Constructive recognition of finite classical groups with stingray elements

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der RWTH Aachen University zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von

Daniel Rademacher, M.Sc.

aus Wesel

Berichter: Universitätsprofessorin Dr. Alice C. Niemeyer

Universitätsprofessor Dr. Max Horn

Emer. Professor of Mathematics Dr. Cheryl E. Praeger

Tag der mündlichen Prüfung: 25. September 2024

Diese Dissertation ist auf den Internetseiten der Universitätsbibliothek online verfügbar.

ABSTRACT OF THE DISSERTATION

Constructive recognition of finite classical groups with stingray elements

Daniel Rademacher

In 1988 Joachim Neubüser posed a matrix group related question in Oberwolfach which was answered by Peter Neumann and Cheryl E. Praeger in 1992. This initiated an international research effort, the *matrix group recognition project*, within the area of computational group theory with the aim of answering fundamental questions about arbitrary matrix groups over finite fields.

One possible method is a data structure called *composition tree*. In this approach, computations of a large matrix group are decomposed into computations for smaller matrix groups until this process cannot be repeated anymore. The remaining leaf groups are the finite (quasi-)simple groups, which include the classical groups.

Therefore, efficient algorithms to deal with classical groups are essential for the overall performance of the composition tree. One elementary aim is to develop an efficient algorithm for the constructive recognition of these groups.

This thesis presents a novel algorithm for constructively recognising classical groups within their natural representations, building upon preliminary concepts from Ákos Seress and Max Neunhöffer for special linear groups.

The algorithm consists of three subalgorithms:

1) **GoingDown algorithm:** Recursively descends from the input group G to a subgroup U isomorphic to a "base case group" using stingray duos and reaching such a group in significantly fewer steps than traditional methods.

2) **BaseCase algorithm:** Utilises an efficient method for constructively recognising the base case group U forming a starting point for the computation of standard generators of G.

3) **GoingUp Algorithm:** Extends standard generators from the subgroup U to the original group G, employing an original approach to compute generators for intermediate subgroups.

This research contributes to the broader goal of enhancing computational methods for matrix group recognition, with a particular focus on classical groups. It presents efficient algorithms that improve the performance of the composition tree method.

Acknowledgements

I am profoundly grateful for the invaluable guidance, support and expertise provided by my first supervisor, Prof. Dr. Alice C. Niemeyer. Her insightful feedback, encouragement and dedication to excellence have been instrumental in shaping my growth as a researcher in the field of mathematics. Additionally, I am very grateful to Prof. Dr. Alice C. Niemeyer for providing me with the opportunity for a research stay in New Zealand, which has significantly enriched my academic experience. I also express my sincere appreciation to my second supervisor, Prof. Dr. Max Horn, for his scholarly mentorship, technical insights and unwavering support throughout this research journey. His expertise in classical groups has broadened my perspective and enriched the quality of my work.

I extend my gratitude and heartfelt appreciation to Prof. Dr. Eamonn O'Brien for his invaluable mentorship, enlightening discussions and unwavering guidance throughout my enriching five-month research visit at the University of Auckland. Additionally I express my gratitude to the Department of Mathematics at the University of Auckland for fostering a warm and welcoming atmosphere. I also express my sincere appreciation to Prof. Dr. Cheryl E. Praeger for her invaluable support, especially during my visit at the University of Western Australia. Her mentorship has left a profound impact on my academic journey, and I am deeply grateful for her guidance. Additionally, I express my gratitude to Prof. Dr. Csaba Schneider for our insightful discussions on matrix groups and coding aspects.

Ákos Seress and Max Neunhöffer had preliminary ideas for special linear groups leading to the approach for constructive recognition presented in this thesis and I would like to thank Max Neunhöffer for sharing a draft of their project ideas with me.

I am deeply grateful for the opportunity to undertake this dissertation and complete this significant milestone in my academic journey. I would like to extend my sincere thanks to the RWTH Aachen University for providing a conducive environment for research and learning and for the financial support through the SFB-TRR 195 'Symbolic Tools in Mathematics and their Application' of the German Research Foundation (DFG) [Program ID 286237555] that made this work possible.

Furthermore, I extend my gratitude to Sergio Siccha for introducing me to the field of computational group theory, starting from our initial meeting at a pub during the first semester. I appreciate his guidance and the insightful discussions we have had on mathematical topics. I extend my heartfelt thanks to Dominik Bernhardt for his mentorship throughout my time as a Bachelor's and Master's student. His warm welcome to the department and inclusion of me in various conferences as a student have been invaluable experiences. I also express my gratitude to my colleagues and friends at the Chair of ART for the wonderful and enjoyable times we shared, especially during our board game and karaoke nights. I extend many thanks to Giulia Iezzi for the enriching discussions we shared during breaks and to Tom Görtzen for the insightful bouldering instructions. I also extend my thanks to Wolfgang Krass, who not-so-secretly runs our department from behind the scenes. His welcoming atmosphere and enjoyable conversations have made my time at the department truly wonderful. I extend special thanks to Friedrich Rober for being an exceptional office partner and friend, sharing humorous moments during conferences and in Australia and, of course, engaging in insightful mathematical discussions with me. Moreover, I express my gratitude to Verity Mackscheidt, affectionately known as Pikachu, for our many enjoyable and insightful discussions on a variety of (non-mathematical) topics.

I express my gratitude to the many people who shared the journey of studying mathematics with me. First, I extend my heartfelt thanks to Anna Katharina Dora and Lucas Fabian Wollenhaupt, who remained steadfast companions throughout our algebra lectures. You have been the best friends one could wish for and I treasure every experience we shared during our time in Aachen. I wish you both and Ada, all the best in your future endeavours. Secondly, I thank Alena Meyer, Astrid Hagemeyer, Marius Graumann, Duc Khuat and Linh An Vu for their exceptional teamwork in tackling algebra lectures and exercises with me. Moreover, I express my gratitude to Paul Geuchen, Miriam Chlumsky-Harttmann, Mona Kuntz, Katharina Felderhoff, Theresa Victoria Hausen, Luca Polzius and Felix Pennig, who were my steadfast companions throughout my time studying in Aachen.

I wish to thank my friends for their companionship and for sharing such wonderful times together. Many, many thanks also to my exceptional and wonderful friends Sven Argo, Simon Berger and Niko Molke, who have been my closest friends since we met in school. Thank you very much for

all the wonderful memories we share together and for your unwavering support.

Furthermore, I express my heartfelt thanks to my wonderful parents, Gudrun and Helge Rademacher, who have supported and encouraged me since the day I was born. Their unwavering support has made my academic journey possible and I am endlessly grateful. I extend my gratitude to my grandparents and all other members of my family for their support and encouragement.

I express my sincere gratitude to my wonderful brother, Tobias Rademacher, for his incredible encouragement throughout my time as a PhD student. The memories we've created together are unforgettable, filled with laughter and joy. I hope to be as good a brother to you as you are to me.

Lastly, I would like to extend my deepest gratitude to my girlfriend, Julia Schmitz, whose kindness and warmth have added balance to my life. Her love and belief in me have made even the most challenging moments bearable and I am forever grateful for her presence in my life.

Eidesstattliche Erklärung

Ich, Daniel Rademacher, erkläre hiermit, dass diese Dissertation und die darin dargelegten Inhalte die eigenen sind und selbstständig, als Ergebnis der eigenen originären Forschung, generiert wurden. Hiermit erkläre ich an Eides statt

1. Diese Arbeit wurde vollständig oder größtenteils in der Phase als Doktorand dieser Fakultät und Universität angefertigt;

2. Sofern irgendein Bestandteil dieser Dissertation zuvor für einen akademischen Abschluss oder eine andere Qualifikation an dieser oder einer anderen Institution verwendet wurde, wurde dies klar angezeigt;

3. Wenn immer andere eigene- oder Veröffentlichungen Dritter herangezogen wurden, wurden diese klar benannt;

4. Wenn aus anderen eigenen- oder Veröffentlichungen Dritter zitiert wurde, wurde stets die Quelle hierfür angegeben. Diese Dissertation ist vollständig meine eigene Arbeit, mit der Ausnahme solcher Zitate;

5. Alle wesentlichen Quellen von Unterstützung wurden benannt;

6. Wenn immer ein Teil dieser Dissertation auf der Zusammenarbeit mit anderen basiert, wurde von mir klar gekennzeichnet, was von anderen und was von mir selbst erarbeitet wurde;

7. Ein Teil oder Teile dieser Arbeit wurden zuvor veröffentlicht und zwar in [49] Max Horn and Alice C. Niemeyer and Cheryl E. Praeger and Daniel Rademacher. *Constructive Recognition of Special Linear Groups*. 2024. arXiv: 2404.18860 [math.GR].

October 17, 2024

Authorship Declaration

This thesis contains work that has been submitted to a journal.

Details of the work: [49] Max Horn and Alice C. Niemeyer and Cheryl E. Praeger and Daniel Rademacher. *Constructive Recognition of Special Linear Groups*. 2024. arXiv: 2404.18860 [math.GR].

Student contribution to work: The author of this thesis contributed fully to all mathematical ideas. The author of this thesis was responsible for the writing of [49, Section 2 - 8] and the mathematical ideas of these sections. The sections [49, Section 2 - 8] are a condensed version of Chapter 2, Chapter 3, Chapter 5 and Chapter 11 of this thesis and were written by the author of this thesis. The author of this thesis implemented the theory developed in [49]. His implementation can be found in the GAP-package [82].

Dedication

For my family and friends.

Contents

Chapter 1

Introduction

"The essence of math is not to make simple things complicated, but to make complicated things simple"[1], a principle mirrored in algorithms since they serve as the computational backbone of mathematics, translating abstract concepts into precise, step-by-step instructions for problem-solving. Fundamentally, algorithms encode the instructions of how we think mathematically, expressing the art of simplifying complex mathematical processes into manageable computational steps.

Algorithmic group theory is the intersection of abstract algebra and computational mathematics, aiming to develop tools for studying the properties, structures, and representations of groups. The roots of group theory can be traced back to the late 18th and early 19th centuries when famous mathematicians such as Carl Friedrich Gauss, Évariste Galois and Augustin-Louis Cauchy established foundational work. They introduced formal notations for abstract algebra which is the starting point of more formalised studies.

During the progression of group theory, the study and classification of finite groups, in particular finite simple groups, gained prominence as finite simple groups are the building blocks of finite groups. In the mid-20th century, fundamental work by mathematicians such as Camille Jordan, Émile Mathieu, Wilhelm Killing and many others advanced the subject. However, as more and more complex groups had to be investigated, the need for computer assisted approaches instead of pen and paper-based calculations increased.

[1] Stanley P. Gudder

In the 20th century significant advancements in computational mathematics, such as Schreier-Sims and Johnson's algorithms, were the starting point for computational group theory as a distinct field. Nowadays mathematicians and computer scientists are developing algorithms and techniques to study groups and their properties and representations on the computer. The use of computations instead of theoretical proofs marked a central moment in the history of algorithmic group theory.

Classifying finite simple groups through their matrix representations proved to be an enormous effort. The aim of classifying finite simple groups profited from algorithmic group theory in two ways. Firstly, computational tools were developed to prove the existence of theoretical groups, such as those developed by Charles Sims [87]. Secondly, by providing computational tools that facilitate the exploration of simple groups. Many mathematicians played crucial roles as they developed important algorithms and computational tools for recognising and analysing groups [48].

Accomplishments in computational group theory and the improvement of computational power, contributed to the eventual completion of the classification of finite simple groups (CFSG) in 1980s. The CFSG marks a tremendous achievement in mathematics, yielding beneficial impacts on algorithmic group theory. The proof of the CFSG is highly complex as many results are distributed across hundreds of journal articles and books which additionally rely on deep mathematical insights. Gorenstein, Lyons, Solomon and others are currently working on a series of books on the proof of the CFSG [44]. Nowadays, computational tools continue to play a vital role for exploring properties and representations of these groups.

One exceptional achievement in algorithmic group theory is the work "Atlas of Finite Group" in 1985 [29] containing detailed information on various finite simple groups, their properties, and matrix representations. The Atlas is a significant source of information for researchers, offering crucial data for further investigations. This project started as a collaborative effort by John Conway, Robert Curtis, Simon Norton, Richard Parker and Robert Wilson and has involved contributions by many others since then. Advancements from this project have also lead to the creation of a modular Atlas [54].

As algorithmic group theory advanced with crucial results such as the CFSG and the Atlas, along with the development of efficient algorithms for handling permutation groups [85] and polycyclic

groups [86] in the mid-1980s, algorithms specifically designed for matrix groups were missing.

Joachim Neubüser (1932-2021) was a professor at RWTH Aachen University and the founder of the computational algebra system GAP [37] in 1986. The first algorithms implemented in GAP included algorithms for permutation groups, finitely-presented groups, polycyclic groups, as well as algorithms for character tables of finite groups. However, working with matrix groups was only possible for very small groups as even algorithms for fundamental tasks, such as computing the order of a matrix, were exponential in the degree of the matrix group. At the time, algorithms for finite matrix groups, i.e. subgroups of the general linear group $\mathrm{GL}(d,q)$ over the finite field $\mathbb{F}_q$, relied on algorithms for permutation groups by letting the matrix group act on the 1-dimensional subspaces of $\mathbb{F}_q^d$. Clearly, even small dimensional matrix groups thus yielded permutation groups of large degree, as the number of 1-dimensional subspaces of a d-dimensional space is

$$\binom{d}{1}_q = \frac{(q^d-1)}{(q-1)}.$$

As working even with small dimensional matrix groups seemed out of reach, Joachmin Neubüser posed the following question at a meeting on computer algebra in Oberwolfach in 1988. Let G be a subgroup of the general linear group $\mathrm{GL}(d,q)$ over the finite field $\mathbb{F}_q$. Is there an efficient algorithm to decide whether the special linear group, $\mathrm{SL}(d,q)$, is contained in G, i.e. $\mathrm{SL}(d,q) \leq G$? In 1992 this question was answered by Neumann and Praeger [69]. Their algorithm was randomised and required $\mathcal{O}(d)$ matrix multiplications. The novel and randomised approach of Neumann and Praeger sparked the interest in designing efficient randomised algorithms for working in matrix groups.

Let $X := \{a_1,\ldots,a_k\} \subseteq \mathrm{GL}(d,q)$ and let $G = \langle X\rangle \leq \mathrm{GL}(d,q)$. We would like to have efficient algorithms to answer fundamental problems about G including:

- Determine the order of G.
- Given $g \in \mathrm{GL}(d,q)$, decide whether $g \in G$. This problem is henceforth referred to as the *membership problem*.
- If $g \in G$, then write g as a word in X. This problem is henceforth referred to as the *word problem*.
- Determine representatives of the conjugacy classes of the maximal subgroups of G.

- Given subgroups $H, K \leq G$, compute generators for the group $H \cap K$.
- Determine a generating set for the automorphism group of G.

These problems led to an international research effort in the area of computational group theory, namely the *matrix group recognition project* [5]. The aim of the matrix group recognition project is to acquire information about a matrix group. Since Neumann and Praeger's algorithm [69], many algorithms as well as GAP and Magma code has been published, see [76] for an overview. A first theoretical solution to solve the problems given above, the *black-box group approach*, was presented by Babai and Beals [9, 10] which aims to determine the abstract group-theoretic structure of a group. Another solution, the *geometric approach*, investigates the action of a group on its underlying vector space. In particular, the goal of the geometric approach is to construct a data structure, called *composition tree* for G, that facilitates computations with G. From this data structure one can immediately determine the order of G or answer more difficult questions about G. This PhD thesis forms part in the geometric approach of the matrix group recognition project.

Many mathematicians collaborated within the matrix group recognition project and one highlight of the geometric approach is the following theorem from 2019.

Theorem 1.1: Holt, Leedham-Green and O'Brien, 2019 [47]

Subject to the existence of a *discrete logarithm oracle* and an *integer factorisation oracle*, there is a polynomial-time *Las Vegas algorithm* in q and d that takes as input $G = \langle X \rangle \leq \mathrm{GL}(d, q)$ and returns as output a *composition tree* for G.

Section 1.1 provides an overview of the matrix group recognition project in which the topic of this thesis is located. Various concepts are described, including randomised algorithms, composition trees, black-box groups, and naming algorithms, along with their basic features. The results of this thesis contribute to constructive recognition of classical groups explained in Section 1.1.8. In Section 1.2 we motivate the objective and in Section 1.3 we illustrate a top-down description of the algorithms for constructive recognition and state the main results. Section 1.4 describes the structure of this thesis.

1.1 Background and literature

Before we embark on discussing the composition tree data structure in detail and important partial results leading to Theorem 1.1, we start by introducing some basic concepts and fixing notations. Note that many people have contributed to the matrix group recognition project. However, we only cite and refer to results which are relevant to this thesis and the treatment presented is based on [76]. This section is not an overview of the matrix group recognition project and instead we refer to [76].

1.1.1 Randomised algorithms

The order of the linear group $\mathrm{GL}(d,q)$ is exponential in d, that is $|\mathrm{GL}(d,q)| \in \mathcal{O}(q^{d^2})$, which makes it tricky to deal with these groups in practice. Hence, most of the algorithms for linear groups are *randomised*, i.e. the algorithms use uniformly distributed and independent random elements. Since groups are given by generating sets, which in practice are often very small, we require algorithms to produce these uniformly distributed and independent elements of a group as words in the generators. Algorithms to compute random elements are discussed further in Section 2.5. We follow the treatment of Seress [85, Section 1.3].

Definition 1.2: [85, p. 13]

Given a set of inputs X and a set of outputs Y, a *computational task* is a relation $R \subseteq X \times Y$. If $R(x,y)$ holds for a pair $(x,y) \in X \times Y$, then y is the correct output for input x.

Definition 1.3: [85, p. 13]

Let X be a set of inputs and Y a set of outputs. Let R be a computational task as in Definition 1.2. A *deterministic algorithm* defines a function $f : X \to Y$ and is *correct* if $R(x, f(x))$ holds for all $x \in X$. A *randomised algorithm* is a function $f : X \times S \to Y$ for a set S of random seeds. A randomised algorithm uses a string $r \in S$ of random bits ("coin flippings") and returns the output $f(x,r)$. The output may not be correct for every sequence $r \in S$ and $f(x,r_1)$ may differ from $f(x,r_2)$ for different random strings $r_1, r_2 \in S$.

We distinguish between two types of randomised algorithms, *Monte Carlo algorithms* and *Las Vegas algorithms*, which play an important role in this thesis.

Definition 1.4: Monte Carlo algorithm [85, p. 13]

Let (Ω, Σ, P) be a probability space and let $R \subset X \times Y$ be a computational task. For $\epsilon \in (0, \frac{1}{2})$ we call a randomised algorithm $f : X \times \Omega \to Y$ a *Monte Carlo algorithm*, if for all inputs $x \in X$,

$$P_{r \in \Omega}(R(x, f(x, r)) \text{ holds}) \geq 1 - \epsilon.$$

The probability of an incorrect answer can be bounded from above by ϵ.

Moreover, for decision problems a Monte Carlo algorithm is *one-sided* if at least one return value is guaranteed to be correct.

A particular type of Monte Carlo algorithms, the Las Vegas algorithms was introduced by Babai in 1997.

Definition 1.5: Las Vegas algorithm [85, p. 13]

A *Las Vegas algorithm* is a Monte Carlo algorithm whose output is either correct (with the prescribed probability at least $1 - \epsilon$) or the algorithm reports failure. Here, ϵ may be any given constant less than 1, since the probability of an (always correct) output can be increased to at least $1 - \epsilon^t$ by running the algorithm t times.

The algorithms of this thesis are one-sided Monte Carlo algorithms. For more information about randomised algorithms see [8] and [85].

1.1.2 Oracles

The running time of algorithms to solve computational problems is usually expressed as a function φ in the input size and reflects the *complexity of an algorithm*. If the function φ can be bounded by a polynomial expression, then an algorithm is a *polynomial time* algorithm. We discuss complexity of algorithms in more detail in Section 2.5.

Some computational problems are assumed to be difficult to solve. Two of these problems are *integer factorisation* and the *discrete logarithm*.

Definition 1.6

1) Suppose $n \in \mathbb{N}$ is not a prime. The computational problem to find $a, b \in \mathbb{N}$ with $n = a \cdot b$ and $a, b \neq 1$ is the *integer factorisation problem*.
2) Let q be a prime power, let $\mathbb{F}_q$ be a finite field with q elements and let $\omega \in \mathbb{F}_q$ be a primitive root, i.e. ω generates the multiplicative group $\mathbb{F}_q^*$. Given $\imath \in \mathbb{F}_q^*$ the computational problem to find $k \in \mathbb{N}$ with $\omega^k = \imath$ is the *discrete logarithm problem*.

A famous unanswered questions in computer science is "P versus NP" where P describes the class of problems which can be solved in polynomial time and NP describes the class of problems for which answers can be verified in polynomial time. The question "P versus NP" asks whether the class NP of problems whose answers can be verified in polynomial time equals the class P of problems which can be solved in polynomial time. The question whether P = NP is listed as a Millennium problem and a solution is rewarded with 1,000,000 US dollars. NP-hard problems are at least as difficult as the hardest problems in NP. At this point, NP-hard problems are considered to be difficult, which means that they cannot be solved in reasonable time if the parameters of an NP-hard problem are large enough. Even though it is an open problem whether the two problems described in Definition 1.6 are NP-hard, they are widely used in cryptography, e.g. in ciphers such as RSA and ElGamal, as factorisation and computation of discrete logarithms becomes computationally infeasible for growing n and q using the state-of-the-art algorithms. Therefore, we introduce the concept of an *oracle*.

Definition 1.7

An *oracle* is a machine capable of producing solutions for instances of a specific computational problem in a single operation.

Definition 1.8

1) An *integer factorisation oracle* is an oracle which, given $n \in \mathbb{N}$ such that n is not a prime, produces $a, b \in \mathbb{N}$ such that $n = a \cdot b$ and $a, b \neq 1$.
2) Let q be a prime power, let $\mathbb{F}_q$ be a finite field of size q and let $\omega \in \mathbb{F}_q$ be a primitive root. Given $\imath \in \mathbb{F}_q^*$ a *discrete logarithm oracle* is an oracle which produces $k \in \mathbb{N}$ with $\omega^k = \imath$.

Even though these oracles are not directly required in the algorithms of this thesis, the discrete

logarithm oracle and factorisation oracle are currently needed for constructive recognition of $\mathrm{SL}(2,q)$. Constructive recognition is later described in Section 1.1.8 and a solution for constructive recognition of $\mathrm{SL}(2,q)$ is discussed in more detail in Section 5.2.

1.1.3 Composition trees

Results about computational group theory are given in e.g. [48] and results on computing with matrix groups and their complexity can be found in e.g. [12, 56]. In the introduction of this chapter we suggested that composition trees can be used to facilitate computations with matrix groups. In this section we define composition trees and provide a rough description how this data structure can be computed. For more information about the composition tree see [61, 72].

Definition 1.9

Let G be a group.

1) A subgroup $N \leq G$ is a *normal subgroup* if $n^g := g^{-1}ng \in N$ for all $n \in N, g \in G$. If N is a normal subgroup of G, then this is denoted by $N \trianglelefteq G$.
2) If G is not the trivial group $\{1\}$ and if $\{1\}$ and G are the only normal subgroups of G, then G is a *simple group*.
3) If the quotient group of G by its center $Z(G) := \{z \in G \mid zg = gz \text{ for all } g \in G\}$ is a simple group, then G is a *quasi-simple group*.

Definition 1.10: [76, Section 11]

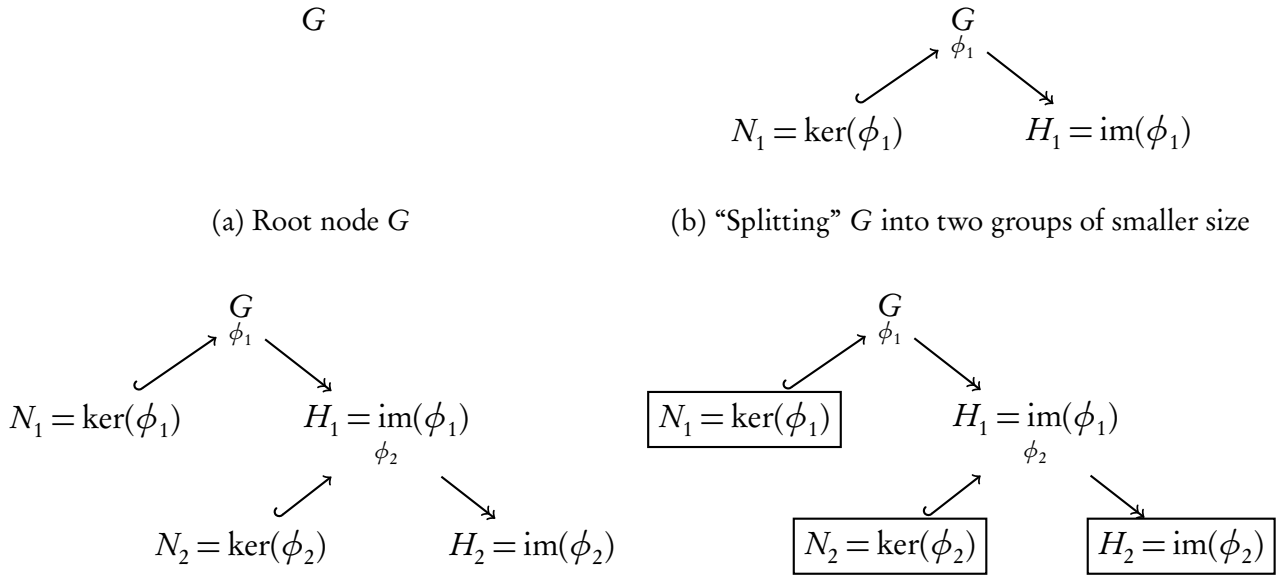
Let G be a group. A *composition tree* of G is a labelled and strict binary tree, where the nodes are tuples of groups and homomorphisms with the following properties:

1) (G, φ_G) is the root node.
2) If a node (K, φ_K) is a leaf node, then K is either simple or quasi-simple and $\varphi_K = \mathrm{Id}$.
3) If a node (K, φ_K) is not a leaf node, then the group of the left child corresponds to a proper normal subgroup $\mathrm{Kernel}(\varphi_K) =: N \trianglelefteq K$ and the group of the right child is $\varphi_K(K)$ isomorphic to K/N.

Remark 1.11

Given a group K, which corresponds to a node in the composition tree, and a group U, any homomorphism $\varphi: K \to U$ with non-trivial kernel, i.e. $\text{Kernel}(\varphi) \notin \{\{1\}, K\}$, yields a left and right child as $N := \text{Kernel}(\varphi) \trianglelefteq K$ with $N \neq K$ and $H := \text{im}(\varphi) \cong K/\text{Kernel}(\varphi) \neq K$. ◀

Composition trees are data structures which can be computed in polynomial time using specific oracles for any matrix group $G \leq \text{GL}(d, q)$ as stated in Theorem 1.1. Most of the algorithms for computing this data structure are randomised and an overview article by Eamonn O'Brien summarises many algorithms involved in more detail [76].



(a) Root node G

(b) "Splitting" G into two groups of smaller size

(c) Complete composition tree of G

(d) Leaf nodes are (quasi-)simple groups

Figure 1.1: Sequence of a computation of a composition tree

Figure 1.1 illustrates the computation of a composition tree for a group G. In Figure 1.1a we initialise a binary tree with root node G. Then we compute $H \leq \text{GL}(d', q')$ and $\phi_1: G \to H$ with non-trivial kernel, i.e. $\ker(\phi_1) \neq G$ and $\ker(\phi_1) \neq \{1_G\}$. Usually we require that computing images and preimages of a homomorphism ϕ_1 is efficient, that is, we can compute generating sets for $\text{im}(\phi_1) = H_1$ and $\ker(\phi_1) = N_1$. Moreover, $1 < |N_1|, |H_1| < |G|$ since ϕ_1 has a non-trivial kernel. The "splitting" of G is displayed in Figure 1.1b. We can repeat the same process for the groups N_1 and H_1. If it is not possible to compute a homomorphism with a non-trivial, proper kernel for a group, then this node becomes a leaf group. Figure 1.1c illustrates how the full composition tree of a group G might look like. In Figure 1.1d the leaf groups are highlighted. These are groups for which every homomorphism has a trivial kernel, i.e. simple groups. In many practical applications

the algorithm also accepts quasi-simple groups as leaf groups.

An important tool in determining the composition tree for a matrix group G is Aschbacher's Theorem [2] which defines nine families of subgroups of the general linear group G such that every maximal subgroup of G lies in one of these classes and gives structural information about each family. That means G is either a classical group or else lies in one of these families. In order to find a composition tree for a matrix group $G \leq \mathrm{GL}(d,q)$, this theorem is employed to compute a homomorphism with non-trivial kernel from G using structural information of the Aschbacher families.

The problem of determining information about G is then reduced to determining information about the two child groups in the composition tree. This process can be repeated with the two new groups and terminates with the leaf node groups, i.e. usually groups closely related to the finite simple groups. Hence, the efficiency of the composition tree algorithm relies heavily on the efficiency of the algorithms for the leaf groups.

Remark 1.12
Given a composition tree for a group $\langle X \rangle = G \leq \mathrm{GL}(d,q)$ we can solve the word problem for G. More precisely, let $H \leq \mathrm{GL}(d',q')$, $\phi\colon G \to H$ with non-trivial kernel, $N_1 = \ker(\phi)$ and $H_1 = \mathrm{im}(\phi)$. By induction suppose we can solve the word problem efficiently for N_1 and H_1. Suppose we want to write $g \in G$ as a word in X. Since we can solve the word problem in H_1 we can write $\phi(g) \in H_1$ as a word in $\phi(X)$. Evaluating this word in X outputs $\tilde{g} \in G$ with $\phi(\tilde{g}) = \phi(g)$. If $\tilde{g} = g$, then we are finished. Otherwise $\phi(g\,\tilde{g}^{-1}) = 1_H$ and, hence, $g\,\tilde{g}^{-1} \in \ker(\phi)$. Since we can solve the word problem in N_1, we can write $g\,\tilde{g}^{-1}$ as a word in a generating set for N_1. But now we can write $g\,\tilde{g}^{-1}$ and $\tilde{g}$ as a word in X and, therefore, also g. ◀

1.1.4 Classification of finite simple groups

Algorithms to handle the leaf groups in a composition tree rely on the classification of the finite simple groups (CFSG). This classification, concluded in 2004 through the research on quasithin groups by Aschbacher and Smith [3, 4], states that a finite simple group is one of the following:

1) a cyclic group of prime order,

2) a finite alternating group of degree at least 5,

3) a finite simple group of Lie type, which includes the classical groups, or

4) one of 26 sporadic simple groups.

While efficient algorithms exist to solve the questions posed in Chapter 1 for finite (quasi-)simple groups in their natural representations, it is not easy to transfer these results to the leaves of a composition tree, since a leaf group can arise in a wide variety of different representations. Hence we need efficient algorithms to answer the following fundamental problems.

Remark 1.13

1) Given a finite simple group, to which finite simple group in the list of all finite simple groups, is it isomorphic? An algorithm, which takes as input a group G and an isomorphism type of finite simple groups and returns true if G is isomorphic to a group of this type, is known as a *naming* algorithm.
2) Suppose it is known that a group G is isomorphic to a finite simple group T. Determine an explicit isomorphism from G to T, thereby transferring various tasks to the natural representations e.g. the membership and word problem. This is known as *constructive recognition*.

◀

1.1.5 Representations

Section 2.4 deals with the basics of representation theory and for now we only state one definition of this section.

Definition 1.14: [68, Definition 1.1.1]

A *linear representation* of a group G on a vector space V over a field $\mathbb{F}_q$ is a group homomorphism

$$\mathscr{X} : G \to \mathrm{GL}(V),$$

i.e. $(g_1 g_2)^{\mathscr{X}} = g_1^{\mathscr{X}} g_2^{\mathscr{X}}$ for all $g_1, g_2 \in G$. A *matrix representation* of G of degree d is a homomorphism

$$\mathscr{X} : G \to \mathrm{GL}(d, q).$$

Groups can be given in a variety of irreducible representations, e.g. special linear groups as in the next example.

Example 1.15

Let $G := \mathrm{SL}(4,5)$ be the special linear group of 4×4 matrices over $\mathbb{F}_5$. A generating set of G is given by the Steinberg generators [88, Theorem 3.14], i.e.

$$G = \left\langle a_1 := \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, a_2 := \begin{pmatrix} 4 & 0 & 0 & 1 \\ 4 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 4 & 0 \end{pmatrix} \right\rangle.$$

Therefore, we have matrix representation of G of degree 4 which is in fact irreducible. In Section 2.4 we define this representation as the *natural representation*. We consider a second group given by a generating set which is

$$H = \left\langle c_1 := \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}, c_2 := \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \right\rangle \leq \mathrm{GL}(6,5).$$

Then $H \cong \mathrm{SL}(4,5) = G$ and there exists an isomorphism $\mathscr{X} : G \to H$ which is a matrix representation of G of degree 6. This irreducible representation of $\mathrm{SL}(4,5)$ given by $\mathscr{X}$ is the *exterior square representation*. ◀

In Example 1.15 we have seen that a group can be given by different irreducible representations. Leaf groups of the composition tree can arise in any irreducible representation of an (almost) simple group, i.e. if one leaf group of a composition tree is isomorphic to $\mathrm{SL}(4,q)$, then the leaf group can, for example, be given as G or H of Example 1.15. Therefore, it is important to be able to handle each of these irreducible representations.

1.1.6 Black-box groups and black-box algorithms

It is well-known that groups can be given in a wide variety of different representations. In order to design algorithms which are applicable to all representations of a specific group, Babai and Szemerédi [12] introduced the concept of *black-box groups*. In this thesis we use the definition of black-box groups given by Seress in [85].

