such that we have $g_{i,j} = e_{i,j} \cdot T$ for some fixed $T \in \mathrm{GL}(d, q)$ and $e_{i,j} \in E \subseteq \mathbb{F}_q^{d \times d}$. It follows that the (i, j)-block of g^{-1} is equal to $T^{-1} \cdot f_{i,j}$, where $(f_{i,j}) \in E^{(n/d)\times(n/d)}$ is the inverse of the matrix $(e_{i,j}) \in E^{(n/d)\times(n/d)}$.
We can now compute that the (i, j)-block of $g^{-1}Cg$ is equal to

$$(g^{-1}Cg)_{i,j} = \sum_{k=1}^{n/d} T^{-1} \cdot f_{i,k} \cdot c \cdot e_{k,j} \cdot T = T^{-1}c \cdot \left(\sum_{k=1}^{n/d} f_{i,k} \cdot e_{k,j}\right) \cdot T = \delta_{i,j} \cdot T^{-1}cT,$$

since $c \in E$ and E is commutative. That is, $g^{-1}Cg$ is a block-diagonal matrix where all diagonal blocks are equal to $T^{-1}cT$. Since $D^{g^{-1}}(h) = T^{-1} \cdot D(h) \cdot T$ for all $h \in N$, we get that $T^{-1}cT$ commutes with all $D^{g^{-1}}(h) = D(g^{-1}hg)$ and thus with all $D(h)$ for $h \in N$, since $g^{-1}hg$ runs

through N if h does. But this means that $T^{-1}cT$ is contained in E and thus is equal to some power of c itself.
Since the conjugation action of G on the set of elements $1 \otimes c^k$ is multiplicative and additive in c^k, we conclude that G acts as field automorphisms on E in this way. The action is not trivial since G itself acts absolutely irreducible. The embedding of the elements of E into the diagonal blocks allows us now to view $V = \mathbb{F}_q^{1\times n}$ as an E-vector space of dimension n/s extending the $\mathbb{F}_q$-vector space structure. Our computations directly show that G is $\mathbb{F}_{q^s}$-semilinear in the sense of VI.(1.3) and thus contained in $\mathcal{D}_3$.
We now consider the absolutely irreducible case $s = 1$. Here we have shown that with respect to the above choice of basis, every element g is equal to a Kronecker product of some matrix $U \in \mathbb{F}_q^{(n/d)\times(n/d)}$ with a matrix $T \in \mathbb{F}_q^{d\times d}$. Since g is invertible, both U and T are invertible.
This provides an $\mathbb{F}_q$-linear isomorphism of $\mathbb{F}_q^n$ and the tensor product $\mathbb{F}_q^{n/d} \otimes_{\mathbb{F}_q} \mathbb{F}_q^d$ such that all $g \in G$ act as a Kronecker product proving the rest of the proposition. □

VI.2.2 Definition/Proposition (Tensor induction of (projective) representations)
For details and proofs see [CR90, 13A] and [Kov90, Section 2].
Let G be a group, K an arbitrary field, $H < G$ a subgroup of index k and $G = \bigcup_{i=1}^k g_i H$ with $g_1, \ldots, g_k \in G$ and $g_1 = 1$. Let $\varphi : G \to H \wr S_k$ be the group monomorphism defined by $\varphi(x) = \pi \cdot (h_1, \ldots, h_k)$ for $x \in G$, where $xg_i = g_{\pi(i)}h_i$ with $\pi \in S_k$ and $h_i \in H$ for $1 \le i \le k$.
Let κ be the group homomorphism $\kappa : \mathrm{GL}(d, K) \wr S_k \to \mathrm{GL}(d^k, K)$, where the direct factors of the base group act on the tensor factors of $(K^d)^{\otimes k} \cong K^{d^k}$ and the top group permutes them. Let $\bar\kappa : \mathrm{PGL}(d, K) \wr S_k \to \mathrm{PGL}(d^k, K)$ be the corresponding map for the projective linear group.
For a representation $\rho : H \to \mathrm{GL}(d, K)$ of H there is a corresponding map $(\rho \wr S_k) : H \wr S_k \to \mathrm{GL}(d, K) \wr S_k$ mapping the base group componentwise by ρ. The *tensor induced representation* $\rho\uparrow^{\otimes G}: G \to \mathrm{GL}(d^k, K)$ is then the composite map $\kappa \circ (\rho \wr S_k) \circ \varphi$. The equivalence class of this representation does not depend on the choices in this construction.
Similarly, for a projective representation $\bar\rho : H \to \mathrm{PGL}(d, K)$ of H there is a corresponding map $(\bar\rho \wr S_k) : H \wr S_k \to \mathrm{PGL}(d, K) \wr S_k$ mapping the base group componentwise by $\bar\rho$. The *tensor induced projective representation* $\bar\rho\uparrow^{\otimes G}: G \to \mathrm{PGL}(d^k, K)$ is then the composite map $\bar\kappa \circ (\bar\rho \wr S_k) \circ \varphi$. The equivalence class of this projective representation does not depend on the choices in this construction.

VI.2.3 Proposition ($\mathcal{D}_7$ implies "projective tensor induced")
If a group $G \le \mathrm{GL}(n, q)$ lies in $\mathcal{D}_7$, then the projective representation afforded by its natural module is tensor induced.

Proof: Let G be in $\mathcal{D}_7$ and let N, T, Z and V be as in Section VI.(1.7). We can use the proof of [Kov90, Tensor Induction Theorem]. Although the hypotheses of the theorem contain that the natural module is not induced from any proper subgroup, this is only used in the proof to ensure that the constituents of the restricted natural module $V|_N$ are all equivalent and absolutely irreducible. However, we have this by assumption and the proof goes through. □

VI.2.4 Remark ($\mathcal{D}_7$ does *not* imply tensor induced)
Let G be the semidirect product $(\mathrm{SL}(2, 7) \otimes_{\mathbb{F}_7} \mathrm{SL}(2, 7)).S_2 < \mathrm{GL}(4, 7)$, where the S_2 permutes the tensor factors. The natural module $V := \mathbb{F}_7^4$ is absolutely irreducible also when restricted to

the normal subgroup $N := \mathrm{SL}(2,7) \otimes_{\mathbb{F}_7} \mathrm{SL}(2,7)$ of index 2. The centre $Z := \langle -\mathrm{id}\rangle$ of G has two elements and N/Z is a minimal normal subgroup of G/Z which is a direct product of two copies of the non-abelian simple group $\mathrm{PSL}(2,7)$. Thus G lies in $\mathcal{D}_7$ by construction.
However, G does not have a subgroup of index 4, the only subgroup of G of index 2 is N and N does not have an irreducible representation of dimension 2 over $\mathbb{F}_7$. Thus the natural module is not tensor induced.
Of course, using Proposition VI.2.3 we conclude that the projective representation afforded by the natural module is in fact tensor induced from the two-dimensional projective absolutely irreducible representation of N over $\mathbb{F}_7$.

VI.3 Finding reductions

The purpose of this section is to give an overview of how reductions can be found algorithmically, provided that a matrix group lies in one of the classes $\mathcal{D}_1$ to $\mathcal{D}_7$. We do not give full details here but instead sketch the methods used and collect references to the literature.
We continue to use the notation for a group $G \leq \mathrm{GL}(n,q)$ that $Z := Z(\mathrm{GL}(n,q)) \cap G$ is the subgroup of scalar matrices and that $V := \mathbb{F}_q^{1\times n}$ is the natural right module. For the complexity statements we assume that G is given by m generators $g_1, \dots, g_m \in \mathrm{GL}(n,q)$.
Note that for a matrix group there is always the determinant map into the ground field and the natural homomorphism $G \to G/Z$. We can compute both explicitly and efficiently.
In the following we assume that we have already used the determinant homomorphism and then distinguish the cases, in which of the classes $\mathcal{D}_1$ to $\mathcal{D}_7$ the group G lies. Whenever it is convenient we can switch over to the projective group G/Z using the natural map as reduction.

(3.1) Finding a reduction in the reducible case: $\mathcal{D}_1$

Using the MeatAxe (see [HR94, IL00, Par84]) we can decide efficiently whether the natural module V is irreducible. If not, we find a proper invariant subspace $0 < W < V$ thereby finding a reduction as described in Section VI.(1.1). If V is irreducible, we either prove that it is absolutely irreducible or find an element that generates the endomorphism ring as a field.
Under the assumptions in Section VII.7.1 the MeatAxe provides a Las Vegas algorithm which terminates after at most $O(mn^3 \log \delta^{-1})$ elementary field operations, where m is the number of generators of G and $0 < \delta < 1/2$ is an upper bound for the failure probability (see Lemma VII.7.2).

(3.2) Finding a reduction in the semilinear or subfield case: $\mathcal{D}_3/\mathcal{D}_5$

Chapter VII deals with the case that G does not lie in $\mathcal{D}_1$ but is contained in $\mathcal{D}_3$ or in $\mathcal{D}_5$. The algorithms described there sometimes even find a reduction if G lies neither in $\mathcal{D}_3$ nor in $\mathcal{D}_5$. Note that, as it is common in the literature, we say in that chapter that a group "lies in $\mathcal{C}_i$" if it is a subgroup of one of the groups in Aschbacher class $\mathcal{C}_i$. For $\mathcal{C}_1$, $\mathcal{C}_3$ and $\mathcal{C}_5$ this then amounts to the same as talking about $\mathcal{D}_1$, $\mathcal{D}_3$ and $\mathcal{D}_5$ respectively.
The algorithm presented in Chapter VII is Las Vegas and is guaranteed to terminate in

$$O(n^3 \log(\delta^{-1})(m + \log(\delta^{-1} \log n)) + n^4 \log(\delta^{-1} \log n) \log q)$$

elementary field operations, if $0 < \delta < 1/2$ is the prescribed upper bound for the failure probability (see Section VII.7).

We conclude this section by pointing to some earlier work regarding reductions for groups in these classes. In [HR94] it is described how to test whether a matrix group acts absolutely irreducibly and in [HLGOR96a] the general case of a group in $\mathcal{C}_3$ is considered.

For the class $\mathcal{C}_5$ there is an algorithm described in [GH97] which decides whether a matrix group can be conjugated to one over a subfield. In [GLGO06] the general case of a group lying in $\mathcal{C}_5$ is considered.

(3.3) Finding a reduction in the imprimitive and tensor cases: $\mathcal{D}_2$ and $\mathcal{D}_4$

The content of this section is a bit experimental and speculative. We mainly describe a new idea to find reductions for the classes $\mathcal{D}_2$ and $\mathcal{D}_4$. At the end of this section we give references to the literature to describe the current state of the art.

The descriptions of the classes $\mathcal{D}_2$ and $\mathcal{D}_4$ have one property in common: They involve a non-scalar normal subgroup N of G such that the restriction of the natural module to N is reducible. That is, the MeatAxe can detect N in the sense that if we have a list of elements of G that happen to generate N, the MeatAxe applied to these elements will return a proper $\mathbb{F}_q N$-submodule. Since we can compute normal closures (see [Ser03, Theorem 2.3.9]) it is in fact enough to produce one non-central element of N to find a reduction for these cases.

However, simply producing random elements in G and hoping to hit elements in N does not work efficiently. Therefore we propose a new method to produce non-central elements of a normal subgroup. This method is put together from different well-known techniques and seems to work extremely well in practice. We give a few heuristic reasons why this works so well below, but a complete analysis of this method still has to be done at the time of this writing. In the following description of Algorithm 9 “INVOLUTIONJUMPER” we include a slight variation which will be used in Section VI.(3.4) to resolve the $\mathcal{D}_7$ case as well.

VI.3.1 Proposition (The Involution Jumper)

*Let G be a finite group. If Algorithm 9 “*INVOLUTIONJUMPER*” is called with G and an involution x, the return value u is either an involution in G with $xu = ux$ or* FAIL.

Proof: This method is heavily inspired by the “dihedral trick” to compute elements for involution centralisers (see [Bra00, 2.2]). We briefly repeat some arguments from there: The element x is an involution and for $c := x^{-1}y^{-1}xy$ we have $xc = c^{-1}x$ and thus $xc^k = c^{-k}x$ for all k. Thus, if c has even order $2k$, then c^k is an involution commuting with x and c^k lies in $[H, G]$ for every subgroup $H \leq G$ with $x \in H$ by construction (compare Remark VI.3.2).

Now assume that c has odd order $2k + 1 > 1$. Then $xyc^k = yxcc^k = yc^{-k-1}x = yc^k x$ and $z := yc^k$ is an element in the involution centraliser $C_G(x)$. In fact, Richard Parker has observed that if y is uniformly distributed in G, then z is uniformly distributed in $C_G(x)$ in this case. Thus, if the order of z is even, then Algorithm 9 “INVOLUTIONJUMPER” returns an involution as well.

If $c = 1$ and thus has order 1, then y lies in $C_G(x)$ by chance. If y has even order $2k$, then y^k is an involution commuting with x.

The whole algorithm fails, if it fails to produce enough elements of even order in $C_G(x)$. Note however that if $v \neq x$ is any involution in G, then $\langle x, v\rangle$ is a dihedral subgroup of G which contains at least one involution commuting with and different from x. □

VI.3.2 Remark (The boolean argument *b*)
The boolean argument b will usually be TRUE, however, it can be set to FALSE, in which case the algorithm will return a u that is contained in the group $[H, G]$ for every subgroup $H \leq G$ containing x. This option will be used in Section VI.(3.4) to handle the case of a group in $\mathcal{D}_7$.

Algorithm 9 INVOLUTIONJUMPER

```
Input: A group $G$, an involution $x \in G$, an integer $t$, and a boolean $b$.
Output: An involution in $G$ or FAIL.
$i := 0$
repeat
  $i := i + 1$
  $y :=$ RANDOM($G$)          {produce a random element of $G$}
  $c := [x, y]$ and $o :=$ ORDER($c$)
  if $o > 1$ then
    if $o$ is even then
      Return $c^{o/2}$
    else if $b =$ TRUE then
      $z := y \cdot c^{(o-1)/2}$ and $o :=$ ORDER($z$)
      if $o$ is even then
        Return $z^{o/2}$
      end if
    end if
  else if $b =$ TRUE then
    $o :=$ ORDER($y$)
    if $o$ is even then
      Return $y^{o/2}$
    end if
  end if
until $i \geq t$
Return FAIL
```

VI.3.3 Definition/Proposition (A Markov chain on involution classes)
Let G be a finite group of even order and $\mathcal{M}$ be the set of conjugacy classes of involutions in G. Then Algorithm 9 "INVOLUTIONJUMPER" defines a Markov chain with vertex set $\mathcal{M}$ in the following way: A step in the Markov chain starting at an involution class $C \in \mathcal{M}$ involves choosing an arbitrary involution in C and running Algorithm 9 "INVOLUTIONJUMPER" on it in the group G until a value not equal to FAIL is returned. The resulting involution class is the conjugacy class of the result. The probability to reach a certain involution class does not depend on the choice of the involution in the starting class.

Proof: Only the last statement needs to be proved. The result of

$$\text{INVOLUTIONJUMPER}(G, x, \infty, \text{TRUE})$$

can be summarised in the following formula. It is evaluated for the first random element y generated in the algorithm for which f is defined:

$$f(x,y) := \begin{cases} [x,y]^k & \text{if } \mathrm{ORDER}([x,y]) = 2k \\ (y[x,y]^k)^l & \text{if } \mathrm{ORDER}([x,y] = 2k+1 \text{ for } k \geq 1 \text{ and } \mathrm{ORDER}(y[x,y]^k) = 2l \\ y^k & \text{if } xy = yx \text{ and } \mathrm{ORDER}(y) = 2k \end{cases} .$$

Since we have $f(x^g, y^g) = f(x,y)^g$ whenever $f(x,y)$ is defined and y is supposed to be uniformly distributed in G and thus in each conjugacy class of G, this reasoning proves that the probability to reach involution class $B \in \mathcal{M}$ from class $A \in \mathcal{M}$ does in fact not depend on the choice of the representative for A. □

VI.3.4 Proposition (The Markov chain in Definition VI.3.3 is irreducible and aperiodic)
*Let G be a finite group of even order. If Algorithm 9 "*INVOLUTIONJUMPER*" is called with the boolean argument $b =$ *TRUE*, then the Markov chain on the involution classes $\mathcal{M}$ of G defined in Definition VI.3.3 is irreducible and aperiodic and thus has a stationary distribution in which every vertex has a non-zero probability.*

Proof: If G has only one class of involutions there is nothing to show.
Let $x, v \in G$ be involutions in different classes. With non-zero probability the random element y is equal to v. Assume that this happens. If x and v commute then Algorithm 9 returns v itself. If x and v do not commute, the subgroup $\langle x, v\rangle$ of G is a dihedral group. In this case, the order of $[x, v]$ is even, because if it were odd, all involutions in $\langle x, v\rangle$ were conjugate contradicting our assumption that x and v are not conjugate in G. Thus the central involution of $\langle x, v\rangle$ is returned which commutes with both x and v, and in the next step v could be returned with non-zero probability. This proves that the Markov chain is irreducible.
As to aperiodicity, this immediately follows from the fact that the random element y in the algorithm could be x itself in which case x is returned.
Therefore, by [Ser03, Theorem 2.2.1] the Markov chain has a stationary distribution in which every vertex has a non-zero probability, because every vertex can be reached from every vertex in at most two steps with non-zero probability. □

Obviously, the Involution Jumper needs an involution to start with. To this end we use a very simple minded algorithm to find one, which is given as Algorithm 10 "INVOLUTIONMAKER".

VI.3.5 Proposition (Involution Maker)
*Let G be a finite group. Then the return value u of Algorithm 10 "*INVOLUTIONMAKER*" called with the group G is either an involution or *FAIL*. The latter result occurs only if all t random elements generated in the algorithm have odd order. Therefore, the probability of failure is small, as long as there are enough elements in G with even order.*

Proof: Omitted. □

VI.3.6 Remark
Obviously, this algorithm can fail horribly if G contains only very few elements of even order. This happens for example in groups like $\mathrm{PSL}(2, 2^k)$ for large k.

Algorithm 10 INVOLUTIONMAKER

```
Input: A finite group $G$ and an integer $t$.
Output: An involution in $G$ or FAIL.
$i := 0$
repeat
  $i := i + 1$
  $x :=$ RANDOM($G$)        {produce a random element of $G$}
  $o :=$ ORDER($x$)
  if $o$ is even then
    Return $x^{o/2}$
  end if
until $i \geq t$
Return FAIL
```

To apply all this to our situation with $G \leq \mathrm{GL}(n, q)$ and $Z = G \cap Z(\mathrm{GL}(n, q))$ we simply compute in G/Z using comparison modulo scalars and the projective order of an element instead of the order. To indicate this trick we call Algorithms 9 and 10 with the group G/Z but then consider the result as an element of G in the sequel.
The following definition helps in formulations.

VI.3.7 Definition (Projective involution)
Let $G \leq \mathrm{GL}(n, q)$ and $Z := G \cap Z(\mathrm{GL}(n, q))$ be the subgroup of scalar matrices. An element $x \in G \setminus Z$ with $x^2 \in Z$ is called *projective involution in G*.

We summarise the method to find a reduction in the case that G lies in $\mathcal{D}_2$ or $\mathcal{D}_4$ in Algorithm 11 "FINDHOMD247". This algorithm already contains a line to check for $\mathcal{D}_7$ which we will explain in Section VI.(3.4). We do not analyse this algorithm here but only describe some observations and give a few heuristic arguments to try to explain why it works so well. The reader is kindly referred to future publications of the author for a more detailed analysis.

VI.3.8 Observation (Algorithm 11 "FINDHOMD247" works well in practice)
We have implemented Algorithm 11 "FINDHOMD247" in GAP and have performed many experiments. To generate random elements we use the RATTLE algorithm (a variant of product replacement, see [LGM02, LGO97a] and Section I.4.1). In all cases of groups G in $\mathcal{D}_2$ or $\mathcal{D}_4$ we looked at, only very few steps of Algorithm 9 "INVOLUTIONJUMPER" were needed to produce an element of the normal subgroup N and thus to find a reduction. In typical examples like $(\mathrm{GL}(3, 3)^{\times 6}).S_6 \leq \mathrm{GL}(18, 3)$ (imprimitive $\mathcal{D}_2$) or $\mathrm{Sp}(6, 3) \otimes \mathrm{O}(7, 3) \leq \mathrm{GL}(48, 3)$ (tensor decomposable $\mathcal{D}_4$) in average only slightly more than one step was needed to produce an element of the normal subgroup. Thus trying 9 times seems to be enough to rule out $\mathcal{D}_2$ and $\mathcal{D}_4$.

VI.3.9 Explanation (Heuristic explanation for Algorithm 11 "FINDHOMD247")
For being contained in a normal subgroup $N \trianglelefteq G$ only the conjugacy class of an element $x \in G$ is relevant since N is a union of conjugacy classes of G. The Involution Jumper (see Algorithm 9) basically performs a random walk through the involution classes of G/Z, as is explained in Definition VI.3.3. The corresponding Markov chain seems to have very good properties with respect

to visiting each vertex. To begin with, Proposition VI.3.4 shows that it is irreducible and has a stationary distribution in which every class has a non-zero probability. Even better, the vertices representing involution classes in normal subgroups seem to attract the Involution Jumper. The fact that we are computing commutators seems to add to this effect because we can hope to go down along the derived series of G. For example in all cases where the commutator $[x, y]$ of the previous involution x with a random element y has even order but has odd order in G/N, the Involution Jumper immediately jumps into N!