Definition 1.16: [85, Chapter 2]

A *black-box group* G is a group whose elements are encoded as strings of length at most N over an alphabet Q, for some positive integer N and finite set Q. We do not require that group elements have a unique representation as a string and not all strings need to correspond to group elements. Group operations are performed by an oracle (the black box). Given strings representing $g, h \in G$, we can

1) compute a string representing gh,
2) compute a string representing g^{-1} and
3) decide whether $g = 1_G$.

We do not know anything about the elements of black-box groups and cannot make any assumptions as we have to use oracles for basic group operations. For example computing the order of an element $g \in G$ of a black-box group can only be computed by seeking the smallest $k \in \mathbb{N}$ for which $g^k = 1_G$. In contrast, provided a factorisation oracle, computing the order of a matrix is quite efficient using the pseudo-order algorithm by Celler and Leedham-Green [24]. Nevertheless, algorithms which can deal with black-box groups are very useful as they are applicable to every representation of a group even though this comes often at the price of a worse complexity as it is not possible to exploit properties of a specific representation.

Definition 1.17: [85, Chapter 2]

A *black-box group algorithm* is an algorithm that does not use specific features of the group representation or particulars of how the group operations are performed. It can use only the operations described in Definition 1.16.

Black-box algorithms are studied in more detail by Seress [85].

1.1.7 Naming algorithms

In Section 1.1.3 we introduced the data structure composition tree and noted in Remark 1.13 the following two problems for leaf groups:

1) To which finite simple group is a leaf group isomorphic? This is known as *naming*.

2) Determine an explicit isomorphism of a leaf group onto its natural representation, thereby transferring e.g. the membership and word problems to the natural representations. This is known as *constructive recognition*.

In general, naming algorithms have better run-times than constructive recognition algorithms. Therefore, we first apply naming algorithms to leaf groups to obtain the name of the leaf group as in the classification of finite simple groups, see Section 1.1.4. Afterwards, we call a constructive recognition algorithm which is specifically designed to handle leaf groups of that name, see Section 1.1.8. In this section we deal with naming algorithms which are defined as follows.

Definition 1.18: Naming algorithm

Let $G \leq \mathrm{GL}(d,q)$ and H a simple group. An algorithm is a *naming algorithm for H* if it answers for an input group G the question if $G \cong H$.

After a question posed by Neubüser, a first algorithm to decide whether a subgroup of $\mathrm{GL}(d,q)$ contains $\mathrm{SL}(d,q)$ was presented by Neumann and Praeger [69]. Note that this algorithm can be regarded as a naming algorithm since if $G \leq \mathrm{GL}(d,q)$ with $\mathrm{SL}(d,q) \leq G$ and the generators of G have determinant 1, then $G = \mathrm{SL}(d,q)$. In 1998, Niemeyer and Praeger [74] developed algorithms to answer corresponding questions for the remaining classical groups in their natural representation. Many naming algorithms of the composition tree are randomised and one-sided Monte Carlo algorithms.

In Section 1.1.5 we noted that leaf groups can be given in a variety of representations and, therefore, introduced the concept of black-box groups in Section 1.1.6. The concept of black-box groups was used by Babai et al. [11] to present an important black-box algorithm for naming.

Theorem 1.19: Black-box naming [11]

Given a black-box group isomorphic to a simple group of Lie type of known characteristic, then the standard name can be computed using a polynomial-time Monte Carlo algorithm.

Note that Theorem 1.19 requires that the characteristic is known. Liebeck and O'Brien [63] developed an algorithm to determine the characteristic of a black-box group.

1.1.8 Constructive recognition

In this section we assume that a leaf group G of the composition tree is given and the name of G is known, see Section 1.1.7. In this section we deal with 2) of Remark 1.13 which is constructive recognition and defined as follows.

Definition 1.20: [76, Section 3]

A constructive recognition algorithm takes as input a generating set X of a group G. We assume G is known to be an irreducible representation and isomorphic to a "golden copy" H of a target group, usually a simple or quasi-simple group, using naming algorithms, see Section 1.1.7. The "golden copy" H is a specific irreducible representation. The aim of *constructive recognition* is to compute an epimorphism

$$\varphi: G \to H$$

which allows efficient computations of images and preimages under φ.

The problem of constructive recognition has to be solved for all possible leaf groups and, therefore, for all groups in the classification of finite simple groups, see Section 1.1.4. A constructive recognition algorithm for alternating groups as black-box groups is given by Beals et al. in [14] and a constructive recognition algorithm for alternating groups in unknown degree by Leuner et al. in [53]. In [93] Wilson introduces standard generators for sporadic groups. Wilson and others solve the constructive membership problem for the Monster group given as a black-box group in [92] and for each sporadic group O'Brien and Wilson provide a black-box constructive-membership algorithm in [46]. Constructive recognition for exceptional groups has been solved by Liebeck and O'Brien [64].

In this thesis we consider the important family of classical groups which plays a crucial role within the classification of finite simple groups. The first constructive recognition algorithm for $\mathrm{SL}(d,q)$ in its natural representation was published by Celler and Leedham-Green [23] in 1988 and has complexity $\mathcal{O}(d^4q)$. In 2001 Kantor and Seress [55] published a black-box constructive recognition algorithm for classical groups, though that algorithm is more theoretical than practical. In [19] Brooksbank gives another constructive recognition algorithm for classical groups in their natural representation having complexity $\mathcal{O}(d^5\log^2(q))$. This algorithm uses an $\mathrm{SL}(2,q)$ oracle. Leedham-Green and O'Brien [59] introduced a randomised constructive recognition algorithm for classical

groups in odd characteristic in 2009 and Dietrich, Leedham-Green, Lübeck and O'Brien (DLLO) [32] for even characteristic in 2013. Their algorithms were also adapted to classical groups given as black-box groups in [33]. The DLLO algorithm is currently used in application for the composition tree in Magma [16].

Remark 1.21

Note that constructive recognition is extremely helpful and important for transferring information from "golden copies" into non-natural irreducible representations. For example the computation of maximal subgroups of simple groups is extremely challenging. Computing all maximal subgroups of the monster group in a "golden copy" was only finished with the work of Dietrich, Lee and Popiel [31] in 2023. Using constructive recognition the maximal subgroups of a non-"golden copy" irreducible representation of a simple group can be computed by setting up an isomorphism between a "golden copy" and a given irreducible representation of a simple group. We then write the generators of the maximal subgroup as words in the "golden copy" and evaluate these words in the given irreducible representation. The same procedure can be used for computing conjugacy classes. ◀

1.1.9 MSLP

In the introduction of this chapter we presented the word problem which, given $G = \langle X \rangle$ and $g \in G$, asks to write g as a word in X. An important tool for this problem are *straight line programs* (SLP) which can be used to encode a word in X and additionally to write words as products of precomputed subwords resulting in an expression which is less computationally intensive to evaluate. Babai and Szemeredi [12] prove that every element of G can be encoded in an SLP for any generating set X of G of length at most $\mathcal{O}(\log^2(|G|))$. An extended version of SLPs are *memory straight line programs* (MSLP) which are described in Section 2.6.

1.1.10 Rewriting procedures

In Section 1.1.8 we described the problem of constructive recognition for a group G in Definition 1.20. Let G be isomorphic to the "golden copy" H. The problem of computing an efficient isomorphism $\varphi\colon G \to H$ is often solved based on the following three steps:

1) Specify a computationally useful set $S \subseteq H$ of generators, called *standard generators*.

2) Express $S' := \varphi^{-1}(S) \subseteq G$ as words in X (without the knowledge of φ).

3) Set up an isomorphism $\varphi: G \to H$ by mapping S' to S.

After these three steps, we have only constructed a partial isomorphism from G to H which is given by the images and preimages of the standard generators S and S'. To finalise this map, efficient algorithms for the following two tasks are required:

a) Given $h \in H$, write h as a word in S.

b) Given $g \in G$, write g as a word in S'.

Thus we have to solve the word problem both in G and in H. In Section 1.1.9 we introduced SLPs and MSLPs to encode words in generating sets. Given $h \in H$, we write h as a word in S, i.e. $h = h_1 h_2 \dots h_\ell$ with $h_i \in S \cup S^{-1}$, and encode this word in an SLP $\mathfrak{S}$. If we evaluate $\mathfrak{S}$ in S this outputs h. Moreover,

$$\varphi^{-1}(h) = \varphi^{-1}(h_1 h_2 \dots h_\ell) = \varphi^{-1}(h_1)\varphi^{-1}(h_2)\dots\varphi^{-1}(h_\ell)$$

and $\varphi^{-1}(h_i) \in S' \cup (S')^{-1}$. This means by knowing the preimage S' of S and having an efficient algorithm for writing h as a word in S, we can compute $\varphi^{-1}(h)$ by evaluating the same SLP $\mathfrak{S}$ in S'. Analogously, for $g \in G$ we can compute $\varphi(g)$ by writing g as a word in S'.

The word problem for classical groups in their natural representation, i.e. a), has been solved by Elliot Costi [30] and by the author in his Bachelor's and Master's thesis [83, 84]. The word problem for classical groups given by b) has been solved by Csaba Schneider and a publication is in progress.

1.1.11 Presentations

Since many of the algorithms for the composition tree are randomised, it is important to have tools to check whether results of randomised algorithms are correct. The verification process, i.e. the process of checking whether the output of a randomised algorithm is correct, for groups of the composition tree is often done using *presentations* which can be defined as follows.

Definition 1.22: [48]

1) Let G be a group and let $R \subseteq G$. Then $\langle R \rangle_G = \langle r^g \mid r \in R, g \in G \rangle$ is the normal subgroup of G generated by R.
2) Let X be a set, F a free group over X and $R \subseteq F$. Then let

$$G = \langle X \mid R \rangle := F / \langle R \rangle_F.$$

Then we say G is generated by X subject to the *defining relations* R. More generally, an arbitrary group G is called *finitely presented* if there are finite sets X and $R \subseteq F$ such that $G = \langle X \mid R \rangle$. In that case $\{X \mid R\}$ is a *finite presentation* for G.

Example 1.23

The cyclic group C_n of order n can be finitely represented as $C_n = \langle a \mid a^n \rangle$. Suppose two groups $G_1 = \langle (1,2,3) \rangle$ and $G_2 = \langle (1,2) \rangle$ are given and we want to use the presentation to check whether one of the given groups is isomorphic to a cyclic group of order 3 and given by a cyclic generator. The presentation $\langle a \mid a^n \rangle$ indicates that we require a generating set of size 1 as C_3 is only described by a. This is the case for G_1 and G_2 as G_1 is generated by $(1,2,3)$ and G_2 is generated by $(1,2)$. The second task is to verify the relations given by the presentation which is $a^3 = 1$ for the presentation of C_3. For G_1 we identify a with $(1,2,3)$ and compute $1 = a^3 = (1,2,3)^3 = ( )$ and, therefore, we know that there exists a unique epimorphism from C_3 to G_1, i.e. G_1 is isomorphic to a quotient of C_3. Moreover, G_1 is isomorphic to C_3 since G_1 and C_3 have size 3 and G_1 is generated by one cyclic generator. For G_2 we identify a with $(1,2)$ and compute $1 = a^3 = (1,2)^3 = (1,2)$ and, therefore, we know that G_2 is not a cyclic group of order 3 or that $(1,2)$ is not a cyclic generator of G_2. ◀

Now, if $\langle X \mid R \rangle$ is simple, then verifying presentations is enough to prove that groups are isomorphic as $\text{Kernel}(\varphi) = \langle 1 \rangle$ for any non-trivial epimorphism φ. Therefore, we can use a presentation $G = \langle X \mid R \rangle$ of a simple group to verify that a group H is isomorphic to G and that H is given with a specific generating set which satisfies the relations R.

In order to verify presentations of the leaf groups, we are not interested in any presentation but rather *short presentations*. Short means that the length of the presentation is in $\mathcal{O}(\log^2(|G|))$. Presentations for leaf groups are given by Babai et al. [6], Hulpke and Seress [50] and Suzuki [90] which can be

summarised in the next theorem.

Theorem 1.24: [76, Theorem 10.3]

For every finite simple group except ${}^2G_2(q)$ there is a known short presentation.

Another important result for presentations of permutation groups is given Conder, Leedham-Green and O'Brien in [17].

1.2 Motivation

The availability of efficient constructive recognition algorithms for the leaf groups of the composition tree has an enormous impact on the complexity of computations involving composition trees. Since classical groups appear frequently as leaf groups, constructive recognition of classical groups has a huge impact on the performance of the composition tree. Let G be a classical group and let H be the natural representation of G. As described in Section 1.1.8 the aim of constructive recognition is to compute an epimorphism

$$\varphi\colon G \to H$$

which allows efficient computations of images and preimages under φ. In this thesis we are dealing with classical groups $\langle X\rangle = G = \mathrm{CL}(d,q)$ in their natural representation and, hence, the problem of constructive recognition reduces to the task of computing a base change matrix $\mathscr{L} \in \mathrm{GL}(d,q)$ and writing a specific set $S \subset G$ as words in X such that $S^{\mathscr{L}}$ are standard generators of $G^{\mathscr{L}}$ as in Section 1.1.8.

The current state-of-the-art solution for randomised constructive recognition of classical groups in their natural representation by Leedham-Green and O'Brien in odd characteristic [59] and by Dietrich, Leedham-Green, Lübeck and O'Brien in even characteristic [32] relies on centralisers of involutions and it is difficult to analyse its complexity. Let ζ denote an upper bound on the number of field operations for computing a random element and let $\mathfrak{X}(q)$ denote an upper bound on the number of field operations for solving the discrete logarithm. In [59] it is shown that the complexity measured in field operations of their algorithm for constructive recognition of a classical group

CL(d,q) for q odd is at least

$$\mathcal{O}(d(\zeta + d^3\log(d) + d^2\log(d)\log(\log(d))\log(q) + \mathfrak{X}(q)))$$

if CL(d,q) is not an orthogonal group of minus type and

$$\mathcal{O}(d(\zeta + d^3\log(d) + d^2\log(d)\log(\log(d))\log(q) + \mathfrak{X}(q)) + \mathfrak{X}(q^2))$$

if CL(d,q) is an orthogonal group of minus type. Based on new publications for special linear groups [35] and for unitary groups [41] the complexity can be improved in these cases and is at least

$$\mathcal{O}(\log(d)(\zeta + d^3\log(d) + d^2\log(d)\log(\log(d))\log(q)) + d\mathfrak{X}(q)).$$

In this thesis we consider a new approach for a randomised constructive recognition algorithm for classical groups based on some preliminary ideas by Ákos Seress and Max Neunhöffer for the special linear group. As the efficiency of the composition tree relies heavily on efficient algorithms for leaf groups and classical groups appear repeatedly, the goal is to design an algorithm for constructive recognition of classical groups in their natural representation and prove that the complexity of the algorithm is even better than the assumed complexity of the current state-of-the-art algorithms in [32, 59].

1.3 Summary of the main results

Here, we describe the subalgorithms of the constructive algorithm of this thesis only for special linear groups $\langle X\rangle = \mathrm{SL}(d,q)$ and a detailed version for the other classical groups is given in Chapter 3. We start by specifying a computationally useful set S of generators for SL(d,q) in its natural representation, called *standard generators*. Their careful choice ensures that the number of required group operations during the algorithm is small. Subsequently, we express S as words in X.

The constructive recognition algorithm consists of different subalgorithms.

(1) GoingDown algorithm: The aim of this step is to find a subgroup $U \leq G$ with $U \cong \mathrm{SL}(2,q)$. In order to do this, a chain of subgroups is constructed

$$\mathrm{SL}(2,q) \cong U_k \leq U_{k-1} \cong \mathrm{SL}(d_{k-1},q) \leq \ldots \leq U_1 \cong \mathrm{SL}(d_1,q) \leq U_0 = G$$

where $d_i \leq 4\lceil \log(d_{i-1}) \rceil$. To descend from U_i to U_{i+1} the algorithm seeks a random duo of elements of U_i having a large common 1-eigenspace (called *stingray duos*) and takes U_{i+1} as the group generated by the duo. To analyse such an algorithm, we must determine the probability of finding a stingray duo and the probability that a stingray duo generates a group isomorphic to $\mathrm{SL}(d_{i+1},q)$. Significant results for these questions have already been published [39, 42, 75]. Note that this GoingDown algorithm takes significantly fewer than $\log(d)$ steps to reach a base case group.

(2) BaseCase algorithm: An efficient algorithm for constructive recognition of $\mathrm{SL}(2,q)$ is given by Conder, Leedham-Green and O'Brien in [27] with outstanding performance. This algorithm computes standard generators for a group isomorphic to $\mathrm{SL}(2,q)$.

(3) GoingUp algorithm: Once the standard generators for $U \cong \mathrm{SL}(2,q)$ have been found, they must be extended to a standard generating set for the original group G. For this another basic step is applied. Let $H_1 \leq G$ with $H_1 \cong \mathrm{SL}(n,q)$ for $n < d$ and assume that standard generators of H_1 are known. The basic step computes standard generators of a subgroup $H_2 \leq G$ with $H_2 \cong \mathrm{SL}(\min\{2n-1,d\},q)$. The algorithm we use for this step is original and has not been published previously.

The standard generators of G are expressed as words in X by MSLPs throughout the algorithm. Since some words in the standard generators are not evaluated directly during the constructive recognition algorithms, it is important to perform a precise complexity analysis of the resulting MSLPs and prove that their length is bounded by $\mathcal{O}(d\log(q))$.

The main result of this thesis is given in the following theorem.

Theorem 1.25

Let $\langle X \rangle = G \in \{\mathrm{SL}(d,q), \mathrm{Sp}(d,q), \mathrm{SU}(d,q), \Omega(d,q)\}$, except $\mathrm{Sp}(d,q)$ for q and d even, be a classical group in its natural representation and $\epsilon \in (0,1)$. There is a one-sided Monte Carlo algorithm which, given input $G = \langle X \rangle$ and ϵ, outputs with probability at least $1-\epsilon$ an MSLP $\mathfrak{S}$ and base change matrix $\mathscr{L}$ such that $\mathfrak{S}$ evaluates from $X^{\mathscr{L}}$ to the standard generators of $G^{\mathscr{L}}$.

Theorem 1.26

Suppose Conjecture 10.33 is true. The complexity of the algorithm stated in Theorem 1.25 is

$$\mathcal{O}(d^3\log(d)+d^2\log(d)\log(\log(d))\log(q)+\frac{\log(d)}{\log(\log(d))}\zeta+\mathfrak{Y}(q)+\mathfrak{Z}(q))$$

where ζ denotes an upper bound on the number of field operations required for computing a random element, $\mathfrak{Y}(q)$ denotes an upper bound on the number of field operations for constructively recognising a base case group as in Definition 3.2 and $\mathfrak{Z}(q)$ denotes an upper bound on the required number of field operations for the final step as in Remark 3.7. For a unitary or orthogonal group the complexity of Algorithm StandardGenerators [Alg. 3] as stated in Theorem 3.10 is

$$\mathcal{O}(d^3\log(d)+d^2\log(d)\log(\log(d))\log(q)+\frac{\log(d)}{\log(\log(d))}\zeta+\mathfrak{Y}(q)+\mathfrak{Z}(q)+\log(d)\mathfrak{V}(q))$$

where $\mathfrak{V}(q)$ is an upper bound on the number of field operations for computing a square root.

All algorithms of this thesis have been implemented in GAP [37] and are available at [82].

1.4 Outline of thesis

This thesis is structured as follows: In Chapter 2 we introduce many notations used for the remainder of this thesis and summarise well-known preliminary results about classical groups, representation theory, complexity theory and MSLPs. In the section about complexity we also restate some important complexity results of algorithms used in this thesis.

In Chapter 3 we describe the fundamental ideas of the constructive recognition algorithm of this thesis for all classical groups except $\mathrm{Sp}(d,q)$ for q and d even. The chapter is divided into four sections where Section 3.1 deals with the GoingDown algorithm, Section 3.2 with the BaseCase algorithm and Section 3.3 with the GoingUp algorithm as outlined in Section 1.3. In Section 3.4 the GoingDown, BaseCase and GoingUp algorithm are combined into one single algorithm StandardGenerators which is the constructive recognition algorithm of this thesis.

In Chapter 4 important elements for the GoingDown algorithm are introduced. Moreover, we

design algorithms for computing theses elements and summarise important results regarding their proportion.

In Chapter 5 the constructive recognition algorithm is discussed in detail for special linear groups. In this chapter we also state some small interesting variants of the presented algorithms. Moreover, using a running example the functionality of the algorithms is illustrated. The constructive recognition algorithm is discussed for symplectic groups in odd characteristic in Chapter 6, for unitary groups in Chapter 7 and for orthogonal groups in Chapter 8. We mostly focus on the differences between the special linear group and the other classical groups in these chapters and do not restate every detail.

In Chapter 9 an alternative GoingUp algorithm is presented for all classical groups in odd characteristic. This GoingUp algorithm is based on involutions and the output MSLPs are shorter than the output MSLPs of the GoingUp algorithms presented in Chapter 5 to Chapter 8. Even though we do not perform a full complexity analysis in this thesis, we suspect that the shorter MSLPs of the GoingUp algorithm in Chapter 9 come at the price of a worse runtime complexity.

In Chapter 10 complexity results about the algorithms of this thesis are proven. We prove the complexity of the GoingDown algorithm and discuss complexity results of the base case algorithms from the literature. For the GoingUp algorithm we propose a conjecture based on practical results and prove further complexity results based on this conjecture.

The algorithms of this thesis have been implemented by the author and are available at [82]. In Chapter 11 we discuss details of this implementation and present run-time comparisons of the constructive recognition of this thesis and the state-of-the-art algorithm [32, 59].

Lastly, further interesting projects based on the results of this thesis are presented in Chapter 12.

Chapter 2

Preliminaries

This chapter introduces basic concepts and defines notation. The first section focuses on well-known fundamentals of algebra, especially linear algebra. In the second section we define the classical groups over finite fields with bilinear forms. For this we first treat bilinear forms and their properties. The third section deals with particular elements in the classical groups, namely *transvections* and *Siegel transformations*. They play a major role in the generation of classical groups, solving the word problem and the algorithms of this thesis. The fourth section introduces basics of representation theory which are used to define the *natural representation* of classical groups. The last two sections focus less on mathematical concepts and rather on technical implementation. In the fifth section concepts of complexity analysis, e.g. *Landau symbols*, are defined and well-known important algorithms and their complexity results are summarised. The last section explains an efficient approach to encode words in terms of generators. This method is used in this thesis and results about properties, e.g. the length of words, are proven in Chapter 10.

All concepts discussed in this chapter are well-known and can be found in sources such as [48, 52, 68, 91]. The treatment presented here is based on these references.

2.1 Basics and notation

We start this section by introducing notations for well-known concepts.

Definition 2.1

Let V be a d-dimensional $\mathbb{F}$-vector space.

1) $\mathrm{End}(V) := \{\varphi \mid \varphi\colon V \to V,\ \varphi \text{ is a vector space homomorphism}\}$,
2) $\mathrm{GL}(V) := \{\varphi \mid \varphi \in \mathrm{End}(V) \text{ and bijective}\}$,
3) $\mathrm{SL}(V) := \{\varphi \mid \varphi \in \mathrm{GL}(V) \text{ and } \det(\varphi) = 1\}$,
4) $\mathrm{GL}(d,q) := \{a \in \mathbb{F}_q^{d\times d} \mid a \text{ is invertible}\}$,
5) $\mathrm{SL}(d,q) := \{a \in \mathbb{F}_q^{d\times d} \mid a \text{ is invertible and } \det(a) = 1\}$.

Remark 2.2

1) Notice that $\mathrm{GL}(d,q)$ is well-defined since all finite fields of order $q = p^f$ are isomorphic.
2) $\mathrm{GL}(V)$ and $\mathrm{SL}(V)$ are groups with respect to the composition of maps. $\mathrm{GL}(d,q)$ and $\mathrm{SL}(d,q)$ are groups with respect to matrix multiplication.
3) Note that in this thesis all groups act from the right which means that we use row vectors $v \in \mathbb{F}_q^d$ and thus have $vg \in \mathbb{F}_q^d$ for $g \in \mathrm{GL}(d,q)$. Representing elements of $\mathbb{F}_q^d$ as row vectors and acting from the right aligns with the convention commonly used in software programs e.g. `Magma` [16] and `GAP` [37]. ◀

There is an important well-known connection between these groups.

Theorem 2.3

Let V be a d-dimensional $\mathbb{F}$-vector space. Let $\mathscr{B} \in V^d$ be a basis of V. The map

$$\mathrm{End}(V) \to \mathbb{F}^{d\times d},\ \varphi \mapsto {}^{\mathscr{B}}\varphi^{\mathscr{B}}$$

is an algebra isomorphism where ${}^{\mathscr{B}}\varphi^{\mathscr{B}}$ is the matrix representation of φ with respect to the basis $\mathscr{B}$.

Corollary 2.4

Let V be a d-dimensional $\mathbb{F}$-vector space. Let $\mathscr{B} \in V^d$ be a basis of V. Then

$$\mathrm{GL}(V) \to \mathrm{GL}(d,q),\ \varphi \mapsto {}^{\mathscr{B}}\varphi^{\mathscr{B}}$$

is an isomorphism.

We continue this section by fixing some basic notations for the remainder of this thesis.

- Groups are denoted by G, H, U where U is generally a subgroup of G or H. Correspondingly,

we use g, h, u as group elements and in general $g \in G$, $h \in H$ and $u \in U$ but occasionally also $g, h, u \in G$.

- The set of all natural numbers is denoted by $\mathbb{N} = \{1, 2, 3, \dots\}$, the set of all primes by $\mathscr{P}$ and variables i, j, k, r denote natural numbers.
- $\mathbb{F}$ (or $\mathbb{F}_q$) is a finite field of characteristic $p \in \mathscr{P}$ and size $q = p^f$ for $f \in \mathbb{N}$. We will frequently assume $\mathrm{char}(\mathbb{F}) \neq 2$. We denote by $\mathbb{F}_p$ the sub-field of size p and by $(\omega_1, \dots, \omega_f)$ a basis of $\mathbb{F}_q$ as an $\mathbb{F}_p$-vector space. If $\mathbb{F}$ admits a unique automorphism of order 2, then it is denoted by $\overline{\ }: \mathbb{F} \to \mathbb{F},\ \lambda \mapsto \overline{\lambda}$. We use $\imath, \jmath, \lambda$ to denote elements of $\mathbb{F}$. The algebraic closure of $\mathbb{F}$ is denoted by $\overline{\mathbb{F}}$.
- Both V and W are finite dimensional $\mathbb{F}$-vector spaces and often $W \leq V$. In general, V denotes a d-dimensional vector space. A vector space homomorphism is denoted by φ and $\varphi|_W$ is the restriction of φ on $W \leq V$. We use $v, w, \nu, b, e \in V$ to denote vectors, while $\mathscr{B} = (b_1, \dots, b_d)$ denotes a basis of V and $(e_1, \dots, e_d)$ is the standard basis of $\mathbb{F}_q^d$. A base change matrix is denoted by $\mathscr{L} \in \mathrm{GL}(d, q)$ and $\mathscr{B}^{\mathscr{L}} = \mathscr{B}'$ for bases $\mathscr{B}$ and $\mathscr{B}'$ of $\mathbb{F}_q^d$. Moreover, $V^* = \{\vartheta : V \to \mathbb{F} \mid \vartheta \text{ is a homomorphism}\}$ is the dual space of V.
- By $\mathrm{SX}(d, q)$ we denote the special linear group $\mathrm{SL}(d, q)$, the symplectic group $\mathrm{Sp}(d, q)$, the special unitary group $\mathrm{SU}(d, q)$ or the special orthogonal group $\mathrm{SO}(d, q)$. We use $\mathrm{CL}(d, q)$ for a classical group, i.e. $\mathrm{SL}(d, q)$, $\mathrm{Sp}(d, q)$, $\mathrm{SU}(d, q)$ or the omega group $\Omega(d, q)$. In general, $G = \mathrm{CL}(d, q)$ and $U \leq G$ with $U \cong \mathrm{CL}(n, q)$ for $n < d$. The transpose of $a \in \mathrm{GL}(d, q)$ is denoted by a^{Tr} and the identity matrix by $I_d \in \mathrm{GL}(d, q)$. Matrices are denoted by $a, c \in \mathrm{GL}(d, q)$ where c is used as a conjugating element in many cases. Finite generating sets of a matrix group G are usually denoted by X, Y, so $\langle X \rangle = G \leq \mathrm{GL}(d, q)$. Let $I_{i,j}$ be the matrix which satisfies in the basis $\mathscr{B}$ that $(I_{i,j})_{ij} = 1$ and all other entries of $I_{i,j}$ are equal to 0. For $i, j \in \{1, \dots, d\}$ with $i \neq j$ and $\lambda \in \mathbb{F}$ let $E_{i,j}(\lambda) \in \mathrm{GL}(V)$ be the group element $I_d + I_{i,j}(\lambda)$ with respect to the basis $\mathscr{B}$. If a matrix group G is given with respect to a basis $\mathscr{B}$ and $\mathscr{L}$ is a base change matrix, then $G^{\mathscr{L}}$ denotes G with respect to the basis $\mathscr{B}^{\mathscr{L}}$.
- x denotes an indeterminate, $P \in \mathbb{F}[x]$ a polynomial and χ_a the characteristic polynomial of $a \in \mathrm{GL}(d, q)$.

Typically, G is a subgroup of $\mathrm{GL}(d, q)$. In the following we assume that a finite generating set

$X \subseteq \mathrm{GL}(d,q)$ of G is known, i.e. $G = \langle X \rangle$, and that $V = \mathbb{F}_q^d$. Clearly G acts on V by right multiplication.

The following lemma is well-known as a dimension formula for subspaces of vector spaces.

Lemma 2.5

Let W_1, W_2 be two subspaces of a finite-dimensional vector space V. Then

$$\dim(W_1) + \dim(W_2) = \dim(W_1 + W_2) + \dim(W_1 \cap W_2).$$

Proof. [91, p. 13] □

Definition 2.6

Let $G \leq \mathrm{GL}(d,q)$ and $V = \mathbb{F}_q^d$. For $g \in G$ let $\mathrm{Fix}(g) := \{v \in V \mid v^g = v\}$ denote its fixed space and for a subgroup U of G we write $\mathrm{Fix}(U)$ for the subspace of vectors which are fixed by all elements of U, i.e. $\mathrm{Fix}(U) := \{v \in V \mid v^u = v \text{ for all } u \in U\}$.

Definition 2.7

Let $(b_1, \ldots, b_d)$ be a basis of V and $1 \leq n \leq d$. Then we define

1) $V_n := \langle b_1, b_2, \ldots, b_n \rangle$ and
2) $F_{d-n} := \langle b_{n+1}, \ldots, b_d \rangle$.

2.2 Forms and classical groups

This section introduces basic definitions and notations that are important in all subsequent chapters. The main objects we consider in this thesis are the finite classical groups which can be defined using forms on vector spaces. The definitions and results of this chapter are well-known, for example in [45, 91], and the treatment presented here is based on these references.

2.2.1 Forms

First, we give some definitions that are needed later to define and describe the classical groups. For the remainder of this chapter, $\mathbb{F}$ denotes a finite field.

Definition 2.8

Let V be an $\mathbb{F}$-vector space.

1) A map $\Phi\colon V \times V \to \mathbb{F}$, $(v,w) \mapsto \Phi(v,w)$ is a *bilinear form* on V if Φ is linear in each component, that is,
   (i) $\Phi(\imath v_1 + \jmath v_2, w) = \imath \cdot \Phi(v_1,w) + \jmath \cdot \Phi(v_2,w)$ for all $\imath,\jmath \in \mathbb{F}$ and $v_1,v_2,w \in V$ and
   (ii) $\Phi(v, \imath w_1 + \jmath w_2) = \imath \cdot \Phi(v,w_1) + \jmath \cdot \Phi(v,w_2)$ for all $\imath,\jmath \in \mathbb{F}$ and $v,w_1,w_2 \in V$.
2) A map $Q\colon V \to \mathbb{F}$ is a *quadratic form* if there is an associated bilinear form Φ_Q, called the *polar form* of Q, such that
   (i) $Q(\lambda v) = \lambda^2 Q(v)$ for all $\lambda \in \mathbb{F}, v \in V$ and
   (ii) $\Phi_Q(v,w) = Q(v+w) - Q(v) - Q(w)$ for all $v,w \in V$.

Remark 2.9

From now on, we refer to a bilinear form or a quadratic form only as a *form*. ◀

We consider some connections between quadratic forms and the corresponding polar form.

Remark 2.10

Let Q be a quadratic form on an $\mathbb{F}$-vector space V and Φ_Q the polar form of Q.

1) Φ_Q is uniquely determined by condition (ii) of Definition 2.8. Moreover, Φ_Q is symmetric since for all $v,w \in V$ we have
$$\Phi_Q(v,w) = Q(v+w) - Q(v) - Q(w) = Q(w+v) - Q(w) - Q(v) = \Phi_Q(w,v).$$
2) Let $\mathrm{char}(\mathbb{F}) \neq 2$ and let Φ be a symmetric form on V. Define $Q_\Phi\colon V \to \mathbb{F}$, $v \mapsto 2^{-1}\Phi(v,v)$. We verify that Q_Φ is a quadratic form with polar form Φ.
   (i) $Q_\Phi(\lambda v) = 2^{-1}\Phi(\lambda v, \lambda v) = \lambda^2 2^{-1}\Phi(v,v) = \lambda^2 Q_\Phi(v)$ for all $\lambda \in \mathbb{F}, v \in V$.
   (ii) Let $v,w \in V$, then

$$\begin{aligned} & Q_\Phi(v+w) - Q_\Phi(v) - Q_\Phi(w) \\ &= 2^{-1}(\Phi(v+w,v+w) - \Phi(v,v) - \Phi(w,w)) \\ &= 2^{-1}(\Phi(v,v) + \Phi(w,w) + 2\Phi(v,w) - \Phi(v,v) - \Phi(w,w)) \\ &= 2^{-1}2\Phi(v,w) = \Phi(v,w). \end{aligned}$$

Hence, Q_Φ defines a quadratic form.