Algorithm 11 FINDHOMD247

```
Input: A group $G \leq \mathrm{GL}(n, q)$ known to act absolutely irreducibly.
Output: A reduction homomorphism or FAIL.
$x :=$ INVOLUTIONMAKER($G/Z$, 100)
if $x =$ FAIL then
   Return FAIL
end if
$i := 0$
while TRUE do
   $i := i + 1$
   Compute the normal closure $N := \langle x \rangle^G$ in $G$
   Run the MeatAxe on $V|_N$
   if $V|_N$ is reducible then
      if $V|_N$ is semisimple and has more than one homogeneous component then
         Return action on homogeneous components ($\mathcal{D}_2$)
      else if $V|_N$ is semisimple but has only one homogeneous component then
         Return tensor product decomposition ($\mathcal{D}_4$)
      end if
   else
      Perform check for $\mathcal{D}_7$ (see Section VI.(3.4) below)
   end if
   if $i \geq 9$ then
      Return FAIL
   end if
   $y :=$ INVOLUTIONJUMPER($G/Z$, $x$, 100, TRUE)
   if $y =$ FAIL then
      Return FAIL
   end if
end while
```

The experimental evidence suggests that it is feasible to analyse this procedure and to come up with a proof that it is in fact a Las Vegas algorithm that works with high probability for all or at least most groups in $\mathcal{D}_2$ and $\mathcal{D}_4$. It certainly has a good complexity in terms of n, the number m of generators of G and $\log q$ (ignoring as usual the discrete logarithm (see Section IV.4) and integer factorisation (see Section IV.5) problems in order computations). The major question to be resolved is, for which groups in $\mathcal{D}_2$ or $\mathcal{D}_4$ this does not work well or not at all.

We conclude this section with a few references to the existing literature. In [HLGOR96a] an algorithm is described to decide whether or not a group is acting imprimitively. The ideas in that paper have actually inspired lots of the methods presented in this chapter. An algorithm to decide whether a matrix group preserves a tensor decomposition of the natural module is presented in [LGO97b] and [LGO97a].

(3.4) Finding a reduction in the tensor induced case: $\mathcal{D}_7$

The definition of class $\mathcal{D}_7$ is similar to the ones for $\mathcal{D}_2$ and $\mathcal{D}_4$ in the sense that a normal subgroup N of $G \leq \mathrm{GL}(n, q)$ with $Z \leq N$ for $Z = G \cap Z(\mathrm{GL}(n, q))$ is involved. However, the restriction $V|_N$ of the natural module V to N is not reducible in the $\mathcal{D}_7$ case. Thus the approach described in Section VI.(3.3) does not work immediately. However, the observation that Algorithm 9 "InvolutionJumper" produces an element of N with relatively high probability also holds for the $\mathcal{D}_7$ case. To some extent it seems to work even better in this case, since the factor group G/N has an action on k points, where $n = r^k$ for some r and k, which makes k relatively small in real world situations.

There are basically two problems. The first is that we cannot easily detect that the normal closure of our involution produced by the Involution Jumper is equal to N, because $V|_N$ is absolutely irreducible. The second is that even if we assume this, we have to find the action of G/N on the direct factors of N/Z (recall that in the $\mathcal{D}_7$ case the group N/Z is isomorphic to a direct product of pairwise isomorphic non-abelian simple groups that are permuted transitively by the conjugation action of G/N on the set of direct factors of N/Z). In this section we present a new idea to solve these two problems which works very well in practice. As for the $\mathcal{D}_2$ and $\mathcal{D}_4$ cases, the reader is kindly referred to future publications by the author for a complete analysis.

Algorithm 11 "FindHomD247" already contains a statement "Perform check for $\mathcal{D}_7$" in a situation in which we have produced a candidate for the normal subgroup N. We now describe how to perform this check, which is summarised in Algorithm 12 "FindHomD7".

First of all, if n does not have a decomposition as $n = r^k$, then we do nothing since G cannot lie in $\mathcal{D}_7$. Otherwise, we need a relatively quick check whether N/Z is a direct product of pairwise isomorphic non-abelian simple groups acting irreducibly. To this end, we simply play the same trick again, since if G is in $\mathcal{D}_7$, then N is in $\mathcal{D}_4$. That is, we run Algorithm 9 "InvolutionJumper" on N/Z. Note that we already have an involution in N/Z, namely the one we used to produce N. In comparison to the method for $\mathcal{D}_2$ and $\mathcal{D}_4$, we change two things, firstly, we set the boolean argument b to False, and secondly, we call the Involution Jumper three times in a row (whenever it returns Fail, we stick to the previous involution). Both changes should improve our chances to produce an involution in N/Z that is trivial in all but one of the direct factors of N/Z. With b set to False we only take commutators with random elements of N. This means that once a component in the direct decomposition is trivial it will stay trivial. Due to the fact that the number of direct factors is relatively small, doing three steps will usually suffice to produce an involution whose normal closure in N/Z is exactly one direct factor.

Let now T be the normal closure of the final involution of N/Z we produce. We run the MeatAxe on the restriction $V|_T$ of the natural module and immediately give up if $V|_T$ is irreducible, not semisimple or not homogeneous. Otherwise, we determine the dimension d of an irreducible submodule. If $n = d^k$ for some k, then we assume that T is in fact equal to one of the k central factors of N.

Algorithm 12 FINDHOMD7

```
Input: A group $G \leq \mathrm{GL}(n, q)$, a normal subgroup $N$ known to act absolutely irreducibly
        and a projective involution $x \in N$.
Output: A reduction homomorphism or FAIL.
if $n$ is not a $k$-th power for some $k \geq 2$ then
  Return FAIL
end if
for $i = 1, \dots, 3$ do
  $y :=$ INVOLUTIONJUMPER($N/Z, x, 100,$ FALSE)
  if $y \neq$ FAIL then
    $x := y$
  end if
end for
Compute the normal closure $T := \langle x \rangle^N$ in $N$
Run the MeatAxe on $V|_T$
if $V|_T$ irreducible or not semisimple or not homogeneous then
  Return FAIL
end if
Let $W$ be an irreducible $\mathbb{F}_q T$-submodule of $V|_T$ and $d := \dim_{\mathbb{F}_q}(W)$
if $n$ is not a power of $d$ then
  Return FAIL
end if
Compute $k$ with $n = d^k$
Enumerate the orbit of $T$ under the conjugation action of $G$
if orbit length $\neq k$ or no proper action then
  Return FAIL
end if
Return action homomorphism as reduction
```

We then try to enumerate the orbit of G acting on the central factors of T, immediately giving up if this orbit has a different length than k.

The only problem to solve for this is how to represent the conjugates of T on the computer and how to compare them. Since we know that in the $\mathcal{D}_7$ case T/Z is a non-abelian simple group, this is both relatively easy. We represent T and its conjugates by a short generating set. To start this, we produce 3 random elements of T making sure that the first one does not commute with the two others. We then conjugate three-tuples of elements with the generators of G. To compare two such three-tuples, we simply check, whether or not the first element of the first tuple commutes with every element in the second tuple. If so, the two conjugates of T commute since T/Z is a non-abelian simple group with trivial centre. Thus the two groups generated by our tuples are not the same. Otherwise, we know that the two conjugates are equal. This test can be done very quickly. There is of course a small error possibility if the first three-tuple accidentally generates a proper subgroup of T.

This whole procedure finds the action of G on the direct factors of N/Z with high probability and thus returns a reduction provided G is in $\mathcal{D}_7$ and N is the normal subgroup from the definition of the class $\mathcal{D}_7$.
Of course, all of this is no proof that this method works in all cases and even less a full complexity analysis. However, in real life examples this method seems to work extremely well and quickly fails if G is not a $\mathcal{D}_7$ group. Further work to analyse these ideas is required.
We conclude this section with a few references to the existing literature. In [LGO02] an algorithm is described to determine algorithmically whether or not a group is projectively tensor induced. This paper uses the methods in [LGO97b] and [LGO97a]. Note that this covers the more general case of a subgroup of a group in Aschbacher class $\mathcal{C}_7$.

(3.5) Finding a reduction in the extraspecial case: $\mathcal{D}_6$

Efficient methods to find a reduction in the $\mathcal{D}_6$ case can be found in [BNS06].

Chapter VII

Finding reductions for subfield and semilinear groups

This chapter is about a polynomial-time reduction algorithm for groups of semilinear or subfield class. The contents of this chapter are joint work with Jon F. Carlson and Colva M. Roney-Dougal and are already published as [CNRD08].
Note that we use the original Aschbacher classes $\mathcal{C}_1$ to $\mathcal{C}_9$ instead of our new classes $\mathcal{D}_1$ to $\mathcal{D}_9$ in this chapter, because this makes the text readable independently from this book. For the purposes of finding reductions for groups in the classes $\mathcal{D}_3$ and $\mathcal{D}_5$ the methods below for $\mathcal{C}_3$ and $\mathcal{C}_5$ can be applied without change.

VII.1 Introduction

The matrix group recognition project was begun some years ago by Neumann and Praeger in a groundbreaking paper [NP92]. Their results answered the question of how one can determine computationally whether a given set of invertible matrices with entries in a finite field $\mathbb{F}_q$ generates the group $\mathrm{SL}(d, q)$. Since then many algorithms for computing with matrix groups over finite fields have been developed. Given a collection $g_1, \ldots, g_m$ of matrices in $\mathrm{GL}(d, q)$, the basic problem is to find a composition series for the group G that they generate and to be able to express arbitrary group elements as straight line programs in the generators. An overview of the aims of the recognition project is given in [LG01].
The overall approach of the project relies on a fundamental theorem of Aschbacher [Asc84] on the maximal subgroups of classical groups. The theorem says that every subgroup of $\mathrm{GL}(d, q)$ lies in at least one of nine classes $\mathcal{C}_1, \ldots, \mathcal{C}_9$. The classes $\mathcal{C}_1, \ldots, \mathcal{C}_8$ are treated by reducing to some sort of easier setting, and there are algorithms for these cases. However, the complexity of some of them has not been analysed and many do not run in polynomial time. The overall project is currently in a second phase, producing provably polynomial-time algorithms for each class.
The basic approach (see [LG01, NS06]) is first to reduce the problem by either finding a proper nontrivial homomorphism from G or finding an isomorphism to a representation with a smaller ambient group (for example to $\mathrm{GL}(d, q_0)$ for $q_0 < q$). If a homomorphism is found then the kernel and image are treated separately, eventually producing a *composition tree*, whose leaves are either simple groups or groups that can be constructively recognised by other means.

Let $\overline{G} \leq \mathrm{PGL}(d, q)$ be the corresponding projective group. In this chapter we present a fully analysed, polynomial-time Las Vegas algorithm to find a reduction for the case that G or $\overline{G}$ is not in $\mathcal{C}_1$, but is in $\mathcal{C}_3$ or $\mathcal{C}_5$, or has non-absolutely-irreducible derived group. Class $\mathcal{C}_1$ (reducible groups) is completely under control using MeatAxe methods [HR94, IL00, Par84]. In the case that G or $\overline{G}$ is in $\mathcal{C}_3$ or $\mathcal{C}_5$ we either find a proper nontrivial homomorphism from G to a permutation, matrix or projective group, or an isomorphism writing G or $\overline{G}$ over a smaller field, or in a smaller dimension. In addition, for some groups in the classes $\mathcal{C}_2$, $\mathcal{C}_4$ or $\mathcal{C}_6$, we find a nontrivial reduction homomorphism: This is important as there is as yet no fully polynomial-time analysis for these classes.

Our algorithms are efficient in the sense that they use a number of field operations that is bounded by a low-degree polynomial in m (the number of generators), d and $\log q$: the input size is $O(md^2 \log q)$. We analyse the complexity of all algorithms during the course of the chapter, and have implemented our work in GAP [GAP07]. We avoid the use of a discrete logarithm oracle. We use 3 for the exponent of matrix multiplication, as although the theoretical exponent is lower than this, for practical implementations this is more realistic.

In addition to developing reduction algorithms, we characterise the groups which have a faithful absolutely irreducible module on which the derived group acts by scalar matrices. We also develop efficient Monte Carlo methods (see Section I.3) for generating subgroups of matrix groups that behave like normal subgroups. The use of Clifford's Theorem upgrades these algorithms to Las Vegas.

One of the motivations for this work is a recent article by Glasby, Leedham-Green and O'Brien [GLGO06], who develop an algorithm to recognise groups G in class $\mathcal{C}_5$, generalising [GH97]. The algorithm in [GLGO06] is polynomial time provided that the commutator subgroup acts absolutely irreducibly. Here we address the case where G' is not absolutely irreducible by providing fully analysed algorithms for all actions of G', including the case where G' consists only of scalars. In the paper [GLGO06] the authors appear to misstate the complexity of generating G'. To the best of our knowledge, the best published complexity for this is $O(d^7 \log^2 q)$. In this chapter, we develop Las Vegas methods to generate a normal subgroup of G that is contained in G' in $O(d^4 \log q)$ field operations.

Our approach in Sections VII.(6.4) to VII.(6.6) is heavily influenced by SMASH [HLGOR96a] and we have reused many subroutines. There are two main differences between these sections and the original treatment in SMASH. Firstly, we have analysed the probability of having generators for a subgroup that has the same submodule lattice as a normal subgroup of G. Secondly, we have improved algorithms and complexity estimates for finding an irreducible submodule of a normal subgroup. Hence we are able to derive tighter upper bounds on the complexity of our algorithm.

The layout of this chapter is as follows. In Section VII.2 we present basic definitions and our main result. In Section VII.3 we characterise the groups with faithful absolutely irreducible representations over arbitrary fields whose derived group is mapped into the scalar matrices. In Section VII.4 we prove probabilistic results about the number of random elements required to generate a matrix group. In Section VII.5 we explain an algorithm for writing irreducible matrix groups over a smaller field. In Section VII.6 we present the main body of our algorithm, followed by Section VII.7 which summarises the complexity results and Section VII.8 which reports on our implementation of these algorithms.

VII.2 Definitions and main result

Throughout this chapter (except for Sections VII.3 and VII.5), we assume that $g_1, \ldots, g_m \in \mathrm{GL}(d, q)$ and let $G = \langle g_1, \ldots, g_m \rangle \leq \mathrm{GL}(d, q)$ be the corresponding matrix group. By considering each of $g_1, \ldots, g_m$ to be defined only up to scalar multiplication, we also define a group $\overline{G} = \langle \overline{g_1}, \ldots, \overline{g_m} \rangle \leq \mathrm{PGL}(d, q)$, which is the projective group generated by the given matrices. Two matrices represent the same elements of $\overline{G}$ if one is a scalar multiple of the other, so replacing any of the g_i by scalar multiples will alter the matrix group but not the projective group. We assume throughout that G acts irreducibly on the natural module $V = \mathbb{F}_q^d$.

VII.2.1 Definition

The group G lies in $\mathcal{C}_3$ (the class of semilinear groups) if there is a divisor e of d with $1 < e < d$, and an $\mathbb{F}_q$-vector space identification between $\mathbb{F}_q^d$ and $\mathbb{F}_{q^e}^{d/e}$ such that for $1 \leq i \leq m$ there exist automorphisms $\alpha_i \in \mathrm{Gal}(\mathbb{F}_{q^e}/\mathbb{F}_q)$ with

$$(v + \lambda w)g_i = vg_i + \lambda^{\alpha_i} wg_i$$

for all $v, w \in \mathbb{F}_{q^e}^{d/e}$ and all $\lambda \in \mathbb{F}_{q^e}$. The group $\overline{G}$ lies in $\mathcal{C}_3$ if and only if G lies in $\mathcal{C}_3$: Note that multiplying $g_1, \ldots, g_m$ by scalars from $\mathbb{F}_q$ does not affect the semilinearity of G.

If the α_i generate a proper subgroup of $\mathrm{Gal}(\mathbb{F}_{q^e}/\mathbb{F}_q)$ then multiplication by elements of the corresponding invariant subfield produces $\mathbb{F}_q G$-endomorphisms that are not $\mathbb{F}_q$-scalar, so V is not absolutely irreducible. Conversely, if V is not absolutely irreducible, then there is a divisor e' of d such that we can view V as a d/e'-dimensional vector space over $\mathbb{F}_{q^{e'}}$ on which the action of G is $\mathbb{F}_{q^{e'}}$-linear. That is, G lies in class $\mathcal{C}_3$ with trivial automorphisms.

VII.2.2 Definition

The group G lies in $\mathcal{C}_5$ if there exists a subfield $\mathbb{F}_{q_0} \subsetneq \mathbb{F}_q$, a $t \in \mathrm{GL}(d, q)$, and $\beta_1, \ldots, \beta_m \in \mathbb{F}_q^\times$ such that $t^{-1} g_i t = \beta_i h_i$ with $h_i \in \mathrm{GL}(d, q_0)$. The group $\overline{G}$ lies in $\mathcal{C}_5$ if and only if G lies in $\mathcal{C}_5$: Note that multiplying $g_1, \ldots, g_m$ by scalars from $\mathbb{F}_q$ does not change the membership of G in $\mathcal{C}_5$.

If $\beta_i = 1$ for all i then G can be written over $\mathbb{F}_{q_0}$. In general, G lies in $\mathcal{C}_5$ if G can be written over $\mathbb{F}_{q_0}$ modulo scalars. Note that $\overline{G}$ being in $\mathcal{C}_5$ implies that $\overline{G} \cong \langle \overline{h_1}, \ldots, \overline{h_m} \rangle$ embeds naturally in $\mathrm{PGL}(d, q_0)$.
We assume that the input to our algorithm is an irreducible group G: See Lemma VII.7.2 for the complexity of proving this. The MeatAxe run which shows G to be irreducible also computes the endomorphism ring $E = \mathrm{End}_{\mathbb{F}_q G}(V)$. If G is irreducible but not absolutely irreducible, the ring E is an extension field of $\mathbb{F}_q$. This provides an explicit E-vector space structure on V and an E-linear action of the group generators.

We now summarise our algorithm, see the relevant sections for more details.

1. Let G be irreducible with endomorphism ring E of degree $e \geq 1$ over $\mathbb{F}_q$. If $e > 1$ find an explicit base change to express the generators over $\mathbb{F}_{q^e}$.

2. Check whether G can be written over a subfield $\mathbb{F}_{q_0}$ with $\beta_i = 1$ for $1 \leq i \leq m$, using the standard basis technique described in Section VII.5, and find the degree f of $\mathbb{F}_q$ over $\mathbb{F}_{q_0}$.

3. If $e > f^2$ then return a homomorphism into $\mathrm{GL}(d/e, q^e)$. Otherwise, if $f > 1$ then return a monomorphism into $\mathrm{GL}(d, q_0)$.
4. Compute 10 commutators of random elements of G. If they are all scalar, choose any non-scalar generator g_i and check whether $[g_i, g_j]$ is scalar for all j. If so, jump to step 10.
5. Compute a normal subgroup N of the derived subgroup G' of G as in Section VII.(6.1).
6. If N is absolutely irreducible, check whether N can be written over a smaller field, as in Section VII.(6.3). If G is not contained in $\mathcal{C}_5$, return `false` as G is not in $\mathcal{C}_3$ or $\mathcal{C}_5$.
7. If N is irreducible but not absolutely irreducible, find a semilinear decomposition of G as in Section VII.(6.4).
8. If N is reducible with more than one homogeneous component, find an imprimitive decomposition of G as in Section VII.(6.5).
9. If N is reducible with a single homogeneous component with irreducible N-submodules of dimension greater than 1, find a tensor decomposition of G as in Section VII.(6.6).
10. If $[g_i, g_j]$ is scalar for some non-scalar g_i and all j, find a nontrivial homomorphism from G to $\mathbb{F}_q^\times$ as in Sections VII.3 and VII.(6.7).

We will show that all of our methods can be applied to both matrix and projective groups, because the success or failure of each step is unaffected by multiplying generating matrices by scalars. The following theorem summarises the main algorithmic results of this chapter.

VII.2.3 Theorem (Main Theorem)
Let $G = \langle g_1, \dots, g_m\rangle \le \mathrm{GL}(d, q)$ or $\overline{G} = \langle \bar{g}_1, \dots, \bar{g}_m\rangle \le \mathrm{PGL}(d, q)$ be an absolutely irreducible group that lies in $\mathcal{C}_3$, $\mathcal{C}_5$ or whose derived group is not absolutely irreducible. There exists an $O(d^4 \log(\log d) \log q + md^3)$ Las Vegas algorithm to find a nontrivial reduction of G or $\overline{G}$, respectively.

The complete procedure is Las Vegas in that we can prescribe an upper bound δ for the failure probability. The algorithm can succeed by returning a homomorphism or reporting `false`; or it can report `fail` with a prescribed probability bound δ. If success is reported, the result is guaranteed to be correct. If the algorithm reports `false` then some additional information may be deduced: For example, that if G lies in $\mathcal{C}_2$ then G' is transitive on all possible sets of blocks.

VII.3 Characterisation of groups with scalar derived group

In this section we do not require the global hypothesis that G is given by matrices or projective matrices. Instead, we assume only that G is a finite group with a faithful representation ρ. We investigate groups G which satisfy the following hypothesis.

VII.3.1 Hypothesis
The group G is a finite group, given by $G = \langle g_1, \dots, g_m\rangle$. Assume that G has a faithful absolutely irreducible representation $\rho : G \longrightarrow \mathrm{GL}(d, k)$ for some $d > 1$ and field k such that for all $g \in G'$, the matrix $\rho(g)$ is scalar.

Equivalently, the group G has a faithful nonlinear absolutely irreducible representation ρ such that $\rho(G)$ is projectively abelian.
Let V be the module afforded by ρ. The hypothesis implies the following facts.

VII.3.2 Lemma
Suppose that G, ρ and V are as above. The following all hold.

1. *The derived group G' is contained in the centre $Z(G)$.*
2. *The group G is nilpotent of class 2 and hence is a direct product of its Sylow subgroups.*
3. *The centre and the derived group of G are cyclic.*
4. *If the characteristic of k is $p > 0$ then the order of G is not divisible by p.*
5. *Let $[g, h] = g^{-1}h^{-1}gh$ be the commutator of elements g and h in G. Then*
$$[g, h_1h_2] = [g, h_1][g, h_2], \qquad [h_1h_2, g] = [h_1, g][h_2, g]$$
for all g, h_1, h_2 in G.
6. *Let $k^\times$ denote the multiplicative group of k. For any $g \in G$ there is a homomorphism $\psi_g : G \longrightarrow k^\times$ given by $\psi_g(h) = \rho([h, g])$. These homomorphisms satisfy $\psi_{g_1g_2}(x) = \psi_{g_1}(x)\psi_{g_2}(x)$ for all $x, g_1, g_2 \in G$. Moreover, ψ_g is a constant function if and only if $g \in Z(G)$.*

Proof: Part (1) is true because ρ is faithful. Part (2) follows from (1). It is well known that finite nilpotent groups are the direct product of their Sylow subgroups.
Part (3) is true because $\rho(G)$ is absolutely irreducible and ρ is faithful, so $Z(\rho(G))$ is a group of scalar matrices, which must be cyclic. Part (4) then follows from the fact that a Sylow p-subgroup of G would contribute a factor of p to the centre of $\rho(G)$, and hence to the centre of G.
Part (5) is a straightforward calculation. That is,
$$h_1h_2g[g, h_1h_2] = gh_1h_2 = h_1g[g, h_1]h_2 = h_1gh_2[g, h_1] = h_1h_2g[g, h_1][g, h_2]$$
since $[g, h_1], [g, h_2] \in Z(G)$ by (1). The second equation follows by inverting the first. Part (6) is a direct consequence of (5). □

The lemma allows us to prove the following.

VII.3.3 Proposition
Let z be the order of $Z(G)$, and let c be the order of G'. Then

1. *the exponent of $G/Z(G)$ is at most c,*
2. *the exponent of G is at most cz, and*
3. *the order of G is a divisor of $c^l z$, where $l = |\{i \mid g_i \notin Z\}|$.*

Proof: The group G is generated by elements $g_1, \dots, g_m$. Let $\psi_i : G \to \rho(G')$ be given by $\psi_i(h) = \rho([h, g_i])$. Then ψ_i is a homomorphism by Lemma VII.3.2.6, and the kernel K_i of ψ_i has index in G at most c, since $\mathrm{Im}(\psi_i)$ has order dividing c. Let ψ_i^c be defined by $\psi_i^c(x) := \psi_i(x)^c$. Then ψ_i^c is the constant homomorphism from G to $\{1\}$. From this and Lemma VII.3.2.6 it follows that g_i^c is in the centre of G for all i. This proves (1).
Part (2) is a direct consequence of (1) since the exponent of $Z(G)$ is z. Finally, (3) is a simple count based on the fact that $\cap_i K_i \leq Z(G)$. □

We know by hypothesis that V is absolutely irreducible and by Lemma VII.3.2.2 that G is a direct product $G = S_1 \times S_2 \times \cdots \times S_t$ of its Sylow subgroups. From this we get the next lemma.

VII.3.4 Lemma
The module V is a tensor product

$$V \cong V_1 \otimes \cdots \otimes V_t,$$

where each V_i is an absolutely irreducible module for S_i on which S_j acts trivially for $i \neq j$.

Proof: It is well known that irreducible modules of direct products are tensor products (see for instance [CR62, 51.13]). Since V is absolutely irreducible, so is each factor. □

VII.3.5 Proposition (Eigenspace decomposition)
Let $\rho(g) \in \mathrm{GL}(n, k)$ have all eigenvalues in k. Then V is a vector space direct sum

$$V = V_{\lambda_1} \oplus V_{\lambda_2} \oplus \cdots \oplus V_{\lambda_s},$$

where $\lambda_1, \dots, \lambda_s$ are the eigenvalues of $\rho(g)$ and V_{λ_i} is the λ_i-eigenspace. Moreover, if $h \in G$, then $V_{\lambda_i}\rho(h) = V_{\lambda_j}$, where $\lambda_j = \psi_g(h)\lambda_i$.

Proof: The space V is a direct sum of eigenspaces of $\rho(g)$ since $k\langle g\rangle$ is semi-simple and split by k. For the next statement, let $v \in V_{\lambda_i}$. Then as asserted

$$(v\rho(h))\rho(g) = v\rho(gh[h, g]) = \lambda_i v\rho([h, g])\rho(h) = \lambda_i \psi_g(h) v\rho(h).$$

□

If $g \in Z(G)$, then in Proposition VII.3.5, V is only a single eigenspace for the action of g.

VII.3.6 Theorem (Non-central element of prime order)
Let r be a prime and G an r-group satisfying Hypothesis VII.3.1. Then either G is the quaternion group of order 8 or G has an element g of order r such that $g \notin Z(G)$.

Proof: Since $Z(G)$ is cyclic, it suffices to prove that either G is quaternion of order 8 or G contains more than one cyclic subgroup of order r.
It is well-known (see for instance [Zas58, 3.15]) that the only r-groups which contain a single subgroup of order r are the cyclic groups and the generalised quaternion groups. Since $\rho(G)$ is absolutely irreducible and $d > 1$, the group G is not cyclic and so either G has a non-central element of order r or G is generalised quaternion.

The generalised quaternion group of order 2^i for $i \geq 3$ has presentation $\langle a, b \mid a^{2^{i-1}} = b^4 = a^{2^{i-2}}b^{-2} = b^{-1}aba = 1\rangle$. A short calculation shows that the derived group contains non-central elements for $i > 3$, and hence if G is a generalised quaternion group then $|G| = 8$. □

We finish this section with our characterisation of the groups satisfying Hypothesis VII.3.1.

VII.3.7 Theorem (Characterisation Theorem)
Let G have a faithful absolutely irreducible representation ρ of dimension $d > 1$ over an arbitrary field k such that the derived group of G is mapped into the set of scalar matrices. Then either $d = 2$ and $\rho(G)$ is an extension by scalars of the quaternion group of order 8 acting semilinearly, or $\rho(G)$ is imprimitive.

Proof: By Lemma VII.3.4 we may consider V as a tensor product with a distinct Sylow subgroup of G acting on each tensor factor. Let V_i be one such factor.
The first possibility is that V_i is 1-dimensional, and S_i is cyclic. Secondly, if the action on V_i is Q_8 then the dimension of V_i is 2 (see for instance [CR62, §47]).
Otherwise, by Theorem VII.3.6 the action has a non-central element $\rho(g)$ of order r. The group G also contains a central element of order r, which is mapped to a scalar. This shows that the field k contains primitive r-th roots of unity. Hence all of the eigenvalues of $\rho(g)$ lie in k. Since $\rho(g)$ is not central, it has more than one eigenvalue. It follows from Proposition VII.3.5 that the elements of G permute the eigenspaces. Since G acts irreducibly, this action on the eigenspaces is transitive and hence $\rho(G)$ is imprimitive.
If at least one of the induced actions is imprimitive then the action of $\rho(G)$ is imprimitive. Not all of the Sylow subgroups can be cyclic since $d > 1$. □

Extraspecial r-groups over finite fields $\mathbb{F}_q$ with $d = r^n$ and r dividing $(q-1)$ lie in this class, but so do other r-groups. For example, the subgroup G of GL(3, 19) of order 3^4 generated by

$$\begin{bmatrix} 0 & 0 & 1 \\ 17 & 0 & 0 \\ 0 & 11 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & 7 & 0 \\ 0 & 0 & 11 \end{bmatrix}$$

satisfies $G' \leq Z(\mathrm{GL}(3, 19))$ but does not contain an extraspecial group of order 3^{1+2}.

VII.4 Generation of matrix groups by random elements

In this section we analyse the generation of a subgroup $H = \langle g_1, \ldots, g_n\rangle$ of a normal subgroup N of a matrix group G, and in particular provide bounds on n for H to have the same submodule structure and endomorphism ring as N with probability at least $1 - \delta$. Perhaps surprisingly, we do this via results for permutation groups.

VII.4.1 Lemma
Let $X_1, X_2, \ldots$ be a sequence of 0-1 valued random variables such that $\mathrm{Prob}(X_i = 1) \geq p$ for any values of the previous X_j (but the distribution of X_i may depend on the outcome of the X_j for $j < i$).

Then, for all integers t and all $0 < \epsilon < 1$,

$$\mathrm{Prob}\left(\sum_{i=1}^{t} X_i \leq (1-\epsilon)pt\right) \leq e^{-\epsilon^2 pt/2}.$$

Proof: See [BCF+95, Corollary 2.2] or [Ser03, Lemma 2.3.3]. □

The following proposition is based on [Ser03, 2.3.7], where it is proved for the case G transitive. Note that the hypotheses here are slightly more general than in [Ser03, 2.3.7], where G is given as a group of permutations. Our Proposition VII.4.2 can for example be applied to any finite group equipped with a permutation action.

VII.4.2 Proposition (Correct orbits of subgroup)
Suppose that a finite group $G = \langle S \rangle$ acts on a finite set Ω, with α orbits. Let $1 > \delta > 0$ be arbitrary. Then with probability at least $1-\delta$, a sequence of $O(\log \delta^{-1} + \log|\Omega|)$ uniformly distributed random elements of G generates a subgroup of G that has the same orbits on Ω as G.

Proof: Let $t = c\log|\Omega|$, where $c \geq \max\{24\log\delta^{-1}/\log|\Omega|, 45\}$. Let $g_1, \ldots, g_t$ be uniformly distributed random elements of G. For $1 \leq i \leq t$ let $G_i = \langle g_1, \ldots, g_i \rangle$, let N_i be the number of G_i-orbits on Ω, and let M_i be the number of G_i-orbits that coincide with G-orbits. Let $k_i = N_i - M_i$. Note that $N_i \geq \alpha$ for $1 \leq i \leq t$, that $M_i \leq \alpha$ for $1 \leq i \leq t$, and that $N_i = \alpha$ if and only if $M_i = \alpha$. Hence k_i is either 0 or at least 2.
We claim that if $N_{i-1} > \alpha$, then

$$\mathrm{Prob}\left(k_i \leq \frac{7}{8}k_{i-1}\right) \geq \frac{1}{3}.$$

To see this, let $k = k_{i-1}$ and let $\Delta_1, \ldots, \Delta_k$ be the G_{i-1} orbits on Ω that are *not* G-orbits. For $1 \leq j \leq k$, let $X_j = 1$ if $\Delta_j^{g_i} \neq \Delta_j$ and let $X_j = 0$ otherwise. Now, Δ_j lies in an orbit of length at least two in the action of G on subsets of Ω. Therefore at most half of the elements of G fix Δ_j, and so $E(X_j) \geq 1/2$ for $1 \leq j \leq k$. Let $X = \sum_{j=1}^{k} X_j$, then $E(X) \geq k/2$.
Let p be the probability that $X \leq k/4$. Then with probability p the variable X takes value at most $k/4$ whilst X takes value greater than $k/4$ and less than or equal to k with probability $1-p$. Therefore

$$\frac{pk}{4} + (1-p)k \geq E(X) \geq k/2,$$

so $p \leq 2/3$. Hence, with probability at least $1/3$, at least $k/4$ of the G_{i-1}-orbits that are not G-orbits are proper subsets of orbits of G_i. Thus, with probability at least $1/3$, the number of orbits of G_i, which are not orbits of G, is at most $7k/8$, and the claim follows.
Define $Y_1, \ldots, Y_t$ by

$$\begin{array}{ll} Y_i = 0 & \text{if } k_i > 0 \text{ and } k_i > \frac{7}{8}k_{i-1} \\ Y_i = 1 & \text{if } k_i = 0 \text{ or } k_i \leq \frac{7}{8}k_{i-1}. \end{array}$$

By the previous claim, $\mathrm{Prob}(Y_i = 1) \geq 1/3$.
Now, $N_0 = |\Omega|$, so $k_0 \leq |\Omega|$. Clearly, $k_t \leq k_0 \left(\frac{7}{8}\right)^{\sum_{i=1}^{t} Y_i}$. The group G_t has the same orbits as G if and only if $k_t \leq 1$, which will follow if $|\Omega| \left(\frac{7}{8}\right)^{\sum_{i=1}^{t} Y_i} \leq 1$. In turn this simplifies to $|\Omega| \leq \left(\frac{8}{7}\right)^{\sum Y_i}$,

which gives

$$\sum_{i=1}^{t} Y_i \geq \frac{\log|\Omega|}{\log\frac{8}{7}}.$$

Then by Lemma VII.4.1, with $p = 1/3$, $t = c\log|\Omega|$ and $\epsilon = 1 - 3/(c\log(8/7))$ we get

$$\mathrm{Prob}(\sum_{i=1}^{t} Y_i \leq \frac{\log|\Omega|}{\log\frac{8}{7}}) \leq e^{-\frac{1}{6}(1-\frac{3}{c\log(8/7)})^2 c\log|\Omega|} \leq e^{-c\log|\Omega|/24}$$

for $c \geq 45$. In turn this is less than or equal to δ. □

We now apply the previous proposition to matrix groups by considering their action on vectors.

VII.4.3 Theorem (Correct action of subgroup)
Let $G \leq \mathrm{GL}(d,q)$, and let $1 > \delta > 0$ be arbitrary. With probability at least $1-\delta$, a sequence of $O(\log\delta^{-1} + d\log q)$ uniformly distributed random elements of G generate a subgroup H of G with the same submodule lattice as G on $V = \mathbb{F}_q^d$. Furthermore, if G is irreducible then $\mathrm{End}_{\mathbb{F}_qG}(V) = \mathrm{End}_{\mathbb{F}_qH}(V)$ with probability $1-\delta$.

Proof: First consider G as a permutation group on $|\Omega| = q^d$ points. By Proposition VII.4.2, any group H generated by $O(\log\delta^{-1} + d\log q)$ uniformly distributed random elements of G has the same orbits as G with probability $1-\delta$.
A submodule for G is a union of orbits of G in its action on vectors that is closed under addition and scalar multiplication, so the first claim follows.
For the second claim, let G be irreducible and $\mathrm{End}_{\mathbb{F}_qG}(V) = \mathbb{F}_{q^f}$. Let H_0 be generated by $O(\log\delta^{-1} + d\log q)$ random elements of G, so that H_0 is irreducible with probability $1-\delta/2$ (adjusting the constant in the capital O). Let $t = c\log d$ for $c \geq \max\{22, 16\log(2\delta^{-1})/(3\log d)\}$ and let $h_1,\dots,h_t$ be further random elements of G. For $1 \leq i \leq t$ let $H_i = \langle H_0, h_1,\dots,h_i\rangle$. Notice that $H = H_t$ is generated by $O(\log\delta^{-1} + d\log q)$ elements of G.
Since H_0 is irreducible, $\mathrm{End}_{\mathbb{F}_qH_0}(V) = \mathbb{F}_{q^s}$ for some s that is a multiple of f and divides d. For $1 \leq i \leq t$ let N_i be the degree of $\mathrm{End}_{\mathbb{F}_qH_i}(V)$ over $\mathbb{F}_{q^f}$.
We claim first that if $N_{i-1} > 1$ then $\mathrm{Prob}(N_i \leq N_{i-1}/2) \geq 1/2$. To see this let x generate $\mathrm{End}_{\mathbb{F}_qH_{i-1}}(V)$. Then since x is not centralised by G, at most half of the elements of G commute with x. If $[h_i, x] \neq 1$ then N_i is a proper divisor of N_{i-1} so the claim follows.
Now for $1 \leq i \leq t$ define $Y_i = 0$ if $N_{i-1} > 1$ and $N_i = N_{i-1}$ and $Y_i = 1$ otherwise. If $\sum_{i=1}^{t} Y_i \geq \log_2 d \geq \log_2(d/f)$ then $\mathrm{End}_{\mathbb{F}_qH}(V) = \mathrm{End}_{\mathbb{F}_qG}(V)$. Now, $\mathrm{Prob}(Y_i = 1) \geq 1/2$ by the claim, so by Lemma VII.4.1 with $p = 1/2$, $t = c\log d$ and $\epsilon = 1 - 2/(c\log 2)$,

$$\mathrm{Prob}(\sum_{i=1}^{t} Y_i \leq \log d) \leq e^{-\frac{1}{4}(1-\frac{2}{c\log 2})^2 c\log d} \leq e^{-\frac{3}{16}c\log d}$$

since $c \geq 22$. This is at most $\delta/2$ so the result follows. □

VII.4.4 Corollary (Correct action of normal subgroup)
For $G \leq \mathrm{GL}(d,q)$, let $N \trianglelefteq G$ and let $1 > \delta > 0$ be arbitrary. With probability $1-\delta$ any group H generated by $O(\log\delta^{-1} + d\log q)$ uniformly distributed elements of N has the same

submodule lattice as N, the same homogeneous components as N and, if N is irreducible, then $\mathrm{End}_{\mathbb{F}_q H}(V) = \mathrm{End}_{\mathbb{F}_q N}(V)$.

We will apply this result to commutators from G. A *normal* generating set for $N \trianglelefteq G$ is a set of elements of N which generate a group whose normal closure is N. Any set of elements of G' form a normal generating set for a normal subgroup of G which is contained in G'.

VII.5 Writing G over a smaller field

In this section, unless indicated otherwise, we let K be a finite field, let $G = \langle g_1, \ldots, g_m \rangle \leq \mathrm{GL}(d, K)$, and let $V = K^{1\times d}$ be the natural right KG-module. We assume that V is irreducible but not necessarily absolutely irreducible. We want to determine whether there exists a $t \in \mathrm{GL}(d, K)$ such that for $1 \leq i \leq m$ the matrices $t^{-1}g_i t$ have entries over some proper subfield of K. If such a t exists, we want to construct it for the smallest possible subfield F of K. We first analyse when such a t exists.
The K-algebra $K \otimes_F FG$ is isomorphic as a K-algebra to the group algebra KG by the F-linear map given by $x \otimes g \mapsto xg$ for $x \in K$ and $g \in G$. This isomorphism makes the tensor product $K \otimes_F \tilde{V}$ into a KG-module, for any FG-module $\tilde{V}$.

VII.5.1 Lemma
There exists a $t \in \mathrm{GL}(d, K)$ such that $t^{-1}Gt \in \mathrm{GL}(d, F)$ if and only if there exists an irreducible FG-module $\tilde{V}$ such that $V \cong K \otimes_F \tilde{V}$ as KG-modules.

Proof: If there exists a $\tilde{V}$ such that $V \cong K \otimes_F \tilde{V}$ as KG-modules then there is an irreducible representation of G over F which is K-equivalent to the natural representation of G on V, hence there is a t as required. On the other hand, such a t gives rise to a representation of G over F and thus to an FG-module $\tilde{V}$. The extension of scalars $K \otimes_F \tilde{V}$ of $\tilde{V}$ to K is isomorphic to V. If $\tilde{V}$ had a non-trivial FG-invariant subspace then V would have a non-trivial KG-invariant subspace, thus $\tilde{V}$ is irreducible. $\square$

For a subfield F of K we denote by $F[G]$ the set of F-linear combinations of the elements of G as an F-subalgebra of $K^{d\times d}$. This is also called the *F-enveloping algebra of G*. We denote the prime field of K by K_0.

VII.5.2 Proposition (Prime field enveloping algebra I)
Let $G \leq \mathrm{GL}(d, F)$ for F finite, and let $V := F^{1\times d}$ be irreducible. Let $E := \mathrm{End}_{F[G]}(V) = \mathrm{End}_{FG}(V)$ with $e = [E : F]$ and $d' = d/e$. Identify V with $E^{1\times d'}$ so that $F[G] = E^{d'\times d'}$. Set $L := F_0[G] \cap (E \cdot \mathbf{1})$. Then $F_0[G] \cong L^{d'\times d'}$ as an F_0-algebra.

Proof: Clearly $e = 1$ and $E = F$ if and only if V is absolutely irreducible.
Choosing an F-basis $(c_1, \ldots, c_e)$ of E we can express each element of E as an $(e \times e)$-matrix over F. The set V is an E-vector space and if $(b_1, \ldots, b_{d'})$ is an E-basis of V, then $(c_i b_j)_{i,j}$ is an F-basis of V. This choice of basis fixes an embedding $E^{d'\times d'} \subseteq F^{d\times d}$. Since the action of G on V is E-linear, we may assume that $G \leq \mathrm{GL}(d', E) \leq \mathrm{GL}(d, F)$. By the Density Theorem (see [CR90, (3.27)]), since V is an irreducible $F[G]$-module, $F[G] = E^{d'\times d'}$.

Now consider $B := F_0[G]$, which is an F_0-subalgebra of $F[G]$ such that $FB = E^{d'\times d'} = F[G]$. We first show that B is a simple algebra. If J is a nilpotent two-sided ideal of B, then FJ is a nilpotent two-sided ideal in $FB = E^{d'\times d'}$, contradicting its simplicity. So B has no nilpotent two-sided ideals and hence is semi-simple. It follows that B is a direct sum of simple algebras. The identity elements in these simple summands form an orthogonal set of central idempotents in B. A central idempotent in B is also central in $FB = E^{d'\times d'}$, and hence is the identity. Consequently, B is a simple algebra.
By the usual Wedderburn Theorems there exists an isomorphism $\psi : L^{s\times s} \to B$ for some s, and some extension L of F_0. The field L does not need to contain F. However, the elements of B corresponding to scalar matrices in $L^{s\times s}$ are central in B and hence also central in $FB = E^{d'\times d'}$. Therefore we can identify L with the centre of B and thus with some subfield of E. That is, $L = F_0[G] \cap (E \cdot \mathbf{1}) \subseteq E^{d'\times d'}$.
This produces a homomorphism of rings

$$\varphi : E^{s\times s} \cong E \otimes_L L^{s\times s} \cong E \otimes_L F_0[G] \to E^{d'\times d'} = F[G]$$

given by $\varphi(x \otimes b) = xb$. Since φ is surjective and $E^{s\times s}$ is simple, φ is an isomorphism and thus $s = d'$. □

VII.5.3 Proposition (Prime field enveloping algebra II)

Let G be as in Proposition VII.5.2, and suppose additionally that there is no proper subfield D of F such that there exists $t \in \mathrm{GL}(d, F)$ with $G^t \leq \mathrm{GL}(d, D)$. Then $F_0[G] = F[G]$.