Note that division by 2 occurs in the definition of Q_Φ and, therefore, Q_Φ is undefined in characteristic 2. Hence, we cannot recover Q from Φ_Q this way in characteristic 2. ◀

To define orthogonal groups in arbitrary characteristic one needs quadratic forms. But due to Remark 2.10, we can instead use bilinear forms in odd characteristic, which is what we do in Section 2.2.2. However, defining classical groups requires forms with additional properties.

Definition 2.11

Let V be an $\mathbb{F}$-vector space equipped with a bilinear form Φ or a quadratic form Q.

1) Φ is *non-singular* or *non-degenerate* if for each $v \in V\backslash\{0\}$ there is a $w \in V$ with $\Phi(v,w) \neq 0$. Otherwise, it is *degenerate*.

   Q is *non-degenerate* if the polar form of Q is non-degenerate. Otherwise, Q is *degenerate*.

2) Φ is *symmetric* if $\Phi(v,w) = \Phi(w,v)$ for all $v,w \in V$.

3) Φ is *skew-symmetric* if $\Phi(v,w) = -\Phi(w,v)$ for all $v,w \in V$.

4) Φ is *alternating* if $\Phi(v,v) = 0$ for all $v \in V$.

5) A non-singular and alternating bilinear form is *symplectic*.

6) Let $\mathbb{F}$ be a field admitting a field automorphism $\bar{}$ of order 2.

   A map $\Phi\colon V \times V \to \mathbb{F}$, $(v,w) \mapsto \Phi(v,w)$ is a *Hermitian form* on V with respect to $\bar{}$ if

   (i) $\Phi(\iota v_1 + \jmath v_2, w) = \iota \cdot \Phi(v_1,w) + \jmath \cdot \Phi(v_2,w)$ for all $\iota,\jmath \in \mathbb{F}$ and $v_1,v_2,w \in V$ and

   (ii) $\Phi(v,w) = \overline{\Phi(w,v)}$ for all $v,w \in V$.

   Analogous to 1) a Hermitian form Φ is non-singular if for each $v \in V\backslash\{0\}$ there is a $w \in V$ with $\Phi(v,w) \neq 0$. A non-singular and Hermitian form is a *unitary form*.

7) If $\mathrm{char}(\mathbb{F}) \neq 2$, then a non-singular and symmetric bilinear form is *orthogonal*.

Remark 2.12

1) An alternating form is skew-symmetric.

2) If $\mathrm{char}(\mathbb{F})$ is odd or zero, then a skew-symmetric form is also alternating. This is not generally true if $\mathrm{char}(\mathbb{F})$ is even. ◀

To work with classical groups, we need to introduce a few additional definitions that extend those in

Definition 2.11.

Definition 2.13

Let V be an $\mathbb{F}$-vector space equipped with a symmetric, skew-symmetric or unitary form Φ or equipped with a polar form Φ of a quadratic form Q on V.

1) $v, w \in V$ are *orthogonal* with respect to Φ if $\Phi(v,w) = 0$. We denote this case by $v \perp w$.
2) Let $A \subseteq V$. Then
$$A^\perp := \{v \in V \mid \Phi(v,w) = 0 \text{ for all } w \in A\}$$
is the *orthogonal complement of* A. We call $\mathrm{rad}(\Phi) = V^\perp$ the *radical of* V and for a quadratic form $\mathrm{rad}(Q) = \{v \in \mathrm{rad}(\Phi) \mid Q(v) = 0\}$ the *radical of* Q.
3) $v \in V\backslash\{0\}$ is *isotropic* if $\Phi(v,v) = 0$.
4) $v \in V\backslash\{0\}$ is *singular* if $Q(v) = 0$.
5) $W \leq V$ is *non-singular* if $\Phi|_W$ is non-singular.
6) $W \leq V$ is *totally isotropic* if $\Phi|_W = 0$.
7) $W \leq V$ is *totally singular* if $Q(w) = 0$ for all $w \in W$.
8) $W \leq V$ is a *maximally totally isotropic* (respectively *maximally totally singular*) subspace of V if W is totally isotropic (totally singular) and there is no proper totally isotropic (totally singular) subspace W' of V containing W.
9) $W \leq V$ is *anisotropic* if $\Phi(w,w) \neq 0$ for all $w \in W\backslash\{0\}$.
10) $W \leq V$ is *non-degenerate* if $W \cap W^\perp = \{0\}$.

Lemma 2.14

Let V be a finite-dimensional $\mathbb{F}$-vector space and let Φ be a non-singular symmetric, skew-symmetric or unitary form on V. Let $W \leq V$.

1) $(W^\perp)^\perp = W$.
2) $\dim(W) + \dim(W^\perp) = \dim(V)$.
3) If $W \cap W^\perp = \{0\}$, then $V = W \oplus W^\perp$.

Remark 2.15

Let $W_1, W_2 \leq V$. If $W_1 \oplus W_2 = V$ and $\Phi(w_1, w_2) = 0$ for all $w_1 \in W_1, w_2 \in W_2$, then we write $W_1 \perp W_2$. ◀

2.2.2 Classical groups

In this chapter classical groups are defined based on forms. There are special endomorphisms on an $\mathbb{F}$-vector space equipped with a bilinear or quadratic form, namely isometries, which are used to define the classical groups.

Definition 2.16

Let V be an $\mathbb{F}$-vector space equipped with a form Φ or equipped with a quadratic form Q. A map $\varphi \in \mathrm{End}(V)$ is an *isometry* from V to V if $\Phi(v^\varphi, w^\varphi) = \Phi(v,w)$ for all $v,w \in V$ or $Q(v^\varphi) = Q(v)$ for all $v \in V$.

At this point it is possible to define the classical groups.

Definition 2.17

Let V be a d-dimensional $\mathbb{F}_q$-vector space equipped with a non-singular form Φ or non-degenerate quadratic form Q.

1) If Φ is symplectic, then the group of all bijective isometries of (V,Φ) is the *symplectic group* and denoted by $\mathrm{Sp}(V,\Phi)$.
2) If Φ is unitary, then the group of all bijective isometries of (V,Φ) is the *unitary group* and denoted by $\mathrm{U}(V,\Phi)$. The subgroup $\mathrm{U}(V,\Phi) \cap \mathrm{SL}(V)$ is the *special unitary group* and denoted by $\mathrm{SU}(V,\Phi)$.
3) The group of all bijective isometries of (V,Q) is the *orthogonal group* and denoted by $O(V,\Phi)$. The subgroup $\mathrm{O}(V,\Phi) \cap \mathrm{SL}(V)$ is the *special orthogonal group*, denoted by $\mathrm{SO}(V,\Phi)$. The derived subgroup of $\mathrm{SO}(V,\Phi)$ is denoted by $\Omega(V,\Phi) := \mathrm{SO}(V,\Phi)'$.

If it is clear from the context which form is intended, then we use $\mathrm{Sp}(V)$, $\mathrm{U}(V)$, $\mathrm{SU}(V)$, $\mathrm{O}(V)$, $\mathrm{SO}(V)$ and $\Omega(V)$ instead of $\mathrm{Sp}(V,\Phi)$, $\mathrm{U}(V,\Phi)$, $\mathrm{SU}(V,\Phi)$, $\mathrm{O}(V,\Phi)$, $\mathrm{SO}(V,\Phi)$ and $\Omega(V,\Phi)$.

Remark 2.18

1) For symplectic groups it is shown that $\mathrm{Sp}(V) \subseteq \mathrm{SL}(V)$ and, thus, $\mathrm{Sp}(V) = \mathrm{Sp}(V) \cap \mathrm{SL}(V)$ in [91, Corollary 8.6].
2) Note that if $\mathrm{char}(\mathbb{F}) \neq 2$, then the orthogonal group is the group of isometries of the non-singular symmetric bilinear form Φ_Q. Therefore, we can also consider an orthogonal form, see Remark 2.10.

3) Note that the symplectic, unitary and orthogonal groups are independent of the chosen bilinear or quadratic form [91, p. 138f.].
4) In the following $\mathrm{CL}(V)$ denotes one of the classical groups $\mathrm{SL}(V)$, $\mathrm{Sp}(V)$, $\mathrm{SU}(V)$ or $\Omega(V)$. There are three types of orthogonal groups which are defined in more detail in this chapter. Nevertheless, if a result is independent of the type of the underlying orthogonal form, then $\Omega(V)$ and $\mathrm{CL}(V)$ are used. ◀

We consider three cases for the orthogonal group with the Witt index being an important distinguishing feature.

Definition 2.19

Let V be an $\mathbb{F}$-vector space with a non-degenerate symplectic or unitary form Φ or a non-degenerate quadratic form Q in which case Φ denotes the polar form of Q. A pair of vectors $(v,w) \in V^2$ is a *hyperbolic pair* if $\dim(\langle v,w\rangle) = 2$, $\Phi(v,w) = 1$ and $\Phi(v,v) = \Phi(w,w) = 0$. The subspace $\langle v,w\rangle$ spanned by a hyperbolic pair (v,w) is a *hyperbolic plane*.

Definition 2.20

Let V be an $\mathbb{F}$-vector space with a non-singular symplectic or unitary form Φ or a quadratic form Q, in which case Φ is the polar form of Q. The dimension of a maximally totally isotropic subspace of V or, in case of a quadratic form, of a maximally totally singular subspace of V, is the *Witt index* of (V,Φ) or (V,Q).

Theorem 2.21

Let Φ be a symplectic bilinear form on an $\mathbb{F}$-vector space V with $\dim(V) \geq 2$. Then V admits a basis $\mathscr{B} = (e_1,\ldots,e_m,f_m,\ldots,f_1)$ where $\langle e_i,f_i\rangle$ are hyperbolic planes for all $i \in \{1,\ldots,m\}$ such that

$$V = \langle e_1,f_1\rangle \perp \ldots \perp \langle e_m,f_m\rangle$$

and V has Witt index m.

Proof. [91, p. 69]. □

Theorem 2.22

Let Φ be a unitary form on an $\mathbb{F}$-vector space V with $\dim(V) \geq 2$. There exist vectors $e_1, \ldots, e_m, f_1, \ldots, f_m \in V$ such that

$$V = \langle e_1, f_1 \rangle \perp \ldots \perp \langle e_m, f_m \rangle \perp V_0$$

where $\langle e_i, f_i \rangle$ are hyperbolic planes for all $1 \leq i \leq m$ and $V_0 \leq V$ is anisotropic, such that V has Witt index m. If V has Witt index m, then one of the following two cases holds:

1) $\dim(V_0) = 0$ and $\dim(V) = 2m$ or
2) $\dim(V_0) = 1$ and $\dim(V) = 2m + 1$. For $V_0 = \langle w \rangle$ we can assume $\Phi(w, w) = 1$.

Proof. [91, p. 116]. □

Remark 2.23

Even though Theorem 2.22 proves that there are two different cases for unitary groups, the type of a unitary group is uniquely determined by the dimension of V. If the dimension is even, then only case 1) is possible and if the dimension is odd, then only case 2) is possible. Therefore, we do not need to distinguish them by different notations. Nevertheless, in Chapter 7 we have to distinguish between both cases from an algorithmic point-of-view. ◀

Theorem 2.24

Let $\mathrm{char}(\mathbb{F})$ be odd and let Φ be an orthogonal form on an $\mathbb{F}$-vector space V. Then there exist vectors $e_1, \ldots, e_m, f_1, \ldots, f_m \in V$ such that

$$V = \langle e_1, f_1 \rangle \perp \ldots \perp \langle e_m, f_m \rangle \perp V_0$$

where $\langle e_i, f_i \rangle$ is a hyperbolic plane and $V_0 \leq V$ is anisotropic, such that V has Witt index m and exactly one of the following holds:

1) $\dim(V_0) = 0$ and $\dim(V) = 2m$.
2) $\dim(V_0) = 1$ and $\dim(V) = 2m + 1$. For $V_0 = \langle w \rangle$ we can assume that $\Phi(w, w) = -2^{-1}$.
3) $\dim(V_0) = 2$ and $\dim(V) = 2m + 2$. For $V_0 = \langle w_1, w_2 \rangle$ we can assume that $\Phi(w_1, w_1) = -2$,

$\Phi(w_1, w_2) = 0$ and $\Phi(w_2, w_2) = 2\omega$ where ω is a primitive element of $\mathbb{F}$.

Proof. [91, pp. 138–139]. □

We introduce names for the different cases.

Definition 2.25

Let Φ be an orthogonal form on an $\mathbb{F}$-vector space V. Let

$$V = \langle v_1, w_1 \rangle \perp \ldots \perp \langle v_m, w_m \rangle \perp V_0$$

as in Theorem 2.24. We define the following types of orthogonal groups:

1) If $\dim(V_0) = 0$, then we denote the group of all bijective isometries of (V, Φ) by $\mathrm{O}^+(V, \Phi)$ and say that the group is an *orthogonal group of plus type* and the form Φ is *hyperbolic*. Moreover, we denote $S(V, \mathbb{F}, \Phi)$ by $\mathrm{SO}^+(V, \Phi)$ and say that the group is a *special orthogonal group of plus type*. With respect to the basis introduced in Theorem 2.24, the group $\mathrm{O}^+(V, \Phi)$ is denoted by $\mathrm{O}^+(2m, q)$ and $\mathrm{SO}^+(V, \Phi)$ by $\mathrm{SO}^+(2m, q)$.
2) If $\dim(V_0) = 1$, then we denote the group of all bijective isometries of (V, Φ) by $\mathrm{O}^\circ(V, \Phi)$ and say that the group is an *orthogonal group of circle type* and the form Φ is *parabolic*. Moreover, we denote $S(V, \mathbb{F}, \Phi)$ by $\mathrm{SO}^\circ(V, \Phi)$ and say that the group is a *special orthogonal group of circle type*. With respect to the basis introduced in Theorem 2.24, the group $\mathrm{O}^\circ(V, \Phi)$ is denoted by $\mathrm{O}^\circ(2m+1, q)$ and $\mathrm{SO}^\circ(V, \Phi)$ by $\mathrm{SO}^\circ(2m+1, q)$.
3) If $\dim(V_0) = 2$, then we denote the group of all bijective isometries of (V, Φ) by $\mathrm{O}^-(V, \Phi)$ and say that the group is an *orthogonal group of minus type* and the form Φ is *elliptic*. Moreover, we denote $S(V, \mathbb{F}, \Phi)$ by $\mathrm{SO}^-(V, \Phi)$ and say that the group is a *special orthogonal group of minus type*. With respect to the basis introduced in Theorem 2.24, the group $\mathrm{O}^-(V, \Phi)$ is denoted by $\mathrm{O}^-(2m+2, q)$ and $\mathrm{SO}^-(V, \Phi)$ by $\mathrm{SO}^-(2m+2, q)$.

Remark 2.26

Based on Definition 2.17 and 2.25 we define $\Omega^+(d, q) := \mathrm{SO}^+(d, q)'$, $\Omega^\circ(d, q) := \mathrm{SO}^\circ(d, q)'$ and $\Omega^-(d, q) := \mathrm{SO}^-(d, q)'$. ◀

Definition 2.27

Let V be a d-dimensional $\mathbb{F}_q$-vector space equipped with a non-singular form Φ or non-degenerate quadratic form Q with polar form Φ. Moreover, let $\mathcal{B} = (b_1, \ldots, b_d)$ be a basis of V. Then

$$b := (\Phi(b_i, b_j))_{i,j=1,\ldots,d}$$

is the *Gram-matrix* of Φ with respect to $\mathcal{B}$.

Remark 2.28

The classical groups except orthogonal groups in characteristic 2 can be defined in terms of a Gram-matrix b of the form as in Definition 2.17 with respect to the corresponding basis of Theorem 2.21, 2.22 and 2.24. These groups can be written as $\mathrm{SX}(d,q) = \{a \in \mathrm{SL}(d,q) \mid aba^{\mathrm{Tr}} = b\}$ if $\mathrm{X} \in \{\mathrm{p}, \mathrm{O}^+, \mathrm{O}^\circ, \mathrm{O}^-\}$ and $\mathrm{SU}(d,q) = \{a \in \mathrm{SL}(d,q^2) \mid aba^* = b\}$ where $a^* = \overline{(a^{\mathrm{Tr}})} = (\overline{a})^{\mathrm{Tr}}$ and

- if $\mathrm{X} = \mathrm{p}$, then

$$b := \left(\begin{array}{ccc|ccc} & & & & & 1 \\ & & & & \iddots & \\ & & & 1 & & \\ \hline & & & & & \\ & & -1 & & & \\ & \iddots & & & & \\ -1 & & & & & \end{array}\right)$$

  with respect to a basis $(e_1, \ldots, e_m, f_m, \ldots, f_1)$ as in Theorem 2.21.

- if $\mathrm{X} = \mathrm{U}$, then

$$b := \left(\begin{array}{ccc|ccc} & & & & & 1 \\ & & & & \iddots & \\ & & & 1 & & \\ \hline & & & & & \\ & & [1] & & & \\ \hline & & 1 & & & \\ & \iddots & & & & \\ 1 & & & & & \end{array}\right)$$

  with respect to a basis $(e_1, \ldots, e_m, [w,]f_m, \ldots, f_1)$ as in Theorem 2.22 where $[1]$ and $[w,]$ are omitted if d is even.

- if $\mathrm{X} = \mathrm{O}^+$ or $\mathrm{X} = \mathrm{O}^\circ$, then

$$b := \left(\begin{array}{c|c|c} & & \begin{smallmatrix} & & 1 \\ & \iddots & \\ 1 & & \end{smallmatrix} \\ \hline & [-2^{-1}] & \\ \hline \begin{smallmatrix} & & 1 \\ & \iddots & \\ 1 & & \end{smallmatrix} & & \end{array}\right)$$

with respect to a basis $(e_1, \ldots, e_m, [w,]f_m, \ldots, f_1)$ as in Theorem 2.24 where $[-2^{-1}]$ and $[w,]$ are omitted if d is even.

- if $\mathrm{X} = \mathrm{O}^-$, then

$$b := \left(\begin{array}{c|c|c} & & \begin{smallmatrix} & & 1 \\ & \iddots & \\ 1 & & \end{smallmatrix} \\ \hline & \begin{smallmatrix} -2 & 0 \\ 0 & 2\omega \end{smallmatrix} & \\ \hline \begin{smallmatrix} & & 1 \\ & \iddots & \\ 1 & & \end{smallmatrix} & & \end{array}\right)$$

with respect to a basis $(e_1, \ldots, e_m, w_1, w_2, f_m, \ldots, f_1)$ as in Theorem 2.24 where ω is a primitive element of $\mathbb{F}_q$. ◀

2.3 Transvections and Siegel transformations

Transvections and Siegel transformations are certain elements in classical groups which play a major role in solving different problems algorithmically. These elements have a simple structure in the natural representation, so operations with them are easy to follow. Moreover, it is well known that these elements generate classical groups. Here we observe why transvections are used in SL, Sp and SU and Siegel transformations in SO. The definitions and results of this chapter are also well-known and can be found in [45, 91].

2.3.1 Transvections

We now define transvections in a matrix group G. In classical groups transvections play a role similar to the role played by elementary matrices in Gaussian elimination. Moreover, some of these elements occur among the *standard generators* defined in this thesis and play an important role in the GoingUp algorithm.

Definition 2.29

Let V be a d-dimensional $\mathbb{F}$-vector space. Let $W \leq V$ such that $\dim(W) = d-1$. A non-identity matrix $T \in \mathrm{GL}(V)$ is a *transvection* with respect to the *hyperplane* W if

1) $w^T = w$ for all $w \in W$ and
2) $v^T - v \in W$ for all $v \in V$.

Transvections play an important role as they generate $\mathrm{SL}(d,q)$, $\mathrm{SU}(d,q)$ and $\mathrm{Sp}(d,q)$, see Lemma 2.35. Since matrices for elementary row and column operations of the Gaussian algorithm are not contained in $\mathrm{SU}(d,q)$ and $\mathrm{Sp}(d,q)$, transvections in these groups can be seen as a substitute for them.

Transvections can be completely characterised by the following lemma.

Lemma 2.30

Let V be a d-dimensional $\mathbb{F}$-vector space. Let $T \in \mathrm{GL}(V)$ be a transvection with respect to a hyperplane $W \leq V$. Let $\vartheta : V \to \mathbb{F} \in V^*$ with $W = \ker(\vartheta)$.

1) There is a vector $w \in W$ such that $v^T = v - v^\vartheta \cdot w$ for all $v \in V$.
2) If $w \in V \setminus \{0\}$ with $w \in \ker(\vartheta)$, then $V \to V$, $v \mapsto v - v^\vartheta \cdot w$ is a transvection.

Proof. [91, p. 20]. □

Definition 2.31

Let V be a d-dimensional $\mathbb{F}$-vector space with basis $\mathcal{B} = \{v_1, \ldots, v_d\}$.

1) We denote by $\langle \cdot \mid \cdot \rangle$ the *standard scalar product* on V with respect to the basis $\mathcal{B}$, that is, $\langle v_i \mid v_j \rangle = \delta_{i,j}$.
2) For linearly independent $w_1, w_2 \in V$, we denote the transvection $v \mapsto v + \langle v \mid w_1 \rangle w_2$ by T_{w_1,w_2}.

Note that ${}^{\mathscr{B}}(T_{\lambda v_i, v_j})^{\mathscr{B}} = E_{i,j}(\lambda)$.

Lemma 2.32

Let $v, w \in V$ and let $c \in \mathrm{GL}(V)$ be arbitrary. Then $T^c_{v,w} = T_{vc^{-\mathrm{Tr}}, wc}$, where $c^{-\mathrm{Tr}}$ is the inverse transposed matrix of c, written with respect to the basis $\mathscr{B}$.

Proof. If $\tilde{v} \in V$ then

$$\begin{aligned}\tilde{v}T^c_{v,w} = \tilde{v}c^{-1}T_{v,w}c = (\tilde{v}c^{-1} + \langle \tilde{v}c^{-1} \mid v\rangle w)c &= \tilde{v}c^{-1}c + \langle \tilde{v}c^{-1} \mid v\rangle wc \\ &= \tilde{v} + \langle \tilde{v} \mid vc^{-\mathrm{Tr}}\rangle wc = \tilde{v}T_{vc^{-\mathrm{Tr}}, wc}.\end{aligned}$$

□

Lemma 2.33

Let $q = p^f$ and let V be an $\mathbb{F}_q$-vector space. Let $v, w \in V$ with $\langle w \mid v\rangle = 0$ and let $0 \leq k \leq p-1$. Then $T^k_{v,w} = T_{v,kw}$.

Proof. If $\tilde{v} \in V$ then

$$\begin{aligned}\tilde{v}T^2_{v,w} = \tilde{v}T_{v,w}T_{v,w} &= (\tilde{v} + \langle \tilde{v} \mid v\rangle w)T_{v,w} \\ &= (\tilde{v} + \langle \tilde{v} \mid v\rangle w) + \langle (\tilde{v} + \langle \tilde{v} \mid v\rangle w) \mid v\rangle w \\ &= \tilde{v} + \langle \tilde{v} \mid v\rangle w + \langle \tilde{v} \mid v\rangle w \\ &= \tilde{v} + \langle \tilde{v} \mid v\rangle 2w = \tilde{v}T_{v,2w}\end{aligned}$$

since $\langle (\tilde{v} + \langle \tilde{v} \mid v\rangle w) \mid v\rangle = \langle \tilde{v} \mid v\rangle + \langle \tilde{v} \mid v\rangle\langle w \mid v\rangle = \langle \tilde{v} \mid v\rangle$. By induction on k the claim follows. □

Lemma 2.34

Let $w_1, w_2, \ldots, w_k, v \in V$ with $\langle v \mid w_i\rangle = 0$. Then the transvections $T_{w_i,v}$ for $1 \leq i \leq k$ commute pairwise and their product is the transvection $T_{\sum_{i=1}^k w_i, v}$.

Proof. Let $i \neq j$. For all $\tilde{v} \in V$

$$\begin{aligned}\tilde{v}T_{w_i,v}T_{w_j,v} = (\tilde{v} + \langle \tilde{v} \mid w_i \rangle v)T_{w_j,v} &= \tilde{v} + \langle \tilde{v} \mid w_i \rangle v + \langle \tilde{v} + \langle \tilde{v} \mid w_i \rangle v \mid w_j \rangle v \\ &= \tilde{v} + \langle \tilde{v} \mid w_i \rangle v + \langle \tilde{v} \mid w_j \rangle v \\ &= \tilde{v} + \langle \tilde{v} \mid w_i + w_j \rangle v = \tilde{v}T_{w_i+w_j,v}\end{aligned}$$

since $\langle \tilde{v} + \langle \tilde{v} \mid w_i \rangle v \mid w_j \rangle = \langle \tilde{v} \mid w_j \rangle + \langle \tilde{v} \mid w_i \rangle \langle v \mid w_j \rangle = \langle \tilde{v} \mid w_j \rangle$. This proves the commutativity and the claim about the product follows by induction. □

Lemma 2.35

Let $d \in \mathbb{N}$ with $d > 1$ and q a prime power.

1) $\mathrm{SL}(d,q)$ is generated by transvections.
2) $\mathrm{Sp}(d,q)$ is generated by transvections.
3) $\mathrm{SU}(d,q)$ is generated by transvections.

Proof. [51, Theorem 1.2 and Remark] □

The following result shows that we cannot work with orthogonal transvections.

Lemma 2.36

Let q be odd. Then $O(d,q)$ contains no transvections. If $\mathbb{F}$ has characteristic 2 then a transvection for a quadratic form Q has the form

$$v \mapsto v - Q(w)^{-1}\Phi_Q(v,w)w$$

where $Q(w) \neq 0$ and Φ_Q is the polar form of Q.

Proof. [91, p. 137 and p. 145]. □

Corollary 2.37

Let q be odd. Then $O(d,q)$ is not generated by transvections.

2.3.2 Siegel transformations

In orthogonal groups Siegel transformations are used to carry out row and column operations. Since they are more complex than transvections, many proofs and algorithms for orthogonal groups are more complicated.

Definition 2.38

Let V be an $\mathbb{F}$-vector space with quadratic form Q and polar form Φ_Q. Let $v \in V \setminus \{0\}$ be isotropic, i.e. $Q(v) = 0$, and $w \in \langle v \rangle^\perp$. Then the map

$$\rho_{v,w} : V \to V,\ \tilde{v} \mapsto \tilde{v} + \Phi_Q(\tilde{v}, w)v - \Phi_Q(\tilde{v}, v)w - Q(w)\Phi_Q(\tilde{v}, v)v$$

is a *Siegel transformation*.

Remark 2.39

That a Siegel transformation $\rho_{v,w}$ is a well-defined element of $\mathrm{SO}(V)$ for fields of all characteristics is shown in [91, p. 148]. ◀

Example 2.40

Let $f \in \mathbb{N}$ and $q = p^f$ a prime power. Then

$$\rho_{e_1,e_3} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{pmatrix} \in \mathbb{F}_q^{4\times 4}$$

is a Siegel transformation in $\mathrm{SO}^+(4, q)$. ◀

Lemma 2.41

Let V be an $\mathbb{F}$-vector space with quadratic form Q. Let $v \in V$ be isotropic, let $w, w_1, w_2 \in \langle v \rangle^\perp$, let $\varphi \in O(V)$ and let $\jmath \in \mathbb{F}^*$. Then we have:

1) $\rho_{\jmath v,w} = \rho_{v,\jmath w}$,
2) $\rho_{v,w_1}\rho_{v,w_2} = \rho_{v,w_1+w_2}$,
3) $\varphi\rho_{v,w}\varphi^{-1} = \rho_{v^\varphi,w^\varphi}$.

Proof. [91, Theorem 11.19]. □

The following result states that orthogonal groups are generated by Siegel transformations.

Theorem 2.42

Let V be an $\mathbb{F}$-vector space with quadratic form Q. If $\dim(V) \geq 3$, the Witt index of V is at least 1 and $\Omega(V) \neq \Omega^+(4,2)$, then $\Omega(V)$ is generated by the Siegel transformations of (V,Q).

Proof. [91, Theorem 11.46]. □

2.4 Representation theory

The goal of this chapter is the definition of the natural representation of classical groups which corresponds to the definition of classical groups in Section 2.2. This chapter follows [68] for the introduction of basic concepts of representation theory. In this section we assume all groups to be finite.

Definition 2.43: [68, Definition 1.1.1]

A *linear representation* of a group G on a vector space V over a field $\mathbb{F}$ is a group homomorphism

$$\mathscr{X} : G \to \mathrm{GL}(V),$$

i.e. $(g_1 g_2)^{\mathscr{X}} = g_1^{\mathscr{X}} g_2^{\mathscr{X}}$ for all $g_1, g_2 \in G$. The *degree* of the representation is given by $\dim(V)$ and V is a *G-module* with respect to the action $(v,g) \mapsto v g^{\mathscr{X}}$. A *matrix representation* of G of degree d is a homomorphism

$$\mathscr{X} : G \to \mathrm{GL}(d,q).$$

Remark 2.44: [68, p. 1]

Linear representations and matrix representations describe the same concept. Let V be an $\mathbb{F}_q$-vector space of dimension d. By choosing a basis of V we obtain a group isomorphism $\mathrm{GL}(V) \to \mathrm{GL}(d,q)$ as stated in Corollary 2.4. ◀

Example 2.45: [68, Example 1.1.17 and 1.1.19]

1) The map $G \to \mathrm{GL}(1,q),\ g \mapsto 1$ is the *trivial representation*.
2) Let G be a finite group and V be a vector space over a field $\mathbb{F}_q$ of dimension $|G|$ with basis $(e_g)_{g\in G}$. We define
$$\mathscr{X}: G \to \mathrm{GL}(V),\ g \mapsto (V \to V,\ e_h \mapsto e_{hg^{-1}})$$
which is the *regular representation* of G. ◀

Representation theory can also be viewed from another perspective. For this we need group algebras.

Definition 2.46: [68, Definition 1.1.6]

Let $\mathbb{F}$ be a field and G a group. We put $\mathbb{F}G := \mathbb{F}^G$, the set of all maps from G to $\mathbb{F}$ which is an $\mathbb{F}$-module. For $g \in G$ we define $\overline{g} \in \mathbb{F}G$ by
$$h^{\overline{g}} := \begin{cases} 0, \text{ for } h \in G\backslash\{g\}, \\ 1, \text{ for } h = g. \end{cases}$$
The map
$$\eth: G \to \mathbb{F}G,\ g \mapsto \overline{g}$$
defines an embedding of G into $\mathbb{F}G$. In particular, the elements of $\mathbb{F}G$ are of the form $\sum_{g\in G} \lambda_g g$ for $\lambda_g \in \mathbb{F}$.
Then $\mathbb{F}G$ becomes an $\mathbb{F}$-algebra with the multiplication
$$\Big(\sum_{g\in G} \imath_g \overline{g}\Big)\cdot\Big(\sum_{h\in G} \jmath_h \overline{h}\Big) := \sum_{k\in G}\Big(\sum_{g,h\in G, gh=k} \imath_g \jmath_h\Big)\overline{k}$$
for $\imath_g, \jmath_h \in \mathbb{F}$ for all $g,h \in G$ and $\mathbb{F}G$ is the *group algebra* of G over $\mathbb{F}$.

Remark 2.47: [68, Remark 1.1.7]

The group algebra $\mathbb{F}G$ is an $\mathbb{F}$-vector space with basis $\{\overline{g}\}_{g\in G}$. ◀

Definition 2.48: [68, p. 4]

Let G be a group and $\mathbb{F}$ be a field. An $\mathbb{F}$-algebra homomorphism $\mathbb{F}G \to \mathrm{End}_{\mathbb{F}}(V)$ is a *representation of $\mathbb{F}G$ on V*.

Lemma 2.49: [68, pp. 4/5]

Let V be a finite dimensional vector space over $\mathbb{F}$ and let G be a group.

1) Let $\mathscr{X}: G \to \mathrm{GL}(V)$ be a representation. Then V is an $\mathbb{F}G$-module under the action
$$v^{\varrho} := v^{\varrho^{\hat{\mathscr{X}}}}, \quad \text{for } \varrho \in \mathbb{F}G,\ v \in V$$
where $\hat{\mathscr{X}}: \mathbb{F}G \to \mathrm{End}_{\mathbb{F}}(V)$ defines the extension of $\mathscr{X}$ to $\mathbb{F}G$.

2) Let V be an $\mathbb{F}G$-module. There exists a representation $\hat{\mathscr{X}}: \mathbb{F}G \to \mathrm{End}_{\mathbb{F}}(V)$. The restriction of $\hat{\mathscr{X}}$ to $G \subseteq (\mathbb{F}G)^*$ yields a representation $\mathscr{X}: G \to \mathrm{GL}(V)$, $g \mapsto g^{\hat{\mathscr{X}}}$. Then $\mathscr{X}$ is the *representation of G afforded by V*.

Remark 2.50: [68, p. 4]

The set of G-modules over $\mathbb{F}$ can be canonically identified with the set of $\mathbb{F}G$-modules. ◀

The next definition defines how representations can be compared.

Definition 2.51: [68, Definition 1.1.2]

Let $\mathscr{X}$ and $\mathscr{Y}$ be representations of G on $\mathbb{F}$-vector spaces V and W, respectively. The representations $\mathscr{X}$ and $\mathscr{Y}$ are *equivalent* if there exists an $\mathbb{F}$-isomorphism $\varphi: V \to W$ such that
$$g^{\mathscr{X}} = \varphi g^{\mathscr{Y}} \varphi^{-1}$$
for all $g \in G$.

One main goal of representation theory is the study of irreducible representations which are defined as follows.

Definition 2.52: [68, Definition 1.1.16]

Let R be a ring. An R-module is *simple* if it has exactly two R-submodules.

Definition 2.53: [68, Definition 1.1.16]

Let $\mathscr{X}: G \to \mathrm{GL}(V)$ be a representation of a group G on an $\mathbb{F}$-vector space V. The representation $\mathscr{X}$ is *irreducible* if $V \neq \{0\}$ and V is simple as an $\mathbb{F}G$-module. If $\mathscr{X}$ is not irreducible, then $\mathscr{X}$ is *reducible*.

We introduce another important property of representations and the connection to irreducibility.

Definition 2.54: [68, Definition 1.1.16]

Let $\mathscr{X}: G \to \mathrm{GL}(V)$ be a representation of a group G on an $\mathbb{F}$-vector space V. Then V is *decomposable* if it is a direct sum of non-trivial G-invariant subspaces. Otherwise, it is *indecomposable*.

Remark 2.55: [68, pp. 6/7]

A decomposable representation is reducible. The converse is not generally true. ◀

We are finally in a position to define the natural representation of classical groups.

Definition 2.56

Let V be an $\mathbb{F}$-vector space admitting a non-degenerate bilinear form Φ and let G be the group preserving Φ. Then $G \leq \mathrm{GL}(V)$ and the canonical embedding $\varphi: G \to \mathrm{GL}(V)$ is the *natural representation of G*. In this case, V is the *natural G-module*.

Theorem 2.57

The natural representations of $\mathrm{SL}(d,q)$, $\mathrm{Sp}(d,q)$, $\mathrm{SU}(d,q)$ and $\Omega(d,q)$ are irreducible.

Proof. [57]. □

2.5 Complexity and algorithms

In order to analyse the algorithms of this thesis we first define which operations are counted. It is sometimes quite tedious to perform precise inquiries which is why the Landau notation $\mathcal{O}$ is introduced. Some algorithms of this thesis rely on well-known methods and procedures. The most important ones are presented in this section and complexity results are summarised from literature. The treatment presented here is based on [48].

We start by specifying which operations are counted in this thesis.

Remark 2.58

In this thesis we count elementary finite field operations, i.e. addition, subtraction, multiplication and inverting over $\mathbb{F}_q$. Note that we do not differentiate between "additive" and "multiplicative" operations. ◀

When performing a complexity analysis of an algorithm the goal is to compute the worst case number of finite field operations in comparison to the input size. In most algorithms of this thesis the input size depends on the degree d of the input matrices and $\log(q)$ where q is the size of the finite field $\mathbb{F}_q$. The worst-case number of finite field operations is helpful to assess the run-time of an instance depending on the input.

In order to express complexity more accessible the well-known Landau notation $\mathcal{O}$ is introduced.

Definition 2.59: Landau notation $\mathcal{O}$

Let $\varphi_1, \varphi_2 \colon \mathbb{R}^+ \to \mathbb{R}$ be functions where $\mathbb{R}^+$ denotes the set of positive real numbers. Then $\varphi_1 \in \mathcal{O}(\varphi_2)$ if there are constants $k_1, k_2 \in \mathbb{R}^+$ such that $|\varphi_1(x)| \leq k_1|\varphi_2(x)|$ for all $x \geq k_2$.

Example 2.60

In the standard algorithm to multiply $g_1, g_2 \in \mathrm{GL}(d,q)$, we must compute d^2 entries and for each entry perform d multiplications and $d-1$ additions. This yields an overall result of $(2d-1)d^2$ elementary field operations. Hence, this algorithm has complexity $\mathcal{O}(d^3)$. ◀

Remark 2.61

Note that there are algorithms to multiply matrices which need less than $(2d-1)d^2$ elementary finite field operations. For example Strassen's algorithm [89] for multiplying two $d \times d$ matrices from 1969 has complexity $\mathcal{O}(d^{\log_2(7)}) = \mathcal{O}(d^{2.8074})$ which is one of the best known algorithms and which has only been beaten recently by an AI for some small d. Therefore, we say that matrix multiplication has complexity $\mathcal{O}(d^\xi)$ with $\xi \leq \log_2(7)$. In comparison, multiplying two permutations of $\mathrm{Sym}(d)$ has complexity $\mathcal{O}(d)$ which makes efficient computations much more important. ◀

Additionally to the logarithm function log we are sometimes referring to the iterated logarithm $\log^*$ which is defined as follows.

Definition 2.62

Let $k \in \mathbb{R}$ with $k > 0$. The iterated logarithm $\log^*(k)$ is defined recursively as follows

$$\log^*(k) := \begin{cases} 0, & \text{if } k \leq 1 \text{ and} \\ 1 + \log^*(\log(k)), & \text{if } k > 1. \end{cases}$$

Remark 2.63

The iterated logarithm counts the number of times the logarithm function must be applied recursively until a non-negative real number is equal or less than 1. For the remainder of this thesis the iterated logarithm $\log^*(k)$ is used to base 2. ◀

One important tool for this thesis is the characteristic polynomial $\chi_a(x) = \det(a - xI_d)$ of a matrix $a \in \mathrm{GL}(d,q)$. See for example [36, 71] for an algorithm of complexity $\mathcal{O}(d^3)$ to compute the characteristic polynomial. There could be more efficient algorithms but an upper bound of $\mathcal{O}(d^3)$ for the complexity of computing characteristic polynomials is sufficient for this thesis.

Since many algorithms of this thesis are randomised it is crucial to have algorithms for computing uniformly distributed and independent random elements of a group $\langle X \rangle = G$. Note that we also require that the uniformly distributed and independent random elements of G can be written as a word in X. The theoretically best algorithm for computing ϵ-uniformly distributed and independent random elements is given by Babai [7]. The complexity of Babai's algorithm is in $\mathcal{O}(\log^5(|G|))$ which is too high for practical applications.

Another well-known algorithm for computing random elements is *ProductReplacement* [25]. There have been some improvements of the `ProductReplacement` algorithm in the last years, see e.g. [58, 67], but we restrict ourselves to the basic version given in [25] which is presented in pseudo-code as Algorithm ProductReplacement [Alg. 1].

The Algorithm ProductReplacement [Alg. 1] uses a list S of n group elements of G as an internal state denoted by $S = [\hat{g}_1, \ldots, \hat{g}_n] \in G^n$. If the list S is not bound at the first call of Algorithm ProductReplacement [Alg. 1], then S is set to be the list of length n containing the generators X of G multiple times until the list is full. In each call Algorithm ProductReplacement [Alg. 1] updates one element of S by multiplying two elements at random positions i and j of S and storing the product in position i. Additionally, we use a coin flip to decide whether the element at position j is inverted and whether to multiply the element at position i with the element at position j or vice versa. After calling Algorithm ProductReplacement [Alg. 1] repeatedly the list S contains elements very close to being uniformly distributed and independent. Note that in an implementation we would remember how each element of S was computed as a word in X which is

possible as the list was originally initialised as a list only containing the generators X. Therefore, we assume in the remainder of the thesis, that we can write uniformly distributed and independent random elements of a group in a given generating set.

Algorithm 1: PRODUCTREPLACEMENT

Input: ▶ A group $G = \langle g_1, \ldots, g_k \rangle$
▶ A natural number $n \geq \max(\{k, 2\})$

Output: ▶ $g \in G$

function PRODUCTREPLACEMENT(G, n)

```
    // Internal state: S = [ĝ_1, ..., ĝ_n] ∈ G^n
1   if not IsBound(S) then                          // i.e. S is not assigned a value
2       S ← [g_1, ..., g_k, g_1, g_2, ...]
3   i, j ← RANDOM({1, ..., n})
4   r ← RANDOM({0, 1}) AND t ← RANDOM({−1, 1})
5   if r = 0 then
6       S[i] ← S[i] · S[j]^t
7   else
8       S[i] ← S[j]^t · S[i]
9   return S[i]
```

Note that each call to `ProductReplacement` performs only one group multiplication. However, the algorithm must be called a certain number of times to mix the initial state S. We call this initial period the *convergence time*. After the convergence time, the cost of each call to `ProductReplacement` is one group operation.

Practical applications observed that n should be at least $2k$ and that the convergence time can be taken to be 50 - 70 steps. After this `ProductReplacement` appears to return elements that are very close to being uniformly distributed and independent for most groups and particularly for classical groups [25]. To get an overview of the `ProductReplacement` algorithm, we refer to [77]. Since the theoretical results do not support such a strong statement so far, we introduce a variable for the complexity of computing a random element instead and refer to [34] for more details about generating random elements in finite groups.

Remark 2.64

For the remainder of this thesis the complexity of computing a uniformly distributed and independent random element of a group is denoted ζ. ◀

The computation of the centraliser and projective centraliser of involutions is another important aspect. John Bray developed an efficient algorithm for this purpose in [18]. The algorithm of Bray was analysed in [46, 78] leading to the following result.

Theorem 2.65

Let G be a simple group of Lie rank r defined over a field of odd characteristic. The centraliser in G of an involution can be computed in time $\mathcal{O}(r(\gimel+\zeta)\log(1/\epsilon)+\wp r^2)$ with probability of success at least $1-\epsilon$ for $\epsilon > 0$, where $\wp$ denotes the cost of a group operation and $\gimel$ denotes the cost of solving the order oracle.

Remark 2.66

We summarise the complexity results of this section in the following table.

Algorithm	Complexity
Matrix multiplication	$\mathcal{O}(d^\xi)$
Characteristic polynomial	$\mathcal{O}(d^3)$
Random element	ζ
Algorithm of Bray	$\mathcal{O}(r(\gimel+\zeta)\log(1/\epsilon)+\wp r^2)$

◀

2.6 MSLP

In this section, we present an efficient way to express a group element as a word in specified generators using basic group operations. Therefore, it is possible to encode group elements as words over a given generating set. Rather than storing the word as a string, we write it as a word in other subwords. This allows us to evaluate the encoded word efficiently under a group homomorphism.

We start by defining which operations can be used. The way we describe the word can be viewed as a program in some programming language. The program has no loops, no conditional statements, no comparisons or other special techniques from programming languages. We restrict ourselves to

multiplying and inverting. Therefore, we call the resulting description of the word a *straight-line program* or *SLP* for short. The treatment presented here is based on [73].

Remark 2.67

MSLPs play an important role in the matrix group recognition project (MGRP) which is presented in more detail in Chapter 1 and even more information about the MGRP can be found in [5]. For the introduction of MSLPs we describe a basic but important subtask of the MGRP in the following. In the MGRP some matrices $a_1,\dots,a_k \in \mathrm{GL}(d,q)$ over a finite field $\mathbb{F}_q$ are given and we consider the group $G = \langle a_1,\dots,a_k\rangle$. We could ask whether $a \in \mathrm{GL}(d,q)$ is an element of G and if this is the case, how a can be expressed in terms of the generators $a_1,\dots,a_k$. The expression of writing a in terms of the generators $a_1,\dots,a_k$ is a *word* and in order to store and evaluate words efficiently MSLPs are used. ◀

Definition 2.68: [73, Section 2.1]

Let G be a group, $\mathfrak{b} \in \mathbb{N}_0$ and $\mathscr{M} = [\mathfrak{m}_1,\dots,\mathfrak{m}_\mathfrak{b}]$ an ordered list, i.e. a list in which the order of the items matter, of $\mathfrak{b}$ elements of G. A modification $\mathscr{I}$ of $\mathscr{M}$ is an *instruction* if it has one of the following forms:

(i) $\mathfrak{m}_k \leftarrow \mathfrak{m}_i$ with $i,k \in \{1,\dots,\mathfrak{b}\}$. This instruction stores $\mathfrak{m}_i$ in the list $\mathscr{M}$ in slot k.

(ii) $\mathfrak{m}_k \leftarrow \mathfrak{m}_i \cdot \mathfrak{m}_j$ with $i,j,k \in \{1,\dots,\mathfrak{b}\}$. This instruction stores $\mathfrak{m}_i \cdot \mathfrak{m}_j$ in the list $\mathscr{M}$ in slot k.

(iii) $\mathfrak{m}_k \leftarrow \mathfrak{m}_i^{-1}$ with $i,k \in \{1,\dots,\mathfrak{b}\}$. This instruction stores $\mathfrak{m}_i^{-1}$ in the list $\mathscr{M}$ in slot k.

(iv) Show(A) where $A \subseteq \{1,\dots,\mathfrak{b}\}$. The slots specified by A are displayed.

Remark 2.69

Some programming languages, e.g. GAP [37], allow additional instructions which can be derived from the instructions (i) to (iii) as short cut. One additional type of instruction introduced by GAP is to set up the initial memory which we use with the following notation

$$\mathfrak{m}_i \leftarrow g$$

in this thesis. This means that the element g is stored in the memory slot $\mathfrak{m}_i$. ◀

We introduce a version of an SLP which also records its memory usage.

Definition 2.70: [73, Section 2.1]

Let G be a group, let $\mathfrak{b} \in \mathbb{N}_0$, let $\mathscr{M} = [\mathfrak{m}_1, \ldots, \mathfrak{m}_\mathfrak{b}]$ be an ordered list of $\mathfrak{b}$ elements of G and $\Upsilon \in \mathbb{N}_0$. A *straight-line program with memory* (MSLP) is a sequence $\mathfrak{S} = [\mathscr{I}_1, \ldots, \mathscr{I}_\Upsilon]$ of instructions $\mathscr{I}_r$ with $1 \leq r \leq \Upsilon$.

The number $\mathfrak{b} \in \mathbb{N}_0$ is the *memory quota* of $\mathfrak{S}$ and $\mathfrak{S}$ is a $\mathfrak{b}$-MSLP. The number Υ is the *length* of $\mathfrak{S}$.

The empty sequence is permitted with length 0.

Remark 2.71

Υ is the Greek capital upsilon and $\mathfrak{S}$ is S in the fraktur font. ◀

Let X be a set and F_X the free group on X. An obvious way to record a word in F_X is to simply store the sequence of elements of X in the word. This, however, is often not very efficient if we evaluate the image of the word under a group homomorphism $\varphi: F_X \to G$. For example, if G is a matrix group and the word has length k, then this would entail $k-1$ multiplications. The aim of an SLP is to give an algorithm how to evaluate homomorphisms on this word efficiently, for example by computing and storing subwords that occur repeatedly only once.

Example 2.72: [73, Section 2.2(ii)]

In most computations with groups, elements need to be raised to some powers. To shorten this, *fast exponentiation* can be used.

Let G be a group, $g \in G$ and $k \in \mathbb{N}$. Suppose we want to write an MSLP that raises a group element to its k-th power. We can express k in binary form as $k = \sum_{i=0}^{r} a_i \cdot 2^i$ with $a_i \in \{0, 1\}$ for $0 \leq i \leq r$ and $r \in \mathbb{N}$. Then

$$g^k = g^{\left(\sum_{i=0}^{r} a_i \cdot 2^i\right)} = \prod_{i=0}^{r} (g^{2^i})^{a_i}.$$

A way to construct an MSLP for fast exponentiation with memory quota 2 is given in Algorithm 2 in pseudo-code. ◀

Definition 2.73: [73, Section 2.1]

Suppose that $\mathscr{M} \in G^\mathfrak{b}$ and we have given an $\mathfrak{b}$-MSLP $\mathfrak{S} = [\mathscr{I}_1, \ldots, \mathscr{I}_\Upsilon]$. The *evaluation of an MSLP* consists of the following recursive steps. First, for $j \leq \Upsilon$ we denote the sequence obtained by truncating $\mathfrak{S}$ after j instructions by $\mathfrak{S}_j = [\mathscr{I}_1, \ldots, \mathscr{I}_j]$. Then we define the evaluation map $\mathrm{Eval}_\mathscr{M}$

on $\mathfrak{S}$ by:

$$\mathrm{Eval}_{\mathscr{M}}(\mathfrak{S}) = \begin{cases} \mathscr{M}, & \text{if } \mathfrak{S} = [\,], \\ [a'_1,\ldots,a'_{k-1},a'_i,a'_{k+1},\ldots,a'_{\mathfrak{b}}], & \text{where } [a'_1,\ldots,a'_{\mathfrak{b}}] = \mathrm{Eval}_{\mathscr{M}}(\mathfrak{S}_{\Upsilon-1}) \\ & \text{and if } \mathscr{I}_\Upsilon = \mathfrak{m}_k \leftarrow \mathfrak{m}_i, \\ [a'_1,\ldots,a'_{k-1},a'_ia'_j,a'_{k+1},\ldots,a'_{\mathfrak{b}}], & \text{where } [a'_1,\ldots,a'_{\mathfrak{b}}] = \mathrm{Eval}_{\mathscr{M}}(\mathfrak{S}_{\Upsilon-1}) \\ & \text{and if } \mathscr{I}_\Upsilon = \mathfrak{m}_k \leftarrow \mathfrak{m}_i \cdot \mathfrak{m}_j, \\ [a'_1,\ldots,a'_{k-1},(a'_i)^{-1},a'_{k+1},\ldots,a'_{\mathfrak{b}}], & \text{where } [a'_1,\ldots,a'_{\mathfrak{b}}] = \mathrm{Eval}_{\mathscr{M}}(\mathfrak{S}_{\Upsilon-1}) \\ & \text{and if } \mathscr{I}_\Upsilon = \mathfrak{m}_k \leftarrow \mathfrak{m}_i^{-1}, \\ [a'_1,\ldots,a'_{\mathfrak{b}}]|_A, & \text{where } [a'_1,\ldots,a'_{\mathfrak{b}}] = \mathrm{Eval}_{\mathscr{M}}(\mathfrak{S}_{\Upsilon-1}) \\ & \text{and if } \mathscr{I}_\Upsilon = \mathrm{Show}(A). \end{cases}$$

Thus evaluating the MSLP $\mathfrak{S}$ amounts to executing its instructions.

In the evaluation process, $\mathscr{M}$ is used as memory for the elements that are needed. The instruction (i) of Definition 2.68 can be used to overwrite a slot and minimise the memory quota. The idea of an MSLP is to reduce the memory required for evaluating a particular word in a group by repeatedly overwriting memory slots which are no longer used. The length Υ of an MSLP describes the number of operations during the entire evaluation.

Remark 2.74

The instructions of Definition 2.68 use elements of $\{1,\ldots,\mathfrak{b}\}$. The memory $\mathscr{M}$ is secondary in the description. This implies that an MSLP is independent of the group, i.e. $\mathrm{Eval}_{\mathscr{M}}(\mathfrak{S})$ can be computed for every $\mathscr{M} \in G^{\mathfrak{b}}$ and every group G. Hence, it is possible to encode an element as a word in one group and to evaluate the constructed SLP in another group. ◀

Algorithm 2: POWER

Input: ▶ $k \in \mathbb{N}_0$ a natural number

Output: ▶ An MSLP $\mathfrak{S}$ which computes g^k using fast exponentiation for $g \in G$

```
function Power(k)
1   i ← 1
2   while k > 0 do
3       if k odd then
4           I_i ← (m_1 ← m_1 · m_2)        // The factor in the binary representation is not 0
5           i ← i + 1
6       if k > 1 then
7           I_i ← (m_2 ← m_2 · m_2)        // Square the element in slot 2
8           i ← i + 1
9       k ← Floor(k/2)
10  I_i ← Show(1)
11  return [I_1, ..., I_i]
```

Example 2.75

Recall the additional instruction introduced in Remark 2.69 for initialising the memory. For example, Algorithm 2 produces the following MSLP for $k = 8$:

$$\mathfrak{S} = [[\mathfrak{m}_2 \leftarrow g], [\mathfrak{m}_1 \leftarrow 1_G], [\mathfrak{m}_2 \leftarrow \mathfrak{m}_2 \cdot \mathfrak{m}_2], [\mathfrak{m}_2 \leftarrow \mathfrak{m}_2 \cdot \mathfrak{m}_2],$$
$$[\mathfrak{m}_2 \leftarrow \mathfrak{m}_2 \cdot \mathfrak{m}_2], [\mathfrak{m}_1 \leftarrow \mathfrak{m}_1 \cdot \mathfrak{m}_2], [\mathrm{Show}(1)]]$$

In this case, we need 7 instructions which nearly is as much as k. However, for increasing k, the length of the SLPs converges to $2\log_2(k)$ which is in $\mathcal{O}(\log_2(k))$.

Now, let $G_1 := (\mathbb{F}_5)^*$ and $\mathscr{M}_1 := [1, 2] \in G_1^2$. If we evaluate the MSLP $\mathfrak{S}$ with $\mathscr{M}_1$, then the evaluation function returns $\mathrm{Eval}_{\mathscr{M}_1}(\mathfrak{S}) = [2^8] = [1]$. But we can also choose $G_2 := C_9$ the cyclic group with 9 elements and $\mathscr{M}_2 := [(), (1, \ldots, 9)] \in G_2^2$. If we evaluate the MSLP $\mathfrak{S}$ with $\mathscr{M}_2$, then the evaluation function returns $\mathrm{Eval}_{\mathscr{M}_2}(\mathfrak{S}) = [(1, \ldots, 9)^8] = [(1, \ldots, 9)^{-1}]$. ◀

For more details on MSLPs, see [73].

Chapter 3

Outline of the Algorithm

All algorithms presented in this thesis for constructively recognising special linear groups, symplectic groups, unitary groups and orthogonal groups are based on the same overall concept with relatively small differences resulting from the underlying structure of the classical groups. A group G is a classical group if $G \in \{\mathrm{SL}(d,q), \mathrm{Sp}(d,q), \mathrm{SU}(d,q), \Omega(d,q)\}$ as in Section 2.2 and we denote all of these by $\mathrm{CL}(d,q)$ where CL stands for SL, Sp, SU or Ω and represents the respective type. The goal of this chapter is to describe and explain the overall concept and structure of the algorithms for constructive recognition of classical groups of this thesis. We start by giving a detailed description of all subalgorithms. Afterwards, these subalgorithms are combined into a single algorithm, namely StandardGenerators, presented as pseudo code in Section 3.4, which is the foundation for the constructive recognition algorithms of classical groups presented in this thesis.

The input of the algorithm StandardGenerators is a set X of invertible $d \times d$ matrices over $\mathbb{F}_q$ such that the group $G = \langle X \rangle$ is a classical group in its natural representation. That a given group G is indeed a classical group in its natural representation can be verified using naming algorithms which are explained in Section 1.1.7.

The output of the algorithm StandardGenerators is a base change matrix $\mathscr{L} \in \mathrm{GL}(d,q)$ and an MSLP $\mathfrak{S}$ such that if $\mathfrak{S}$ is evaluated in $X^{\mathscr{L}}$, then the output of $\mathfrak{S}$ is a specific generating set, namely the *standard generators*, of $\mathrm{CL}(d,q)^{\mathscr{L}}$, which are defined as follows.

Definition 3.1

Let $S \subset \mathrm{CL}(d,q)$. Then S is a set of *standard generators* of $\mathrm{CL}(d,q)$ if there is a base change matrix $\mathscr{L} \in \mathrm{GL}(d,q)$ such that $S^{\mathscr{L}} = \{g^{\mathscr{L}} \mid g \in S\}$ is equal to a specific subset of $\mathrm{CL}(d,q)$ as defined in

- Definition 5.1 for $\mathrm{CL}(d,q)$ being a special linear group,
- Definition 6.3 for $\mathrm{CL}(d,q)$ being a symplectic group,
- Definition 7.3 for $\mathrm{CL}(d,q)$ being a special unitary group or
- Definition 8.3 for $\mathrm{CL}(d,q)$ being an orthogonal group.

We are interested in these generating sets because they yield a variety of advantages compared to arbitrary generating sets of classical groups. Firstly, these generating sets contain the DLLO standard generators defined in [32, 59] which can be used to verify relations of a finite presentation of a classical group, see [60], and, therefore, the correctness of the output of a randomised constructive recognition algorithm. Secondly, it is possible to perform efficient rewriting procedures as in Section 1.1.10 starting with standard generators while it is unclear how to perform rewriting with arbitrary generators.

The algorithms of this thesis are designed to deal with *large* classical groups defined as follows.

Definition 3.2

Let $\mathrm{CL}(d,q)$ be a classical group in its natural representation. Then $\mathrm{CL}(d,q)$ is a *base case group* if

- $\mathrm{CL}(d,q)$ is a special linear group and $d \leq 2$,
- $\mathrm{CL}(d,q)$ is a symplectic group and $d \leq 4$,
- $\mathrm{CL}(d,q)$ is a special unitary group and $d \leq 4$ or
- $\mathrm{CL}(d,q)$ is an orthogonal group and $d \leq 6$.

Moreover, $\mathrm{CL}(d,q)$ is a *large group* if it is not a base case group.

In this thesis we do not deal directly with base case groups, but instead refer the reader to results and algorithms already existing in the literature, see Table 3.4. An exception are the special linear groups where we give a detailed description of a constructive recognition algorithm for $\mathrm{SL}(2,q)$ in Section 5.2 in order to gain an understanding of how constructive recognition algorithms for base case groups work and outline their underlying mathematical problems.

In summary, base case groups are classical groups in small dimension for which specialised and efficient constructive recognition algorithms exist. If the input group G is a base case group, then we call an algorithm specifically designed for constructive recognition of the base case group G. If G is not a base case group, then we apply the corresponding constructive recognition algorithm for $\mathrm{CL}(d,q)$ described in this thesis. The algorithm StandardGenerators, which is given in Section 3.4, relies for all classical groups on the same three basic subalgorithms, namely

(1) a GoingDown algorithm which is explained in Section 3.1,

(2) a BaseCase algorithm which is explained in Section 3.2 and

(3) a GoingUp algorithm which is explained in Section 3.3.

After dealing with all subalgorithms in the following sections the structure of the combined algorithm StandardGenerators is outlined in Section 3.4.

Another notation we use frequently throughout this thesis is given in the following definition. As usual, we denote a conjugate of a group G by a a base change matrix $\mathscr{L} \in \mathrm{GL}(d,q)$ by $G^{\mathscr{L}} = \{\mathscr{L}^{-1} g \mathscr{L} \mid g \in G\}$.

Definition 3.3

Let $G \leq \mathrm{GL}(d,q)$ and $U \leq G$. Then U is *stingray embedded of degree n in G* for $n \leq d$ if there exists a group $H \leq \mathrm{GL}(n,q)$, a base change matrix $\mathscr{L} \in \mathrm{GL}(d,q)$ and an isomorphism $\sigma_{\mathscr{L}}$ from H to $U^{\mathscr{L}}$ such that

$$\sigma_{\mathscr{L}} : H \to U^{\mathscr{L}},\ a \mapsto \mathrm{diag}(a, I_{d-n}) = \begin{pmatrix} a & 0 \\ 0 & I_{d-n} \end{pmatrix}.$$

We denote a stingray embedding by

$$U^{\mathscr{L}} = \begin{pmatrix} H & 0 \\ 0 & I_{d-n} \end{pmatrix} \leq G^{\mathscr{L}}.$$

3.1 GoingDown algorithm

The GoingDown algorithm is the first subalgorithm which is called by the StandardGenerators algorithm. The input of the GoingDown algorithm is a generating set X such that $\langle X \rangle = G = \mathrm{CL}(d,q)$ which is also the input of the StandardGenerators algorithm. In this section, we exclude $\mathrm{Sp}(d,q)$ for q even from $\mathrm{CL}(d,q)$ which is further elaborated in Remark 6.1. The GoingDown algorithm is a randomised algorithm and relies on finding *stingray elements* with a random procedure of elements of G. Therefore, there is an additional input $N \in \mathbb{N}$ which is used as an upper bound for the maximal number of chosen random elements. If the limit N is exceeded, then the GoingDown algorithm returns `fail`. The GoingDown algorithm of this thesis is a one-sided Monte-Carlo algorithm, therefore we also discuss how to choose $N \in \mathbb{N}$ for a given $0 < \epsilon < 1$ such that the GoingDown algorithm succeeds with probability $1 - \epsilon$. Stingray elements, which are used by the GoingDown algorithm, are discussed in detail in Chapter 4. The output of the GoingDown algorithm is a base change matrix $\mathscr{L}$, an MSLP $\mathfrak{S}$ and a subgroup $\langle X_U \rangle = U \leq G$ with $U \cong \mathrm{CL}(n,q)$ a base case group such that

$$U^{\mathscr{L}} = \left(\begin{array}{c|c} \mathrm{CL}(n,q) & 0 \\ \hline 0 & I_{d-n} \end{array}\right).$$

Moreover, evaluating the MSLP $\mathfrak{S}$ in X yields the generators X_U of U.

The GoingDown algorithm repeatedly applies a subroutine, which is the GoingDown basic step. The input of the GoingDown basic step subroutine is a stingray embedded subgroup U of G (initially G itself), and it returns a subgroup H of U which is stingray embedded of smaller degree in G than its predecessor U. This is used as input for the next GoingDown basic step invocation, and so on, until eventually a *terminal group* is reached which is defined below in Definition 3.6. We denote by U_{i-1} the input and by U_i the output of the i-th iteration of the basic step. Before the first iteration U_0 is defined to be G. After each iteration of the basic step, the subgroup U_i satisfies $U_i \leq U_{i-1}$ and U_i is the input for the next iteration of the basic step that is

$$U_i \coloneqq \textsc{GoingDownBasicStep}(U_{i-1}, N), \quad \text{where } U_i \leq U_{i-1}.$$

The input N is as for the GoingDown algorithm used to control the maximal number of chosen random elements. The basic step is applied until U_i is isomorphic to a terminal group. In this case

we set $k \coloneqq i$ and the GoingDown basic step returns a descending chain

$$U_k \leq U_{k-1} \leq \ldots \leq U_1 \leq U_0 = G, \tag{3.1.1}$$

where U_k is a terminal group and k is roughly $\log^*(d)$ which is the iterated logarithm as in Definition 2.62. To be more precise, each of the groups U_i of the descending chain is isomorphic to a classical group $U_i \cong \mathrm{CL}(d_i, q)$ of the same type as G and is stingray embedded in G for $d_i \leq d$ and $d_k < d_{k-1} < \ldots < d_1 < d_0 = d$. Thus there is $\mathscr{L}_i \in \mathrm{GL}(d, q)$ such that

$$U_i = \begin{pmatrix} \mathrm{CL}(d_i, q) & 0 \\ 0 & I_{d-d_i} \end{pmatrix}^{\mathscr{L}_i} \leq G.$$

The base change matrix $\mathscr{L}_i$ can also be computed by the GoingDown basic step by taking the base change matrix $\mathscr{L}_{i-1}$ of U_{i-1} and modifying it to a base change matrix $\mathscr{L}_i$ of U_i. Moreover, the GoingDown basic step also returns an MSLP $\mathfrak{S}_i$ such that when $\mathfrak{S}_i$ is evaluated in the generators of U_{i-1} it returns generators of U_i that is

$$(U_i, \mathscr{L}_i, \mathfrak{S}_i) \coloneqq \textsc{GoingDownBasicStep}(U_{i-1}, \mathscr{L}_{i-1}, N), \quad \text{where } U_i \leq U_{i-1}.$$

Lastly, we add d_{i-1} with $U_{i-1} \cong \mathrm{CL}(d_{i-1}, q)$ as input parameter for optimisation to control the length of the descending chain that is

$$(U_i, \mathscr{L}_i, d_i, \mathfrak{S}_i) \coloneqq \textsc{GoingDownBasicStep}(U_{i-1}, \mathscr{L}_{i-1}, d_{i-1}, N), \quad \text{where } U_i \leq U_{i-1}.$$

We additionally require that $d_i \leq 4\lceil \log(d_{i-1}) \rceil$ holds. This restriction ensures that the length k of the descending chain is in $\mathcal{O}(\log^*(d))$. The descending chain given in (3.1.1) is the *descending recognition chain* and a sub-chain of the *full descending recognition chain* given in Definition 3.4.

The GoingDown basic step is based on stingray elements which are defined and treated in detail in Chapter 4. The basic step uses stingray elements as follows: In U_{i-1} we seek two stingray elements $s_1, s_2 \in U_{i-1}$, using the Algorithms 5 and 7 of Chapter 4. In Chapter 10 we discuss that $\langle s_1, s_2 \rangle \cong \mathrm{CL}(d_i, q)$ holds with high probability. One reason why the algorithm uses stingray elements

is that $\langle s_1, s_2 \rangle$ is automatically stingray embedded in $\mathrm{CL}(d,q)$, i.e.

$$U_i = \langle s_1, s_2 \rangle = \begin{pmatrix} \mathrm{CL}(d_i,q) & 0 \\ 0 & I_{d-d_{i+1}} \end{pmatrix}^{\mathscr{L}_i} \leq G.$$

The base change matrix to achieve this block structure can be computed using the algorithms of Section 4.3. Since the basic step relies on finding stingray elements by a random procedure and on generating classical groups using stingray elements, the GoingDown algorithm is a randomised algorithm. Therefore, the GoingDown algorithm returns `fail` after selecting at most N random elements. The probability that the GoingDown algorithm succeeds completely relies on N and we analyse in Chapter 10 how to choose N in order to have a success rate of at least $1-\epsilon$ for a given $\epsilon \in (0,1)$.

We introduce a notion for a descending recognition chain in the next definition.

Definition 3.4

Let $\mathrm{CL}(d,q)$ be a classical group in its natural representation. Then a descending chain of subgroups of $\mathrm{CL}(d,q)$ is a *full descending recognition chain* with

$$U_k \leq U_{k-1} \leq \ldots \leq U_1 \leq U_0 = \mathrm{CL}(d,q),$$

if $U_i \cong \mathrm{CL}(d_i,q)$ of the same type as $\mathrm{CL}(d,q)$ is stingray embedded in $\mathrm{CL}(d,q)$, $d_i < d_{i-1}$ and $d_i \leq 4\lceil \log(d_{i-1}) \rceil$ for all $1 \leq i \leq k$, the subgroup U_k of $\mathrm{CL}(d,q)$ is isomorphic to a base case group and the subgroup U_{k-1} is isomorphic to a terminal group. The sub-chain

$$U_{k-1} \leq \ldots \leq U_1 \leq U_0 = \mathrm{CL}(d,q),$$

is the *descending recognition chain*.