Proof: Let $\psi : L^{d'\times d'} \to F_0[G]$ be the isomorphism given by Proposition VII.5.2. Let $e_{i,j} \in F_0[G]$ for $1 \leq i, j \leq d'$ be the image under ψ of a set of matrix units in $L^{d'\times d'}$. Then $e_{i,j}e_{k,l} = \delta_{j,k}e_{i,l}$ and the $e_{i,j}$ are an L-basis of $F_0[G]$.
We claim that the $e_{i,j}$ are an E-basis of $E^{d'\times d'}$. To see linear independence, let

$$\sum_{i,j=1}^{d'} \lambda_{i,j} e_{i,j} = 0,$$

then multiplying on the left by $e_{k,k}$ and on the right by $e_{l,l}$ shows that $\lambda_{k,l}e_{k,l} = 0$ and thus $\lambda_{k,l} = 0$ for all k, l. On the other hand, since $F_0[G]$ is the F_0-span of the elements of G, the $e_{i,j}$ span $E[G] = E^{d'\times d'}$ as an E-vector space. Thus they are an E-basis of $E^{d'\times d'}$.
Since $\mathbf{1}_{\mathbf{d'\times d'}} = \sum_{i=1}^{d'} e_{i,i}$ gives rise to a decomposition of $E^{1\times d'}$ as an E-vector space in which the direct summands are the row spaces of the $e_{i,i}$, it follows that these row-spaces are all one-dimensional.
Let $b'_1 \in E^{1\times d'}$ such that $\langle b'_1\rangle_E$ is the row space of $e_{1,1}$ and set $b'_i := b'_1 e_{1,i}$. Then $b'_i e_{j,k} = \delta_{i,j} b'_k$. If $t^{-1} \in E^{d'\times d'}$ has rows $b'_1, b'_2, \ldots, b'_{d'}$, then $t^{-1}e_{i,j}t$ has 1_E in position (i, j) and zeroes elsewhere. Thus $F_0[G^t] = L^{d'\times d'}$.
If V is absolutely irreducible then $E = F$ and so $L \leq F$. By assumption F is the smallest subfield of F over which G can be written, so $L = F = E$ as required.
Now consider the case $F < E$ and let $l := [E : L]$. Since $F[G^t] = F[G] = E^{d'\times d'}$ it follows that $F \cdot L^{d'\times d'} = E^{d'\times d'}$. Therefore, E is the smallest field containing both F and L implying

that l and $e = [E : F]$ are coprime. Let $D := F \cap L$. Then D is a field with $[F : D] = l$ and $[L : D] = e$.
We claim that G can be written over D. Let $(c'_1, \dots, c'_e)$ be a D-basis of L. Then it is also an F-basis of E, since every element of E is an F-linear combination of elements of L. Now change basis in $V = F^{1\times d}$ from the F-basis $(c_i b'_j)_{i,j}$ to $(c'_i b'_j)_{i,j}$ using a base change $s \in \mathrm{GL}(d, F)$, to get $G^{ts} \le D^{d\times d}$.
However, by our assumption that F is the smallest subfield of F over which G can be written, we get $D = F \cap L = F$ and thus $F \le L$. From $F \cdot L^{d'\times d'} = E^{d'\times d'}$ follows now immediately $L = E$.
Since we have proved $L = E$ in both cases and we already know that $F_0[G]$ is isomorphic to $L^{d'\times d'}$ as F_0-algebras and contained in $F[G] = E^{d'\times d'}$ the proposition follows. □

VII.5.4 Theorem (Characterisation of smallest possible field)
Let $G \le \mathrm{GL}(d, K)$ act irreducibly on its natural module. Then there is a unique smallest subfield F of K such that there exists $t \in \mathrm{GL}(d, K)$ with $G^t \le \mathrm{GL}(d, F)$. This F is uniquely determined by $K_0[G] \cap (K \cdot \mathbf{1}_{d\times d}) = F \cdot \mathbf{1}_{d\times d}$. Furthermore, $K_0[G] \cong E^{(d/e)\times(d/e)}$, where $E = \mathrm{End}_{F[G^t]}(F^{1\times d})$ is an extension field of F of degree e.

Proof: Since K is finite, there is a smallest subfield F of K such that there exists a $t \in \mathrm{GL}(d, K)$ with $G^t \le \mathrm{GL}(d, F)$. By Lemma VII.5.1 the natural $F[G^t]$-module $V := F^{1\times d}$ is irreducible. Then Proposition VII.5.3, applied to G^t, shows that $F_0[G^t] = F[G^t] = E^{(d/e)\times(d/e)}$. Since the F-scalar matrices are central in $F^{d\times d}$, the theorem follows immediately. □

From now on we assume that the equivalent statements in Lemma VII.5.1 hold for some subfield F of K. We now develop some theory which leads to an algorithm that finds a t and the smallest possible subfield F, or proves that none exists.
Let $B = \{b_1, \dots, b_f\}$ be an F-basis for K, such that $b_1 = 1$. We start by noting that if the natural KG-module V is isomorphic to $K \otimes_F \tilde{V}$ as KG-modules, then $V \cong \bigoplus_{i=1}^f b_i \otimes_F \tilde{V}$ as FG-modules. We therefore identify V with $K \otimes_F \tilde{V}$ and $\tilde{V}$ with $1 \otimes_F \tilde{V}$, respectively via this second isomorphism and thus write $b_i v$ instead of $b_i \otimes_F v$ and v for $1 \otimes_F v \in \tilde{V}$.

VII.5.5 Lemma
Let F be a subfield of K such that the equivalent statements in Lemma VII.5.1 hold and let $\tilde{V}$ be as above. Then the F-dimension of $\mathrm{End}_{FG}(\tilde{V})$ is equal to $e = \dim_F(\mathrm{End}_{KG}(V))$.

Proof: This result is a consequence of the fact that

$$\mathrm{End}_K(K \otimes_F \tilde{V}) \cong K \otimes_F \mathrm{End}_F(\tilde{V}),$$

see for example [CR90, (2.38)]. To assist the reader and set up some notation, we first prove that $\mathrm{End}_K(V) \cong K \otimes_F \mathrm{End}_F(\tilde{V})$. If $(m_1, \dots, m_d)$ is an F-basis of $\tilde{V}$, then $(b_i m_j)_{1\le i\le f, 1\le j\le d}$ is an F-basis of V and $(m_j)_{1\le j\le d} = (1 \otimes_F m_j)_{1\le j\le d}$ is a K-basis of V. Hence every $\varphi \in \mathrm{End}_K(V)$ can be written in a unique way as

$$\varphi = \sum_{i=1}^{f} b_i \varphi_i$$

with $\varphi_i \in \mathrm{End}_F(\tilde{V})$. Therefore $\mathrm{End}_K(V) \cong K \otimes_F \mathrm{End}_F(\tilde{V})$.

Next we show that $\mathrm{End}_{KG}(V) \cong K \otimes_F \mathrm{End}_{FG}(\tilde{V})$. If $\psi \in \mathrm{End}_K(V)$ then $\psi \in \mathrm{End}_{KG}(V)$ if and only if $\psi(vg) - \psi(v)g = 0$ for all $g \in G$ and $v \in V$. By K-linearity, it suffices to check this for $v \in (m_j)_{1 \le j \le d}$. Writing $\psi = \sum_{i=1}^{f} b_i \psi_i$ with $\psi_i \in \mathrm{End}_F(\tilde{V})$ shows that

$$\psi \in \mathrm{End}_{KG}(V) \Leftrightarrow \sum_{i=1}^{f} b_i(\psi_i(m_j g) - \psi_i(m_j)g) = 0$$

for all $1 \le j \le d$ and all $g \in G$ and by the uniqueness above this in turn is equivalent to $\psi_i(m_j g) - \psi_i(m_j)g = 0$ for all i, j, and g. This proves that $\mathrm{End}_{KG}(V) \cong K \otimes_F \mathrm{End}_{FG}(\tilde{V})$.
Now the F-dimension of $K \otimes_F \mathrm{End}_{FG}(\tilde{V})$ equals $f \dim_F(\mathrm{End}_{FG}(\tilde{V}))$ and the F-dimension of $\mathrm{End}_{KG}(V)$ is ef, so $e = \dim_F(\mathrm{End}_{FG}(\tilde{V}))$, as required. □

VII.5.6 Lemma

Let F be a subfield of K such that the equivalent statements in Lemma VII.5.1 hold and let $\tilde{V}$ and $B = \{b_1, \dots, b_f\}$ be as above. Let $w \in \tilde{V}$ and $x_1, \dots, x_k \in FG$. The set of vectors $\{\sum_{i=1}^{f} b_i w x_j \mid 1 \le j \le k\}$ is linearly independent over K if and only if it is linearly independent over F.

Proof: For $1 \le j \le k$ let $c_j = \sum_{i=1}^{f} b_i w x_j$. Let $a = \sum_{i=1}^{f} b_i$ and for $1 \le j \le k$ let $d_j = a^{-1} c_j = w x_j \in \tilde{V}$. The d_j are linearly independent over K if and only if the c_j are linearly independent over K. Since each c_j has been multiplied by the *same* element $a^{-1} \in K$, the same statement is true over F.
The set $\{d_j : 1 \le j \le k\}$ is linearly independent over F if and only if $\{1 \otimes d_j : 1 \le j \le k\}$ is linearly independent over K. The result follows from the identification of the two sets. □

We are now in a position to attack the original problem of this section. The method for finding the matrix t as in Lemma VII.5.1 is an instance of the "standard basis method" which is usually used for finding homomorphisms from irreducible modules into arbitrary modules. In fact, we use the FG-module isomorphism $V \cong \bigoplus_{i=1}^{f} b_i \tilde{V}$ and then find an FG-homomorphism from $\tilde{V}$ to V. We will show that the K-span of the image is V such that every F-basis of $\tilde{V}$ is mapped to a K-basis of V. The representing matrices with respect to such a basis are the same as those on $\tilde{V}$ and thus are over F.
To describe this, we first define the term "standard basis", which is most easily done by means of an algorithm. Note that this concept was described by Parker in [Par84, Section 6].

VII.5.7 Definition (Standard basis)

Let $G = \langle g_1, \dots, g_m \rangle$ be a group, let V be a right FG-module, and let $0 \neq v \in V$. The *standard basis* starting at v with respect to $(g_1, \dots, g_m)$ is a list of vectors.
Starting with (v), successively apply each of $g_1, \dots, g_m$ in this order to each vector in the list, finding all m images of one vector before progressing to the next. Whenever the result is not contained in the F-linear span of the previous vectors, add it to the end of the list. This produces a basis $SB(V, v, (g_1, \dots, g_m))$ for a non-trivial G-invariant subspace, which is V itself if V is irreducible.

We next present a theorem which is useful for isomorphism testing with an irreducible module. Although the ideas are described in [Par84, Section 6], we include the exact formulation and a proof, since these arguments are used in an intricate way later in the determination of the matrix t from Lemma VII.5.1.

VII.5.8 Theorem (Isomorphism test)
Let $G = \langle g_1, \ldots, g_m \rangle$ be a group, V a finite-dimensional, irreducible FG-module and $E := \mathrm{End}_{FG}(V)$ its endomorphism ring. Furthermore let W be a finite-dimensional FG-module, and $c \in FG$ an element such that $\dim_F(\ker_V(c)) = \dim_F(E)$. Let $N := \ker_W(c) = \{w \in W : wc = 0\}$. There are two possibilities:

- *If $N = \{0\}$, then $V \ncong W$ as FG-modules.*
- *If $N \neq \{0\}$, let $0 \neq w \in N$. Then $SB(W, w, (g_1, \ldots, g_m))$ spans W if and only if $V \cong W$. If $V \cong W$ as FG-modules, then the F-linear map $\varphi : V \to W$ mapping $SB(V, v, (g_1, \ldots, g_m))$ to $SB(W, w, (g_1, \ldots, g_m))$ is an FG-isomorphism.*

  Hence, if $V \cong W$, then for any non-zero $w_1, w_2 \in N$ there is an FG-automorphism of W mapping w_1 to w_2.

Remark: This provides an efficient algorithm to test whether $V \cong W$ as FG-modules, and if so, to construct an explicit isomorphism, provided V is known to be irreducible and $\dim_F(E)$ is known. The algorithm finds c, computes $SB(V, v, (g_1, \ldots, g_m))$, and then computes N, looking for $0 \neq w \in N$. If an appropriate c is found and $N \neq 0$ then the algorithm computes $SB(W, w, (g_1, \ldots, g_m))$. This computation verifies whether φ is an FG-isomorphism. Thus the algorithm either computes an isomorphism φ or proves that none exists.

Proof: The kernel $\ker_V(c)$ is E-invariant and thus a vector space over E. By assumption its E-dimension is 1. Every FG-module isomorphism between V and W maps $\ker_V(c)$ into N. If $W \cong V$, then $\dim_F(N) = \dim_F(\ker_V(c))$ and so N is a 1-dimensional vector space over $E' := \mathrm{End}_{FG}(W)$. Therefore, for all $(w, w') \in N \times N$ with $w \neq 0 \neq w'$ there is an automorphism $e' \in E'$ with $e'(w) = w'$. Thus, if we pick any $0 \neq v \in \ker_V(c)$ and any $0 \neq w \in N$, then there is an isomorphism $\varphi : V \to W$ that maps v to w. This isomorphism necessarily maps $SB(V, v, (g_1, \ldots, g_m))$ to $SB(W, w, (g_1, \ldots, g_m))$ proving our claims. □

VII.5.9 Theorem (Writing G over a smaller field)
Let the global assumptions for this section on G apply. Let again $e := \dim_K(\mathrm{End}_{KG}(V))$ be the degree of the splitting field. We assume that e is already computed.
There exists a $c \in K_0G$ such that $\dim_K \ker_V(c) = e$. Let $w \in \ker_V(c)$ with $w \neq 0$, let $B := SB(V, w, (g_1, \ldots, g_m))$ and let $t^{-1} \in \mathrm{GL}(d, K)$ have the vectors in B as rows.
Let F be the smallest subfield of K for which there is an $r \in \mathrm{GL}(d, K)$ such that $r^{-1}Gr \leq \mathrm{GL}(d, F)$ (see Theorem VII.5.4), then $t^{-1}Gt \leq \mathrm{GL}(d, F)$ as well. That is, $t^{-1}Gt$ writes G over the smallest possible field.
Let $1 > \delta > 0$ be arbitrary. There is an algorithm to find c and construct t in Las Vegas $O(md^3 \log \delta^{-1})$ field operations with failure probability at most δ. If the algorithm is allowed to run indefinitely, then it finishes with probability 1 and the expected number of attempts to find c is bounded above by a constant which does not depend on d or $|K|$. Each attempt needs $O(md^3)$ field operations.

Proof: Let $B = \{b_1, \ldots, b_f\}$ and $\tilde{V}$ be as in the paragraph before Lemma VII.5.5 and let $E := \mathrm{End}_{F[G^r]}(\mathbb{F}^{1\times d})$, then by Lemma VII.5.5, we have $e = [E : F]$ and know it in advance, since we know $\mathrm{End}_{KG}(V)$ by a MeatAxe run.
We apply the standard basic technique to $V \cong \bigoplus_{i=1}^{f} b_i \tilde{V}$, where $\{b_1, \ldots, b_f\}$ is an F-basis for K, as before. Note that we assume that the isomorphism exists but do not yet have it explicitly! We attempt to compute an FG-homomorphism $\varphi : \tilde{V} \to \bigoplus_{i=1}^{f} b_i \tilde{V}$, and will either succeed or show that $f = 1$.
1. First we look for $c \in FG$ such that $\dim_F(\ker_{\tilde{V}}(c)) = \dim_F(\mathrm{End}_{FG}(\tilde{V}))$. We do not know F nor have $\tilde{V}$, but by Lemma VII.5.5, $e = \dim_F(\mathrm{End}_{FG}(\tilde{V}))$ and

$$f \cdot \dim_F(\ker_{\tilde{V}}(c)) = \dim_F(\ker_V(c)) = f \cdot \dim_K(\ker_V(c)).$$

Given a possible c, we can compute $\dim_K(\ker_V(c))$, and stop if this is equal to e.
To find c we produce random K_0-linear (and hence F-linear) combinations c of elements of G and stop if $\dim_K(\ker_V(c)) = e$. The results in [HR94] and [IL00] show that there is an upper bound b not depending on $|K|$, $|F|$ and d for the probability that a random element $c \in K_0[G] = F_0[G] \cong F[G^t] \cong E^{(d/e)\times(d/e)}$ (compare Theorem VII.5.4) has $\dim_K(\ker_V(c)) \neq e$. Thus $\log_b \delta^{-1}$ tries will succeed with probability at least $1-\delta$. Note that these arguments prove that such a $c \in K_0[G]$ actually exists!
2. Assume that we have found such a $c \in FG$, given in its action on V. We compute a nonzero $w \in \ker_V(c)$. This has the form $w = \sum_{i=1}^{f} b_i w_i$ for some $w_i \in \tilde{V}$, and since $\ker_V(c) = \bigoplus_{i=1}^{f} b_i \ker_{\tilde{V}}(c)$ all of the w_i lie in $\ker_{\tilde{V}}(c)$.
We can now use the standard basis algorithm from Definition VII.5.7 with w in place of v, testing for K-linear independence of the resulting vectors. In fact, this will test for F-linear independence as is required — note that this works without knowing F explicitly, provided we only take linear combinations over K_0 to find $c \in FG$! We need to prove these claims.
By Theorem VII.5.8 and the fact that all summands $b_i \tilde{V}$ are isomorphic to $\tilde{V}$ as FG-modules, we conclude that for all w and all nonzero $v \in \ker_{\tilde{V}}(c)$, there is a unique FG-monomorphism from $\tilde{V}$ into V, mapping $SB(\tilde{V}, v, (g_1, \ldots, g_m))$ to $SB(V, w, (g_1, \ldots, g_m))$. Note that the latter is a basis for the image of this FG-homomorphism, which is an F-subspace, and that these standard bases are defined using F-linear independence.
We do yet know F, so we cannot yet test for F-linear independence. We now make some observations which allow us to apply Lemma VII.5.6. By the last statement of Theorem VII.5.8, for $1 \leq i \leq f$ there is an automorphism $\alpha_i \in \mathrm{End}_{FG}(\tilde{V})$ mapping w_i to w_1. Thus, there is an automorphism α of the FG-module $V \cong \oplus_{i=1}^{f} b_i \tilde{V}$ with $\alpha(b_i w_i) = b_i w_1$ for all i and thus $\alpha(w) = \sum_{i=1}^{f} b_i w_1$.
By Lemma VII.5.6, with $x_1, x_2, \ldots, x_k \in FG$, the tuple $t := (\sum_{i=1}^{f} b_i w_1 x_j)_{1\leq j\leq k}$ is F-linearly independent if and only if it is K-linearly independent. This in turn holds if and only if the tuple $(w x_j)_{1\leq j\leq k}$ is K-linearly independent, since it is mapped to t by α.
This proves our claim that we in fact compute $SB(V, w, (g_1, \ldots, g_m))$ with testing for F-linear independence.
3. By the above arguments, the result $SB(V, w, (g_1, \ldots, g_m))$ is an F-basis of an FG-submodule of V that is isomorphic to $\tilde{V}$. In particular, the representing matrices for the g_i expressed with respect to this basis contain only coefficients from F. Since $SB(V, w, (g_1, \ldots, g_m))$ is K-linearly independent it clearly is a K-basis of V and we have found our base change matrix t explicitly.

Finally, we determine the smallest subfield F of K containing all coefficients of $t^{-1}g_it$ for $1 \leq i \leq m$.
All individual steps are $O(md^3)$ proving our claims. If the search for c is repeated indefinitely, the probability of success tends to 1 and the expected number of tries is $1/(1-b)$. □

In summary, our Las Vegas algorithm to write G over a subfield proceeds as follows. We assume that we have already tested V for absolute irreducibility, and hence know the degree $e = \dim_K(\mathrm{End}_{KG}(V))$ of the splitting field.

1. Choose a uniformly distributed random element $c \in K_0G$ in its action on V and compute $\ker_V(c)$. Repeat this until $\dim_K(\ker_V(c)) = e$ or fail after $O(\log \delta^{-1})$ tries.
2. Take $0 \neq w \in \ker_V(c)$ and compute $B := SB(V, w, (g_1, \ldots, g_m))$ using K-linear independence.
3. Let $t^{-1} \in \mathrm{GL}(d, K)$ have the vectors in B as rows, and find the smallest subfield of K containing all entries of all $t^{-1}g_it$.

By Theorem VII.5.9 this algorithm either fails in step 1 with bounded probability or finds the smallest possible subfield F of K together with an explicit base change matrix t to write G over F. If G cannot be written over a smaller field then $F = K$ in step 3.