Remark 3.5

Note that a descending recognition chain can be computed using the GoingDown basic step repeatedly. ◀

A terminal group is similar to a base case group and defined as follows.

Definition 3.6

Let $\mathrm{CL}(d,q)$ be a classical group in its natural representation. Then $\mathrm{CL}(d,q)$ is a *terminal group* if

- $\mathrm{CL}(d,q)$ is a special linear group and $d = 4$,
- $\mathrm{CL}(d,q)$ is a symplectic group and $d = 8$,
- $\mathrm{CL}(d,q)$ is a special unitary group, $d = 6$ and q is odd,
- $\mathrm{CL}(d,q)$ is a special unitary group, $d = 10$ and q is even or
- $\mathrm{CL}(d,q)$ is an orthogonal group and $d = 8$.

Remark 3.7

It is not possible to reach a base case group using the GoingDown basic step because their degrees d_i of $U_i \cong \mathrm{CL}(d_i,q)$ are too small to find stingray elements or generating classical groups with these elements. Therefore, we stop the GoingDown basic step at a terminal group and the GoingDown algorithm relies on alternative algorithms to identify a stingray embedded base case group which is the *final step* of the GoingDown algorithm and the last step for computing a full descending recognition chain from the descending recognition chain of the GoingDown basic step such that

$$\underbrace{U_k \leq \overbrace{U_{k-1} \leq \ldots \leq U_1 \leq U_0 = G}^{\text{GoingDown basic step}}}_{\text{Final step}}$$

where U_k is a base case group and U_{k-1} a terminal group. The degrees of the terminal groups of the GoingDown basic step are summarised in Table 3.1. For terminal groups there are no direct constructive recognition algorithms without computing base case groups so far but the GoingDown algorithm can identify base case groups in terminal groups efficiently using the algorithms of the sections given as in Table 3.2. ◀

Type	q even	q odd
SL	4	4
Sp	—	8
SU	10	6
Ω	8	8

Table 3.1: Dimensions of the terminal groups of the GoingDown basic step, i.e. a terminal group as in Definition 3.6 has been reached in the descending chain (3.1.1) of the GoingDown basic step.

Type	Section
SL	Section 5.1.3
Sp	Section 5.1.3 and Section 6.1.3
SU	Section 5.1.3 and Section 7.1
Ω	Section 5.1.3 and Section 8.1

Table 3.2: Sections for the algorithms of the final step of the GoingDown algorithm to compute a base case group from a terminal group of the descending recognition chain.

Remark 3.8

Note that there are some classical groups in their natural representation whose degree lies between the one of a base case group and the one of a terminal group, see Table 3.3. These exceptional classical groups are either not covered by the constructive recognition algorithms of this thesis or additional algorithms to deal with these groups are discussed in the sections about the final step, see Table 3.2. If a group is not covered in this thesis, then this is mentioned at the beginning of each chapter and we refer to algorithms already existing in the literature. ◀

Type	Ranges for d which are not covered	Algorithms in the literature
SL	3	[65]
Sp	6	[20]
SU	5 if q is odd and 5-9 if q is even	[21, 32]
Ω	7	[32, 59]

Table 3.3: Groups whose degree lies between base and terminal groups.

3.2 BaseCase algorithm

The BaseCase algorithm is the second subalgorithm of the StandardGenerators algorithm. Let $G := \mathrm{CL}(d,q)$ be the input of StandardGenerators. Then the input of the BaseCase algorithm is a generating set X of a base case group $\mathrm{CL}(d_k,q) \leq G$ as in Definition 3.2 of the same type as G, e.g. if G is a special linear group, then the base case group is also a special linear group. Moreover, a base change matrix $\mathscr{L} \in \mathrm{GL}(d,q)$ is known such that $\langle X \rangle^{\mathscr{L}}$ is stingray embedded with degree d_k in $G^{\mathscr{L}}$. Since the BaseCase algorithm is also randomised we again use N as additional input to restrict the maximal number of selecting random elements to N. The output of the BaseCase algorithm is an MSLP $\mathfrak{S}$ and a base change matrix $\mathscr{L}' \in \mathrm{GL}(d,q)$ such that if $\mathfrak{S}$ is evaluated in $X^{\mathscr{L}'}$, the output are the standard generators of $\mathrm{CL}(d_k,q)^{\mathscr{L}'}$ as in Definition 3.1.

For base case groups efficient constructive recognition algorithms are known which do not rely on constructive recognition of subgroups. Therefore, the BaseCase algorithm calls an efficient constructive recognition algorithm for $\mathrm{CL}(d_k, q)$ which is possible as $\mathrm{CL}(d_k, q)$ is a base case group. As mentioned earlier efficient algorithms for constructive recognition of base case groups already exist in the literature, see Table 3.4, and are not treated in this thesis except for special linear groups in Section 5.2.

Group	Reference
$\mathrm{SL}(2,q)$	[27]
$\mathrm{Sp}(4,q)$	[20]
$\mathrm{SU}(4,q)$	[21]
$\Omega^{\pm}(6,q)$	[22]

Table 3.4: Constructive recognition algorithms for base case groups

Properties of the BaseCase algorithms e.g. if a given algorithm is randomised (see Section 1.1.1) or if it requires the discrete logarithm oracle (see Section 1.1.2) depend on the choice of the constructive recognition algorithm for the base case group. Note that the constructive recognition algorithms used for base case groups can readily be exchanged when more efficient constructive recognition algorithms become available. The current state-of-the-art algorithms often call other base case algorithms as displayed in Figure 3.1.

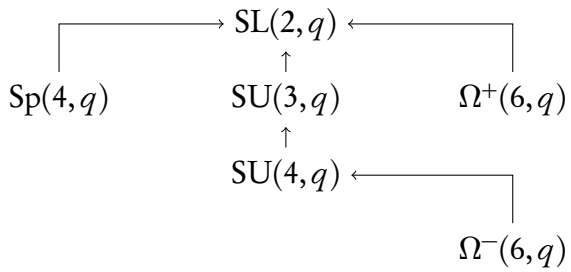


Figure 3.1: Relation of algorithms for constructive recognition of base case groups. We use $B \to A$ to denote that B calls the function A.

The state-of-the-art algorithm for constructive recognition of $\mathrm{SL}(2,q)$ is given in [27]. This algorithm is randomised and uses the discrete logarithm oracle. Since a constructive recognition algorithm to recognise $\mathrm{SL}(2,q)$ is used by all the other constructive recognition algorithms for base case groups, these base case algorithms are also randomised and involve a discrete logarithm oracle. So far, it is unknown whether it is possible to recognise $\mathrm{SL}(2,q)$ constructively without the discrete logarithm oracle.

As for the GoingDown algorithm the success of the BaseCase algorithm depends on the choice of N and for a given $\epsilon \in (0,1)$ we can select N such that the BaseCase algorithm succeeds with probability at least $1-\epsilon$. The choice of N of the BaseCase algorithm is not discussed in this thesis, instead we refer to the literature, see Table 3.4.

3.3 GoingUp algorithm

The last subalgorithm of StandardGenerators is the GoingUp algorithm. The input of this algorithm is X with $\langle X \rangle = G = \mathrm{CL}(d,q)$ in its natural representation and the standard generating set S of a stingray embedded subgroup $\langle S \rangle = H \leq G^{\mathscr{L}}$, where H is a base case group of the same type as G for a known base change matrix $\mathscr{L} \in \mathrm{GL}(d,q)$. Moreover, the standard generators S of H can be written as words in $X^{\mathscr{L}}$, i.e. a constructive recognition algorithm has been used on H. The GoingUp algorithm is also randomised and, therefore, we use N as input to allow at most N selections of random elements. The output of the GoingUp algorithm is an MSLP $\mathfrak{S}$ and a base change matrix $\mathscr{L}'$ such that if $\mathfrak{S}$ is evaluated in $(S \cup X^{\mathscr{L}})^{\mathscr{L}'}$ the output are the standard generators of $G^{\mathscr{L}\mathscr{L}'} = \mathrm{CL}(d,q)$.

Similar to the GoingDown algorithm, the GoingUp algorithm uses a step repeatedly until the standard generators of G have been constructed. The input of the GoingUp step are X and $\widetilde{S}$ with $\langle \widetilde{S} \rangle = \widetilde{H} \leq G^{\widetilde{\mathscr{L}}}$ where $\widetilde{H} \cong \mathrm{CL}(n,q)$ is stingray embedded and of the same type as G for a known base change matrix $\widetilde{\mathscr{L}} \in \mathrm{GL}(d,q)$. The standard generators $\widetilde{S}$ of $\widetilde{H}$ can be written as words in $X^{\widetilde{\mathscr{L}}}$ and are encoded in an MSLP $\widetilde{\mathfrak{S}}$. Moreover, the base change matrix $\widetilde{\mathscr{L}}$ as well as N to control the maximal number of random element selections are additional inputs of the GoingUp step. The output of the GoingUp step is a base change matrix $\mathscr{L} \in \mathrm{GL}(d,q)$ and an MSLP which evaluates in $\widetilde{S} \cup X^{\widetilde{\mathscr{L}}}$ to the standard generators of another stingray embedded classical subgroup K of $G^{\widetilde{\mathscr{L}}}$. Since the output of the GoingUp step is an MSLP evaluating to the standard generators of a subgroup of G we denote this by saying that the input of the i-th step is given by X, $S_{i-1}, \mathscr{L}_{i-1}$ and N and the output group is given by S_i and $\mathscr{L}_i$ with $\langle S_i \rangle = H_{(i)}$ resulting in

$$(S_i, \mathscr{L}_i) \coloneqq \textsc{GoingUpStep}(X, S_{i-1}, \mathscr{L}_{i-1}, N).$$

Note that this is not completely accurate as the output is not directly S_i but rather only an MSLP

which evaluates to the standard generators S_i of $H_{(i)}$. We ignore this small inaccuracy since we know that the MSLP evaluates to S_i. So far we only noted that $H_{(i)} \leq G^{\mathscr{L}_i}$ holds but we know more about the output of the GoingUp step which is $H_{(i-1)}^{\mathscr{L}_{i-1}^{-1}} \leq H_{(i)}^{\mathscr{L}_i^{-1}} \leq G$ and $H_{(i)} \cong \mathrm{CL}(n_i, q)$ is of the same type as G. By setting $H_{(0)} = H$ this yields a chain of subgroups called an *ascending recognition chain*

$$H = H_{(0)}^{\mathscr{L}_0^{-1}} \leq H_{(1)}^{\mathscr{L}_1^{-1}} \leq \ldots \leq H_{(\ell-1)}^{\mathscr{L}_{\ell-1}^{-1}} \leq H_{(\ell)}^{\mathscr{L}_\ell^{-1}} = G^{\mathscr{L}_\ell^{-1}}.$$

In most cases the length of an ascending recognition chain is greater than the length of the descending recognition chain. Since the standard generators of each $H_{(i)}$ are known and $H_{(\ell)} = G$, the standard generators of G are constructed by the GoingUp algorithm.

The GoingUp step is more complicated than the GoingDown basic step and consists of seven phases. Therefore, we are not discussing the details of the GoingUp step in this chapter and refer to a detailed description accompanied by an example in Section 5.3. Instead we continue this chapter by displaying the input and output of the GoingUp step in matrix form.

Each of the $H_{(i)}$ is a stingray embedded classical group of the same type as G, i.e.

$$H_{(i)} = \begin{pmatrix} \mathrm{CL}(n_i, q) & 0 \\ 0 & I_{d-n} \end{pmatrix} \leq G^{\mathscr{L}_i}$$

and the standard generators of $H_{(i)}$ are known. The GoingUp step then roughly proceeds as follows. We identify an element $c \in G^{\mathscr{L}_{i-1}}$ such that

$$\langle H_{(i-1)}, H_{(i-1)}^c \rangle := \begin{pmatrix} \mathrm{CL}(\mathscr{N}(d, n_{i-1}), q) & 0 \\ 0 & I_{d-\mathscr{N}(d,n_{i-1})} \end{pmatrix} \leq G^{\mathscr{L}_i}$$

for a base change matrix $\mathscr{L}_i \in \mathrm{GL}(d, q)$ such that $\langle H_{(i-1)}, H_{(i-1)}^c \rangle$ is stingray embedded as displayed above. The function $\mathscr{N}(d, n)$ depends on the type of the classical group $\mathrm{CL}(d, q)$ and is given in Table 3.5. The reason why $\mathscr{N}(d, n)$ differs for the classical groups is discussed in the corresponding GoingUp chapters of each type of classical group.

Clearly, $H \leq \langle H, H^c \rangle$ and the standard generators for H are known. Using this knowledge and c we can very carefully construct standard generators for $\langle H, H^c \rangle \cong \mathrm{CL}(\mathscr{N}(d, n), q)$. Then we set

Group	$\mathcal{N}(d,n)$
$\mathrm{SL}(d,q)$	$\min\{2n-1,d\}$
$\mathrm{Sp}(d,q)$	$\min\{2n-2,d\}$
$\mathrm{SU}(d,q)$	$\min\{2n-2,d\}$
$\Omega(d,q)$	$\min\{2n-4,d\}$

Table 3.5: Values of $\mathcal{N}(d,n)$ for the classical group $\mathrm{CL}(d,q)$.

$H_{(i)} := \langle H, H^c \rangle$ and the standard generators for $H_{(i)}$ can be derived. Since we want to avoid as many matrix operations as possible the standard generators of $H_{(i)}$ are only encoded in an MSLP instead of performing the actual matrix computations. As for the GoingDown algorithm we introduce a notion for an ascending recognition chain in the next definition.

Definition 3.9

Let $\langle X \rangle = G \in \{\mathrm{SL}(d,q), \mathrm{Sp}(d,q), \mathrm{SU}(d,q), \Omega(d,q)\}$. Then an ascending chain of subgroups of G with

$$H = H_{(0)} \leq H_{(1)} \ldots \leq H_{(\ell-1)} \leq H_{(\ell)} = G,$$

where $H_{(i)} \cong \mathrm{CL}(n_i, q)$ of the same type as G is stingray embedded in G and H is a base case group, is an *ascending recognition chain*.

3.4 StandardGenerators algorithm

In this chapter we describe the fundamental structure of all constructive recognition algorithms of this thesis in a single algorithm StandardGenerators. The StandardGenerators algorithm uses the three subalgorithms GoingDown, BaseCase and GoingUp outlined in the previous sections. The input and output of the StandardGenerators algorithm are as described in the introduction of Chapter 3.

The StandardGenerators algorithm deals with the input $\langle X \rangle = G := \mathrm{CL}(d,q)$ as follows:

1) If G is a base case group, then call the BaseCase algorithm on G and return the output.

2) Call the GoingDown algorithm with input G to construct a base case group U with $U \leq G$.

3) Call the BaseCase algorithm with input U to recognise U constructively.

4) Call the GoingUp algorithm with input U to express the standard generators of G as words in X in an MSLP.

The StandardGenerators algorithm can be displayed in pseudo code as follows.

Algorithm 3: StandardGenerators

Input: ▶ $\langle X \rangle = G \coloneqq \mathrm{CL}(d,q)$ be a classical group except, $\mathrm{Sp}(d,q)$ for d and q even, in its natural representation
▶ $\epsilon \in (0,1)$ such that the algorithm succeeds with probability $1-\epsilon$

Output: **fail OR** $(\mathscr{L}, \mathfrak{S})$ **where**
▶ $\mathscr{L} \in \mathrm{GL}(d,q)$ is a base change matrix and
▶ $\mathfrak{S}$ is an MSLP such that if $\mathfrak{S}$ is evaluated on $X^{\mathscr{L}}$, then the standard generators of $G^{\mathscr{L}}$ are computed

function StandardGenerators(G, ϵ)

1 **if** G **is a base case group then**
2 **return** BaseCase(G)
3 $N_1, N_2, N_3 \leftarrow \mathrm{MaximalRandomSelections}_{\mathrm{CL}}(d,q,\epsilon)$
4 $U, \mathscr{L}_1, \mathfrak{S}_1 \coloneqq \mathrm{GoingDown}(G, N_1)$
5 $\mathfrak{S}_2, \mathscr{L}_2 \coloneqq \mathrm{BaseCase}(U, N_2)$
6 $\mathfrak{S}_3, \mathscr{L}_3 \coloneqq \mathrm{GoingUp}(G^{\mathscr{L}_1\mathscr{L}_2}, U^{\mathscr{L}_2}, N_3)$
7 Concatenate $\mathfrak{S}_1$, $\mathfrak{S}_2$ and $\mathfrak{S}_3$ into a single MSLP $\mathfrak{S}$ **AND** $\mathscr{L} \coloneqq \mathscr{L}_1\mathscr{L}_2\mathscr{L}_3$
8 **return** $(\mathfrak{S}, \mathscr{L})$

Algorithm StandardGenerators yields the structure for the constructive recognition algorithms of this thesis. In the next four chapters we present the details for the subalgorithms GoingDown, BaseCase and GoingUp for each of the classical groups. While the GoingDown algorithm is very similar across the various classical groups we have to deal with different algebraic problems to design efficient GoingUp algorithms in all cases. Moreover, we describe two different GoingUp procedures, one of which is based on linear algebra and the other one is based on involutions. The GoingUp algorithm based on involutions can be proven for all classical groups simultaneously which is done separately in Chapter 9. The big advantage of the linear algebra approach is the efficiency of writing an MSLP from X to the standard generators of $\mathrm{CL}(d,q)$. But this comes at the price of having longer MSLPs than the MSLPs of the approach using involutions. More details about both approaches are

given in Chapter 9. The complexity of the algorithm StandardGenerators in discussed partially in Chapter 10.

We finish this chapter by stating the main theorem for all classical groups of this thesis.

Theorem 3.10

Let $\langle X\rangle = G \in \{\mathrm{SL}(d,q), \mathrm{Sp}(d,q), \mathrm{SU}(d,q), \Omega(d,q)\}$, except $\mathrm{Sp}(d,q)$ for q and d even, and $\epsilon \in (0,1)$. Algorithm StandardGenerators [Alg. 3] is a one-sided Monte Carlo algorithm which given input G and ϵ outputs with probability at least $1-\epsilon$ an MSLP $\mathfrak{S}$ and base change matrix $\mathscr{L}$ such that $\mathfrak{S}$ evaluates from $X^{\mathscr{L}}$ to the standard generators of $G^{\mathscr{L}}$.

Proof. The correctness is proven as displayed in Table 3.6.

	GoingDown	BaseCase	GoingUp
$\mathrm{SL}(d,q)$	5.13	5.27	5.35 and 9.3
$\mathrm{Sp}(d,q)$	6.9	6.10	6.13
$\mathrm{SU}(d,q)$	7.7	7.9	7.11
$\Omega(d,q)$	8.8	8.10	8.12

Table 3.6: Theorems which prove the correctness of the subalgorithms GoingDown, BaseCase and GoingUp used in Algorithm StandardGenerators [Alg. 3].

The function $\mathrm{MaximalRandomSelections}_{\mathrm{CL}}(d,q,\epsilon)$ defines three integers N_1, N_2, N_3 for the maximal number of random selections of the GoingDown, BaseCase and GoingUp algorithm and is used in Line 3 of Algorithm StandardGenerators [Alg. 3]. How N_1, N_2, N_3 are computed is given and justified in Corollary 10.21 for N_1, Remark 10.32 for N_2 and Corollary 10.37 for N_3. Note that we only perform a complexity analysis of the GoingUp algorithm based on a conjecture which is not proven in this thesis. Therefore, we only provide a conjecture about how to choose N_3 based on practical results. □

Theorem 3.11

Suppose Conjecture 10.33 is true. The complexity of Algorithm StandardGenerators [Alg. 3] as stated in Theorem 3.10 is

$$\mathcal{O}(d^3\log(d) + d^2\log(d)\log(\log(d))\log(q) + \frac{\log(d)}{\log(\log(d))}\zeta + \mathfrak{Y}(q) + \mathfrak{Z}(q))$$

where ζ denotes an upper bound on the number of field operations required for computing a random element, $\mathfrak{Y}(q)$ denotes an upper bound on the number of field operations for constructively recognising a base case group and $\mathfrak{Z}(q)$ denotes an upper bound on the required number of field operations for the final step. For a unitary or orthogonal group the complexity of Algorithm StandardGenerators [Alg. 3] as stated in Theorem 3.10 is

$$\mathcal{O}(d^3\log(d)+d^2\log(d)\log(\log(d))\log(q)+\frac{\log(d)}{\log(\log(d))}\zeta+\mathfrak{Y}(q)+\mathfrak{Z}(q)+\log(d)\mathfrak{V}(q))$$

where $\mathfrak{V}(q)$ is an upper bound on the number of field operations for computing a square root.

Proof. Theorem 10.38. □

Chapter 4

Stingray elements

In this chapter we introduce important elements of classical groups which play a critical role in the GoingDown algorithm presented in this thesis as outlined in Chapter 3. Stingray elements have been studied in numerous publications, see [38, 39, 40, 42, 43, 75, 81], and the treatment presented here is based on these references.

In Section 4.1 *stingray elements* are defined and properties of these elements are summarised. Moreover, we introduce randomised methods for computing stingray elements in matrix groups using *pre-stingray candidates*. In Section 4.2 we summarise results from [75] about the proportion of pre-stingray candidates and, therefore, the probability of computing stingray elements by random procedure. In Section 4.3 additional algorithms designed for computing properties of stingray elements are introduced for later use.

4.1 Definition of stingray elements

In this section stingray elements are defined and general properties of stingray elements are summarised. We start with the definition of stingray elements.

Definition 4.1: Definition 2.53

Let $G \leq \mathrm{GL}(d,q)$ and let W be a G-invariant subspace of $\mathbb{F}_q^d$, i.e. $wg \in W$ for all $w \in W$ and $g \in G$. Then G acts *irreducibly* on W if there is no G-invariant subspace of W other than $\{0\}$ and W.

Definition 4.2: [42, Section 3.2]

A matrix $s \in \mathrm{GL}(d,q)$ is a *stingray element* or more precisely a *stingray element of degree m* if s has a $(d-m)$-dimensional 1-eigenspace E_s and s acts irreducibly on a complementary invariant subspace $W_s \leq \mathbb{F}_q^d$ of dimension m. The space W_s is the *stingray body* and E_s is the *stingray tail*.

Example 4.3

Consider the following matrix:

$$\begin{pmatrix}
4 & 1 & 3 & 3 & 2 & 1 & 2 & 4 & 0 & 4 \\
3 & 3 & 4 & 4 & 3 & 4 & 1 & 0 & 3 & 2 \\
2 & 1 & 0 & 4 & 3 & 1 & 1 & 2 & 0 & 3 \\
3 & 1 & 3 & 4 & 4 & 0 & 2 & 0 & 1 & 3 \\
3 & 4 & 1 & 1 & 0 & 3 & 4 & 4 & 1 & 1 \\
1 & 0 & 4 & 4 & 2 & 2 & 1 & 1 & 4 & 2 \\
2 & 4 & 2 & 2 & 0 & 3 & 4 & 2 & 1 & 0 \\
0 & 2 & 2 & 2 & 4 & 0 & 3 & 4 & 2 & 0 \\
3 & 4 & 1 & 1 & 4 & 3 & 4 & 4 & 2 & 1 \\
0 & 2 & 2 & 2 & 2 & 1 & 3 & 2 & 1 & 2
\end{pmatrix}
\xrightarrow{\text{change of basis}}
\left(\begin{array}{cccc|cccccc}
0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 4 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}\right)$$

This element acts irreducibly on a 4-dimensional subspace and fixes a 6-dimensional subspace pointwise. [1] ◀

For an algorithm that uses stingray elements it is important to have an efficient way to find these special elements in a given group. Because the properties of stingray elements are quite rare, a random search cannot be justified and is very unlikely to succeed. Therefore, we introduce two cumulative definitions *stingray candidates* and *pre-stingray candidates* to relax the properties we are looking for such that a random search is much more promising in classical groups. The probability of computing pre-stingray candidates by random selection is summarised in Section 4.2.

Definition 4.4

A matrix $\tilde{s} \in \mathrm{GL}(d,q)$ is a *stingray candidate* if $\tilde{s}$ has a $(d-m)$-dimensional 1-eigenspace and s acts on a complementary invariant subspace.

[1] MediaProduction. *Manta Ray Image*. iStock.com/MediaProduction, Stock ID: 1173588191. Used under standard license. Accessed August 26, 2022.

This raises two relevant questions:

a) How can one find stingray candidates?

b) How can we check if a stingray candidate is a stingray element?

We present a randomised algorithm to compute stingray candidates and determine the probability of successfully finding one. Since stingray candidates still are rare, we relax the properties even further and search for *pre-stingray candidates* from which we can easily construct stingray candidates. The probability of finding pre-stingray candidates is high and has already been studied by Niemeyer and Praeger [75].

The definition of pre-stingray candidates is quite different compared to stingray elements and stingray candidates. The idea is to have a computable property which is useful to construct stingray candidates. The connection is shown in Theorem 4.6.

Definition 4.5: [75, Remark 3.2]

A matrix $\mathfrak{s} \in \mathrm{GL}(d,q)$ is a *pre-stingray candidate* if the characteristic polynomial $\chi_{\mathfrak{s}}(x)$ has an irreducible factor $P(x) \in \mathbb{F}_q[x]$ of degree m over $\mathbb{F}_q$ and no other irreducible factors of degree divisible by m. The irreducible factor $P(x)$ is a *stingray factor*.

The following theorem is the key for Algorithm FindStingrayCandidate [Alg. 4].

Theorem 4.6: [75, Remark 3.2]

Let $G \leq \mathrm{GL}(d,q)$ and let $\mathfrak{s} \in G$ be a pre-stingray candidate, i.e. we can write the characteristic polynomial as $\chi_{\mathfrak{s}}(x) = P_1(x)P_2^{c_2}(x)\dots P_k^{c_k}(x)$ where $P_1,\dots,P_k$ are irreducible polynomials, and $\deg(P_1)$ does not divide $\deg(P_i)$ for $i \geq 2$. Then $\mathfrak{s}^B$ is a stingray candidate where $B = p^\beta \prod_{i=1}^{k}(q^{\deg(P_i)}-1)$ and $\beta = \lceil \log_p(\max\{c_2,\dots,c_k\}) \rceil$.

Proof. [75, Remark 3.2]. □

The following algorithm implements the ideas of Theorem 4.6.

Algorithm 4: FINDSTINGRAYCANDIDATE

Input:
- $\langle X \rangle = H \leq \mathrm{GL}(d,q)$
- A natural number $4 < n \leq d$
- $N \in \mathbb{N}$

Output: **fail OR** $(g, \mathfrak{S}, N')$ **where**
- $g \in H$ is a stingray candidate,
- $\mathfrak{S}$ is an MSLP from X to g and
- $N' \in \mathbb{N}$ where $N - N'$ is the number of random selections that were used

function FINDSTINGRAYCANDIDATE(H, n, N)

```
while N > 0 do
    N ← N − 1
    (h, 𝔖) ← PSEUDORANDOM(H)                    // 𝔖 evaluates on X to h
    χ_h ← characteristic polynomial of h
    {P_i} ← irreducible factors of χ_h
    K ← {P_i | 2 ≤ deg(P_i) ≤ min{2⌈log(n)⌉, ⌈n/2⌉ − 1}}
    if exists P_i ∈ K, m ← deg(P_i) does not divide deg(P_j), j ≠ i then
        B ← as in Theorem 4.6 AND 𝔖' ← MSLP to h^B using 𝔖
        return (h^B, 𝔖', N)
return fail
```

Remark 4.7

Note that the degree of a stingray factor of a pre-stingray candidate is equal to the dimension of the stingray body of a corresponding stingray element. ◀

Thus question a) is solved. For question b) we need to decide efficiently whether a stingray candidate acts irreducibly on the m-dimensional subspace. To examine this, we consider ppd-elements.

Definition 4.8: [80]

Let $\mathfrak{a}, \mathfrak{b} \in \mathbb{N}$ with $\mathfrak{a}, \mathfrak{b} > 1$. A prime r which divides $(\mathfrak{a}^{\mathfrak{b}} - 1)$ is a *primitive prime divisor* (or *ppd* for short) of $\mathfrak{a}^{\mathfrak{b}} - 1$ if $r \nmid \mathfrak{a}^i - 1$ for all $i \in \mathbb{N}$ with $i < \mathfrak{b}$.

Let $q = p^f$ be a prime power and $d, m \in \mathbb{N}$. A matrix $g \in \mathrm{GL}(d,q)$ is a *ppd$(d,q;m)$-element* if there is a primitive prime divisor r of $q^m - 1$ such that r divides the order of g.

Definition 4.9

Let $\mathfrak{s} \in \mathrm{GL}(d,q)$ be a pre-stingray candidate, i.e. the characteristic polynomial $\chi_{\mathfrak{s}}(x)$ has an irreducible factor of degree m over $\mathbb{F}_q$ and no other irreducible factors of degree divisible by m. Then $\mathfrak{s}$ has the *ppd-stingray property* if $\mathfrak{s}$ is a $\mathrm{ppd}(d,q;m)$-element. In that case $\mathfrak{s}^B$ is a *ppd-stingray element* where B is chosen as in Theorem 4.6.

The advantage of a pre-stingray candidate $\mathfrak{s}$ with ppd-stingray property is that the element $\mathfrak{s}^B$ automatically acts irreducibly on the m-dimensional invariant subspace. That means that $\mathfrak{s}^B$ is not only a stingray candidate but rather a stingray element.

Lemma 4.10: [75, Remark 3.2]

Let $G \leq \mathrm{GL}(d,q)$ and let $\mathfrak{s} \in G$ be a pre-stingray candidate with ppd-stingray property. Choose B as in Theorem 4.6. Then $\mathfrak{s}^B$ is a stingray element, i.e. it acts irreducibly on an m-dimensional invariant subspace and fixes a complementary $(d-m)$-dimensional subspace point-wise where $m = \deg(P(x))$ and $P(x)$ is the stingray factor of $\mathfrak{s}$. Moreover, $\mathfrak{s}^B$ is a $\mathrm{ppd}(d,q;m)$-element.

Proof. [75, Remark 3.2]. □

Remark 4.11

Clearly every ppd-stingray element is a stingray element but there are stingray elements which are not ppd-stingray elements. ◀

In order to use ppd-stingray elements in an algorithm, we need an efficient algorithm to check for the ppd-stingray property.

Lemma 4.12

Let $G \leq \mathrm{GL}(d,q)$ and $\mathfrak{s} \in G$ be a pre-stingray candidate, i.e. the characteristic polynomial $\chi_{\mathfrak{s}}(x)$ has an irreducible factor of degree m over $\mathbb{F}_q$ and no other irreducible factors of degree divisible by m. Let $W \leq \mathbb{F}_q^d$ be the $\mathfrak{s}^B$-invariant m-dimensional subspace. Then $\mathfrak{s}$ has the ppd-stingray property if and only if $(\mathfrak{s}|_W)^{(q^m-1)/\Psi} \neq 1$, where $\Psi = \Psi(m,q)$ denotes the product of all primitive prime divisor of q^m-1 with multiplicity.

Proof. The linear transformation $\mathfrak{s}|_W$ induced by $\mathfrak{s}$ on W has order dividing q^m-1. If there is a primitive prime divisor of q^m-1 which also divides $|\mathfrak{s}|_W|$, then $\mathfrak{s}$ has the ppd-stingray property. Since the order of $\mathfrak{s}|_W$ divides q^m-1 and $(\mathfrak{s}|_W)^{(q^m-1)/\Psi} \neq 1$, the natural number Ψ must contain a prime divisor of $|\mathfrak{s}|_W|$. Hence, there exists $r \in \mathbb{N}$ prime with $r \mid \Psi$ and r divides $|\mathfrak{s}|_W|$. Thus, there is a primitive prime divisor of q^m-1 which divides $|\mathfrak{s}|_W|$.

The opposite direction also holds using [75]: Remark 3.2 of [75] states that if $\mathfrak{s} \in G \leq \mathrm{GL}(d,q)$ has the ppd-stingray property, then there exists $r \in \mathbb{N}$ prime with $r \mid q^m-1$ and r divides $|\mathfrak{s}^B|$. Hence, r divides $|(\mathfrak{s}|_W)^B| = |\mathfrak{s}^B|$ and r divides $|\mathfrak{s}|_W|$. Since r is a primitive prime divisor of q^m-1, r does not divide $(q^m-1)/\Psi$ and, therefore, $(\mathfrak{s}|_W)^{(q^m-1)/\Psi} \neq 1$. □

Remark 4.13

$(\mathfrak{s}|_W)^{(q^m-1)/\Psi} \neq 1$ is a non-identity matrix if and only if $x^{(q^m-1)/\Psi} \neq 1$ within the polynomial ring $\mathbb{F}_q[x]/\langle P_1(x)\rangle$ which can be checked efficiently, see [75, Remark 3.2]. ◀

Algorithm FindStingrayCandidate [Alg. 4] and Remark 4.13 can be combined in a single algorithm as Algorithm FindStingrayElement [Alg. 5].

Theorem 4.14 states the correctness of Algorithm FindStingrayElement [Alg. 5] and its termination using at most N random elements. In Chapter 10 we determine how to choose N to control the probability that the algorithm returns an output that is not `fail`.

Theorem 4.14

Algorithm FindStingrayElement [Alg. 5] terminates using at most N random selections and works correctly.

Proof. Clear. □

Algorithm FindStingrayElement [Alg. 5] would require a large amount of tries to reach a high success probability in symplectic and orthogonal groups since pre-stingray candidates with the ppd-stingray property as in Definition 4.9 are quite rare. In order to adapt the stingray elements approach for symplectic and orthogonal groups, the pre-stingray candidates must have an additional property. This is used in Chapter 6 and Chapter 8.

Algorithm 5: FINDSTINGRAYELEMENT

Input:
- $\langle X \rangle = H \leq \mathrm{GL}(d,q)$
- A natural number $4 < n \leq d$
- $N \in \mathbb{N}$

Output: **fail OR** $(g, \mathfrak{S}, N')$ **where**
- $g \in H$ is a ppd-stingray element,
- $\mathfrak{S}$ is an MSLP from X to g and
- $N' \in \mathbb{N}$ where $N - N'$ is the number of random selections that were used

function FINDSTINGRAYELEMENT(H, n, N)

```
while N > 0 do
    N ← N − 1
    (h, 𝔖) ← PSEUDORANDOM(H)                          // 𝔖 evaluates on X to h
    χ_h ← characteristic polynomial of h
    {P_i} ← irreducible factors of χ_h
    K ← {P_i | 2 ≤ deg(P_i) ≤ min{2⌈log(n)⌉, ⌈n/2⌉ − 1}}
    if exists P_i ∈ K, m ← deg(P_i) does not divide deg(P_j), j ≠ i then
        if x^((q^m − 1)/Ψ(m,d)) ≠ 1 within 𝔽_q[x]/⟨P_i(x)⟩ then      // Remark 4.13
            B ← as in Theorem 4.6 AND 𝔖' ← MSLP to h^B using 𝔖
            return (h^B, 𝔖', N)
return fail
```

Definition 4.15: [75, Section 3]

For $\mathbb{F}$ let $\overline{\mathbb{F}}$ denote the algebraic closure. Two polynomials $P_1(x), P_2(x) \in \mathbb{F}_q[x]$ are *conjugate* if for each root $\lambda \in \overline{\mathbb{F}_q}$ of $P_1(x)$ the element λ^{-1} is a root of $P_2(x)$. A polynomial $P(x)$ is *self-conjugate* if $P(x) \in \mathbb{F}_q[x]$ is conjugate to itself.

Definition 4.16

Let $P(x) := \sum_{i=0}^{d} \lambda_i x^i \in \mathbb{F}_q[x]$ be a polynomial of degree d. Then the *reciprocal* or *reflection* of $P(x)$ is $P^R(x) := \sum_{i=0}^{d} \lambda_{d-i} x^i$. We call $P(x)$ *self-reciprocal* or *palindromic* if $P(x) = P^R(x)$ holds.

The connection between self-conjugate and self-reciprocal polynomials shown in the next lemma is

widely-known.

Lemma 4.17

Let $P(x) := \sum_{i=0}^{d} \lambda_i x^i \in \mathbb{F}_q[x]$ be a polynomial of degree d. Then all of the following hold:

1) $P^R(x) = x^d P(x^{-1})$.
2) If $P(x)$ is self-reciprocal, then $P(x)$ is self-conjugate.
3) If $P(x)$ is irreducible, monic, self-conjugate and $d > 1$, then $P(x)$ is self-reciprocal.

Proof.

1) Follows from $x^d P(x^{-1}) = \sum_{i=0}^{d} \lambda_i x^{d-i} = P^R(x)$.

2) By 1) we have $P(x) = x^d P(x^{-1})$ which implies the claim.

3) Let $\iota_1, \ldots, \iota_d \in \overline{\mathbb{F}_q}$ be the roots of $P(x)$, i.e. $P(x) = \prod_{i=1}^{d}(x - \iota_i)$. Since $P(x)$ is irreducible, the ι_i are pairwise distinct. Since $P(x)$ is self-conjugate, for every $1 \leq i \leq d$ there is a unique $1 \leq j \leq d$ such that $\iota_i \iota_j = 1$. Moreover $i \neq j$ as $\iota_i^2 = 1$ would imply $\iota_i = \pm 1$, hence $P(x) = x \pm 1$, but $\deg(P(x)) > 1$. Hence, the roots come in pairs and d is even. Thus the constant term of $P(x)$ is equal to $P(0) = (-1)^d \prod_{i=1}^{d} \iota_i = \prod_{i=1}^{d} \iota_i = 1$.

   Hence, $P^R(x)$ is monic. Moreover, by 1) every root of $P(x)$ is a root of $P^R(x)$. This implies $P(x)$ and $P^R(x)$ have the same roots and the same degree and $P(x)$ divides $P^R(x)$, hence $P = P^R(x)$. □

Algorithm 6: SelfReciprocalCheck

Input: ▶ $P(x) \in \mathbb{F}_q[x]$ with $\sum_{i=0}^{d} \lambda_i x^i = P(x)$

Output: ▶ true if $P(x)$ is self-reciprocal and otherwise false

```
function SelfReciprocalCheck(P)
    if λ_i = λ_{d−i} for all i ∈ {0, ..., ⌊d/2⌋} then
        return true
    else
        return false
```

Lemma 4.17 presents an efficient way to check whether a polynomial of $\mathbb{F}_q[X]$ is self-reciprocal and

Algorithm SelfReciprocalCheck [Alg. 6] is used as a self-reciprocal test in the following.

Remark 4.18

In the following we are interested in self-conjugate, irreducible factors of degree at least 2 of the characteristic polynomial of matrices in $\mathrm{SL}(d,q)$. That means that we are computing monic, irreducible polynomials of degree at least 2 and by Lemma 4.17 being self-conjugate and self-reciprocal is equivalent in these cases. Therefore, we use self-reciprocal instead of self-conjugate in the following. Note that verifying the statement $P(x) = P(x) = x^d P(x^{-1})$ is more costly than comparing equality of λ_i and λ_{d-i} for all $i \in \{0, \ldots, \lfloor \frac{d}{2} \rfloor\}$. ◀

Definition 4.19

Let $G \leq \mathrm{GL}(d,q)$. An element $\mathfrak{s} \in G$ is a *self-reciprocal pre-stingray candidate* if the characteristic polynomial $\chi_{\mathfrak{s}}(x)$ has a self-reciprocal irreducible factor $P(x) \in \mathbb{F}_q[x]$ of degree $2m$ over $\mathbb{F}_q$, no other self-reciprocal irreducible factor of degree divisible by $2m$ and no non-self-reciprocal irreducible factor of degree divisible by m. The irreducible factor $P(x)$ is a *self-reciprocal stingray factor*.

If $\mathfrak{s} \in G$ is a self-reciprocal pre-stingray candidate with the ppd-stingray property, then $\mathfrak{s}^B$ is a *self-reciprocal ppd-stingray element* where B is chosen as in Theorem 4.6.

Self-reciprocal ppd-stingray elements are important for the GoingDown algorithm of the symplectic and orthogonal groups. Since self-reciprocal pre-stingray candidates form a subset of the pre-stingray elements, Algorithm FindStingrayElement [Alg. 5] can be used for self-reciprocal pre-stingray elements while additionally applying Algorithm SelfReciprocalCheck [Alg. 6] to the irreducible factors.

Algorithm 7: FindSelfReciprocalStingrayElement

Input:
- $\langle X \rangle = H \leq \mathrm{GL}(d,q)$
- A natural number $4 < n \leq d$
- $N \in \mathbb{N}$

Output: **fail OR** $(g, \mathfrak{S}, N')$ **where**
- $g \in H$ is a self-reciprocal ppd-stingray element,
- $\mathfrak{S}$ is an MSLP from X to g and
- $N' \in \mathbb{N}$ where $N - N'$ is the number of random selections that were used

function FindSelfReciprocalStingrayElement(H, n, N)

```
while N > 0 do
    N ← N − 1
    (h, 𝔖) ← PseudoRandom(H)                    // 𝔖 evaluates on X to h
    χ_h ← characteristic polynomial of h
    {P_i} ← irreducible factors of χ_h
    K ← {P_i | 2 ≤ deg(P_i) ≤ min{2⌈log(n)⌉, ⌈n/2⌉ − 1}, deg(P_i) is even, SelfReciprocalCheck(P_i)}
    if exists P_i ∈ K, 2m ← deg(P_i) does not divide deg(P_j), j ≠ i if SelfReciprocalCheck(P_j)
     AND m does not divide deg(P_j), j ≠ i if not(SelfReciprocalCheck(P_j)) then
        if x^((q^m − 1)/Ψ(m,d)) ≠ 1 within 𝔽_q[x]/⟨P_i(x)⟩ then
            B ← as in Theorem 4.6 AND 𝔖' ← MSLP to h^B using 𝔖
            return (h^B, 𝔖', N)
return fail
```

Theorem 4.20

Algorithm FindSelfReciprocalStingrayElement [Alg. 7] terminates using at most N random selections and works correctly.

Proof. Clear. □

Lastly, we need one final lemma about the stingray body of stingray elements in this thesis.

Lemma 4.21: [42, Lemma 3.9(a)]

Let $s \in \mathrm{CL}(d,q)$ be a stingray element with stingray body W_s and stingray tail E_s. Then we have $\mathbb{F}_q^d = W_s \oplus E_s$. Moreover, W_s and E_s are non-degenerate if $\mathrm{CL}(d,q)$ preserves a non-degenerate form.

Proof. The decomposition $\mathbb{F}_q^d = W_s \oplus E_s$ is clear since W_s is a complementary subspace of E_s. It is left to show that W_s and E_s are non-degenerate if $\mathrm{CL}(d,q)$ preserves a non-degenerate form Φ. For the proof of that we follow [42, Lemma 3.9(a)]. Let $w \in W_s$ and $v \in E_s$. Then

$$\Phi(w,v) = \Phi(wg, vg) = \Phi(wg, v)$$

which is equivalent to $\Phi(w - wg, v) = 0$. Since $s - 1$ is non-singular on W_s, we can conclude that $\{wg - w \mid w \in W_s\} = W_s$. Therefore, $\Phi(w, v) = 0$ for all $w \in W_s$ and hence $v \in (W_s)^\perp$. Since $v \in E_s$ was arbitrary, $E_s \leq (W_s)^\perp$. Using 2) of Lemma 2.14 we can also conclude that $\dim((W_s)^\perp) = \dim(\mathbb{F}_q^d) - \dim(W_s) = d - \dim(W_s) = \dim(E_s)$ since $\mathbb{F}_q^d = W_s \oplus E_s$ and, hence, it follows that $(W_s)^\perp = E_s$. Thus, $W_s \cap (W_s)^\perp = W_s \cap E_s = \{0\}$ and, therefore, W_s is non-degenerate. Similarly, it can be shown that E_s is non-degenerate. □

In the algorithms of this thesis we are not dealing with single stingray elements but instead with pairs of stingray elements as given in the next definition.

Definition 4.22

If $s_1, s_2 \in \mathrm{GL}(d, q)$ are stingray elements, then (s_1, s_2) is a *stingray pair*. Moreover, in special linear groups a stingray pair (s_1, s_2) is a *stingray duo* if $W_{s_1} \cap W_{s_2} = \{0\}$ where W_{s_i} is the stingray body of s_i for $i \in \{1, 2\}$. In symplectic, unitary and orthogonal groups a stingray pair (s_1, s_2) is a *stingray duo* if W_{s_i} is non-degenerate for $i \in \{1, 2\}$ and $W_{s_1} \cap W_{s_2} = \{0\}$ and $W_{s_1} \oplus W_{s_2}$ is non-degenerate where W_{s_i} is the stingray body of s_i for $i \in \{1, 2\}$.

Remark 4.23

Note that in symplectic, unitary and orthogonal groups if $s_1, s_2 \in \mathrm{GL}(d, q)$ are ppd-stingray elements, then W_{s_i} is non-degenerate for $i \in \{1, 2\}$ using Lemma 4.21 where W_{s_i} is the stingray body of s_i. Hence, we only need to verify that $W_{s_1} \cap W_{s_2} = \{0\}$ and $W_{s_1} \oplus W_{s_2}$ is non-degenerate. ◀

4.2 Proportion results about stingray elements

In order to comment on the efficiency of the algorithms designed in this thesis, it is important to know the proportion of pre-stingray candidates in G, i.e. the probability that a pre-stingray candidate of G is chosen by one random selection in G. Niemeyer and Praeger [75] have already dealt with this question and we summarise the most important results below.

Let $H = \mathrm{CL}(d, q)$. In the GoingDown basic step we select random elements until we find a (self-reciprocal) pre-stingray candidate as outlined in Chapter 3. The self-reciprocal property is omitted in the case of special linear groups and special unitary groups. In this section we summarise results

about the probability that a random element is a (self-reciprocal) pre-stingray candidate based on the results by Niemeyer and Praeger [75]. In Section 10.1 we extend some of the results of this section to be more fitting to the algorithms designed in this thesis.

Since H is finite it is sufficient to compute the size of the set of all (self-reciprocal) pre-stingray candidates in order to compute the probability that a random element of H is a (self-reciprocal) pre-stingray candidate. The sets are defined as follows.

Definition 4.24: [75, Definition 3.1]

Let H be a d-dimensional classical group defined over a finite field with q elements, as in one of the rows of Table 4.1. Let k be an integer such that $1 \leq k \leq d$.

(i) Let $\amalg_k$ denote the set of all elements $h \in H$ with characteristic polynomial $\chi_h(x)$ for which one of the following conditions holds:

(a) $H = \mathrm{SL}(r,q)$ or $\mathrm{SU}(r,q)$ and h is a pre-stingray candidate with stingray factor of degree k where k is odd in case $\mathrm{SU}(r,q)$.

(b) $H = \mathrm{Sp}(2r,q)$, $\mathrm{SO}^{\pm}(2r,q)$ or $\mathrm{SO}^{\circ}(2r+1,q)$ and h is a self-reciprocal pre-stingray candidate with self-reciprocal stingray factor of degree k.

(ii) Let $\amalg_k^{\mathrm{ppd}}$ denote the subset of $\amalg_k$ consisting of those elements h which also have the ppd-stingray property.

The integer r of Definition 4.24 depends on the entry in Table 4.1.

H	d	α	δ
$\mathrm{SL}(r,q)$	r	1	1
$\mathrm{SU}(r,q)$	r	1	2
$\mathrm{Sp}(2r,q)$	$2r$	2	1
$\mathrm{SO}^{\circ}(2r+1,q)$	$2r+1$	2	1
$\mathrm{SO}^{\pm}(2r,q)$	$2r$	2	1

Table 4.1: Groups and constants in Theorem 4.25 and 4.26 [75, Table 1].

$\amalg$	$\amalg_k$	$\amalg_k^{\mathrm{ppd}}$
$\ell_{k,\amalg}$	$1-\frac{2}{q^{k/2}}$	$1-\frac{1}{\alpha k}$
$m_{r,\amalg}$	$1-\frac{2}{q^{\log(r)/2}}$	$1-\frac{1}{\alpha \log(r)}$

Table 4.2: Definitions of $\ell_{k,\amalg}$ and $m_{r,\amalg}$ in Theorem 4.25 [75, Table 2].

Theorem 4.25: [75, Theorem 3.3]

Let q be a prime power and $\mathbb{F}_q$ a finite field with q elements. Let H, d, α and δ be as in one of the rows of Table 4.1. Let k be a positive integer with $\max\{3, \log(r)\} \leq k \leq r/2$ and such that k is odd when $H = \mathrm{SU}(r,q)$. Let $\amalg$ be one of $\amalg_k$ or $\amalg_k^{\mathrm{ppd}}$ such that $\amalg \neq \emptyset$, and let $\ell_{k,\amalg}$ and $m_{r,\amalg}$ be as in Table 4.2. Then

1) $\frac{\ell_{k,\amalg}}{3e\alpha k} \leq \frac{|\amalg|}{|H|} \leq \frac{5}{3\alpha k}$ and
2) if $\log(r) < k \leq 2\log(r)$, then $\frac{m_{r,\amalg}}{6e\alpha \log(r)} \leq \frac{|\amalg|}{|H|} \leq \frac{5}{3\alpha \log(r)}$.

Proof. [75, Theorem 3.3]. □

Theorem 4.26: [75, Theorem 1.1]

Let H be a d-dimensional classical group defined over a finite field with q elements, as in one of the rows of Table 4.1. Let r, α be as in Table 4.1, and let $k \in \mathbb{N}$ such that $\max\{3, \log(r)\} \leq k \leq r/2$ and k odd if $H = \mathrm{SU}(r,q)$. Let $\amalg$ be the set $\amalg_k^{\mathrm{ppd}}$. Each element of $\amalg$ powers up to an element which has a $(d - \alpha k)$-dimensional 1-eigenspace and acts irreducibly on a complementary invariant subspace of dimension αk. Moreover,

$$\frac{2}{9e\alpha k} \leq \frac{|\amalg|}{|H|} \leq \frac{5}{3\alpha k}.$$

Proof. [75, Theorem 1.1]. □

The two theorems above prove that the probability of finding a pre-stingray candidate with ppd-stingray property by random search is promising in classical groups.

Remark 4.27

Notice that there is a slight error in [75, Theorem 1.1] as not all elements of $\amalg_k$ power up to an element that acts irreducibly on a complementary invariant subspace of dimension αk but this has no effect on the results of this thesis as we are only using elements of $\amalg_k^{\mathrm{ppd}}$. ◀

Remark 4.28

Note that the proportion results of Theorem 4.26 are given for special linear groups and unitary groups based on stingray elements and for symplectic groups and orthogonal groups based on

self-reciprocal stingray elements as outlined in Definition 4.24. Therefore, we design algorithms using stingray elements in special linear groups and unitary groups and self-reciprocal stingray elements in symplectic groups and orthogonal groups in this thesis. ◀

4.3 Algorithms for stingray elements

In this section more important algorithms are introduced and analysed, e.g. for computing the stingray body and stingray tail of stingray elements. Moreover, since stingray elements act irreducibly and invariantly on a subspace while fixing a complementary subspace, these elements are stingray embedded and we design an algorithm for extracting the stingray embedded group such that further computations can be performed using matrices of smaller degree and are thus more efficient.

First, given a stingray element, we require an algorithm to compute the stingray body (see Algorithm ComputeStingrayBody [Alg. 9]) and the stingray tail (see ComputeStingrayTail [Alg. 8]). These algorithms rely solely on basic linear algebra.

Algorithm 8: ComputeStingrayTail

Input: ▶ $s \in \mathrm{GL}(d,q)$ a stingray element

Output: ▶ $E_s \leq \mathbb{F}_q^d$ the stingray tail, i.e. the 1-eigenspace, of s

function ComputeStingrayTail(s)

1 $E_s \leftarrow$ Kernel($s - I_d$)

2 **return** E_s

Algorithm 9: ComputeStingrayBody

Input: ▶ $s \in \mathrm{GL}(d,q)$ a stingray element

Output: ▶ $W_s \leq \mathbb{F}_q^d$ the stingray body of s, i.e. s acts irreducibly and invariantly on W_s

function ComputeStingrayBody(s)

1 $W_s \leftarrow$ Image($s - I_d$)

2 **return** W_s

Lemma 4.29

Algorithm ComputeStingrayTail [Alg. 8] and ComputeStingrayBody [Alg. 9] work correctly and terminate.

Proof. Clearly the two algorithms terminate. Since s is a stingray element, $\mathbb{F}_q^d$ decomposes into $\mathbb{F}_q^d = W_s \oplus E_s$ where E_s is the stingray tail of s and W_s is the stingray body. Note that $s - I_d$ is the unique $\langle s \rangle$-invariant submodule of $\mathbb{F}_q^d$ on which s acts non-trivially and irreducible, see [42, Lemma 3.7(c)]. This is exactly what the algorithms compute. □

Using Algorithm ComputeStingrayBody [Alg. 9] it is, therefore, easy to verify whether the stingray bodies of (ppd-)stingray elements intersect trivially and, thus, whether a pair of (ppd-)stingray elements forms a stingray duo. An algorithm to check whether a pair of ppd-stingray elements is a stingray duo is given by Algorithm IsStingrayDuo [Alg. 10].

Algorithm 10: IsStingrayDuo

Input: ▶ $(s_1, s_2) \in \mathrm{CL}(d,q)$ a stingray pair of ppd-stingray elements
▶ Φ a preserved form by $\mathrm{CL}(d,q)$ (omitted if $\mathrm{CL}(d,q) = \mathrm{SL}(d,q)$)

Output: ▶ true if (s_1, s_2) is a stingray duo as in Definition 4.22 and otherwise false

```
function IsStingrayDuo((s_1, s_2), Φ)
1   W_{s_1} ← ComputeStingrayBody(s_1)
2   W_{s_2} ← ComputeStingrayBody(s_2)
3   W ← write basis of W_{s_1} and W_{s_2} into one matrix
4   if Kernel(W) = {0} then
5       if IsNonDegenerate(W, Φ) then        // Note that this check is omitted in special linear groups
6           return true
7   return false
```

The performance of the constructive recognition algorithms of this thesis can be improved by using lower-dimensional matrices. This can be achieved by observing the action of stingray elements on their stingray bodies. This corresponds to extracting the non-trivial block of a stingray element after applying a suitable base change, e.g. extracting the non-trivial block of the right matrix of Example 4.3. An algorithm to compute the action of a (ppd-)stingray element on its stingray body is given

by Algorithm InducedActionRepresentation [Alg. 11].

Algorithm 11: InducedActionRepresentation

Input: ▶ $s \in \mathrm{GL}(d,q)$ a (ppd-)stingray element

Output: ▶ $s' \in \mathrm{GL}(m,q)$ which corresponds to the linear action of s on its stingray body

function InducedActionRepresentation(s)

1 $W_s \leftarrow$ ComputeStingrayBody(s)

2 $\mathcal{B} \leftarrow$ Basis(W_s)

3 $s' \leftarrow$ Apply($\mathcal{B}$, $b \to$ Coefficients($\mathcal{B}$, $b \cdot s$))

4 **return** s'

Remark 4.30

Line 3 of Algorithm InducedActionRepresentation [Alg. 11] is executed as follows: Since the stingray body W_s of a (ppd-)stingray element $s \in \mathrm{GL}(d,q)$ is invariant under the action of s, we can choose a basis $(b_1,\ldots,b_m)$ of W_s and $b_i s \in W_s$ for $1 \leq i \leq m$. Hence, for $1 \leq i \leq m$ we can express $b_i s$ as a linear combination of $(b_1,\ldots,b_m)$ and write the coefficients of this linear combination as a row in a matrix which results in an $m \times m$ matrix describing the action of s on the stingray body W_s. ◀

Lastly, we represent a group which is generated by stingray elements of $\mathrm{GL}(d,q)$ as lower-dimensional matrices by observing the action of the group on an irreducible and invariant subspace of $\mathbb{F}_q^d$. As a group generated by stingray elements is also stingray embedded in $\mathrm{GL}(d,q)$, computing this new representation corresponds to extracting H of Definition 3.3. We cannot call Algorithm InducedActionRepresentation [Alg. 11] on both stingray elements because we assume that the stingray bodies of stingray elements intersect trivially and, therefore, Algorithm InducedActionRepresentation [Alg. 11] uses different base change matrices for extracting the non-trivial block of the stingray elements. We solve this problem by observing the action of the stingray elements on the sum of their stingray bodies which is achieved by Algorithm InducedActionRepresentationGroup [Alg. 12].

Algorithm 12: InducedActionRepresentationGroup

Input: ▶ $U \leq \mathrm{GL}(d,q)$ generated by (ppd-)stingray elements

Output: ▶ $\tilde{U} \leq \mathrm{GL}(\tilde{d},q)$ which is generated by the linear action of the generators of U on the sum of their stingray bodies

```
function InducedActionRepresentationGroup(U)
    A ← [ ]
    for s ∈ Generators(U) do
        W_s ← ComputeStingrayBody(s)
        Append(A, W_s)
    B ← Basis(⟨A⟩)
    Ũ ← [ ]
    for s ∈ Generators(U) do
        Append(Ũ, Apply(B, b → Coefficients(B, b · s)))
    return ⟨Ũ⟩
```

Chapter 5

Special linear group

This chapter details an efficient realisation of the strategy outlined in Chapter 3 for the special linear group $\mathrm{SL}(d,q)$ in its natural representation for $d \geq 4$. If $d = 2$, then we refer to [27], and if $d = 3$, then we refer to [65]. This chapter builds upon preliminary concepts and ideas from Ákos Seress and Max Neunhöffer who had the idea of using stingray elements for a constructive recognition algorithm.

We start this chapter by introducing standard generators for special linear groups and proving how arbitrary transvections can be computed as words in this generating set. Afterwards we describe the subalgorithms as outlined in Section 3.1.

Section 5.1 deals with the GoingDown algorithm and is divided into four subsections. Given the matrix group $\langle X \rangle = G = \mathrm{SL}(d,q)$, the aim is to compute a subgroup U of G with $U \cong \mathrm{SL}(2,q)$. Section 5.1.1 introduces the GoingDown basic step, while Section 5.1.2 applies the GoingDown basic step iteratively in the main GoingDown algorithm. Given $U_i \leq G$ with $U_i \cong \mathrm{SL}(d_i,q)$ the goal of the GoingDown basic step is to compute a subgroup U_{i+1} of U_i with $U_{i+1} \cong \mathrm{SL}(d_{i+1},q)$ and $d_{i+1} \leq 4\lceil \log(d_i) \rceil$. Stingray elements play a major role in this process. In Section 5.1.2 the GoingDown basic step is employed to design an algorithm for computing a descending recognition chain of subgroups of G as in Definition 3.4 terminating in a subgroup isomorphic to $\mathrm{SL}(4,q)$, i.e. a terminal group as in Definition 3.6. The correctness of this algorithm is also proven. In Section 5.1.3 we discuss why the GoingDown basic step in Section 5.1.1 cannot be used to compute

a subgroup isomorphic to $\mathrm{SL}(2,q)$. Therefore, we introduce another approach by Leedham-Green and O'Brien and explain why their methods are applicable. The approach by Leedham-Green and O'Brien is used as the final step and for computing the full descending recognition chain as outlined in Chapter 3. In Section 5.1.4 a variation of the GoingDown basic step is discussed which yields a faster practical run-time and incurs no additional costs regarding the MSLPs. Additionally, another idea is illustrated to improve the run-time even further at the cost of longer MSLPs. Lastly, in Section 5.1.5 we combine the GoingDown basic step of Section 5.1.2 and final step of Section 5.1.3 into a single GoingDown algorithm for special linear groups.

Section 5.2 deals with the BaseCase algorithm, i.e. given $\langle X_U \rangle = U = \mathrm{SL}(2,q)$ the standard generators of U are expressed as words in X_U. For this we rely on an algorithm for constructive recognition of $\mathrm{SL}(2,q)$, see [27]. We do not discuss the details of [27] but justify it is applicable in our situation.

Section 5.3 introduces the GoingUp algorithm. We prove the correctness of two different approaches for GoingUp basic steps. The first one is based on linear algebra and uses ideas which are completely new and have not been published previously. The second approach produces shorter MSLPs and is based on involutions and ideas from [59] but in contrast to our first approach is only applicable when the characteristic is odd and has worse run-time in practice.

We start this chapter by introducing the standard generators of special linear groups. As in Chapter 2, $q = p^f$ denotes a prime power, $(\omega_1, \ldots, \omega_f)$ an $\mathbb{F}_p$-basis of $\mathbb{F}_q$ and d, n are natural numbers with $n \leq d$. Let $V = \mathbb{F}_q^d$ with basis $\{b_1, \ldots, b_d\}$. The matrices $E_{i,j}(\iota)$ are as defined in Section 2.1.

Definition 5.1

Let $d \geq 2$ and let $S \subseteq \mathrm{SL}(d,q)$. Then S is a set of *standard generators* for $\mathrm{SL}(d,q)$ if S is conjugate to the following set consisting of $2f+2$ elements:

- $E_{1,2}(\omega_i)$ for $1 \leq i \leq f$,
- $E_{2,1}(\omega_i)$ for $1 \leq i \leq f$,
- a permutation matrix z_1 corresponding to the permutation $(1, d, d-1, \ldots, 2)$ with the entry $(z_1)_{1,d}$ changed to -1 if d is even and
- a permutation matrix z_2 corresponding to the permutation $(2, d, d-1, \ldots, 3)$ with the entry $(z_2)_{2,d}$ changed to -1 if d is odd.

Remark 5.2

The negative entries are introduced to ensure that the elements are contained in $\mathrm{SL}(d,q)$. ◀

The following lemma shows how arbitrary transvections can be expressed in terms of the standard generators and computes upper bounds for the length of these words.

Lemma 5.3

Let $S \subseteq \mathrm{SL}(d,q)$ be the set of standard generators with respect to the basis $(b_1,\ldots,b_d)$ and let $\mathscr{M} = [S, I_d, I_d, I_d]$.

1) Let $\iota \in \mathbb{F}_q$ and $\iota = \sum_{j=1}^{f} \lambda_j \omega_j$ for $\lambda_j \in \mathbb{F}_p$. By identifying the finite field element $\lambda_j \in \mathbb{F}_p$ with an integer in $\{0,\ldots,p-1\}$, this yields

$$E_{1,2}(\iota) = \prod_{j=1}^{f} E_{1,2}(\omega_j)^{\lambda_j} \qquad \text{and} \qquad E_{2,1}(\iota) = \prod_{j=1}^{f} E_{2,1}(\omega_j)^{\lambda_j}.$$

Moreover, there exists a $(2f+5)$-MSLP $\mathfrak{S}$ of length at most $f - 1 + 2\log_2(q)$ such that $\mathrm{Eval}_{\mathscr{M}}(\mathfrak{S}) = [E_{1,2}(\iota)]$ or $\mathrm{Eval}_{\mathscr{M}}(\mathfrak{S}) = [E_{2,1}(\iota)]$.

2) Moreover,

$$E_{1,i}(\iota)^{z_2^{-1}} = E_{1,i+1}(\iota) \qquad \text{for } 2 \le i \le d-1,$$
$$E_{i,1}(\iota)^{z_2^{-1}} = E_{i+1,1}(\iota) \qquad \text{for } 2 \le i \le d-1 \text{ and}$$
$$E_{i,j}(\iota)^{z_1^{-1}} = E_{i+1,j+1}(\iota) \qquad \text{for } 2 \le j \le d-1 \text{ and } 1 \le i \le d-2.$$

Proof. 1) If $k \in \{0,\ldots,p-1\}$, then $E_{1,2}(\omega_j)^k = E_{1,2}(k\omega_j)$ and hence

$$E_{1,2}(\iota) = E_{1,2}\Big(\sum_{j=1}^{f} \lambda_j \omega_j\Big) = \prod_{j=1}^{f} E_{1,2}(\lambda_j \omega_j) = \prod_{j=1}^{f} E_{1,2}(\omega_j)^{\lambda_j}.$$

Using repeated squaring the computation of $E_{1,2}(\omega_j)^{\lambda_j}$ needs at most $2\log_2(p)$ operations. Since computing each of the f matrices $E_{1,2}(\omega_j)^{\lambda_j}$ thus requires at most $2\log_2(p)$ operations and multiplying the resulting f matrices requires further $f-1$ operations, we need at most $f - 1 + f\,2\log_2(p) = f - 1 + 2\log_2(q)$ operations. This can be done analogously for $E_{2,1}(\iota)$.

2) First, we prove that $E_{1,i}(\iota)^{z_2^{-1}} = E_{1,i+1}(\iota)$ by showing $b_k E_{1,i}(\iota)^{z_2^{-1}} = b_k E_{1,i+1}(\iota)$ for $1 \le k \le d$.