VII.6 Restriction to a subgroup of the derived group

We now consider the case of a matrix group $G = \langle g_1, \ldots, g_m \rangle \leq \mathrm{GL}(d, q)$ acting absolutely irreducibly on the natural module $V = \mathbb{F}_q^d$, that cannot be written over a smaller field with trivial scalars. Our algorithm finds a reduction provided that G lies in $\mathcal{C}_3$ or $\mathcal{C}_5$, or the derived group of G is not absolutely irreducible. If none of these is the case, it might still find a reduction but might also report that G does not lie in $\mathcal{C}_3$ or $\mathcal{C}_5$ and that G' is absolutely irreducible.
In Sections VII.(6.2) and following we refer to a normal subgroup N of G that is contained in the derived group G'. In Section VII.(6.1) we describe a method of computing a subgroup H of such an N which can be used instead. In each of the following sections, we analyse the complexity of the algorithms used in terms of number of field operations.
Note that some of these complexity results involve a prescribed bound δ for the failure probability. If we do several such steps consecutively, we have to adjust the individual bounds because the complete procedure fails if any of the intermediate steps fails. We analyse the overall picture in Section VII.7.

(6.1) Computing a normal subgroup of the derived group

We proceed differently depending on whether G has at most 7 non-scalar generators. If this holds then we start by making all commutators of generators. If these are all scalar then G' is scalar, since G modulo scalars is abelian. Thus the methods of Section VII.(6.7) apply with G' known to be scalar. If any of them is found to be non-scalar we proceed to generate H as described below.
If there are more than 7 non-scalar generators then we compute 10 commutators of random elements of G. If these are all scalar then for the first non-scalar generator g_i of G we test whether $[g_i, g_j]$ is scalar for $j > i$. If this holds for all j then Proposition VII.6.9 applies.

Otherwise, we now have a nonempty set S of non-scalar commutators. We compute a subgroup H of a normal subgroup N of G that is contained in G' by the methods of Section VII.4. Namely, we use a variant of RATTLE [LGM02, LGO97a] to produce a set T of $O(\log\delta^{-1} + d\log q)$ elements of $N = \langle S\rangle^G$. Computing T has complexity $O(d^3\log\delta^{-1} + d^4\log q)$ field operations, since one random element can be produced at cost $O(d^3)$ after initial setup.
By Corollary VII.4.4 the group $H = \langle T\rangle \leq N$ has the same submodule structure and (if N is irreducible) centraliser algebra as N with probability at least $1-\delta$. That is, we can use H instead of N in the methods described in subsequent sections.

(6.2) A case analysis for $\mathbf{N \trianglelefteq G}$ with $\mathbf{N \leq G'}$

From now on we assume that H is given by s generators and is a subgroup of a normal subgroup N of G that is contained in G'. Note that $s = O(\log\delta^{-1} + d\log q)$ if we use the method from Section VII.(6.1), but our algorithms in Sections VII.(6.3) to VII.(6.6) can be applied to any normal subgroup. The group H might be smaller than N, but with probability $1-\delta$ the structure of the natural module is the same for both groups.
Since $N \trianglelefteq G$ there are only five possibilities, by Clifford's Theorem.

1. N is absolutely irreducible on V.
2. N is irreducible but not absolutely irreducible.
3. N is reducible, and V is a direct sum of more than one homogeneous component.
4. N is reducible, and V splits into a direct sum of isomorphic irreducible N-submodules of dimension greater than 1, so that in particular N is non-scalar.
5. N is reducible, and V splits into a direct sum of isomorphic 1-dimensional submodules, so that N is scalar.

We proceed differently in each of these five cases, but, assuming G is in $\mathcal{C}_3$ or $\mathcal{C}_5$ or N is not absolutely irreducible, in each case we find a reduction with probability δ of failure. By a reduction we mean a non-trivial homomorphism onto a smaller group or an isomorphism to a situation with smaller input size. To distinguish these cases, we first run the MeatAxe on H in place of N. This uses $O(d^3 s\log\delta^{-1})$ field operations since H is given by s generators, where δ is the upper bound for the failure probability for this step. This MeatAxe run decides whether we run the algorithms for case 1, case 2, or one of cases 3 and 4. In case 2, it returns a field generator of the endomorphism algebra. In cases 3 and 4, it returns a proper H-submodule. Note that if we use the methods in Section VII.(6.1) to compute H, then case 5 is never found here because it is detected earlier on (see Section VII.(6.7)).

(6.3) Absolutely irreducible normal subgroup

We continue to assume that N is a normal subgroup of G that is contained in G', and add the assumption that the MeatAxe has shown that H and hence N act absolutely irreducibly.
We first note the following lemma, which rules out case 1 for $\mathcal{C}_3$.

VII.6.1 Lemma
If G lies in class $\mathcal{C}_3$ then G', and hence H and N, are not absolutely irreducible.

Proof: Assume that there is an $\mathbb{F}_{q^e}$-vector space structure on V, such that G acts semilinearly. Then G' acts $\mathbb{F}_{q^e}$-linearly and thus $\mathrm{End}_{\mathbb{F}_q G'}(V) \neq \mathbb{F}_q$. Thus G' is not absolutely irreducible. □

We can therefore assume in the case of this section that G lies in class $\mathcal{C}_5$.

VII.6.2 Lemma (Compare [GLGO06, Lemma 4.1])
If G can be written over $\mathbb{F}_{q_0}$ modulo scalars in $\mathbb{F}_q$, then $N \le G'$ can be written over $\mathbb{F}_{q_0}$.

Proof: Multiplying each of $g, h \in G$ by a fixed scalar fixes the value of $[g, h]$. □

VII.6.3 Theorem (Recognition of $\mathcal{C}_5$)
Consider $G = \langle g_1, \dots, g_m \rangle \le \mathrm{GL}(d, q)$ or its projective version and let $1 > \delta > 0$ be given. Let $H = \langle n_1, \dots, n_s \rangle \le N \trianglelefteq G$ with $N \le G'$ and let H be known (by a MeatAxe run) to be absolutely irreducible. Then in $O(sd^3 \log \delta^{-1} + md^3)$ field operations we can construct a homomorphism from G to $\mathrm{PGL}(d, q_0)$ for minimal q_0 or prove that G and $\overline{G}$ are not in $\mathcal{C}_5$. The algorithm returns `fail` with probability at most δ.

Proof: Since H is absolutely irreducible we use the methods of Section VII.5 to find a matrix t such that $t^{-1}Ht \le \mathrm{GL}(d, q')$ with δ as an upper bound for the failure probability. This automatically finds the smallest prime power q' with this property. Notice that the vector chosen in the kernel of c by the standard basis method is unique up to multiplication by elements of $\mathrm{End}_{\mathbb{F}_q H}(V) = \mathbb{F}_q$. From this point on the algorithm is guaranteed to determine whether G lies in $\mathcal{C}_5$.
Now examine $h_i := t^{-1}g_i t$ and check whether it can be written as a product of a scalar $\lambda_i \in \mathbb{F}_q$ and an element of $\mathrm{GL}(d, q_0)$ for $q' < q_0 < q$. To do so, notice that if $h_i \in \lambda_i \mathrm{GL}(d, q_0)$, then the quotient between any two nonzero entries in h_i lies in $\mathbb{F}_{q_0}$. Therefore we may take λ_i to be any nonzero entry of h_i and then find the minimal field $\mathbb{F}_{q_0}$ containing all entries of h_i/λ_i. This enables us to set up a homomorphism from G to $\mathrm{PGL}(d, q_0)$ with kernel $G \cap Z(\mathrm{GL}(d, q))$, and so a reduction has been completed. For the projective group $\overline{G}$, we get a homomorphism into $\mathrm{PGL}(d, q_0)$, which could be an isomorphism. Even if this is the case, we have reduced to a smaller field.
If no smaller field is found, the procedure reports that G does not lie in $\mathcal{C}_3$ or $\mathcal{C}_5$ and that G' is absolutely irreducible. □

Note that although we work with H instead of N, since H is absolutely irreducible so is N.

(6.4) Irreducible but not absolutely irreducible

We continue to assume that N is a normal subgroup of G that is contained in G'. As described in Section VII.(6.2) we assume that the MeatAxe has proved that H acts irreducibly but not absolutely irreducibly, and so N is guaranteed to act irreducibly, but with probability at most δ the endomorphism ring $\mathrm{End}_{\mathbb{F}_q N}(V)$ may be smaller than $\mathrm{End}_{\mathbb{F}_q H}(V)$. We will deal with this possibility at the end of this section, and in general talk about N rather than H.

VII.6.4 Proposition
If G is absolutely irreducible and $N \trianglelefteq G$ is irreducible but not absolutely irreducible then G is semilinear.

Proof: Since N is irreducible but not absolutely irreducible, $E = \mathrm{End}_{\mathbb{F}_q N}(V) = \mathbb{F}_{q^e}$ for some $e > 1$. Let $C \in \mathrm{GL}(d, q)$ generate the multiplicative group of E.
For all $h \in N$, $g \in G$, by definition $hC = Ch$, thus $h^g C^g = C^g h^g = h_1 C^g = C^g h_1$, for some $h_1 \in N$. As h varies over N the element h_1 takes every value in N, therefore $\langle C\rangle^g = \langle C\rangle$, and so $C^g = C^k$ for some k. Suppose that $C^i + C^j = C^l$, then $(C^i)^g + (C^j)^g = (C^l)^g$ so g acts as field automorphisms on $\mathbb{F}_{q^e}$ and thus G is semilinear. □

VII.6.5 Theorem (Recognition of $\mathcal{C}_3$)
Let $G = \langle g_1, \dots, g_m\rangle \leq \mathrm{GL}(d, q)$ or its projective version. Let $N = \langle n_1, \dots, n_s\rangle \trianglelefteq G$ be known to be irreducible but not absolutely irreducible. In deterministic $O(d^4 \log q + md^3)$ field operations we can construct two homomorphisms, one to the cyclic group of order e for some divisor e of d and a second from the kernel of the first to $\mathrm{GL}(d/e, q^e)$ or $\mathrm{PGL}(d/e, q^e)$.

Proof: When N is irreducible but not absolutely irreducible, the MeatAxe returns a generator C of the field $\mathbb{F}_{q^e} = \mathrm{End}_{\mathbb{F}_q N}(V)$ realised as a matrix in $\mathrm{GL}(d, q)$ together with e. Note that $e \leq d$. The matrix C does not need to generate the multiplicative group of $\mathbb{F}_{q^e}$, but its powers $C^0, C^1, \dots, C^{e-1}$ are $\mathbb{F}_q$-linearly independent.
As shown in Proposition VII.6.4, the group G acts by conjugation as field automorphisms on $\mathbb{F}_{q^e} = \mathrm{End}_{\mathbb{F}_q N}(V)$ and thus on the group $\langle C\rangle$. We can immediately read off this action using $O(md^3)$ field operations by computing the matrices $C, C^q, C^{q^2}, \dots, C^{q^{e-1}}$, conjugating C with the generators of G and looking up the result. Computing these matrices requires at most $O(d^4 \log q)$ field operations and space for $O(d)$ matrices, since $e \leq d$. Computing this action provides a homomorphism from G to the cyclic group of order e, because the above mentioned matrices are the possible images of C under automorphisms of $\mathbb{F}_{q^e}$.
In addition, C gives an explicit $\mathbb{F}_{q^e}$-vector space structure on V. To get the $\mathbb{F}_{q^e}$-span of a vector $v \in V$ we compute $v, vC, vC^2, \dots, vC^{e-1}$. In this way we can perform a spinning algorithm for V as an $\mathbb{F}_{q^e}$-vector space. All computations are with vectors over $\mathbb{F}_q$, but whenever we produce a new vector v that does not lie in the $\mathbb{F}_q$-span of what we already have, we not only add v but also $vC, vC^2, \dots, vC^{e-1}$ by repeatedly multiplying with C. This spinning algorithm gives us a base change to an $\mathbb{F}_{q^e}$-adapted basis. It needs at most $O(md^3)$ field operations.
The kernel of the action as field automorphisms acts $\mathbb{F}_{q^e}$-linearly on the original space and we read off this action using the above base change to the $\mathbb{F}_{q^e}$-adapted basis. This therefore also leads to a reduction for the kernel by reducing the input size to $(d/e) \times (d/e)$-matrices over $\mathbb{F}_{q^e}$.
Altogether, we have found a significant reduction using $O(d^3(d \log q + m))$ field operations and memory for $O(d)$ matrices.
Note that since scalars from $\mathbb{F}_q$ do not alter the action of elements of G as field automorphisms on $\mathbb{F}_{q^e}$ the same procedure works for the projective case $\overline{G}$.
The homomorphism is the same as in the matrix case, the kernel is a subgroup of $\mathrm{PGL}(d, q)$ and we construct a map from the kernel into $\mathrm{PGL}(d/e, q^e)$ by writing the matrices over the bigger field. This map in turn has a kernel, since we divide out more scalars. However, this second kernel only contains $\mathbb{F}_{q^e}$-scalars modulo $\mathbb{F}_q$-scalars, which can be handled easily. Thus this case can be handled in the projective situation. □

We finish with a discussion of the possibility that $\mathbb{F}_{q^e} = \mathrm{End}_{\mathbb{F}_q H}(V) \neq \mathrm{End}_{\mathbb{F}_q N}(V)$, which happens with probability bounded by δ. If $\mathrm{End}_{\mathbb{F}_q H}(V) \neq \mathrm{End}_{\mathbb{F}_q N}(V)$ then the elements of G

need not act as field automorphisms on $\mathbb{F}_{q^e}$, which we notice during the above computation. However, if G is contained in $\mathcal{C}_3$ then they *will* act as field automorphisms on some subfield of $\mathbb{F}_{q^e}$ that properly contains $\mathbb{F}_q$. Thus, if e has not too many divisors, we test for each divisor i of e whether G acts as field automorphisms on $\mathbb{F}_{q^i}$. In theory we could use the fact that e has $O(\log d)$ prime divisors (by the Prime Number Theorem) to find a subfield $\mathbb{F}_{q^r} \subseteq \mathrm{End}_{\mathbb{F}_q N}(V)$ and hence a reduction in $O(\log d)$ attempts, but in practice we always find the divisors of e.
To find a generating element C' for the field $\mathbb{F}_{q^i}$ we proceed as follows. First we find a polynomial $h \in \mathbb{F}_q[x]$ of degree less than e such that $h(C)$ has order $q^e - 1$. This involves only polynomial arithmetic using the minimal polynomial of C. Evaluating $h(C)$ can be done in at most $O(d^4)$ field operations. We then power up $h(C)$ to exponent $(q^e - 1)/(q^i - 1)$, which can be done using at most $O(d^4 \log q)$ field operations.
If no divisor works then the algorithm reports `fail`, that G is not in $\mathcal{C}_3$ and that N and the derived group are absolutely irreducible. Note that if N is not absolutely irreducible then the algorithm is guaranteed to find that G is in $\mathcal{C}_3$ at this point, therefore if failure is reported the algorithm adds generators to H until it is absolutely irreducible, and then returns to the test of Section VII.(6.3). However this can only happen with probability δ.
If there are too many divisors, we simply report `fail`. This happens with probability at most δ by Corollary VII.4.4 and never in practice.

(6.5) More than one homogeneous component

We continue to assume that N is a normal subgroup of G that is contained in G'. As described in Section VII.(6.2) we assume that the MeatAxe has proved that H acts reducibly by finding an explicit proper nontrivial submodule V' of the natural module.
First we prove a lemma which will eventually be used to find an irreducible H-submodule that with probability $1 - \delta$ is an N-submodule.

VII.6.6 Lemma (Finding an irreducible module)
Let $N = \langle n_1, \dots, n_s \rangle \leq \mathrm{GL}(d, q)$ act reducibly on the natural module V and let $1 > \delta > 0$ be given. Given a submodule $V' < V$, an irreducible N-subfactor can be found in Las Vegas $O(sd^3 \log(\delta^{-1} \log d))$ field operations, with probability of failure at most δ.

Proof: We repeatedly use the MeatAxe to find an irreducible subfactor of $V|_{\mathbb{F}_q N}$. Initially, we have a submodule $V' < V$. We run the MeatAxe either on V/V' or on V', whichever has the smaller dimension. If we find a proper submodule, we repeat the same technique. Since we halve the dimension in each step, this terminates after at most $\log_2 d$ runs of the MeatAxe using at most $O(d^3 s 2^{-3i} \log \delta'^{-1})$ field operations in step i, where δ' is an upper bound for the failure probability in each step. To bound the overall failure probability of this whole procedure by δ, we define $\delta' := \delta / \log_2 d$. Since $\sum_{i=1}^{\infty} 2^{-3i} < 1$, the overall cost for finding an irreducible subfactor is $O(sd^3 \log(\delta^{-1} \log d))$. □

VII.6.7 Theorem (Construction of a block action)
Let $G = \langle g_1, \dots, g_m \rangle \leq \mathrm{GL}(d, q)$ or its projective version, and let $1 > \delta > 0$ be given. Let $N = \langle n_1, \dots, n_s \rangle \trianglelefteq G$ be known (by a MeatAxe run) to be reducible.

In Las Vegas $O(sd^3\log(\delta^{-1}\log d))$ field operations we can either construct a homomorphism from G to a permutation group with kernel the pointwise stabiliser of the set of homogeneous components of $V|_{\mathbb{F}_qN}$ or prove that $V|_{\mathbb{F}_qN}$ has a single homogeneous component. The probability of failure is bounded from above by δ.

Proof: By assumption $V|_{\mathbb{F}_qN}$, as an $\mathbb{F}_qN$-module, is a direct sum of homogeneous components $C_1,\dots,C_k$ with $k>1$. The C_i form a block system exhibiting an imprimitive action of G and N is a normal subgroup of the kernel of the action on blocks. We only have to find the action on this block system to find a reduction.

By Lemma VII.6.6 we can find an irreducible N-subfactor in $O(sd^3\log(\delta^{-1}\log d))$ field operations with probability of failure δ. This subfactor is an irreducible module $\tilde{S}$ and we can now apply the isomorphism testing procedure described in Theorem VII.5.8 once to give a homomorphism of $\tilde{S}$ into $V|_{\mathbb{F}_qN}$ and thus an irreducible submodule S with $O(sd^3)$ field operations. Note that when we proved the final subfactor in the procedure described in Lemma VII.6.6 to be irreducible, we constructed the algebra word c that is needed for isomorphism testing, namely a word describing an algebra element with nullity the dimension of the centraliser of N.

Such an irreducible module S is all we need to run the MinBlocks procedure described in [HLGOR96b] which needs $O(sd^3)$ field operations to compute the block system or reports that there is none. If the latter occurs, then there is a single homogeneous constituent and we apply the algorithms of Section VII.(6.6). Otherwise this provides a non-trivial homomorphism onto a permutation group and thus a reduction. The overall complexity is $O(sd^3\log(\delta^{-1}\log d))$.

Since $\mathbb{F}_q$-scalars act trivially on the set of homogeneous components, the homomorphism onto the permutation group has all scalars in its kernel. Therefore we can use the same homomorphism for the projective situation with $\overline{G}$. Thus, this case can be handled in the projective situation. □

Of course in practice we work with a subgroup H of N. If H has a submodule that is *not* an N-submodule then it is possible that we will not be able to find a homomorphism from the irreducible H-subfactor to V. In this case the algorithm will report `fail`: Note that this occurs with probability at most δ. However, it is possible to rerun the algorithm starting at Sections VII.(6.2) with a new version of H that has submodule structure closer to that of N. To see this, note that the subfactor is described by two H-submodules of V, at least one of which is not preserved by N. Therefore a simple argument shows that at least half of the elements of N must fail to fix at least one of the two H-submodules. Thus we add a new generator to H and return to the MeatAxe run of Section VII.(6.2) to determine whether the new H is (absolutely) irreducible.

(6.6) Isomorphic irreducible submodules of dimension at least 2

We continue to assume that N is a normal subgroup of G that is contained in G'. As described in Section VII.(6.2) we assume that the MeatAxe has proved that H acts reducibly by finding an explicit proper nontrivial submodule V' of the natural module.

As described in Theorem VII.6.7 we first find an irreducible H-submodule S and run the MinBlocks procedure. If this fails to find a block system, then (assuming H has the same submodule structure as N) there is only one homogeneous component, corresponding to S.

VII.6.8 Theorem (Reduction for single homogeneous component)

Let $G = \langle g_1, \ldots, g_m \rangle \leq \mathrm{GL}(d, q)$ or its projective version be absolutely irreducible, and let $1 > \delta > 0$ be given. Let $N = \langle n_1, \ldots, n_s \rangle \trianglelefteq G$ be reducible, with a single homogeneous component of dimension $n > 1$, and let an irreducible N-submodule S be given. In deterministic $O((s+m)d^3)$ field operations we can construct a proper nontrivial homomorphism from G into $\mathrm{PGL}(d/n, q^e)$ for e the $\mathbb{F}_q$-dimension of $\mathrm{End}_{\mathbb{F}_q N}(S)$.

Proof: We first find an explicit decomposition of $V|_{\mathbb{F}_q N}$ as a direct sum of copies of S. This can be done using a variant of the isomorphism testing procedure described in Theorem VII.5.8 to compute a basis of the space of all homomorphisms of S into $V|_{\mathbb{F}_q N}$. Namely, we compute the action on V of the algebra word $c \in \mathbb{F}_q N$ that proved that S is simple and determine its kernel K. Since $V|_{\mathbb{F}_q N}$ is isomorphic to a direct sum of copies of S, we can choose an arbitrary nonzero vector from K, compute the standard basis with respect to the generators of N starting at that vector and thereby find a summand S_1 of $V|_{\mathbb{F}_q N}$ together with an explicit isomorphism of S to S_1. By choosing further vectors from K that are not contained in the direct sum of previous copies of S and repeating this procedure, we inductively get an explicit direct sum decomposition of $V|_{\mathbb{F}_q N}$ into summands that are all isomorphic to S. This automatically leads to a base change such that every element of N is represented by a block diagonal matrix in which all diagonal blocks are identical of size $n := \dim_{\mathbb{F}_q}(S)$ in $O(sd^3)$ field operations.