Note that $b_1 z_2 = b_1$. Hence,

$$b_1 E_{1,i}(\imath)^{z_2^{-1}} = b_1 z_2 E_{1,i}(\imath) z_2^{-1} = b_1 E_{1,i}(\imath) z_2^{-1} = (b_1 + \imath b_i) z_2^{-1} = b_1 z_2^{-1} + \imath b_i z_2^{-1} = b_1 + \imath b_{i+1},$$
$$b_2 z_2 E_{1,i}(\imath) z_2^{-1} = b_d E_{1,i}(\imath) z_2^{-1} = b_d z_2^{-1} = b_2 \quad \text{and}$$
$$b_k z_2 E_{1,i}(\imath) z_2^{-1} = b_{k-1} E_{1,i}(\imath) z_2^{-1} = b_k \quad \text{for } k \in \{3, \ldots, d\}.$$

Hence, $E_{1,i}(\imath)^{z_2^{-1}} = E_{1,i+1}(\imath)$ and analogously it can be shown that $E_{i,1}(\imath)^{z_2^{-1}} = E_{i+1,1}(\imath)$. Second, we show that $E_{i,j}(\imath)^{z_1^{-1}} = E_{i+1,j+1}(\imath)$. We have

$$b_{i+1} E_{i,j}(\imath)^{z_1^{-1}} = b_{i+1} z_1 E_{i,j}(\imath) z_1^{-1} = (b_i + \imath b_j) z_1^{-1} = b_{i+1} + \imath b_{j+1},$$
$$b_1 z_1 E_{i,j}(\imath) z_1^{-1} = b_d E_{i,j}(\imath) z_1^{-1} = b_1 \quad \text{and}$$
$$b_k z_1 E_{i,j}(\imath) z_1^{-1} = b_{k-1} E_{i,j}(\imath) z_1^{-1} = b_k \quad \text{for } k \in \{2, \ldots, d\} \setminus \{i+1\}.$$

Hence, $E_{i,j}(\imath)^{z_1^{-1}} = E_{i+1,j+1}(\imath)$. □

Example 5.4

We visualise the algorithms of this chapter using the following example matrices of $\mathbb{F}_5^{16\times 16}$:

$$\underbrace{\begin{pmatrix}
3&2&1&3&4&0&3&1&2&1&4&2&4&0&1&0\\
2&2&0&4&2&2&4&1&3&2&4&4&1&4&2&3\\
3&4&1&1&3&3&1&4&2&3&1&1&4&1&3&2\\
4&1&4&0&2&1&4&2&2&0&3&0&4&2&0&4\\
0&4&4&1&4&2&1&0&1&1&0&2&2&4&1&3\\
1&2&4&3&4&4&3&3&0&2&2&4&0&1&2&2\\
4&3&1&2&1&2&3&2&0&3&3&1&0&4&3&3\\
2&4&3&1&3&1&1&2&0&4&4&3&0&2&4&4\\
3&1&2&4&2&4&4&4&1&1&1&2&0&3&1&1\\
0&1&1&4&2&3&4&0&4&0&0&3&3&1&4&2\\
4&4&2&1&3&0&1&2&4&2&4&4&3&0&2&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0\\
0&2&2&3&4&1&3&0&3&3&0&1&2&2&3&4\\
4&3&1&2&1&2&2&2&0&3&3&1&0&0&3&3\\
0&1&1&4&2&3&4&0&4&4&0&3&3&1&0&2\\
2&0&4&0&0&4&0&1&4&3&4&1&3&3&3&2
\end{pmatrix}}_{:= a^{\Xi_1}},
\underbrace{\begin{pmatrix}
1&0&3&2&0&4&4&3&3&1&2&1&4&4&2&1\\
2&1&2&3&2&4&1&2&3&1&2&2&0&0&0&0\\
1&2&1&1&1&1&1&4&3&2&1&4&2&0&0&3\\
4&1&0&0&1&2&3&0&3&4&1&2&3&0&2&0\\
1&0&3&0&0&3&3&0&4&1&3&0&0&1&1&1\\
0&2&3&3&3&2&4&3&0&3&1&3&4&0&1&2\\
0&1&3&1&0&1&4&0&4&2&1&4&3&0&3&0\\
3&4&4&0&2&0&0&4&4&0&2&2&1&1&2&4\\
1&0&2&3&0&4&4&4&0&0&0&3&2&1&0&1\\
4&1&1&0&1&1&3&1&4&2&4&3&2&0&1&2\\
0&4&0&3&4&2&3&1&2&0&4&4&4&3&3&3\\
4&3&2&1&3&0&0&0&1&3&4&3&0&3&1&4\\
1&2&1&3&1&4&2&2&3&2&1&2&3&0&1&0\\
3&4&0&0&4&1&0&4&3&2&2&1&2&3&4&2\\
0&3&1&3&3&1&0&3&3&0&3&0&4&1&1&4\\
1&1&1&3&3&4&3&3&4&3&3&4&4&3&0&0
\end{pmatrix}}_{:= a^{\Xi_2}}$$

and $G^{\Xi} := \langle a^{\Xi_1}, a^{\Xi_2} \rangle$. The symbol Ξ, which is the Greek capital letter xi, is used in this chapter to refer to groups and group elements mentioned in previous examples. The symbol itself has no meaning except for recalling these previous examples. ◀

5.1 GoingDown algorithm

Stingray elements are key to the GoingDown algorithm as they are used in the GoingDown basic step to find a subgroup of $\mathrm{SL}(d,q)$ isomorphic to $\mathrm{SL}(d',q)$ quickly, where $d' \leq 4\lceil \log(d) \rceil$. Later in this section we study how the GoingDown basic step can be implemented while the idea of the basic step is illustrated in Figure 5.1. Repeated calls to the GoingDown basic step yield a descending recognition chain as in Definition 3.4, that is

$$\mathrm{SL}(4,q) \cong U_k \leq U_{k-1} \cong \mathrm{SL}(d_{k-1},q) \leq \ldots \leq U_1 \cong \mathrm{SL}(d_1,q) \leq U_0 = G,$$

where $d_i \leq 4\lceil \log(d_{i-1}) \rceil$ for $2 \leq i \leq k$. In Section 5.1.2 Algorithm GoingDownToDim4 [Alg. 14] implements this strategy and we prove its correctness and analyse its complexity. Unfortunately, the GoingDown basic step is not applicable to find a subgroup U of G isomorphic to the base case group $\mathrm{SL}(2,q)$. The reason for this is explained in more detail in Remark 5.16. In Section 5.1.3 we introduce a method of Leedham-Green and O'Brien [59] to construct a subgroup of $\mathrm{SL}(4,q)$ isomorphic to $\mathrm{SL}(2,q)$ which is our final step as outlined in Chapter 3.

Lastly, a variation of the GoingDown basic step is introduced in Section 5.1.4 which yields a practical run-time improvement while having no effect on the length and memory quota of the produced MSLPs. In Section 5.1.4 we also discuss an approach for improving the run-time at the cost of longer MSLPs.

5.1.1 GoingDown basic step

Let $G = \mathrm{SL}(d,q)$. In this section we describe and prove the correctness of one GoingDown basic step. Given a stingray embedded subgroup H of G with $H \cong \mathrm{SL}(d',q)$ for $d' \leq d$ as in Definition 3.3, the aim of the basic step is to compute a stingray embedded subgroup U of H, and thus U is also a stingray embedded subgroup of G, with $U \cong \mathrm{SL}(d'',q)$ and $d'' \leq 4\lceil \log(d') \rceil$. The algorithm to compute the subgroup U is a randomised algorithm using stingray elements. Therefore, this section heavily relies on Chapter 4 and the algorithms presented there.

We start with a sketch of the idea. Let $s_1, s_2 \in \mathrm{GL}(d,q)$ be two stingray elements, let $W_{s_i} \leq \mathbb{F}_q^d$ be their respective stingray bodies, $\dim(W_{s_i}) = n_i$ and E_{s_i} their respective stingray tails for $i \in \{1,2\}$, where $n_1 + n_2 \leq 4\lceil \log(d) \rceil$. As in Example 4.3, there are base change matrices $\mathscr{L}_i$ such that the

stingray element $s_i^{\mathscr{L}_i}$ is a matrix with a non-trivial $n_i \times n_i$ block in the upper left hand corner and 1s on the remaining diagonal. The base change matrix $\mathscr{L}_i$ can be computed by appending a basis for the stingray tail E_{s_i} to a basis for the stingray body W_{s_i} for $i \in \{1,2\}$. This is illustrated by the upper left gray block in Figure 5.1. With high probability the base change matrices $\mathscr{L}_1$ and $\mathscr{L}_2$ are different. However, since n_1 and n_2 are relatively small compared to d, the elements s_1 and s_2 must have a large common 1-eigenspace, namely $E_{s_1} \cap E_{s_2}$. We consider a base change matrix $\mathscr{L}_{s_1,s_2} \in \mathrm{GL}(d,q)$ that arises if we choose a basis for $W_{s_1} + W_{s_2}$ and append a basis for the large common 1-eigenspace $E_{s_1} \cap E_{s_2}$ of s_1 and s_2. Let $n := \dim(W_{s_1} + W_{s_2}) = d - \dim(E_{s_1} \cap E_{s_2})$. The actions of the stingray elements $s_1^{\mathscr{L}_{s_1,s_2}}$ and $s_2^{\mathscr{L}_{s_1,s_2}}$ on $W_{s_1} + W_{s_2}$ are represented with orange blocks in Figure 5.1. It is important that the base change matrix is identical for s_1 and s_2. With high probability $W_{s_1} \cap W_{s_2} = \{0\}$, $\langle s_1, s_2 \rangle$ acts irreducibly on $W_{s_1} + W_{s_2}$ and $\langle s_1, s_2 \rangle \cong \mathrm{SL}(n,q)$ as illustrated in Figure 5.1. The probability results are summarised in Chapter 10 and the action of the stingray pair (s_1, s_2) is discussed in more detail in Remark 5.5.

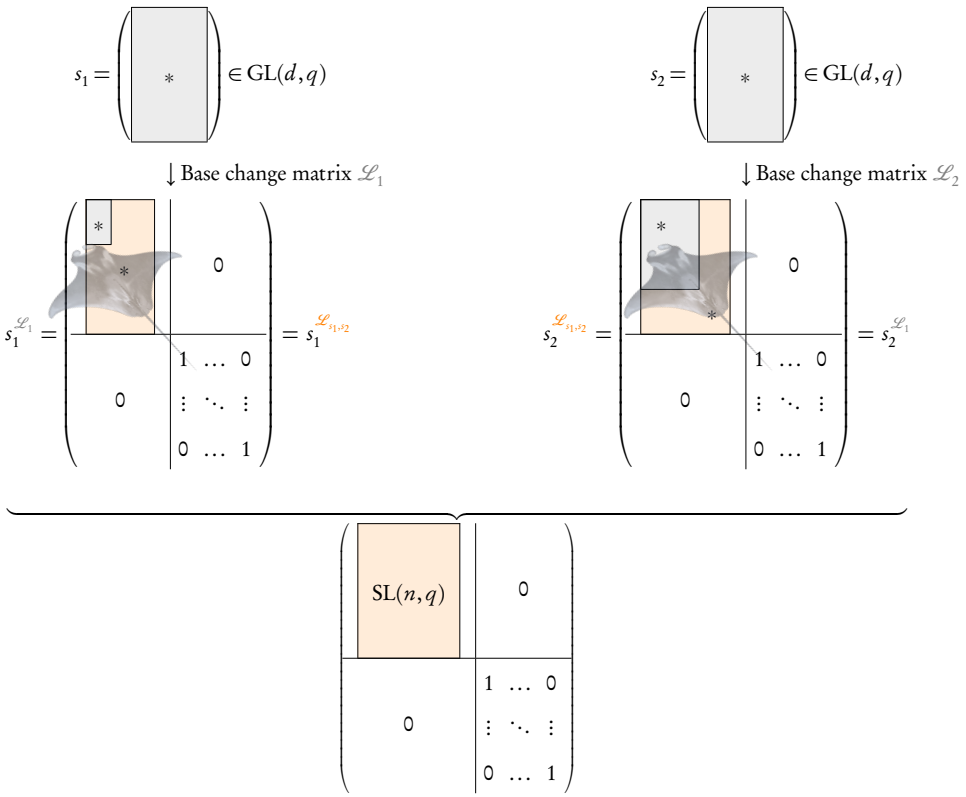


Figure 5.1: Visualisation of generating special linear groups using stingray duos.[1]

[1]MediaProduction. *Manta Ray Image*. iStock.com/MediaProduction, Stock ID: 1173588191. Used under standard license. Accessed August 26, 2022.

Two interesting questions are why the matrices in the orange block "interact" with each other and how the stingray elements can be displayed under the described base change of Figure 5.1. The next remark deals with this and the following example illustrates it.

Remark 5.5

Let $s_1, s_2 \in \mathrm{SL}(d,q)$ be two stingray elements and $W_{s_1}, W_{s_2} \leq \mathbb{F}_q^d$ with $W_{s_1} \cap W_{s_2} = \{0\}$ where W_{s_i} is the stingray body of s_i. First, note that every element of the intersection of the stingray tails $E_{s_1} \cap E_{s_2}$ is fixed by s_1 and s_2 and that $E_{s_1} \cap E_{s_2}$ is a complementary subspace of $W_{s_1} + W_{s_2}$. Second, let $w_1 \in W_{s_1}$ and $w_2 \in W_{s_2}$. Then

$$w_1 s_1 \in W_{s_1}$$

as s_1 leaves W_{s_1} invariant. Before we compute $w_2 s_1$ notice that $w_2 = \tilde{w}_1 + \tilde{v}$ where $\tilde{w}_1 \in W_{s_1}$ and $\tilde{v} \in E_{s_1}$ and

$$\begin{aligned} w_2(s_1 - I_d) &= (\tilde{w}_1 + \tilde{v})(s_1 - I_d) \\ &= \tilde{w}_1(s_1 - I_d) + \tilde{v}(s_1 - I_d) \\ &= \tilde{w}_1(s_1 - I_d) \\ &= \tilde{w}_1 s_1 - \tilde{w}_1 I_d \in W_{s_1}. \end{aligned}$$

Thus, $w_2 s_1 = \tilde{w}_1 s_1 - \tilde{w}_1 + w_2 = w' + w_2$ where $w' = \tilde{w}_1 s_1 - \tilde{w}_1 \in W_{s_1}$. ◀

Example 5.6

Let $G = \mathrm{SL}(6,5)$. Then

$$s_1 := \begin{pmatrix} 1 & 1 & 2 & 4 & 2 & 3 \\ 2 & 4 & 0 & 4 & 4 & 1 \\ 3 & 3 & 3 & 0 & 3 & 2 \\ 4 & 3 & 4 & 2 & 2 & 3 \\ 1 & 4 & 0 & 2 & 3 & 3 \\ 1 & 0 & 2 & 1 & 4 & 2 \end{pmatrix}, \qquad s_2 := \begin{pmatrix} 3 & 2 & 1 & 1 & 3 & 3 \\ 3 & 3 & 0 & 4 & 0 & 3 \\ 0 & 3 & 3 & 0 & 1 & 2 \\ 3 & 1 & 1 & 0 & 3 & 4 \\ 0 & 2 & 3 & 0 & 0 & 3 \\ 1 & 0 & 4 & 3 & 2 & 1 \end{pmatrix}$$

are two stingray elements with $s_1, s_2 \in G = \mathrm{SL}(6,5)$ and additionally $\langle s_1, s_2 \rangle \cong \mathrm{SL}(4,5)$. Here $W_{s_1} = \langle (2,1,1,1,0,0), (4,4,1,0,4,1) \rangle$ and $W_{s_2} = \langle (3,0,2,4,1,0), (1,4,0,3,0,1) \rangle$ such that the dimension $n_i := \dim(W_{s_i}) = 2$. Moreover, $W_{s_1} \cap W_{s_2} = \{0\}$ and thus $\dim(W_{s_1} + W_{s_2}) = 4$. Since $W_{s_1} \cap W_{s_2} = \{0\}$

and $\dim(W_{s_i}) = 2$ we know that $\dim(E_{s_i}) = \dim(\mathbb{F}_5^6) - \dim(W_{s_i}) = 6-2 = 4$ and $\dim(E_{s_1} \cap E_{s_2}) = \dim(\mathbb{F}_5^6) - \dim(W_{s_1}) - \dim(W_{s_1}) = 6-2-2 = 2$. We can also compute a basis for the common 1-eigenspace of s_1 and s_2 which is given by $E_{s_1} \cap E_{s_2} = \langle (1,4,0,2,4,0), (0,1,3,0,2,2) \rangle$. This yields a basis $\mathscr{B} := ((2,1,1,1,0,0), (4,4,1,0,4,1), (3,0,2,4,1,0), (1,4,0,3,0,1), (1,4,0,2,4,0), (0,1,3,0,2,2))$ for $\mathbb{F}_5^6$. With respect to this basis $\mathscr{B}$, the elements s_1 and s_2 are represented as the following matrices

$$ {}^{\mathscr{B}}s_1^{\mathscr{B}} = \left(\begin{array}{cccc|cc} 4 & 2 & 0 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 & 0 & 0 \\ 3 & 3 & 1 & 0 & 0 & 0 \\ 4 & 2 & 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right), \quad {}^{\mathscr{B}}s_2^{\mathscr{B}} = \left(\begin{array}{cccc|cc} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 3 & 0 & 0 \\ 0 & 0 & 3 & 2 & 0 & 0 \\ 0 & 0 & 4 & 3 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right). $$

In order to improve the run-time, it is reasonable to represent the matrices s_1 and s_2 as elements of an (n_1+n_2)-dimensional special linear group which is described in more detail in Section 5.1.4. Note that in this example $\mathrm{SL}(4,q) \cong \langle s_1, s_2 \rangle$ and that $\langle s_1, s_2 \rangle$ is stingray embedded of degree 4 in $\mathrm{SL}(6,5)$.◀

The next algorithm implements the GoingDown basic step, i.e. computes the next subgroup of a descending recognition chain

$$\mathrm{SL}(4,q) \cong U_k \leq U_{k-1} \cong \mathrm{SL}(d_{k-1},q) \leq \ldots \leq U_1 \cong \mathrm{SL}(d_1,q) \leq U_0 = G$$

where $d_i \leq 4\lceil \log(d_{i-1}) \rceil$ for $1 \leq i \leq k$.

Remark 5.7

Some randomised algorithms described in this thesis call other randomised algorithms. In order to bound the overall number of computed random elements, we use a "global" variable N. That means if a randomised algorithm A uses the variable N and calls a randomised algorithm B with input N, then A and B "share" the identical N, i.e. if N is decreased by B and B returns N, then the N of A is "updated" and thus also decreased. The variable N is very useful in this context to bound the maximal number of random elements computed by the input N. ◀

Remark 5.8

Some algorithms of this thesis call other randomised algorithms which can either return a valid output or `fail`. Whether an algorithm returns valid output or `fail` needs to be checked before the output is given as an argument to the next function. If a randomised algorithm A calls a randomised algorithm B and B returns `fail`, then A also terminates and returns `fail`. The check whether B returns `fail` requires an if statement in an implementation. To provide better readability of algorithms we sometimes omit this if-statement and refer to this remark instead. ◀

Algorithm 13: GOINGDOWNBASICSTEPSL

Input:
- ▶ $d_1 \in \mathbb{N}$ with $d_1 > 4$
- ▶ $\langle X \rangle = G \leq \mathrm{GL}(d,q)$ with $G \cong \mathrm{SL}(d_1,q)$
- ▶ $N \in \mathbb{N}$

Output: **fail OR** $(d_2, U, \mathfrak{S}, N')$ **where**
- ▶ $d_2 \in \mathbb{N}$ with $4 \leq d_2 \leq 4\lceil \log(d_1) \rceil$,
- ▶ $U \leq G$ with $U \cong \mathrm{SL}(d_2,q)$,
- ▶ $\mathfrak{S}$ is an MSLP from X to generators of U and
- ▶ $N' \in \mathbb{N}$ where $N - N'$ is the number of random selections that were used

function GOINGDOWNBASICSTEPSL(d_1, G, N)

```
1  while N > 0 do                                                    // Remark 5.7
2      (s_1, S_1, N) ← FINDSTINGRAYELEMENT(G, d_1, N)                // Remark 5.8
3      W_{s_1} ← COMPUTESTINGRAYBODY(s_1)
4      repeat
5          (s_2, S_2, N) ← FINDSTINGRAYELEMENT(G, d_1, N)            // Remark 5.8
6          W_{s_2} ← COMPUTESTINGRAYBODY(s_2)
7      until ISSTINGRAYDUO((s_1, s_2))
8      d_2 ← dim(W_{s_1}) + dim(W_{s_2})
9      if ⟨s_1, s_2⟩ ≅ SL(d_2, q) then        // Using a naming algorithm, see Section 1.1.7
10         S ← an MSLP from X to (s_1, s_2) using S_1 and S_2
11         return (d_2, ⟨s_1, s_2⟩, S, N)
12 return fail
```

Remark 5.9

1) The input parameter N of Algorithm GoingDownBasicStepSL [Alg. 13] is used to control the maximal number of random elements chosen. Even though Algorithm GoingDownBasicStepSL [Alg. 13] does not compute random elements itself, Algorithm FindStingrayElement [Alg. 5] is used which is a randomised algorithm to compute stingray elements, see Chapter 4. The Algorithm GoingDownBasicStepSL [Alg. 13] is a one-sided Monte Carlo algorithm and its success depends on the control parameter N. Thus if N is too small, then the algorithm can fail. How N has to be chosen in order to obtain a success probability of at least $1-\epsilon$ for $\epsilon \in (0,1)$ is discussed in Chapter 10.
2) The condition of the `if` statement in Line 9 of Algorithm GoingDownBasicStepSL [Alg. 13], which is $\langle s_1, s_2 \rangle \cong \mathrm{SL}(d_2,\ q)$, can be verified using a naming algorithm as in Section 1.1.7. ◀

We now establish the correctness of Algorithm GoingDownBasicStepSL [Alg. 13] and its termination using at most N random elements.

Theorem 5.10

Algorithm GoingDownBasicStepSL [Alg. 13] terminates using at most N random selections and is correct.

Proof. We start by proving that the algorithm terminates. Algorithm GoingDownBasicStepSL [Alg. 13] contains two loops which start in Line 1 and in Line 4. The loop of Line 1 terminates if $N \leq 0$. For every computation of a random element N is decreased and every call of Algorithm FindStingrayElement [Alg. 5] requires at least one computation of a random element. Therefore, the statement $N \leq 0$ becomes true after at most N executions of FindStingrayElement [Alg. 5]. In this case Algorithm GoingDownBasicStepSL [Alg. 13] returns `fail`. The loop of Line 4 also returns `fail` if $N \leq 0$ by Algorithm FindStingrayElement [Alg. 5] in Line 5.
The correctness is clear since the algorithm either returns `fail` or a subgroup which is isomorphic to $\mathrm{SL}(d_2, q)$ as this is verified in Line 9. We only have to prove that $4 \leq d_2 \leq 4\lceil \log(d_1) \rceil$. The non-trivial and irreducible subspace of the stingray element returned by Algorithm GoingDownBasicStepSL [Alg. 13] has dimension bounded by 2 and $2\lceil \log(d_1) \rceil$. Since d_2 is the sum of two such integers it lies between 4 and $4\lceil \log(d_1) \rceil$. □

Example 5.11

We apply Algorithm GoingDownBasicStepSL [Alg. 13] to the group G^{Ξ} already known from Example 5.4. Using Algorithm FindStingrayElement [Alg. 5] we can identify the following two stingray elements of G^{Ξ}:

$$\underbrace{\begin{pmatrix}
4&1&0&0&3&0&0&3&2&2&3&0&1&1&2&3\\
1&2&2&1&2&0&3&1&4&4&4&1&2&2&3&4\\
2&2&2&0&0&2&4&2&1&1&1&4&0&1&0&2\\
0&3&4&3&2&0&1&0&0&0&1&2&0&0&3&1\\
4&2&0&3&0&3&0&4&3&3&4&4&2&1&1&3\\
1&4&1&3&4&1&4&1&4&4&0&3&2&2&1&0\\
2&3&0&3&2&3&1&2&0&0&2&4&3&2&3&1\\
4&4&1&1&2&3&4&0&3&3&3&2&2&1&3&2\\
2&4&3&2&1&3&2&2&1&0&4&3&3&2&4&3\\
3&2&4&3&0&1&1&3&1&2&4&0&4&2&0&2\\
1&1&1&2&4&4&4&1&0&0&1&0&4&1&1&2\\
0&0&4&1&2&4&1&0&1&1&1&0&2&4&3&3\\
3&1&2&3&4&2&3&3&0&0&1&2&3&3&1&2\\
1&4&4&0&3&3&1&1&1&1&2&1&1&1&2&1\\
2&3&0&3&2&3&0&2&0&0&2&4&3&2&4&1\\
4&4&4&3&1&1&1&4&0&0&0&0&1&4&4&4
\end{pmatrix}}_{:=s_1}
\text{ and }
\underbrace{\begin{pmatrix}
1&1&4&1&4&2&3&0&4&0&3&1&1&2&2&3\\
0&2&4&1&4&2&3&0&4&0&3&1&1&2&2&3\\
2&1&0&4&1&0&3&2&0&3&2&1&0&2&0&0\\
1&4&1&4&2&2&2&1&4&4&4&4&1&3&2&3\\
2&4&0&0&2&0&2&0&0&4&2&1&2&4&3&2\\
2&2&2&3&3&2&1&0&2&4&1&4&0&0&4&1\\
1&3&2&2&3&0&0&1&0&4&1&3&0&1&0&0\\
1&1&0&2&2&2&3&4&4&3&4&4&4&1&4&1\\
0&1&0&3&1&3&3&2&2&4&2&4&2&1&0&0\\
3&4&1&1&4&0&2&3&0&3&3&4&0&3&0&0\\
0&1&1&0&3&4&3&4&3&3&2&2&3&0&3&2\\
0&3&4&2&1&3&4&4&1&3&2&0&0&4&2&3\\
1&1&0&2&2&2&3&3&4&3&4&4&0&1&4&1\\
2&2&3&0&0&2&1&2&4&3&0&2&1&0&2&3\\
1&2&0&0&3&0&1&0&0&2&1&3&1&2&0&1\\
0&1&0&3&1&3&3&2&1&4&2&4&2&1&0&1
\end{pmatrix}}_{:=s_2}.$$

For $i \in \{1,2\}$ the element s_i acts irreducibly on a 3-dimensional subspace W_{s_i} and fixes a complementary 13-dimensional subspace E_{s_i}, i.e. $\dim(W_{s_i}) = 3$ and $\dim(E_{s_i}) = 13$. For each of s_1 and s_2 we compute a matrix $\mathscr{L}_i$ where the first 3 row vectors form a basis for W_{s_i} and the last 13 row vectors form a basis for E_{s_i}.

$$\underbrace{\begin{pmatrix}
1&0&3&3&0&4&2&1&0&0&3&1&4&1&0&0\\
4&4&0&2&4&2&0&4&4&4&4&1&4&0&1&0\\
3&1&4&1&0&4&1&3&3&3&4&4&3&0&0&1\\
4&3&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
4&2&1&0&1&0&0&0&0&0&0&0&0&0&0&0\\
4&2&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
3&2&1&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&4&1&0&0&0&0&1&0&0&0&0&0&0&0&0\\
1&3&1&0&0&0&0&0&1&0&0&0&0&0&0&0\\
2&2&2&0&0&0&0&0&0&1&0&0&0&0&0&0\\
0&3&3&0&0&0&0&0&0&0&1&0&0&0&0&0\\
0&4&3&0&0&0&0&0&0&0&0&1&0&0&0&0\\
4&2&4&0&0&0&0&0&0&0&0&0&1&0&0&0\\
4&0&1&0&0&0&0&0&0&0&0&0&0&1&0&0\\
3&2&1&0&0&0&0&0&0&0&0&0&0&0&1&0\\
0&2&2&0&0&0&0&0&0&0&0&0&0&0&0&1
\end{pmatrix}}_{:=\mathscr{L}_1}
\text{ and }
\underbrace{\begin{pmatrix}
2&4&4&3&4&4&2&3&3&0&3&3&1&0&0&0\\
1&3&2&2&3&0&4&1&0&4&1&3&0&1&0&0\\
2&2&2&3&3&1&1&0&2&4&1&4&0&0&4&1\\
4&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
4&0&2&1&0&0&0&0&0&0&0&0&0&0&0&0\\
2&0&0&0&4&1&0&0&0&0&0&0&0&0&0&0\\
0&0&2&0&0&0&1&0&0&0&0&0&0&0&0&0\\
4&0&1&0&1&0&0&1&0&0&0&0&0&0&0&0\\
1&0&4&0&1&0&0&0&1&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&1&0&0&0&0&0&0\\
3&0&3&0&2&0&0&0&0&0&1&0&0&0&0&0\\
1&0&3&0&2&0&0&0&0&0&0&1&0&0&0&0\\
4&0&1&0&1&0&0&0&0&0&0&0&1&0&0&0\\
4&0&4&0&0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&2&0&0&0&0&0&0&0&0&0&1&0\\
1&0&4&0&1&0&0&0&0&0&0&0&0&0&0&1
\end{pmatrix}}_{:=\mathscr{L}_2}.$$

The matrices $\mathscr{L}_1$ and $\mathscr{L}_2$ can be used as base change matrices, that is

$$s_1^{\mathscr{L}_1^{-1}} = \left(\begin{array}{ccc|cccccccccccc} 2&1&4&0&0&0&0&0&0&0&0&0&0&0&0\\ 2&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\ 2&3&4&0&0&0&0&0&0&0&0&0&0&0&0\\ \hline 0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&1&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&1&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&1&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&1 \end{array}\right) = \left(\begin{array}{ccc|c} 2&1&4&0\\ 2&0&0&0\\ 2&3&4&0\\ \hline 0&0&0&I_{13} \end{array}\right)$$

and

$$s_2^{\mathscr{L}_2^{-1}} = \left(\begin{array}{ccc|cccccccccccc} 4&1&3&0&0&0&0&0&0&0&0&0&0&0&0\\ 0&4&4&0&0&0&0&0&0&0&0&0&0&0&0\\ 1&3&2&0&0&0&0&0&0&0&0&0&0&0&0\\ \hline 0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&1&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&1&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&1&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&1 \end{array}\right) = \left(\begin{array}{ccc|c} 4&1&3&0\\ 0&4&4&0\\ 1&3&2&0\\ \hline 0&0&0&I_{13} \end{array}\right).$$

Note that $W_{s_1} \cap W_{s_2} = \{0\}$. Hence, we can construct a base change matrix $\mathscr{L}_3$ which maps the original basis to a basis which consists of a basis for $W_{s_1} + W_{s_2}$ followed by a basis for the 10-dimensional

common fixed space $E_{s_1} \cap E_{s_2}$. We obtain

$$\mathscr{L}_3 = \begin{pmatrix}
1 & 0 & 3 & 3 & 0 & 4 & 2 & 1 & 0 & 0 & 3 & 1 & 4 & 1 & 0 & 0 \\
4 & 4 & 0 & 2 & 4 & 2 & 0 & 4 & 4 & 4 & 4 & 1 & 4 & 0 & 1 & 0 \\
3 & 1 & 4 & 1 & 0 & 4 & 1 & 3 & 3 & 3 & 4 & 4 & 3 & 0 & 0 & 1 \\
2 & 4 & 4 & 3 & 4 & 4 & 2 & 3 & 3 & 0 & 3 & 3 & 1 & 0 & 0 & 0 \\
1 & 3 & 2 & 2 & 3 & 0 & 4 & 1 & 0 & 4 & 1 & 3 & 0 & 1 & 0 & 0 \\
2 & 2 & 2 & 3 & 3 & 1 & 1 & 0 & 2 & 4 & 1 & 4 & 0 & 0 & 4 & 1 \\
1 & 1 & 0 & 1 & 1 & 4 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 3 & 3 & 0 & 4 & 4 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 4 & 1 & 2 & 1 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 2 & 4 & 0 & 2 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 4 & 3 & 2 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 2 & 3 & 3 & 1 & 4 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 4 & 2 & 1 & 4 & 2 & 2 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 4 & 2 & 0 & 2 & 3 & 4 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 4 & 1 & 4 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 3 & 3 & 3 & 2
\end{pmatrix}.$$

Then

$$s_1^{\mathscr{L}_3^{-1}} = \left(\begin{array}{cccccc|c}
2 & 1 & 4 & 0 & 0 & 0 & 0 \\
2 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 3 & 4 & 0 & 0 & 0 & 0 \\
2 & 3 & 4 & 1 & 0 & 0 & 0 \\
2 & 2 & 1 & 0 & 1 & 0 & 0 \\
1 & 3 & 1 & 0 & 0 & 1 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & I_{10}
\end{array}\right)$$

and

$$s_2^{\mathscr{L}_3^{-1}} = \left(\begin{array}{cccccc|c}
1 & 0 & 0 & 4 & 2 & 3 & 0 \\
0 & 1 & 0 & 1 & 3 & 0 & 0 \\
0 & 0 & 1 & 4 & 0 & 0 & 0 \\
0 & 0 & 0 & 4 & 1 & 3 & 0 \\
0 & 0 & 0 & 0 & 4 & 4 & 0 \\
0 & 0 & 0 & 1 & 3 & 2 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & I_{10}
\end{array}\right).$$

Moreover

$$\left\langle \begin{pmatrix}
2 & 1 & 4 & 0 & 0 & 0 \\
2 & 0 & 0 & 0 & 0 & 0 \\
2 & 3 & 4 & 0 & 0 & 0 \\
2 & 3 & 4 & 1 & 0 & 0 \\
2 & 2 & 1 & 0 & 1 & 0 \\
1 & 3 & 1 & 0 & 0 & 1
\end{pmatrix}, \begin{pmatrix}
1 & 0 & 0 & 4 & 2 & 3 \\
0 & 1 & 0 & 1 & 3 & 0 \\
0 & 0 & 1 & 4 & 0 & 0 \\
0 & 0 & 0 & 4 & 1 & 3 \\
0 & 0 & 0 & 0 & 4 & 4 \\
0 & 0 & 0 & 1 & 3 & 2
\end{pmatrix} \right\rangle = \mathrm{SL}(6,5)$$

and, hence, $G_1^{\Xi} := \langle s_1, s_2 \rangle \cong \mathrm{SL}(6,5)$. We record the base change matrix $\mathscr{L}_3$ by setting $\mathscr{L}_1^{\Xi} := \mathscr{L}_3^{-1}$. ◀

5.1.2 Combining GoingDown basic steps

Let $G = \mathrm{SL}(d,q)$. By repeatedly calling the GoingDown basic step, we now have all the methods needed to construct a descending recognition chain as in Definition 3.4 of special linear groups as

$$\mathrm{SL}(4,q) \cong U_k \leq U_{k-1} \cong \mathrm{SL}(d_{k-1},q) \leq \ldots \leq U_1 \cong \mathrm{SL}(d_1,q) \leq U_0 = G$$

where $d_i \leq 4\lceil \log(d_{i-1}) \rceil$ for $1 \leq i \leq k$. Observe that the chain reaches a group isomorphic to $\mathrm{SL}(4,q)$ rapidly as the basic step is reducing the dimension logarithmically. The overall number of GoingDown basic steps required is given by the iterated logarithm $\log^*(d)$.

Remark 5.12

Note that all state-of-the-art algorithms require a lot more steps. The currently best algorithm is an algorithm by Dietrich, Leedham-Green, Lübeck and O'Brien [32, 59], in the following referred to as DLLO algorithm. The DLLO algorithm has two variations from which one is based on a "splitting" step (called "One" in [59]) and the other on a "conjugating" step (called "Two" in [59]). In practice the variation of the DLLO algorithm with the "splitting" step is used. In each call to the "splitting" step of the DLLO algorithm two subgroups of half of the input dimension are computed in best case. Moreover, the DLLO algorithm has to be applied to both computed subgroups. The GoingDown algorithm in this thesis avoids "splitting" by constructing only one subgroup which is isomorphic to $\mathrm{SL}(2,q)$. It should be mentioned that the DLLO algorithm is also applicable for non-natural irreducible matrix representations and for black box settings and, hence, does not exploit properties of the natural representation as we do in our GoingUp algorithms which in turn has an enormous impact on the design of the overall algorithm. ◀

Finally, the following algorithm computes a descending recognition chain by iteratively applying Algorithm GoingDownBasicStepSL [Alg. 13].