As $N \trianglelefteq G$, for all $h \in N$ and $g \in G$, the product $g^{-1}hg \in N$ and thus $g^{-1}hg$ is also a block diagonal matrix in which all $n \times n$-blocks along the diagonal are identical. Fixing g, we conclude that $g \cdot (g^{-1}hg) = hg$ for all $h \in N$. If we now cut g into $n \times n$-blocks, we get:

$$g \cdot (g^{-1}hg) = \left[\begin{array}{c|c|c|c} g_{1,1} & g_{1,2} & \cdots & g_{1,d/n} \\ \hline g_{2,1} & g_{2,2} & \cdots & g_{2,d/n} \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline g_{d/n,1} & g_{d/n,2} & \cdots & g_{d/n,d/n} \end{array}\right] \cdot \left[\begin{array}{c|c|c|c} D^{g^{-1}}(h) & 0 & \cdots & 0 \\ \hline 0 & D^{g^{-1}}(h) & \cdots & 0 \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline 0 & 0 & \cdots & D^{g^{-1}}(h) \end{array}\right]$$

$$= \left[\begin{array}{c|c|c|c} D(h) & 0 & \cdots & 0 \\ \hline 0 & D(h) & \cdots & 0 \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline 0 & 0 & \cdots & D(h) \end{array}\right] \cdot \left[\begin{array}{c|c|c|c} g_{1,1} & g_{1,2} & \cdots & g_{1,d/n} \\ \hline g_{2,1} & g_{2,2} & \cdots & g_{2,d/n} \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline g_{d/n,1} & g_{d/n,2} & \cdots & g_{d/n,d/n} \end{array}\right] = hg,$$

where the $g_{i,j}$ are $n \times n$-matrices, $D(h)$ is a matrix representing h on the module S and $D^{g^{-1}}(h) = D(g^{-1}hg)$ is the same representation twisted by the element g^{-1}. By the block diagonal structure of the matrices in N we get $g_{i,j} \cdot D^{g^{-1}}(h) = D(h) \cdot g_{i,j}$ for all i and j and all $h \in N$.

But by hypothesis, the matrix representations D and $D^{g^{-1}}$ of N are isomorphic. Thus there is a nonzero matrix $T \in \mathbb{F}_q^{n \times n}$ with $T \cdot D^{g^{-1}}(h) = D(h) \cdot T$ for all $h \in N$. By Schur's lemma and since the representation D is irreducible, the matrix T is invertible and unique up to left multiplication by an element of $C_{\mathrm{GL}(n,q)}(D(N))$, which is isomorphic as a group to the group of units of the extension field $\mathrm{End}_{\mathbb{F}_q N}(S) \cong \mathbb{F}_{q^e}$.

This shows that for every pair $(i, j) \in \{1, \ldots, d/n\} \times \{1, \ldots, d/n\}$ there is a unique element $e_{i,j} \in \mathrm{End}_{\mathbb{F}_q N}(S)$ (possibly 0) with $g_{i,j} = e_{i,j} \cdot T$. Thus we have shown that with respect to

the above choice of basis, every element g is equal to a Kronecker product of some matrix in $U \in \mathbb{F}_{q^e}^{d/n \times d/n}$ with a matrix $T \in \mathbb{F}_q^{n \times n}$. Since g is invertible both U and T are invertible.
This provides an explicit embedding of $\mathbb{F}_q^d$ into a tensor product $\mathbb{F}_{q^e}^{d/n} \otimes_{\mathbb{F}_q} \mathbb{F}_q^n$, where one factor can be over an extension field if the $\mathbb{F}_q N$-module S is not absolutely irreducible. This embedding can be computed explicitly because the above base change is constructive. Using another $O(md^3)$ field operations we compute the generators of G after the base change from which we can read off the tensor decomposition.
Thus we get a non-trivial homomorphism of G into $\mathrm{PGL}(d/n, q^e)$ with N lying in the kernel, which is a significant reduction. The kernel of this homomorphism can immediately be reduced further since its elements are block diagonal matrices with identical $n \times n$-diagonal blocks.
The projective situation can be handled identically, by viewing the kernel as a projective group.
□

If H is a proper subgroup of N, then our algorithm can fail in two ways. Firstly, $V|_{\mathbb{F}_q H}$ might not be isomorphic to a direct sum of copies of the irreducible H-module S. In this case there are not enough homomorphisms from S into the socle of $V|_{\mathbb{F}_q H}$ to span the whole of V. Secondly, even if $V|_{\mathbb{F}_q H}$ is a direct sum of copies of S, the generators of G might not be Kronecker products after a corresponding base change, which we detect during the setup of the homomorphism. In both cases, the error is detected and the algorithm reports `fail`. However, by Corollary VII.4.4 this happens with probability at most δ.

(6.7) Normal subgroup is scalar

The remaining case is that the restriction of the natural module to N has only one homogeneous component and all irreducible N-constituents are one-dimensional, so that N consists of scalars. We may also use the algorithms in this section if we have found a non-scalar generator g_i of G such that $[g_i, g_j]$ is scalar for all j.
We start with a proposition giving a homomorphism into the multiplicative group of the field that does not necessarily need to correspond to an imprimitive decomposition of the natural module.

VII.6.9 Proposition (Scalar homomorphism)
Let $G = \langle g_1, \dots, g_m \rangle \leq \mathrm{GL}(d, q)$ or its projective version be an absolutely irreducible group such that the commutator of a non-scalar generator g_i with all other generators is known to be scalar. Then we can construct a nontrivial homomorphism from G into the multiplicative group of $\mathbb{F}_q$ at no further cost.

Proof: We are given a non-scalar generator g_i, such that all commutators of it with all other generators are scalar matrices. Thus g_i is central in G modulo scalars, thus the commutators of g_i with all elements of G are scalar. Therefore, the map $\psi_{g_i} : G \to \mathbb{F}_q, g \mapsto [g, g_i]$ is a group homomorphism into the scalar matrices. This is proved exactly as Lemma VII.3.2.5.
The kernel of ψ_{g_i} is $C_G(g_i)$. Since g_i is non-central, ψ_{g_i} is nontrivial. Multiplying generators by scalars does not change commutators, so these algorithms will also work in the projective case. □

Since $g_i \in C_G(g_i)$, the kernel is not an absolutely irreducible group and may even not be irreducible. If G' is known to be scalar then the derived group of the kernel $C_G(g_i)$ is also central, and hence a hint can be passed to the kernel to return to the techniques of this section once an absolutely irreducible representation has been found.

Finally we give a deterministic decomposition algorithm for groups with scalar derived group that are not r-groups. We can apply this algorithm if G has at most 7 non-scalar generators, so that all commutators of generators have been calculated — in this case the m^2 vanished from the complexity. This algorithm can easily be modified to decompose any black box group with order oracle that is known to be nilpotent and not a p-group. The assumption that the prime factors of $q^i - 1$ are known for $i \leq d$ is reasonable in practice, and is relied upon for many other algorithms: See [BLS$^+$02] for details of currently maintained lists of such factors.

VII.6.10 Lemma
Let $G = \langle g_1, \ldots, g_m \rangle \leq \mathrm{GL}(d, q)$ be an absolutely irreducible group whose derived group consists only of scalars. Suppose that the order of G is divisible by k primes for some $k > 1$, and that the prime divisors of $q^i - 1$ are known for $1 \leq i \leq d$. Then $k < \log(q-1)$ and in $O(m^2 d^3 \log q \log(d \log q))$ field operations we can compute a homomorphism from G whose kernel and image have order divisible by $\lfloor k/2 \rfloor$ and $\lceil (k+1)/2 \rceil$ primes respectively. Both the kernel and image have at most m generators.

Proof: By Proposition VII.3.3.3 the order of G is a divisor of $o := (q-1)^{m+1}$, which is divisible by less than $\log(q-1)$ distinct primes.

The group G is a direct product of its Sylow subgroups by Lemma VII.3.2.2. We compute the order o_i of g_i for $1 \leq i \leq m$ in $O(md^3 \log q \log(d \log q))$ field operations (see [CLG97]), and find a set of primes $\{p_1, \ldots, p_k\}$ such that each o_i is a product of powers of these primes. For $1 \leq i \leq k$ we find the highest exponent α_i such that $p_i^{\alpha_i}$ divides o.

Let $a := \lfloor k/2 \rfloor$ and define $r = p_1^{\alpha_1} \cdots p_a^{\alpha_a}$ and $r' = p_{a+1}^{\alpha_{a+1}} \cdots p_k^{\alpha_k}$. First run the extended Euclidean algorithm to find s and s' such that $1 = sr + s'r'$, then let $N = s'r'$ and $M = sr$. Clearly for all $g \in G$ the order of g^N divides r whilst $|g^M|$ divides r'. Therefore for all $g \in G$ the only way to write g as a product of an element of order dividing r and an element of order dividing r' is $g = g^N g^M$. Since G is nilpotent, the map $x \mapsto x^N$ is a homomorphism from G to $\mathrm{Syl}(G, p_{a+1}) \times \cdots \times \mathrm{Syl}(G, p_k)$ with kernel $\mathrm{Syl}(G, p_1) \times \cdots \times \mathrm{Syl}(G, p_a)$.

Notice that $p_i^{\alpha_i}$ divides o for all i so $N < o$, and hence we can raise each generator to the power N in $O(m^2 d^3 \log q)$ field operations to get generators for the image. Some of them could be trivial, so we get at most m generators. Multiplying each generator of G by the inverse of its image in $O(md^3)$ field operations will produce at most m generators for the kernel, ignoring again trivial ones. □

VII.7 Complexity summary

In this section we summarise our complexity results, mainly for the sake of a good overview, but also to explicitly give our assumptions.

We begin by describing the complexity of a "MeatAxe run". Although this result is well-known we want to say exactly what assumptions we make and what results underlie our complexity analysis.

VII.7.1 Hypothesis
Assume that we can generate a sufficiently evenly distributed random element both in the group generated by a set X of matrices, and in its normal closure in a group $G \leq \mathrm{GL}(d, q)$, in $O(d^3)$ elementary field operations. Assume further that we can generate a sufficiently evenly distributed random element in a matrix algebra $\mathcal{A} \leq \mathbb{F}_q^{d \times d}$ generated by a set X of matrices in $O(d^3)$ elementary field operations.

Note that this assumption is valid in practice by using Product Replacement and RATTLE (see [CLGM+95, LGM02, LGO97a]) for groups and normal closures, and by taking random linear combinations of random products of generators for algebras, at least after an initialisation phase. However, even using such techniques, these assumptions are not proven to hold in general!

VII.7.2 Lemma (MeatAxe)
Let F be a finite field, $\mathcal{A}$ a finite dimensional F-algebra, V an $\mathcal{A}$-module of F-dimension d, given by the action of m generators of $\mathcal{A}$ as matrices in $F^{d \times d}$, and let $0 < \delta < 1$ be given. There is a Las Vegas algorithm with failure probability less than δ that determines whether V is irreducible in $O(md^3 \log \delta^{-1})$ elementary field operations. In the case of success the result is either a proper nontrivial submodule or the answer "irreducible" together with a field generator of the endomorphism ring $\mathrm{End}_{\mathcal{A}}(V)$. Running the algorithm until success gives a deterministic algorithm in which a step needs $O(md^3)$ field operations and the expected value of the number of such steps is bounded by a constant not depending on $|F|$, d and m.

Proof: All of this is proved in [HR94, IL00], since it is shown that a certain percentage (not depending on $|F|$ or d or m) of all matrix algebra elements are usable to reach a decision and all operations in one step are $O(md^3)$. Note that we assume that we are able to create sufficiently random elements as in Hypothesis VII.7.1. □

We now summarise our complexity results, all assuming Hypothesis VII.7.1 and all given in terms of number of field operations. The whole procedure contains several subalgorithms of Las Vegas type, namely in steps 2, 5, 6, 7 and 8. However, at most 4 of them are possibly executed sequentially (namely in the execution path with steps 1, 2, 3, 4, 5, 8, 9). Thus if we prescribe a failure probability of $\delta/4$ in each Las Vegas step, we get a Las Vegas algorithm with overall failure probability bounded from above by δ. Notice that the factor of 4 does not affect the "big O" complexity. We follow the numbering in our summary of the complete procedure in Section VII.2:

1. Assume $G = \langle g_1, \ldots, g_m \rangle \leq \mathrm{GL}(d, q)$ acting irreducibly, $E = \mathrm{End}_{\mathbb{F}_q G}(\mathbb{F}^d) = \mathbb{F}_{q^e}$ and field generator $C \in \mathrm{GL}(d, q)$ of E^* are known. If $e > 1$, find an explicit base change in $O(md^3)$ field operations.

2. Try to write G over a subfield with $\beta_i = 1$ for $1 \leq i \leq m$ in $O(md^3 \log \delta^{-1})$ field operations (see Theorem VII.5.9).

3. Immediately get a reduction in either the non-absolutely irreducible or the subfield case.

4. Compute 10 commutators of random elements of G. If all are scalar test in $O(md^3)$ whether all commutators of some non-scalar g_i with other generators are scalar. If so, jump to step 10.

Table VII.1: Complexity of algorithm for different cases

Path	Cost
1,2,3	$O(md^3 \log \delta^{-1})$
1,2,3,4,10	$O(md^3 \log \delta^{-1})$
1,2,3,4,5,6	$O(md^3 \log \delta^{-1} + d^4 \log q + sd^3 \log \delta^{-1})$
1,2,3,4,5,7	$O(md^3 \log \delta^{-1} + d^4 \log q + sd^3 \log \delta^{-1})$
1,2,3,4,5,8	$O(md^3 \log \delta^{-1} + d^4 \log q + sd^3 \log(\delta^{-1} \log d))$
1,2,3,4,5,8,9	$O(md^3 \log \delta^{-1} + d^4 \log q + sd^3 \log(\delta^{-1} \log d))$

where $s = O(\log \delta^{-1} + d \log q)$.

5. Compute $s = O(d \log q + \log \delta^{-1})$ generators for $H \leq N \trianglelefteq G$ with N contained in G' in $O(d^3 \log \delta^{-1} + d^4 \log q)$ field operations (see Section VII.(6.1)).

   Run the MeatAxe to distinguish cases for N in $O(sd^3 \log \delta^{-1})$ field operations (see Section VII.(6.2)).

6. If N is absolutely irreducible, check whether G is in $\mathcal{C}_5$ in $O(sd^3 \log \delta^{-1} + md^3)$ field operations (see Section VII.(6.3)).

7. If N is irreducible but not absolutely irreducible, check whether G is in $\mathcal{C}_3$ in $O(d^4 \log q + md^3)$ field operations as in Section VII.(6.4).

8. If N is reducible, look for more than one homogeneous component and if so find an imprimitive decomposition of G in $O(sd^3 \log(\delta^{-1} \log d))$ field operations as in Section VII.(6.5).

9. If N is reducible with a single homogeneous component with irreducible N-submodules dimension greater than 1, find a tensor decomposition of G in $O((s+m)d^3)$ field operations as in Section VII.(6.6).

10. If one non-scalar generator has only scalar commutators with other generators, we have already constructed a nontrivial homomorphism from G to $\mathbb{F}_q^\times$ as in Section VII.(6.7).

The above algorithm can stop after either of steps 3, 6, 7, 8, 9 or 10. We summarise the complexity statements for each of the possible paths through the above steps in Table VII.1.
The worst cases are the last two, where the overall complexity is bounded from above by

$$O(md^3 \log \delta^{-1} + d^4 \log q + sd^3 \log(\delta^{-1} \log d)),$$

where $s = O(\log \delta^{-1} + d \log q)$, that is by

$$O(d^3 \log \delta^{-1}(m + \log(\delta^{-1} \log d)) + d^4 \log(\delta^{-1} \log d) \log q).$$

Fixing $\delta > 0$ this simplifies to $O(d^4 \log(\log d) \log q + md^3)$.

Table VII.2: Timing results for a few example groups

No	Group	d	q	m	Case	d'	q'	Time	Total
1	M_{11}	10	3^5	2	Subfield	10	3	9	70
2	$S.M_{11}$	10	3^5	2	$\mathcal{C}_5$	10	3	25	110
3	J_2	13	3^{10}	2	Subfield	13	3^2	25	2791
4	$S.J_2$	13	3^{10}	2	$\mathcal{C}_5$	13	3^2	184	1687
5	Co_3	22	2^{16}	2	Subfield	22	2	103	3509
6	$S.Co_3$	22	2^{16}	2	$\mathcal{C}_5$	22	2	788	4173
7	$2.\mathrm{A}_8$	24	5^6	2	Subfield	24	5^2	119	1147
8	$S.(2.\mathrm{A}_8)$	24	5^6	2	$\mathcal{C}_5$	24	5^2	954	1711
9	$\mathrm{GL}_{90}(7)$	90	7^5	2	Subfield	90	7	8849	—
10	$S.\,\mathrm{GL}_{90}(7)$	90	7^5	2	$\mathcal{C}_5$	90	7	51133	—
11	J_2	28	2	2	NotAbsIrr	14	2^2	52	1255
12	$J_2.2$	28	2	2	$\mathcal{C}_3$	14	2^2	27	888
13	M_{22}	90	3	2	NotAbsIrr	45	3^2	622	10262
14	$(S.M_{24}).A$	69	5	2	$\mathcal{C}_3$	23	5^3	594	3096
15	3.3^9	81	19	256	Scalar	81	19	5140	50085
16	$\mathrm{S}_{14} \wr C_5$	320	2	3	Components	320	2	2117	106100
17	$3^{1+4} \otimes HS$	189	25	8	Tensor	9	25	24053	150226

VII.8 Implementation and performance

All of our algorithms have been implemented in the forthcoming GAP package recog for constructive group recognition. In general we make $\lfloor (d \log q)/20 \rfloor$ random elements when producing generators for N (see Section VII.(6.1)), but always at least 5 and at most 40, which seems to work well in practice. The division by 20 indicates that our analysis of the generation of a sufficiently large subgroup of N underestimates the probability of success in most cases.

In Table VII.2 we give timing results. All experiments have been done on a machine with an Intel Core2 Quad CPU Q6600 running at 2.40GHz with 8 GB of main memory using the development version of GAP and the recog package. All times are in milliseconds and were averaged over several runs. Note that due to randomisation the runtimes can vary significantly between runs.

Columns "d" and "q" give the matrix dimension and field size of the input matrix group. Column "m" states the number of generators. Column "Group" contains a structural description of the input group. The notation "$S.G$" indicates that the input group is a central extension of a group G by all scalars of $\mathbb{F}_q$. The notation "$G.A$" indicates that the input group is a group G extended by a group A of field automorphisms on a centralising matrix.

Column "Case" describes the type of reduction found. Here, "Subfield" means that an immediate base change to write the group over a smaller field was found. "NotAbsIrr" means that the input group did not act absolutely irreducibly and was written over a larger field with smaller dimension. "$\mathcal{C}_5$" means that a subgroup $H < N$ was computed which acted irreducibly and then a base change was found to write the (projective) group over a smaller field. "$\mathcal{C}_3$" means that a subgroup

$H < N$ was computed which acted irreducibly (but not absolutely irreducibly) and an action as field automorphisms was found. This automatically gives the information required to write the kernel in a smaller dimension over the appropriate extension field. “Components” means that an imprimitive action was found. “Tensor” means that a tensor decomposition was found. “Scalar” means that the group was reduced using commutators with a single non-central element.
Columns “d'” and “q'” contain the dimension and field size after the reduction. The “Time” column indicates the time needed for the first reduction. The “Total” column contains the runtime to build a complete composition tree for the given group. This value is occasionally omitted, which indicates that not enough leaf methods are implemented yet to recognise this group fully.
To produce the examples, we first changed the field by embedding or blowing up and then conjugated by a random element in the general linear group over the new field.
To make example 14 we took the Mathieu group M_{24} represented in $\mathrm{GL}_{23}(5)$, multiplied the generators by scalars in $\mathbb{F}_{5^3}$, blew everything up into $\mathrm{GL}_{69}(5)$ and added a new generator that acts as a field automorphism of $\mathbb{F}_{5^3}$ when conjugating the centralising matrix. In example 14 the algorithm then writes the kernel as a subgroup of $\mathrm{GL}_{23}(5^3)$ and finally recognises the $\mathcal{C}_5$ case and goes back to $\mathrm{GL}_{23}(5)$.
Example 15 is an absolutely irreducible 3-group in $\mathrm{GL}_{81}(19)$ such that all commutators are scalar matrices. After several reductions by the commutator action the kernel becomes reducible.
Example 16 is a wreath product of the symmetric group S_{14} with the cyclic group of order 5. We started with an absolutely irreducible 64-dimensional representation of S_{14} over $\mathbb{F}_2$ and made a 320-dimensional absolutely irreducible representation for $\mathrm{S}_{14} \wr C_5$ over $\mathbb{F}_2$. Our algorithm computes the action on the five homogeneous components, then tells the kernel node that the group is reducible and in block form so that a MeatAxe call is not necessary.
Example 17 is a central product of an extraspecial 3-group of order 243 in its irreducible representation of dimension 9 over $\mathbb{F}_{25}$ with the sporadic finite simple group HS in an irreducible representation of dimension 21 over $\mathbb{F}_{25}$. The latter representation came from one over $\mathbb{F}_5$, where the representing matrices of the group generators were multiplied by elements from $\mathbb{F}_{25}$. The extraspecial factor vanishes when computing a subgroup of the derived group, since it is one example of our characterisation in Section VII.3, exhibiting the tensor decomposition. In subsequent nodes the extraspecial factor is taken apart using commutators as in Section VII.(6.7).

Chapter VIII

Leaves of the composition tree

This chapter is about constructive recognition (see Problem V.2.1) of the leaves of a composition tree (see Section V.2). By the arguments in Chapter VI this means that we have to be able to solve Problem V.2.1 for the subgroups $G \leq \mathrm{GL}(n, q)$ that lie in classes $\mathcal{D}_8$ and $\mathcal{D}_9$ (see Sections VI.(1.8) and VI.(1.9)). Because of the homomorphism from $\mathrm{GL}(n, q) \to \mathrm{PGL}(n, q)$ and our setup of the composition tree we only have to work with the projective version $\bar{G} \leq \mathrm{PGL}(n, q)$.
We do not want to describe the details of the methods used here but instead give an overview and refer the reader to the literature. One reason for this is that the current state of the art does not seem to be final, another is that the author has not contributed to the development of algorithms in this area up to now.
This chapter is structured as follows. We begin by describing methods based on permutation groups, which we call “direct methods for constructive recognition” in Section VIII.1. The next Section VIII.2 explains the concept of “non-constructive recognition” which means to determine the isomorphism type of a given group. Once this is known, the concept of standard generators applies, which is introduced in Section VIII.3. These concepts in turn are usually needed to apply more specialised methods for constructive recognition, of which an overview is given in Section VIII.4 for classical groups in their natural representation and in Section VIII.5 for almost simple groups.

VIII.1 Direct methods for constructive recognition

For “small groups” one can use permutation group methods to solve Problem V.2.1 for a group $\bar{G} \leq \mathrm{PGL}(n, q)$. The basic idea is to find a permutation action. This immediately gives a homomorphism into a permutation group which will be an isomorphism if the group is simple. Even if there is a non-trivial kernel the composition tree setup will take care of this.
Matrix groups and projective groups of course act on their natural module and thus on vectors and subspaces. So, finding some action is not difficult. Achieving good performance however can be tricky. To this end we want to find short orbits. Heuristic methods for this for matrix groups can be found in [MO95].
Another possibility is low index methods. Here one would guess the point stabiliser of a point with a short orbit, restrict the natural module to it and use the MeatAxe to find a proper invariant

subspace. The orbit of this subspace would then be relatively short. However, although this approach looks promising and occasionally works, it is not clear how to find generators of such a point stabiliser, even with random methods.
Once we have an action and thus a homomorphism into a permutation group we can either use the methods described in Section V.5 or immediately compute a stabiliser chain for the projective group using the Schreier-Sims method (see [BCFS91] or [Ser03]). Both approaches solve the constructive recognition problem V.2.1 even if the group is not a simple group.
Note that for larger dimensions and in particular for classical groups we do not even want to try these direct methods because any orbit we can possible find would be prohibitively large.

VIII.2 Non-constructive recognition

The term "non-constructive recognition" for a group means finding the isomorphism type. In situations where direct methods for constructive recognition (see the previous Section VIII.1) fail, the usual approach is to determine the isomorphism type of the group first and then use additional knowledge about the group in question to do the constructive part of the recognition. In particular to use the technique of standard generators (see the next Section VIII.3), the isomorphism type has to be known in advance.
For the non-constructive recognition problem we can for example use element order statistics. We produce uniformly distributed elements in the group and compute their orders. If we see an element order that does not occur in a group of a certain isomorphism type, we can immediately rule out this type. If we fail to see an element order that occurs very frequently in a certain isomorphism type after some tries, we can rule out this type with a known error probability.
See Sections VIII.4 and VIII.5 for an overview of the methods for non-constructive recognition for classical groups in their natural representation (class $\mathcal{D}_8$) and for groups in class $\mathcal{D}_9$, respectively.

VIII.3 Standard generators

In this section we give a definition of the term "standard generators". This idea is due to Robert Wilson and was first published in [Wil96]. As in this whole chapter we do not want to go into too much detail but instead give the reader an idea of the concept.
Once the isomorphism type of a group $G = \langle g_1, \ldots, g_k \rangle$ is known (after successful non-constructive recognition as in Section VIII.2), one wants to find an explicit isomorphism of G to a "standard copy $\hat{G}$". Note that such an isomorphism is not automatically "computable" in the sense that we can map group elements back and forth, as we will see below. However, in the end we strive to use previously acquired and stored knowledge about $\hat{G}$ and transfer it over to G via that isomorphism to eventually solve the constructive recognition problem (see V.2.1) for G.
The concept of standard generators serves this purpose.

VIII.3.1 Definition (Standard generators)
Let G be a finite group and $\mathrm{Aut}(G)$ its automorphism group. Then $\mathrm{Aut}(G)$ acts on tuples of elements of G componentwise. We choose one orbit of this action that contains tuples whose entries generate G as a group, and call exactly those tuples in this orbit *standard generators* for

G. This choice is done once and forever for every isomorphism type of finite group and the chosen orbit is described by giving a set of properties of the tuples that uniquely determines the orbit.

VIII.3.2 Remark
Note that this choice has to be done individually for every isomorphism type of finite groups and the properties have to be determined intelligently, such that finding a tuple of standard generators is possible efficiently (see Section VIII.3.6).

This rather vague definition can best be filled with life by an example:

VIII.3.3 Example (Standard generators for the sporadic simple Mathieu group M_{11})
This description is taken from [WWT+99, M_{11} page] and is derived in [Wil96, Example 11]. Standard generators of M_{11} are (a, b), where a has order 2, b has order 4, ab has order 11 and $ababababbababbabb$ has order 4. Note that it is a theorem that these properties uniquely determine an orbit of $\mathrm{Aut}(M_{11})$ on pairs of elements of M_{11}.

To find standard generators for M_{11}:

1. Find an element of order 4 or 8. This powers up to x of order 2 and y of order 4.
   [The probability of success at each attempt is 3 in 8.]
2. Find a conjugate a of x and a conjugate b of y such that ab has order 11.
   [The probability of success at each attempt is 16 in 165.]
3. If $ababbabbb$ has order 3, then replace b by its inverse.
4. Now $ababbabbb$ has order 5, and standard generators of M_{11} have been obtained.

It is a theorem that this procedure produces standard generators and the probabilities can be read off the character table of M_{11}. Of course we assume uniformly distributed random elements for these probabilities to be correct. Note that choosing random conjugates of x that are uniformly distributed in the conjugacy class of x can be achieved by conjugating x with a random element that is uniformly distributed in the group.

VIII.3.4 Proposition (The virtue of standard generators)
If $(s_1, \dots, s_m) \in G^m$ and $(t_1, \dots, t_m) \in G^m$ are both standard generators for a group G, then the equations

$$\varphi(s_i) = t_i \qquad \text{for all } 1 \le i \le m$$

uniquely define an automorphism φ of G.
That is, if we have a tuple of standard generators $(s_1, \dots, s_m)$ for G and $\hat{G}$ is a standard isomorphic copy of G for which we know a tuple of standard generators $(u_1, \dots, u_m)$, then the equations

$$\psi(s_i) = u_i \qquad \text{for all } 1 \le i \le m$$

uniquely define an explicit isomorphism ψ from G to $\hat{G}$.

Proof: Since by definition both tuples lie in the same orbit under $\mathrm{Aut}(G)$ and both tuples generate G, there is exactly one automorphism $\varphi \in \mathrm{Aut}(G)$ mapping s_i to t_i for all $1 \le i \le m$. The hypothesis that $\hat{G}$ is isomorphic to G takes care of the second statement. □

VIII.3.5 Remark (The problem of mapping elements)
Note that even if we have found standard generators in G, the above definition of the explicit isomorphism ψ to $\hat{G}$ does in fact *not* enable us to map arbitrary elements of G via ψ, because for this we would have to express an arbitrary element of G as a straight line program in $(s_1, \ldots, s_m)$, which is exactly the constructive recognition problem we want to solve!
However, if we want to store certain elements or subgroups of $\hat{G}$ beforehand, we can store them as straight line programs in $(u_1, \ldots, u_m)$ and can then evaluate these straight line programs in $(s_1, \ldots, s_m)$ to actually get their images under ψ^{-1} in G. This fact helps to transfer previously acquired knowledge from $\hat{G}$ to G.

VIII.3.6 Remark (Good standard generators)
We want to comment only briefly on this topic. Basically, the choice of the standard generators for an isomorphism type of group, that is the choice of the $\mathrm{Aut}(G)$-orbit in the tuples of elements of G, is "good", if it is relatively easy to find a tuple of standard generators by random methods. The example in Section VIII.3.3 exhibits this. The probabilities to find the right elements in the algorithm presented there are quite good, such that very few random elements will usually lead to success.
A large collection of such good choices of standard generators together with algorithms to find them can be found on the WWW-Atlas of group representations, see [WWT+99].

VIII.3.7 Application (Storing hints for stabiliser chains)
One immediate application of standard generators is the following. If a group has a subgroup U with a relatively low index, then we can store generators for this subgroup as a straight line program in standard generators. Once we have recognised the isomorphism type of G using non-constructive recognition and have found standard generators in G, we can evaluate this straight line program, get a generating set of a subgroup U of G with this low index, restrict the natural module to U and find a proper invariant subspace. Provided that G acts irreducibly on its natural module, the G-orbit of this subspace in the natural action on subspaces then gives a permutation action of G which is isomorphic to the one on cosets of U. Note that the existence of such a proper invariant subspace of course depends not only on the isomorphism type of G but rather also on the actual representation G comes in.
Thus, we can collect hints to find "good" actions for the different absolutely irreducible matrix representations of a group. Then, after we have non-constructively recognised G and of course know the concrete representation it comes in, we can look up which "good" action to use and compute this action using standard generators as described in the previous paragraph.
Eamonn O'Brien and Robert Wilson have for example done exactly this for the sporadic simple groups. Their hints data is available in the MAGMA system (see [BC07]) and will be used in the composition tree implementation in the GAP system as well.

VIII.4 The classical case in natural representation: $\mathcal{D}_8$

The seminal paper by Neumann and Praeger [NP92] which presents an algorithm to decide whether a given group $G \leq \mathrm{GL}(n, q)$ contains the special linear group was the starting point of a whole industry of papers concerned with non-constructive and constructive recognition of groups.

An algorithm to recognise classical groups in their natural representation non-constructively is given in [NP98]. Once this is done, other algorithms apply: In [CLG98] an algorithm to recognise $\mathrm{SL}(n, q)$ in its natural representation constructively is presented with effective cost $O(n^4 q)$. For arbitrary classical groups in their natural representation the results in [Bro03a] give algorithms to solve the constructive recognition problem with effective cost $O(n^5 \log^2 q)$, however these algorithms need an $\mathrm{SL}(2, q)$ oracle, that is, they rely on a solution of the constructive recognition problem for $\mathrm{SL}(2, q)$. Recently a new preprint (see [LGO07]) has appeared which handles the case of odd characteristic and promises better complexity results.

The basic idea of all these constructive recognition algorithms is to find a tuple of standard generators and then perform a base change such that linear algebra methods can be used to express arbitrary elements as straight line programs in the standard generators.

VIII.5 The almost simple case: $\mathcal{D}_9$

This part of the whole group recognition project is probably the one for which the currently known methods are least satisfying. In particular because many isomorphism types of groups have to be dealt with in a case by case fashion a lot of work still needs to be done. In this section we try to give an overview of the known methods by means of references to the literature. We intentionally leave out complexity results for the sake of brevity and because the last word on these does not yet seem to have been spoken. A more detailed account of the state of the art can be found in [O'B06].

We begin with a discussion of the problem with the "almost" in "almost simple".

If a group $G \leq \mathrm{GL}(n, q)$ with $Z := G \cap Z(\mathrm{GL}(n, q))$ is contained in class $\mathcal{D}_9$, there is a non-abelian simple group $\bar{N}$ and a group T with $\bar{N} \leq T \leq \mathrm{Aut}(\bar{N})$ and $G/Z \cong T$. The first problem for both constructive and non-constructive recognition is that G/Z is itself not necessarily simple, after all, G/Z is only "almost simple". However, the Schreier conjecture, which follows from the classification of finite simple groups, says that $\mathrm{Aut}(\bar{N})/\bar{N}$ is solvable for all finite simple groups $\bar{N}$.

In fact, this outer automorphism group is rather small for groups occurring in practice. If $\bar{N}$ is alternating or sporadic, then $|\mathrm{Out}(\bar{N})| = 2$ except for $|\mathrm{Out}(A_6)| = 4$. In [LMM01, Lemma 1.4] it is shown that for a simple group $\bar{N}$ of Lie type in cross-characteristic, $|\mathrm{Out}(\bar{N})| \leq \beta \log n$ for some global constant β. If $\bar{N}$ is a simple group of Lie type in its defining characteristic, then $|\mathrm{Out}(\bar{N})| \leq 2(n+1) \log q$ by [LMM01, Proof of Lemma 1.3].

So the first step for recognition is to find the simple subgroup isomorphic to $\bar{N}$ of G/Z. We denote this by N/Z in the following. One approach is to go down the derived series until a group is reached which is "probably perfect". To test for the latter, one uses the algorithm by Leedham-Green and O'Brien in [LGO02, 5.3]. This algorithm estimates the order of an element in a factor group if the group and the normal subgroup are given. This technique produces generators for N efficiently.

Most publications in this area seem to concentrate on the non-constructive and constructive recognition of simple groups. However, to completely solve Problem V.2.1 for all groups in $\mathcal{D}_9$ one has to be able to do constructive recognition for all almost simple groups. There seems to be no method known to get hold of the factor group G/N in general since in most cases the restriction of the natural module to N is absolutely irreducible. We will ignore this problem here, in particular since G/N is very small in most cases, such that solving the constructive recognition problem for N together with a few coset case distinctions can solve the problem in G satisfactorily.

Next we try to give an overview of the known techniques for non-constructive recognition of simple groups. Alternating groups and sporadic groups can be recognised non-constructively by looking at element orders of random elements. For Lie type groups there is an algorithm for non-constructive recognition in [BKPS02]. One outstanding case in this paper is covered by methods in [AB01]. However, both algorithms need to know the defining characteristic of the Lie type group in advance. To determine this, there are three concurrent methods, one is described in [KS02], a newer one in [LO07] and thirdly there is unpublished work by Seress which uses statistics about the two largest projective element orders occurring.

In 2001 Malle and O'Brien developed a practical implementation of all the algorithms for non-constructive recognition known at the time, which is distributed with the MAGMA system.

Assuming from now on that the non-constructive recognition is achieved we conclude this section with a list of references to work that has been done to solve the constructive recognition problem for classes of simple groups. Note that many of these solutions include the corresponding almost simple groups thereby solving specific cases of $\mathcal{D}_9$ groups completely.

An algorithm to recognise the alternating group A_k and the symmetric group S_k constructively in an arbitrary representation is described in [BLGN$^+$03]. An alternative algorithm was developed in [BP00].

For classical groups in another than their natural representation (see Section VIII.4 for the $\mathcal{D}_8$ case) there is an algorithm in [KS01]. The particularly important case of the special linear group $\mathrm{SL}(2,q)$ of Lie rank 1 in defining characteristic but non-natural representation is dealt with in [CLG01] and [CLGO06] in the sense that an algorithm is described to find an explicit computable epimorphism to $\mathrm{PSL}(2,q)$ in its natural representation. Further work to improve the algorithms for simple classical groups in arbitrary representations can be found in [Bro01], [Bro03b], [BK01], [KS03] and [BK06]. Recently the preprint [MOS08] appeared which deals with recognising small degree representations of general linear groups specifically.

For Lie type groups work on constructive recognition has appeared in [LMO07], [Ryb07] and [Bää06]. An article about constructive recognition of exceptional groups of Lie type by Kantor and Magaard is in preparation. However, this area is still work in progress.

For the sporadic simple groups the data collected by O'Brien and Wilson about subgroup chains (see Section VIII.3.7) can be used to solve the constructive recognition problem after non-constructive recognition using the direct methods mentioned in Section VIII.1.

We finish this section by mentioning two other approaches to the constructive part of the recognition problem.

One is published in the preprint [HLO$^+$08]. It uses involution centralisers and the fact that generators for them can be computed efficiently (see [Bra00]). One instance of the constructive membership test problem for a group G is translated into three instances of the corresponding problem in the centralisers of certain involutions which come up during the run of the algorithm.

Finally we want to mention the paper [ANPS05], which introduces a generic framework for constructive membership testing in a group whose isomorphism type is known. It relies on standard generators (see Section VIII.3) and prepared subgroup chains stored as straight line programs in the standard generators.

Appendix A

Notation and Symbols

Symbols:	
$\emptyset$	the empty set
$\subseteq$	is contained in
$\supseteq$	contains
$\subsetneq$	is contained properly
$\supsetneq$	contains properly
$\circ$	concatenation of mappings (left after right) or central product
$\cong$	isomorphism
$\lceil x \rceil$	smallest integer $\geq x$
$\lfloor x \rfloor$	greatest integer $\leq x$
$(-\|-)$	bilinear form
$\sum$	sum
$\prod$	product
$\bigoplus$	direct sum of modules
$\langle - \rangle_A$	A-span by $-$
$\langle - \rangle$	group generated by $-$
$\langle - \rangle^G$	normal closure in G of $-$
$\oplus$	direct sum
$\otimes$	tensor product
$\times$	direct product or cartesian product
$\mathbb{F}^\times$	multiplicative group of the field $\mathbb{F}$
$\trianglelefteq$	is normal subgroup in
$\varphi^{-1}(N)$	full preimage of the set N under φ
$G \wr H$	wreath product of G with $H \leq S_n$
$[x, y]$	commutator $x^{-1}y^{-1}xy$
A_n	alternating group on n points
$\mathrm{ann}_R(v)$	annihilator in R of v
$\mathrm{Aut}(G)$	automorphism group of G
$\mathcal{C}_1$ to $\mathcal{C}_9$	Aschbacher classes, see Sections VI.(1.1) to VI.(1.9)

$C_G(H)$	centraliser in G of H
$\mathcal{D}_1$ to $\mathcal{D}_9$	modified Aschbacher classes, see Sections VI.(1.1) to VI.(1.9)
$\deg f$	degree of the polynomial f
$\det M$	determinante of the matrix M
$\dim_{\mathbb{F}}(V)$	$\mathbb{F}$-dimension of V
$E(X)$	expected value of the random variable X
$\mathrm{End}_G(M)$	set of endomorphisms of the G-module M
$\exp(x)$	exponential function of x (i.e. e^x)
$F^{n\times m}$	$n \times m$-matrices with entries in F
$\mathbb{F}_q$	field with q elements
$F[G]$	set of linear combinations of the elements of the matrix group G
FG	group algebra of G over F
$\|G\|$	order of the group G
G'	derived subgroup of the group G
$\gcd(a, b)$	greatest common divisor of a and b
$\mathrm{GL}(n, q)$	group of invertible matrices in $\mathbb{F}_q^{n\times n}$
$\Gamma\mathrm{L}(n, q)$	group of semilinear transformations of $\mathbb{F}_q^n \to \mathbb{F}_q^n$
$\mathrm{Hom}_G(M, N)$	set of homomorphism between the G-modules M and N
id	identity mapping
$\mathrm{Im}(f)$	image of the mapping f
$\mathrm{Inn}(G)$	inner automorphism group of the group G
$\ker(f)$	kernel of the mapping f
$\mathrm{lcm}(a, b)$	least common multiple of a and b
$\log_b(x)$	logarithm of x with respect to the basis b ($= e$ if omitted)
$\|M\|$	cardinality of the set M
$\min(M)$	minimum of the set M
$\mu_{M,V}$	minimal polynomial of M in its action on V
$N_G(H)$	normaliser in G of H
$\mathbb{N}$	set of natural numbers, $0 \notin \mathbb{N}$
$\mathrm{ord}_M(v)$	order polynomial of M acting on v
$O(g)$	capital-O notation, see Section I.1.1
$\mathrm{O}^\epsilon(n, q)$	orthogonal group in $\mathrm{GL}(n, q)$, see [CCN+85, Section 2.4]
$\Omega^\epsilon(n, q)$	$\mathrm{O}^\epsilon(n, q)'$ in most cases, see [CCN+85, Section 2.4] for details
$\mathrm{Out}(G)$	outer automorphism group of the group G
$\mathrm{PGL}(n, q)$	full projective group ($\mathrm{GL}(n, q)$ modulo scalars)
$\Phi(G)$	Frattini subgroup of G
$\mathrm{P}\Omega^\epsilon(n, q)$	simple projective orthogonal group (see [CCN+85, Section 2.4])
$\mathrm{Prob}(E)$	probability of the event E
$\mathrm{Prob}(E\|F)$	probability of the event E under the condition F
$\mathrm{PSL}(n, q)$	special projective group ($\mathrm{SL}(n, q)$ modulo scalars)
$\mathrm{PSp}(n, q)$	projective symplectic group ($\mathrm{Sp}(n, q)$ modulo scalars)
$\mathrm{PSU}(n, q)$	projective special unitary group ($\mathrm{SU}(n, q)$ modulo scalars)
$\mathbb{R}$	set of real numbers

$\mathrm{RowSp}(Y)$	row space of the matrix Y
$\rho\uparrow^{\otimes G}$	tensor induced (projective) representation see Section VI.2.2
S_n	symmetric group on n points
$\mathrm{SL}(n, q)$	special linear group, matrices of determinant 1 in $\mathrm{GL}(n, q)$
$\mathrm{Sp}(n, q)$	symplectic group in $\mathrm{GL}(n, q)$
$\mathrm{SU}(n, q)$	special unitary group in $\mathrm{GL}(n, q^2)$
$\mathrm{Syl}(G, p)$	Sylow p-subgroup of G
$\mathrm{U}(n, q)$	unitary group in $\mathrm{GL}(n, q^2)$
$V\|_N$	module V restricted to N
$\chi_{M,V}$	characteristic polynomial of M in its action on V
$\mathbb{Z}$	set of rational integers
$Z(G)$	centre of the group G

Appendix B

List of Figures

Appendix C

List of Tables

Appendix D

Bibliography

[AB01] Christine Altseimer and Alexandre V. Borovik. Probabilistic recognition of orthogonal and symplectic groups. In *Groups and computation, III (Columbus, OH, 1999)*, volume 8 of *Ohio State Univ. Math. Res. Inst. Publ.*, pages 1–20. de Gruyter, Berlin, 2001. 137

[AC97] Daniel Augot and Paul Camion. On the computation of minimal polynomials, cyclic vectors, and Frobenius forms. *Linear Algebra Appl.*, 260:61–94, 1997. 38, 59

[AHU75] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The design and analysis of computer algorithms*. Addison-Wesley Publishing Co., Reading, Mass.-London-Amsterdam, 1975. Second printing, Addison-Wesley Series in Computer Science and Information Processing. 27

[ANPS05] Sophie Ambrose, Max Neunhöffer, Cheryl E. Praeger, and Csaba Schneider. Generalised sifting in black-box groups. *LMS J. Comput. Math.*, 8:217–250 (electronic), 2005. 137

[Asc84] Michael Aschbacher. On the maximal subgroups of the finite classical groups. *Invent. Math.*, 76(3):469–514, 1984. 5, 83, 84, 85, 104

[Bää06] Henrik Bäärnhielm. Recognising the Suzuki groups in their natural representations. *J. Algebra*, 300(1):171–198, 2006. 137

[BC07] Wieb Bosma and John J. Cannon. *Magma, Handbook of Magma Functions, Edition 2.14*, 2007. `http://magma.maths.usyd.edu.au/magma/htmlhelp/MAGMA.htm` . 2, 13, 39, 61, 135

[BCF$^+$95] László Babai, Gene Cooperman, Larry Finkelstein, Eugene Luks, and Ákos Seress. Fast Monte Carlo algorithms for permutation groups. *J. Comput. System Sci.*, 50(2):296–308, 1995. 23rd Symposium on the Theory of Computing (New Orleans, LA, 1991). 111

[BCFS91] László Babai, Gene Cooperman, Larry Finkelstein, and Ákos Seress. Nearly linear time algorithms for permutation groups with a small base. In *ISSAC '91: Proceedings of the 1991 international symposium on Symbolic and algebraic computation*, pages 200–209, New York, NY, USA, 1991. ACM Press. 80, 133

[BK01] Peter A. Brooksbank and William M. Kantor. On constructive recognition of a black box $\mathrm{PSL}(d,q)$. In *Groups and computation, III (Columbus, OH, 1999)*, volume 8 of *Ohio State Univ. Math. Res. Inst. Publ.*, pages 95–111. de Gruyter, Berlin, 2001. 137

[BK06] Peter A. Brooksbank and William M. Kantor. Fast constructive recognition of black box orthogonal groups. *J. Algebra*, 300(1):256–288, 2006. 137

[BKPS02] László Babai, William M. Kantor, Péter P. Pálfy, and Ákos Seress. Black-box recognition of finite simple groups of Lie type by statistics of element orders. *J. Group Theory*, 5(4):383–401, 2002. 137

[BLGN$^+$03] Robert Beals, Charles R. Leedham-Green, Alice C. Niemeyer, Cheryl E. Praeger, and Ákos Seress. A black-box group algorithm for recognizing finite symmetric and alternating groups. I. *Trans. Amer. Math. Soc.*, 355(5):2097–2113 (electronic), 2003. 137

[BLN$^+$08] Peter Brooksbank, Maska Law, Max Neunhöffer, Alice Niemeyer, and Ákos Seress. recog – *a collection of group recognition methods*, 2008. GAP package. (http://www-groups.mcs.st-and.ac.uk/~neunhoef/Computer/Software/Gap/recog.html). 4

[BLS97] László Babai, Eugene M. Luks, and Ákos Seress. Fast management of permutation groups. I. *SIAM J. Comput.*, 26(5):1310–1342, 1997. 81

[BLS$^+$02] John Brillhart, Derrick H. Lehmer, John L. Selfridge, Bryant Tuckerman, and Sam S. Wagstaff, Jr. *Factorizations of* $b^n \pm 1$, volume 22 of *Contemporary Mathematics*. Amer. Math. Soc., Providence, RI, third edition, 2002. http://www.cerias.purdue.edu/homes/ssw/cun/index.html. 127

[BNS06] Peter Brooksbank, Alice C. Niemeyer, and Ákos Seress. A reduction algorithm for matrix groups with an extraspecial normal subgroup. In *Finite geometries, groups, and computation*, pages 1–16. Walter de Gruyter GmbH & Co. KG, Berlin, 2006. 103

[BP00] Sergey Bratus and Igor Pak. Fast constructive recognition of a black box group isomorphic to S_n or A_n using Goldbach's conjecture. *J. Symbolic Comput.*, 29(1):33–57, 2000. 137

[Bra00] John N. Bray. An improved method for generating the centralizer of an involution. *Arch. Math. (Basel)*, 74(4):241–245, 2000. 96, 137

[Bre] Richard Brent. Factor tables. http://wwwmaths.anu.edu.au/~brent/factors.html. 69

[Bro01] Peter A. Brooksbank. A constructive recognition algorithm for the matrix group $\Omega(d, q)$. In *Groups and computation, III (Columbus, OH, 1999)*, volume 8 of *Ohio State Univ. Math. Res. Inst. Publ.*, pages 79–93. de Gruyter, Berlin, 2001. 137

[Bro03a] Peter A. Brooksbank. Constructive recognition of classical groups in their natural representation. *J. Symbolic Comput.*, 35(2):195–239, 2003. 136

[Bro03b] Peter A. Brooksbank. Fast constructive recognition of black-box unitary groups. *LMS J. Comput. Math.*, 6:162–197 (electronic), 2003. 137

[CCN$^+$85] John H. Conway, Robert T. Curtis, Simon P. Norton, Richard A. Parker, and Robert A. Wilson. *Atlas of finite groups*. Oxford University Press, Eynsham, 1985. Maximal subgroups and ordinary characters for simple groups, With computational assistance from J. G. Thackray. 88, 139

[CLG97] Frank Celler and Charles R. Leedham-Green. Calculating the order of an invertible matrix. In *Groups and computation, II (New Brunswick, NJ, 1995)*, volume 28 of *DIMACS Ser. Discrete Math. Theoret. Comput. Sci.*, pages 55–60. Amer. Math. Soc., Providence, RI, 1997. 64, 65, 66, 127

[CLG98] Frank Celler and Charles R. Leedham-Green. A constructive recognition algorithm for the special linear group. In *The atlas of finite groups: ten years on (Birmingham, 1995)*, volume 249 of *London Math. Soc. Lecture Note Ser.*, pages 11–26. Cambridge Univ. Press, Cambridge, 1998. 136

[CLG01] Marston Conder and Charles R. Leedham-Green. Fast recognition of classical groups over large fields. In *Groups and computation, III (Columbus, OH, 1999)*, volume 8 of *Ohio State Univ. Math. Res. Inst. Publ.*, pages 113–121. de Gruyter, Berlin, 2001. 137

[CLGM$^+$95] Frank Celler, Charles R. Leedham-Green, Scott H. Murray, Alice C. Niemeyer, and Eamonn A. O'Brien. Generating random elements of a finite group. *Comm. Algebra*, 23(13):4931–4948, 1995. 128

[CLGO06] Marston D. E. Conder, Charles R. Leedham-Green, and Eamonn A. O'Brien. Constructive recognition of $\mathrm{PSL}(2, q)$. *Trans. Amer. Math. Soc.*, 358(3):1203–1221 (electronic), 2006. 137

[CNRD08] Jon F. Carlson, Max Neunhöffer, and Colva M. Roney-Dougal. A polynomial-time reduction algorithm for groups of semilinear or subfield class. *Journal of Algebra*, 2008. submitted. 5, 89, 92, 104

[CR62] Charles W. Curtis and Irving Reiner. *Representation Theory of Finite Groups and Associative Algebras*. John Wiley & Sons, New York, Chichester, Brisbane, Toronto, Singapore, 1962. 86, 89, 90, 109, 110

[CR90] Charles W. Curtis and Irving Reiner. *Methods of representation theory. Vol. I*. Wiley Classics Library. John Wiley & Sons Inc., New York, 1990. With applications to finite groups and orders, Reprint of the 1981 original, A Wiley-Interscience Publication. 94, 113, 115

[DM96] John D. Dixon and Brian Mortimer. *Permutation groups*, volume 163 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1996. 90

[Ebe00] Wayne Eberly. Asymptotically Efficient Algorithms for the Frobenius form, 2000. Department of Computer Science, University of Calgary Technical Report, (http://pages.cpsc.ucalgary.ca/~eberly/Research/publications.php). 39

[FMR$^+$95] Peter Fleischmann, Gerhard O. Michler, Peter Roelse, Jens Rosenboom, Reiner Staszewski, Clemens Wagner, and Michael Weller. *Linear algebra over small finite fields on parallel machines*, volume 23 of *Vorlesungen aus dem Fachbereich Mathematik der Universität GH Essen [Lecture Notes in Mathematics at the University of Essen]*. Universität Essen Fachbereich Mathematik, Essen, 1995. 18

[GAP07] The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.4.10*, 2007. (http://www.gap-system.org/). 2, 4, 13, 17, 23, 38, 39, 61, 77, 105

[GH97] Stephen P. Glasby and Robert B. Howlett. Writing representations over minimal fields. *Comm. Algebra*, 25(6):1703–1711, 1997. 96, 105

[Gie95] Mark Giesbrecht. Nearly optimal algorithms for canonical matrix forms. *SIAM J. Comput.*, 24(5):948–969, 1995. 38, 39

[GLGO06] Stephen P. Glasby, Charles R. Leedham-Green, and Eamonn A. O'Brien. Writing projective representations over subfields. *J. Algebra*, 295(1):51–61, 2006. 96, 105, 121

[HLGOR96a] Derek F. Holt, Charles R. Leedham-Green, Eamonn A. O'Brien, and Sarah Rees. Computing matrix group decompositions with respect to a normal subgroup. *J. Algebra*, 184(3):818–838, 1996. 89, 96, 101, 105

[HLGOR96b] Derek F. Holt, Charles R. Leedham-Green, Eamonn A. O'Brien, and Sarah Rees. Testing matrix groups for primitivity. *J. Algebra*, 184(3):795–817, 1996. 85, 124

[HLO$^+$08] Petra E. Holmes, Steve A. Linton, Eamonn A. O'Brien, Alexander J.E. Ryba, and Robert A. Wilson. Constructive membership in black-box groups. *Journal of Group Theory*, 2008. Preprint. http://www.math.auckland.ac.nz/~obrien/papers.php. 137

[HR94] Derek F. Holt and Sarah Rees. Testing modules for irreducibility. *J. Austral. Math. Soc. Ser. A*, 57(1):1–16, 1994. 62, 95, 96, 105, 118, 128

[HS08] Derek F. Holt and Mark J. Stather. Computing a chief series and the soluble radical of a matrix group over a finite field. *LMS JCM*, 11:223–251, 2008. 4

[Hup67] Bertram Huppert. *Endliche Gruppen. I.* Die Grundlehren der Mathematischen Wissenschaften, Band 134. Springer-Verlag, Berlin, 1967. 91

[IL00] Gábor Ivanyos and Klaus Lux. Treating the exceptional cases of the MeatAxe. *Experiment. Math.*, 9(3):373–381, 2000. 95, 105, 118, 128

[Jac85] Nathan Jacobson. *Basic algebra. I.* W. H. Freeman and Company, New York, second edition, 1985. 50, 59

[KG85] Walter Keller-Gehrig. Fast algorithms for the characteristic polynomial. *Theoret. Comput. Sci.*, 36(2-3):309–317, 1985. 38

[KL90] Peter Kleidman and Martin Liebeck. *The subgroup structure of the finite classical groups*, volume 129 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, Cambridge, 1990. 85

[Knu98] Donald E. Knuth. *The Art of Computer Programming. Vol. 2.* Addison-Wesley Publishing Co., Reading, Mass., third edition, 1998. Seminumerical algorithms, Addison-Wesley Series in Computer Science and Information Processing. 14, 32, 41

[Koh08] Stefan Kohl. Algorithms for a class of infinite permutation groups. *J. Symbolic Comput.*, 43(8):545–581, 2008. 2

[Kov90] László G. Kovács. On tensor induction of group representations. *J. Austral. Math. Soc. Ser. A*, 49(3):486–501, 1990. 94

[KS01] William M. Kantor and Ákos Seress. Black box classical groups. *Mem. Amer. Math. Soc.*, 149(708):viii+168, 2001. 137

[KS02] William M. Kantor and Ákos Seress. Prime power graphs for groups of Lie type. *J. Algebra*, 247(2):370–434, 2002. 137

[KS03] William M. Kantor and Ákos Seress. Computing with matrix groups. In *Groups, combinatorics & geometry (Durham, 2001)*, pages 123–137. World Sci. Publ., River Edge, NJ, 2003. 137

[LG01] Charles R. Leedham-Green. The computational matrix group project. In *Groups and computation, III (Columbus, OH, 1999)*, volume 8 of *Ohio State Univ. Math. Res. Inst. Publ.*, pages 229–247. de Gruyter, Berlin, 2001. 3, 70, 104

[LGM02] Charles R. Leedham-Green and Scott H. Murray. Variants of product replacement. In *Computational and statistical group theory (Las Vegas, NV/Hoboken, NJ, 2001)*, volume 298 of *Contemp. Math.*, pages 97–104. Amer. Math. Soc., Providence, RI, 2002. 15, 99, 120, 128

[LGO97a] Charles R. Leedham-Green and Eamonn A. O'Brien. Recognising tensor products of matrix groups. *Internat. J. Algebra Comput.*, 7(5):541–559, 1997. 99, 101, 103, 120, 128

[LGO97b] Charles R. Leedham-Green and Eamonn A. O'Brien. Tensor products are projective geometries. *J. Algebra*, 189(2):514–528, 1997. 101, 103

[LGO02] Charles R. Leedham-Green and Eamonn A. O'Brien. Recognising tensor-induced matrix groups. *J. Algebra*, 253(1):14–30, 2002. 103, 136

[LGO07] Charles R. Leedham-Green and Eamonn O'Brien. Constructive recognition of classical groups in odd characteristic, 2007. Preprint.
`http://www.math.auckland.ac.nz/~obrien/papers.php`. 136

[LMM01] Andrea Lucchini, Frederico Menegazzo, and Marta Morigi. On the number of generators and composition length of finite linear groups. *J. Algebra*, 243(2): 427–447, 2001. 136

[LMO07] Frank Lübeck, Kay Magaard, and Eamonn A. O'Brien. Constructive recognition of $\mathrm{SL}_3(q)$. *J. Algebra*, 316(2):619–633, 2007. 137

[LNPS06] Maska Law, Alice C. Niemeyer, Cheryl E. Praeger, and Ákos Seress. A reduction algorithm for large-base primitive permutation groups. *LMS J. Comput. Math.*, 9:159–173 (electronic), 2006. 81

[LO07] Martin W. Liebeck and Eamonn A. O'Brien. Finding the characteristic of a group of Lie type. *J. Lond. Math. Soc. (2)*, 75(3):741–754, 2007. 137

[Lüb01] Frank Lübeck. Conway polynomials for finite fields, 2001.
`(http://www.math.rwth-aachen.de:8001/~Frank.Luebeck/data/ConwayPol/index.html?LANG=en)`. 17, 19

[McC90] Kevin S. McCurley. The discrete logarithm problem. In *Cryptology and computational number theory (Boulder, CO, 1989)*, volume 42 of *Proc. Sympos. Appl. Math.*, pages 49–74. Amer. Math. Soc., Providence, RI, 1990. 68

[MO95] Scott H. Murray and Eamonn A. O'Brien. Selecting base points for the Schreier-Sims algorithm for matrix groups. *J. Symbolic Comput.*, 19(6):577–584, 1995. 132

[MOS08] Kay Magaard, Eamonn A. O'Brien, and Ákos Seress. Recognition of small dimensional representations of general linear groups. *J. Aust. Math. Soc.*, 2008. Preprint.
`http://www.math.auckland.ac.nz/~obrien/papers.php`. 137

[Neb00] Gabriele Nebe. Faktorisieren ganzer Zahlen. *Jahresber. Deutsch. Math.-Verein.*, 102(1):1–14, 2000. 69

[Neu06] Max Neunhöffer. cvec – *compact vectors over finite fields*, 2006. GAP package. (http://www-groups.mcs.st-and.ac.uk/~neunhoef/Computer/Software/Gap/cvec.html). 23

[Nic88] Werner Nickel. Endliche Körper in dem gruppentheoretischen Programmsystem GAP, 1988. Diploma thesis. 17, 19

[NP92] Peter M. Neumann and Cheryl E. Praeger. A recognition algorithm for special linear groups. *Proc. London Math. Soc. (3)*, 65(3):555–603, 1992. 3, 104, 135

[NP98] Alice C. Niemeyer and Cheryl E. Praeger. A recognition algorithm for classical groups over finite fields. *Proc. London Math. Soc. (3)*, 77(1):117–169, 1998. 136

[NP08] Max Neunhöffer and Cheryl E. Praeger. Computing Minimal Polynomials of Matrices. *LMS JCM*, 2008. accepted. 4, 37

[NS06] Max Neunhöffer and Ákos Seress. A Data Structure for a Uniform Approach to Computations with Finite Groups. In *Proceedings of the 2006 International Symposium on Symbolic and Algebraic Computation (Genova)*, pages 254–261 (electronic), New York, 2006. ACM. 5, 70, 104

[NS08] Max Neunhöffer and Ákos Seress. recogbase – *a framework for group recognition*, 2008. GAP package. (http://www-groups.mcs.st-and.ac.uk/~neunhoef/Computer/Software/Gap/recogbase.html). 4

[O'B06] Eamonn A. O'Brien. Towards effective algorithms for linear groups. In *Finite geometries, groups, and computation*, pages 163–190. Walter de Gruyter GmbH & Co. KG, Berlin, 2006. 3, 37, 136

[Odl00] Andrew Odlyzko. Discrete logarithms: the past and the future. *Des. Codes Cryptogr.*, 19(2-3):129–145, 2000. Towards a quarter-century of public key cryptography. 68, 69

[Par84] Richard A. Parker. The computer calculation of modular characters (the meat-axe). In *Computational group theory (Durham, 1982)*, pages 267–274. Academic Press, London, 1984. 17, 62, 95, 105, 116, 117

[Rin94] Michael Ringe. The C-MeatAxe, 1994. (http://www.math.rwth-aachen.de/~MTX/). 17

[Ryb07] Alexander J. E. Ryba. Identification of matrix generators of a Chevalley group. *J. Algebra*, 309(2):484–496, 2007. 137

[Ser03] Ákos Seress. *Permutation Group Algorithms*. Cambridge University Press, Cambridge, 2003. 2, 72, 76, 77, 79, 80, 81, 96, 98, 111, 133

[Sim70] Charles C. Sims. Computational methods in the study of permutation groups. In *Computational Problems in Abstract Algebra*, pages 169–183. Pergamon Press, 1970. 72, 79

[Sta06] Mark J. Stather. *Algorithms for computing with finite matrix groups*. PhD thesis, University of Warwick, 2006. 4

[Sta07] Mark Stather. Constructive Sylow theorems for the classical groups. *J. Algebra*, 316(2):536–559, 2007. 4

[Ste97] Allan Steel. A new algorithm for the computation of canonical forms of matrices over fields. *J. Symbolic Comput.*, 24(3-4):409–432, 1997. Computational algebra and number theory (London, 1993). 39, 61

[Sto98] Arne Storjohann. An $O(n^3)$ algorithm for the Frobenius normal form. In *Proceedings of the 1998 International Symposium on Symbolic and Algebraic Computation (Rostock)*, pages 101–104 (electronic), New York, 1998. ACM. 38, 39

[Sto01] Arne Storjohann. Deterministic computation of the Frobenius form (extended abstract). In *42nd IEEE Symposium on Foundations of Computer Science (Las Vegas, NV, 2001)*, pages 368–377. IEEE Computer Soc., Los Alamitos, CA, 2001. 38, 39

[vzGG03] Joachim von zur Gathen and Jürgen Gerhard. *Modern computer algebra*. Cambridge University Press, Cambridge, second edition, 2003. 41

[Wil83] Robert A. Wilson. *Maximal Subgroups of Some Finite Simple Groups*. PhD thesis, University of Cambridge, 1983. 85

[Wil96] Robert A. Wilson. Standard generators for sporadic simple groups. *J. Algebra*, 184(2):505–515, 1996. 133, 134

[WWT$^+$99] Robert Wilson, Peter Walsh, Jonathan Tripp, Ibrahim Suleiman, Stephen Rogers, Richard Parker, Simon Norton, Simon Nickerson, Steve Linton, John Bray, and Rachel Abbott. The WWW Atlas of Finite Group Representations, 1999. (http://brauer.maths.qmul.ac.uk/Atlas/). 62, 134, 135

[Zas58] Hans J. Zassenhaus. *The theory of groups*. 2nd ed. Chelsea Publishing Company, New York, 1958. 109

Appendix E

Index