Algorithm 14: GoingDownToDim4

Input:
- ▶ $d \in \mathbb{N}$ with $d \geq 4$
- ▶ $\langle X \rangle = G = \mathrm{SL}(d,q)$
- ▶ $N \in \mathbb{N}$

Output: fail **OR** $(U, \mathfrak{S}, N')$ **where**
- ▶ $U \leq G$ with $U \cong \mathrm{SL}(4,q)$,
- ▶ $\mathfrak{S}$ is an MSLP from X to generators of U and
- ▶ $N' \in \mathbb{N}$ where $N - N'$ is the number of random selections that were used

```
function GoingDownToDim4(d, G, N)
1   U ← G AND dim ← d AND 𝔖 ← an MSLP from X to X
2   while dim > 4 do
3       (dim, U, 𝔖′, N) ← GoingDownBasicStepSL(U, dim, N)        // Remark 5.7 and 5.8
4       𝔖 ← Composition of 𝔖 and 𝔖′
5   return (U, 𝔖, N)
```

Theorem 5.13

Algorithm GoingDownToDim4 [Alg. 14] terminates using at most N random selections and is correct.

Proof. The algorithm only contains one loop. The return value of Algorithm GoingDownBasicStepSL [Alg. 13] is either a subgroup which is isomorphic to $\mathrm{SL}(\mathtt{dim}, q)$ or `fail`. If the return value is `fail`, then the function terminates immediately. Let d_1 be equal to `dim` and let d_2 be equal to the first return value in Line 3. Then $d_2 < d_1$ since $4 \leq d_2 \leq \min\{4\lceil\log(d_1)\rceil, \lceil\frac{d_1}{2}\rceil\}$ and, therefore, the algorithm terminates.

If $d = 4$, then $G = \mathrm{SL}(4, q)$. Otherwise we have after every execution of the while loop in Line 2 that $U \cong \mathrm{SL}(\mathtt{dim}, q)$. Since `dim` strictly decreases in each iteration the claim follows. □

Remark 5.14

Recall the descending recognition chain

$$\mathrm{SL}(4,q) \cong U_k \leq U_{k-1} \cong \mathrm{SL}(d_{k-1},q) \leq \ldots \leq U_1 \cong \mathrm{SL}(d_1,q) \leq U_0 = G,$$

which is computed by Algorithm GoingDownToDim4 [Alg. 14]. Algorithm GoingDownToDim4 [Alg. 14] uses Algorithm GoingDownBasicStepSL [Alg. 13] which verifies that a group generated by two stingray elements is isomorphic to a special linear group using naming algorithms. Note that the number of calls to naming algorithms can be reduced by slight variations of the algorithms. Different alternatives for these variations are, e.g.

- always use a naming algorithm, i.e. verify that $U_i \cong \mathrm{SL}(d_i, q)$ for all $i \in \{1, \ldots, k\}$,
- only use a naming algorithm until the first subgroup is computed, i.e. only verify that $U_1 \cong \mathrm{SL}(d_1, q)$,

- never use a naming algorithm and restart from the input group if a maximal number of random selection is reached, i.e. restart from $U_0 = G$,
- never use a naming algorithm but instead of restarting from the input group backtrack one step, i.e. if a maximal number of tries is reached in the process of computing U_{i+1} as a subgroup of U_i, then restart from U_{i-1} and compute another subgroup U_i.

Practical tests of implementations could be used to compare the impact on the running time which is not done in this thesis. ◀

Example 5.15

In Example 5.11 we computed a subgroup $G_1^\Xi \leq G^\Xi$ with $G_1^\Xi \cong \mathrm{SL}(6,5)$. As we have not found a subgroup of G^Ξ which is isomorphic to $\mathrm{SL}(4,5)$ we repeat the GoingDown basic step with input G_1^Ξ. Note that all matrices in the following are given as elements of $(G_1^\Xi)^{\mathscr{L}_1^\Xi}$ of Example 5.11 if not mentioned otherwise.

Again we identify two stingray elements in G_1^Ξ using Algorithm FindStingrayElement [Alg. 5]:

$$s_3 := \left(\begin{array}{cccccc|c} 3&0&1&1&1&4&0\\ 4&4&2&1&0&2&0\\ 0&2&1&1&2&1&0\\ 3&4&4&2&3&3&0\\ 4&2&2&3&0&4&0\\ 0&4&0&2&4&3&0\\ \hline 0&0&0&0&0&0&I_{10} \end{array}\right), \quad s_4 := \left(\begin{array}{cccccc|c} 3&1&1&3&3&0&0\\ 1&4&3&4&4&0&0\\ 4&3&0&2&0&2&0\\ 4&3&4&3&0&2&0\\ 1&3&3&4&0&0&0\\ 2&2&3&4&2&3&0\\ \hline 0&0&0&0&0&0&I_{10} \end{array}\right) \in (G_1^\Xi)^{\mathscr{L}_1^\Xi}$$

The dimension of the stingray tails E_{s_3} and E_{s_4} is 14 and both elements have 2-dimensional stingray bodies W_{s_3} and W_{s_4}. The following base change matrices

$$\mathscr{L}_4 := \left(\begin{array}{cccccc|c} 4&4&2&4&1&0&0\\ 2&4&1&3&0&1&0\\ 3&1&1&0&0&0&0\\ 2&2&0&1&0&0&0\\ 1&1&0&0&1&0&0\\ 1&2&0&0&0&1&0\\ \hline 0&0&0&0&0&0&I_{10} \end{array}\right) \quad \text{and} \quad \mathscr{L}_5 := \left(\begin{array}{cccccc|c} 4&2&2&1&1&0&0\\ 2&4&2&1&0&1&0\\ 2&1&0&0&0&0&0\\ 0&0&4&1&0&0&0\\ 2&0&0&0&1&0&0\\ 1&0&4&0&0&1&0\\ \hline 0&0&0&0&0&0&I_{10} \end{array}\right)$$

satisfy

$$s_3^{\mathscr{L}_4^{-1}} = \left(\begin{array}{cc|c} 0&2&0\\ 2&4&0\\ \hline 0&0&I_{14} \end{array}\right) \quad \text{and} \quad s_4^{\mathscr{L}_5^{-1}} = \left(\begin{array}{cc|c} 0&1&0\\ 4&4&0\\ \hline 0&0&I_{14} \end{array}\right).$$

The next base change matrix maps the given basis to a basis consisting of a basis of $W_{s_3} + W_{s_4}$ and a

basis for $E_{s_3} \cap E_{s_4}$

$$\mathscr{L}_6 = \left(\begin{array}{cccccc|c} 4 & 4 & 2 & 4 & 1 & 0 & 0 \\ 2 & 4 & 1 & 3 & 0 & 1 & 0 \\ 4 & 2 & 2 & 1 & 1 & 0 & 0 \\ 2 & 4 & 2 & 1 & 0 & 1 & 0 \\ 1 & 0 & 3 & 2 & 3 & 0 & 0 \\ 0 & 1 & 0 & 4 & 1 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & I_{10} \end{array}\right).$$

Then

$$s_3^{\mathscr{L}_6^{-1}} = \left(\begin{array}{cccc|c} 0 & 2 & 0 & 0 & 0 \\ 2 & 4 & 0 & 0 & 0 \\ 0 & 4 & 1 & 0 & 0 \\ 3 & 3 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 & I_{12} \end{array}\right) \quad \text{and} \quad s_4^{\mathscr{L}_6^{-1}} = \left(\begin{array}{cccc|c} 1 & 0 & 2 & 2 & 0 \\ 0 & 1 & 4 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 4 & 4 & 0 \\ \hline 0 & 0 & 0 & 0 & I_{12} \end{array}\right).$$

Moreover

$$\left\langle \begin{pmatrix} 0 & 2 & 0 & 0 \\ 2 & 4 & 0 & 0 \\ 0 & 4 & 1 & 0 \\ 3 & 3 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 2 & 2 \\ 0 & 1 & 4 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 4 & 4 \end{pmatrix} \right\rangle = \mathrm{SL}(4,5)$$

and, hence, $G_2^{\Xi} := \langle s_3^{(\mathscr{L}_1^{\Xi})^{-1}}, s_4^{(\mathscr{L}_1^{\Xi})^{-1}} \rangle \cong \mathrm{SL}(4,5)$. Note that $G_2^{\Xi} \leq G^{\Xi}$. We denote the new base change matrix by $\mathscr{L}_2^{\Xi} := \mathscr{L}_1^{\Xi} \mathscr{L}_6^{-1}$. ◀

5.1.3 Final step of the GoingDown algorithm

At this point we assume that we have found $U \leq G$ with $U \cong \mathrm{SL}(4,q)$ which is a terminal group. To obtain a base case group, however, we need $H \leq G$ with $H \cong \mathrm{SL}(2,q)$. Unfortunately, the idea from Section 5.1.2 is not applicable as the next remark shows.

Remark 5.16

Note that stingray elements of special linear groups with a 1-dimensional stingray body are equal to the identity matrix. To see this, apply a base change to obtain a block matrix structure as in Example 4.3 where one block visualises the stingray tail which is an identity matrix of size $(d-1) \times (d-1)$ and the other block visualises the stingray body which has to be $(1) \in \mathbb{F}_q^{1\times 1}$ since the stingray element is contained in $\mathrm{SL}(d,q)$.

Let $G = \mathrm{SL}(4,q)$ and suppose we search for two stingray elements $s_1, s_2 \in G$ with $H = \langle s_1, s_2 \rangle \cong \mathrm{SL}(2,q)$. One idea could be to use the GoingDown basic step, i.e. to search for two stingray elements with 1-dimensional stingray bodies which also intersect trivially and verify $\langle s_1, s_2 \rangle \cong \mathrm{SL}(2,q)$. As we have noted, stingray elements with 1-dimensional stingray body are equal to the identity matrix.

Therefore, it is not possible that $\langle s_1, s_2 \rangle \cong \mathrm{SL}(2,q)$.

Another idea is to compute two stingray elements $s_1, s_2 \in G$ which have the same 2-dimensional stingray body. Note that there are

$$\binom{4}{2}_q = \frac{(q^4-1)(q^4-q)}{(q^2-1)(q^2-q)}$$

different 2-dimensional subspaces of $\mathbb{F}_q^4$. Therefore, the probability that s_1 and s_2 have the same 2-dimensional stingray body is given by

$$\frac{1}{\binom{4}{2}_q} = \frac{(q^2-1)(q^2-q)}{(q^4-1)(q^4-q)} = \frac{1}{(q^2+1)(q^2+q+1)} < \frac{1}{q^4}$$

which decreases as q increases. For example, if $q = 121$, then the probability that s_1 and s_2 have the same 2-dimensional stingray body is 0.0000000046. ◀

Instead we use a method from the DLLO algorithm [32, 59]. In the following we only sketch the method for q odd.

Definition 5.17: [59, Definition 4.1]

Let $G \leq \mathrm{GL}(d,q)$.

1) $g \in G$ is an *involution* if $g \neq 1_G$ and $g^2 = 1_G$.
2) $g \in G$ is a *strong involution* if g is an involution and $\dim(E_1(g)) \in \{\lceil \frac{d}{3} \rceil, \ldots, \lfloor \frac{2d}{3} \rfloor\}$.
3) $g \in G$ is a *strong pre-involution* if $|g|$ is even and $g^{|g|/2}$ is a strong involution.

The probability to find a strong pre-involution by selecting independent uniformly distributed random elements is promising as the next lemma shows.

Lemma 5.18: [66, Theorem 1.1]

Let q be odd and $d \geq 3$. Define $A(d,q) := \{g \in \mathrm{SL}(d,q) \mid g \text{ is a strong pre-involution}\}$. Then

$$\frac{|A(d,q)|}{|\mathrm{SL}(d,q))|} \geq \frac{1}{5000 \log_2(d)}.$$

Proof. [66, Theorem 1.1]. □

Algorithm 15: GoingDownFinalStepSL

Input: ▶ $\langle X\rangle = G = \mathrm{SL}(4,q)$ with q odd

▶ $N \in \mathbb{N}$

Output: **fail OR** $(\mathscr{L}, U, \mathfrak{S}, N')$ **where**

▶ $U \leq G$ with $U \cong \mathrm{SL}(2,q)$,

▶ $\mathscr{L} \in \mathrm{GL}(4,q)$ is a base change matrix such that $U^{\mathscr{L}}$ is stingray embedded in $G^{\mathscr{L}}$,

▶ $\mathfrak{S}$ is an MSLP from X to generators of U and

▶ $N' \in \mathbb{N}$ where $N - N'$ is the number of random selections that were used

function GoingDownFinalStepSL(G, N)

```
repeat                                                  // First find a strong pre-involution
    g ← PseudoRandom(G) AND N ← N − 1
until g is not a strong pre-involution with dim(E_1(g)) = 2 AND N > 0
if N ≤ 0 then
    return fail
g ← g^{|g|/2} AND ℒ ← E_1(g) ∪ E_{-1}(g)
(C, N) ← CentraliserOfInvolution(G, g, N)               // Remark 5.7 and 5.8
gens ← []                                               // Extract a SL(2,q).
while N > 0 do
    g ← PseudoRandom(C^ℒ) AND N ← N − 1
    if exists ℓ ∈ ℕ such that the 2 × 2 bottom right block of g^ℓ is trivial but the 2 × 2 top left
     block is not then
        gens ← gens ∪ {g^ℓ}
        if ⟨gens⟩ ≅ SL(2,q) then                        // Using a naming algorithm, see Section 1.1.7
            return (⟨gens⟩, ℒ, 𝔖, N)
return fail
```

Let $g \in G = \mathrm{SL}(d,q)$ be a strong involution. We seek a strong involution since the derived group of its centraliser satisfies

$$C_G(g)' = \mathrm{SL}(E_1(g)) \times \mathrm{SL}(E_{-1}(g))$$

which is proven in [59]. In our case, G is a natural $\mathrm{SL}(4,q)$ and we seek an involution $g \in G$ with $\dim(\mathrm{SL}(E_1(g))) = 2$ such that a $\mathrm{SL}(2,q)$ can be extracted. We give the algorithm in pseudo-code as Algorithm GoingDownFinalStepSL [Alg. 15].

Theorem 5.19

Algorithm GoingDownFinalStepSL [Alg. 15] is correct and terminates.

Proof. [59, Chapter 11]. □

Example 5.20

We continue Example 5.15. Remember that we computed $G_2^{\Xi} \leq G^{\Xi}$ with $G_2^{\Xi} \cong \mathrm{SL}(4,q)$ and $G_2^{\Xi} = \langle s_3, s_4 \rangle$ where

$$s_3^{\mathscr{L}_2^{\Xi}} = \left(\begin{array}{cccc|c} 0 & 2 & 0 & 0 & 0 \\ 2 & 4 & 0 & 0 & 0 \\ 0 & 4 & 1 & 0 & 0 \\ 3 & 3 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 & I_{12} \end{array}\right) \quad \text{and} \quad s_4^{\mathscr{L}_2^{\Xi}} = \left(\begin{array}{cccc|c} 1 & 0 & 2 & 2 & 0 \\ 0 & 1 & 4 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 4 & 4 & 0 \\ \hline 0 & 0 & 0 & 0 & I_{12} \end{array}\right).$$

Now we are using Algorithm GoingDownFinalStepSL [Alg. 15] to compute a subgroup isomorphic to $\mathrm{SL}(2,q)$. If not stated otherwise, then the matrices are given as elements of $(G_2^{\Xi})^{\mathscr{L}_2^{\Xi}}$. By randomly choosing elements of $(G_2^{\Xi})^{\mathscr{L}_2^{\Xi}}$ we identify

$$a_5 = \left(\begin{array}{cccc|c} 1 & 2 & 0 & 2 & 0 \\ 3 & 3 & 3 & 0 & 0 \\ 2 & 4 & 3 & 1 & 0 \\ 3 & 3 & 0 & 4 & 0 \\ \hline 0 & 0 & 0 & 0 & I_{12} \end{array}\right).$$

Using the characteristic polynomial and a pseudo-order algorithm [76, Section 2.5] this yields that the order of a_5 divides 120. By setting

$$a_6 := a_5^{60} = \left(\begin{array}{cccc|c} 0 & 1 & 2 & 3 & 0 \\ 3 & 1 & 3 & 3 & 0 \\ 4 & 3 & 4 & 1 & 0 \\ 0 & 1 & 3 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & I_{12} \end{array}\right)$$

we observe that a_6 is not the identity and, since the pseudo-order of a_5 is known, that a_6^2 is the identity. Hence, a_6 is an involution. Further, basic computations show that the (-1)-eigenspace of a_6 is 2-dimensional. Hence, we compute a base change matrix $\mathscr{L}_7$ into a basis consisting of a basis of

the (−1)-eigenspace and the 1-eigenspace

$$\mathscr{L}_7 := \left(\begin{array}{cccc|c} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 3 & 4 & 0 \\ 1 & 0 & 4 & 2 & 0 \\ 0 & 1 & 3 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 & I_{12} \end{array}\right).$$

Note that

$$a_6 = \left(\begin{array}{cccc|c} 4 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 & I_{12} \end{array}\right)^{\mathscr{L}_7}.$$

Using Bray's algorithm [18] we can compute $C_{G_2^\Xi}(a_6^{(\mathscr{L}_2^\Xi)^{-1}})$ efficiently. Under the base change matrix $\mathscr{L}_2^\Xi \mathscr{L}_7^{-1}$ an element of $C_{G_2^\Xi}(a_6^{(\mathscr{L}_2^\Xi)^{-1}})$ is a block-diagonal matrix, e.g. the following element is in $C_{G_2^\Xi}(a_6^{(\mathscr{L}_2^\Xi)^{-1}})$

$$\left(\begin{array}{cccc|c} 2 & 3 & 0 & 0 & 0 \\ 4 & 3 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 0 \\ 0 & 0 & 0 & 4 & 0 \\ \hline 0 & 0 & 0 & 0 & I_{12} \end{array}\right).$$

By random search of elements in $C_{G_2^\Xi}(a_6^{(\mathscr{L}_2^\Xi)^{-1}})$ we find the following two elements which can be displayed as follows using the base change matrix $\mathscr{L}_2^\Xi \mathscr{L}_7^{-1}$, namely

$$a_7 := \left(\begin{array}{cccc|c} 3 & 4 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 3 & 4 & 0 \\ 0 & 0 & 2 & 4 & 0 \\ \hline 0 & 0 & 0 & 0 & I_{12} \end{array}\right), \quad a_8 := \left(\begin{array}{cccc|c} 1 & 4 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 2 & 2 & 0 \\ \hline 0 & 0 & 0 & 0 & I_{12} \end{array}\right).$$

The order of $\left(\begin{smallmatrix} 3 & 4 \\ 2 & 4 \end{smallmatrix}\right)$ divides 12 and the order of $\left(\begin{smallmatrix} 3 & 0 \\ 2 & 2 \end{smallmatrix}\right)$ divides 4 using again a pseudo-order algorithm [76, Section 2.5]. Hence, we raise a_7 and a_8 to these integers, respectively, yielding

$$a_9 := a_7^{12} = \left(\begin{array}{cccc|c} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 & I_{12} \end{array}\right) = \left(\begin{array}{cc|c} 1 & 1 & 0 \\ 0 & 1 & 0 \\ \hline 0 & 0 & I_{14} \end{array}\right) \quad \text{and} \quad a_{10} := a_8^4 = \left(\begin{array}{cccc|c} 4 & 1 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 & I_{12} \end{array}\right) = \left(\begin{array}{cc|c} 4 & 1 & 0 \\ 4 & 0 & 0 \\ \hline 0 & 0 & I_{14} \end{array}\right).$$

Since $\mathrm{SL}(2,q) \cong \langle a_9^{(\mathscr{L}_2^\Xi \mathscr{L}_7^{-1})^{-1}}, a_{10}^{(\mathscr{L}_2^\Xi \mathscr{L}_7^{-1})^{-1}} \rangle =: G_3^\Xi \leq G^\Xi$, the algorithm was successful. Again the new base change matrix is stored by setting $\mathscr{L}_3^\Xi := \mathscr{L}_2^\Xi \mathscr{L}_7^{-1}$. ◀

Using an algorithm for constructive recognition of $\mathrm{SL}(2,q)$, we can encode the standard generators from Definition 5.1 as words in the generators form Algorithm GoingDownFinalStepSL [Alg. 15]. A description of a constructive recognition algorithm for $\mathrm{SL}(2,q)$ is given in Section 5.2.

Remark 5.21

In Algorithm GoingDownFinalStepSL [Alg. 15] we compute the 1- and (-1)-eigenspace of an involution which requires that $1 \neq -1$, i.e. $\mathrm{char}(\mathbb{F}) \neq 2$. If $\mathrm{char}(\mathbb{F}) = 2$, then the computation of usable involutions becomes more difficult. Nevertheless, there are constructive algorithms based on involutions in characteristic 2 which are described in [32] but not further discussed in this thesis. ◀

We use similar algorithms for the final step in the other classical groups for which we provide a generalised algorithm in pseudo-code as Algorithm GoingDownFinalStepCL [Alg. 16].

Theorem 5.22

Algorithm GoingDownFinalStepCL [Alg. 16] is correct and terminates.

Proof. [59, Chapter 11]. □

Remark 5.23

A modified version of Algorithm GoingDownFinalStepCL [Alg. 16] can be used as the final step algorithm for $\mathrm{CL}(d,q)$ with q even. This is not discussed in this thesis and we refer to [32, Chapter 5]. ◀

Algorithm 16: GoingDownFinalStepCL

Input:
- $\langle X \rangle = G = \mathrm{CL}(d,q)$ a terminal group as in Definition 3.6 with q odd
- $d' \in \mathbb{N}$ with $2 \leq d' < d$
- $N \in \mathbb{N}$

Output: fail **OR** $(\mathscr{L}, U, \mathfrak{S}, N')$ **where**
- $U \leq G$ with $U \cong \mathrm{CL}(d',q)$ of the same type as G,
- $\mathscr{L} \in \mathrm{GL}(d,q)$ is a base change matrix such that $U^{\mathscr{L}}$ is stingray embedded in $G^{\mathscr{L}}$,
- $\mathfrak{S}$ is an MSLP from X to generators of U and
- $N' \in \mathbb{N}$ where $N - N'$ is the number of random selections that were used

function GoingDownFinalStepCL(G, d', N)

```
repeat                                                    // First find a strong pre-involution
    g ← PseudoRandom(G) AND N ← N − 1
until g is not a strong pre-involution with dim(E_1(g)) = d' AND N > 0
if N ≤ 0 then
    return fail
g ← g^{|g|/2} AND 𝓛 ← E_1(g) ∪ E_{-1}(g)
(C, N) ← CentraliserOfInvolution(G, g, N)                 // Remark 5.7 and 5.8
gens ← []                                                 // Extract a SL(2, q).
while N > 0 do
    g ← PseudoRandom(C^𝓛) AND N ← N − 1
    if exists ℓ ∈ ℕ such that the (d − d') × (d − d') bottom right block of g^ℓ is trivial but the
     d' × d' top left block is not then
        gens ← gens ∪ {g^ℓ}
        if ⟨gens⟩ ≅ CL(d', q) then                        // Using a naming algorithm, see Section 1.1.7
            return (⟨gens⟩, 𝓛, 𝔖, N)
return fail
```

5.1.4 GoingDown basic step with lower-dimensional matrices

In this chapter an extended version of the GoingDown algorithm is introduced and proved to be correct. This method improves the running time asymptotically while the length and memory of the output MSLPs remain the same.

The general idea is to extract the non-trivial blocks of stingray elements, which is possible using a respective base change matrix as displayed in Example 5.6, after each GoingDown step such that all subsequent computations can be performed with smaller-dimensional matrices and, therefore, more efficiently. Algorithm InducedActionRepresentationGroup [Alg. 12] plays a key role for the methods of this chapter.

Recall that Algorithm InducedActionRepresentationGroup [Alg. 12] introduced in Chapter 4 takes as input a subgroup $U \leq \mathrm{GL}(d,q)$ which is generated by a stingray duo (s_1, s_2) with $\dim(W_{s_1}) =$

n_i and outputs a group $\tilde{U} \leq \mathrm{GL}(n_1+n_2,q)$ generated by the actions of s_1 and s_2 on $W_{s_1}+W_{s_2}$. This can be done as all relevant information about $\langle s_1,s_2\rangle$ is already present in the action of $\langle s_1,s_2\rangle$ on $W_{s_1}+W_{s_2}$. Instead of calling the next GoingDown basic step with input group $\langle s_1,s_2\rangle$ we call it with input $\tilde{U} \leq \mathrm{GL}(n_1+n_2,q)$ where $\tilde{U}$ is the output of Algorithm InducedActionRepresentationGroup [Alg. 12]. Continuing with $\tilde{U}$ is more efficient since we are only dealing with $(n_1+n_2)\times(n_1+n_2)$ matrices instead of $d \times d$ matrices. By using this trick repeatedly this refines the descending recognition chain

$$\mathrm{SL}(4,q) \cong U_k \leq U_{k-1} \cong \mathrm{SL}(d_{k-1},q) \leq \ldots \leq U_1 \cong \mathrm{SL}(d_1,q) \leq U_0 = G = \mathrm{SL}(d,q),$$

where $d_i \leq 4\lceil \log(d_{i-1})\rceil$ for $2 \leq i \leq k$ into

$$\mathrm{SL}(4,q) = U_k \lesssim U_{k-1} = \mathrm{SL}(d_{k-1},q) \lesssim \ldots \lesssim U_1 = \mathrm{SL}(d_1,q) \lesssim U_0 = G = \mathrm{SL}(d,q).$$

Here $H \lesssim U$ denotes that H is isomorphic to a stingray embedded subgroup of U. Note that we cannot say that U_i is a subgroup of U_{i-1} any more as the elements of U_i are $d_i \times d_i$ matrices and the elements of U_{i+1} are $d_{i+1} \times d_{i+1}$ matrices.

Using these ideas Algorithm GoingDownToDim4 [Alg. 14] can be improved as follows.

Algorithm 17: GoingDownToDim4LDVersion1

Input:
- $d \in \mathbb{N}$ with $d \geq 4$
- $\langle X_1\rangle = G = \mathrm{SL}(d,q)$
- $N \in \mathbb{N}$

Output: fail **OR** (X_2, N') **where**
- $\langle X_2\rangle = U = \mathrm{SL}(4,q)$ and
- $N' \in \mathbb{N}$ where $N - N'$ is the number of random selections that were used

function GoingDownToDim4LDVersion1(d, G, N)

```
1  U ← G AND dim ← d
2  while dim > 4 do
3      (dim, U, S, N) ← GoingDownBasicStepSL(U, dim, N)        // Remark 5.7 and 5.8
4      U ← InducedActionRepresentationGroup(U)
5  return (U, N)
```

One crucial difference compared to Algorithm GoingDownToDim4 [Alg. 14] is that the output of Algorithm GoingDownToDim4LDVersion1 [Alg. 17] is no longer a subgroup of G. Let $\langle g_1, g_2\rangle = U$ be the output of Algorithm GoingDownToDim4LDVersion1 [Alg. 17]. Then U can be stingray embedded into G by defining $\tilde{g}_i = \mathrm{diag}(g_i, I_{d-4})$ and U is isomorphic to $\tilde{U} = \langle \tilde{g}_1, \tilde{g}_2\rangle \cong \mathrm{SL}(4,q)$. Note, that the stingray embedding is not sufficient to replace GoingDownToDim4 [Alg. 14] by Algorithm GoingDownToDim4LDVersion1 [Alg. 17] yet. By using Algorithm InducedActionRepresentationGroup [Alg. 12] we apply implicitly a base change matrix to naturally embed $U \leq \mathrm{GL}(d,q)$ into $\mathrm{GL}(d,q)$. Hence, we need to keep track of these base change matrices after each GoingDown basic step in course of Algorithm GoingDownToDim4LDVersion1 [Alg. 17]. This corresponds to basic linear algebra through linear combinations of basis vectors of the stingray bodies of stingray elements and results in the following final algorithm.

Theorem 5.24

Algorithm GoingDownToDim4LD [Alg. 18] is correct and terminates.

Proof. The algorithm does the same as Algorithm GoingDownToDim4 [Alg. 14] except the additional computation of a basis of the sum of the stingray bodies of the generators of U at the end of the while loop in Line 7. Since the computation of such a basis has no impact on the GoingDown algorithm the output group is a subgroup isomorphic to $\mathrm{SL}(4,q)$ as for Algorithm GoingDownToDim4 [Alg. 14]. The construction of a suitable base change matrix corresponds to linear combinations of basis vectors after the computation of the stingray bodies of the generators of U as it is done in the Line 5 to 7. □

Algorithm 18: GoingDownToDim4LD

Input:
- $d \in \mathbb{N}$ with $d \geq 4$
- $\langle X\rangle = G = \mathrm{SL}(d,q)$
- $N \in \mathbb{N}$

Output: fail **OR** $(U, \mathscr{L}, \mathfrak{S}, N')$ **where**
- $U \leq G$ with $U \cong \mathrm{SL}(4,q)$,
- $\mathscr{L} \in \mathrm{GL}(d,q)$ is a base change matrix such that $U^{\mathscr{L}}$ is stingray embedded in $G^{\mathscr{L}}$,
- $\mathfrak{S}$ is an MSLP from X to generators of U and
- $N' \in \mathbb{N}$ where $N - N'$ is the number of random selections that were used

function GoingDownToDim4LD(d, G, N)

1 $U \leftarrow G$ **AND** dim $\leftarrow d$ **AND** $\mathscr{B} \leftarrow I_d$ **AND** $\mathfrak{S} \leftarrow$ an MSLP from X to X
2 **while** dim > 4 **do**
3 $(\text{dim}, U, \mathfrak{S}', N) \leftarrow$ GoingDownBasicStepSL(U, dim, N) // Remark 5.7 and 5.8
4 $\mathfrak{S} \leftarrow$ Composition of $\mathfrak{S}$ and $\mathfrak{S}'$
5 $W_{s_1} \leftarrow$ ComputeStingrayBody(s_1) // We assume $U = \langle s_1, s_2 \rangle$
6 $W_{s_2} \leftarrow$ ComputeStingrayBody(s_2)
7 $W \leftarrow \langle W_{s_1} + W_{s_2} \rangle$ **AND** $\mathscr{B} \leftarrow W \cdot \mathscr{B}$
8 $U \leftarrow$ InducedActionRepresentationGroup(U)
9 $\mathscr{L} \leftarrow \mathscr{B}$ with a basis for the common 1-eigenspace
10 **return** $(\langle \text{diag}(g_1, I_{d-4}), \text{diag}(g_2, I_{d-4}) \rangle, \mathscr{L}, \mathfrak{S}, N)$ // Let $\langle g_1, g_2 \rangle = U$

Remark 5.25

The output of Algorithm GoingDownToDim4LD [Alg. 18] is $U \leq G$ with $U \cong \text{SL}(4, q)$ and $U^{\mathscr{L}}$ is stingray embedded in $G^{\mathscr{L}}$. Since U is a terminal group we want to use the algorithm presented in Section 5.1.3 to compute a base case group in G and the algorithms of Section 5.2 to recognise the base case group constructively. We also use Algorithm InducedActionRepresentationGroup [Alg. 12] to continue these computations with 4×4 matrices instead of $d \times d$ matrices. ◀

5.1.5 Complete GoingDown algorithm

We finish this chapter by giving the overall GoingDown algorithm which uses as sub-algorithms Algorithm GoingDownToDim4LD [Alg. 18] and Algorithm GoingDownFinalStepSL [Alg. 15].

Algorithm 19: GoingDown

Input:
- $\langle X \rangle = G = \text{SL}(d, q)$ with $d \geq 4$
- $N \in \mathbb{N}$

Output: fail **OR** $(U, \mathscr{L}, \mathfrak{S})$ **where**
- $U \leq G$ with $U \cong \text{SL}(2, q)$,
- $\mathscr{L}$ is a base change matrix such that $U^{\mathscr{L}}$ is stingray embedded in $G^{\mathscr{L}}$ and
- $\mathfrak{S}$ is an MSLP from X to generators of U

function GOINGDOWN(G, N)

1 $(U, \mathscr{L}_1, \mathfrak{S}_1, N) \leftarrow$ GOINGDOWNTODIM4LD(d, G, N) // Remark 5.7 and 5.8

2 $U \leftarrow$ INDUCEDACTIONREPRESENTATIONGROUP(U)

3 $(U, \mathscr{L}_2, \mathfrak{S}_2, N) \leftarrow$ GOINGDOWNFINALSTEPSL(U, N) // Remark 5.7 and 5.8

4 $\mathfrak{S} \leftarrow$ Composition of $\mathfrak{S}_1$ and $\mathfrak{S}_2$ **AND** $\mathscr{L} \leftarrow \mathscr{L}_1 \operatorname{diag}(\mathscr{L}_2, I_{d-4})$

5 **return** $(\langle \operatorname{diag}(g_1, I_{d-2}), \operatorname{diag}(g_2, I_{d-2}) \rangle, \mathscr{L}, \mathfrak{S})$ // Let $\langle g_1, g_2 \rangle = U$

Theorem 5.26

Algorithm GOINGDOWN [Alg. 19] is correct and terminates.

Proof. Clear. □

5.2 BASECASE algorithm

Let $\langle X \rangle = H = \mathrm{SL}(2, q)$. Our goal is to express the standard generators of $\mathrm{SL}(2, q)$ given in Definition 5.1 as MSLPs in X. For this we use an algorithm by Conder and Leedham-Green [28].

We briefly outline the idea and present pseudo-code for the constructive recognition algorithm of $\mathrm{SL}(2, q)$ given in [28]. All results, concepts and algorithms of this section are well-known.

Algorithm 20: CONSTRUCTIVERECOGNITIONSL2

Input:
- $\langle X \rangle = G = \mathrm{SL}(2, q)$
- $N \in \mathbb{N}$

Output: **fail OR** $(\mathscr{L}, S, \mathfrak{S}, N')$ **where**
- $\mathscr{L} \in \mathrm{GL}(2, q)$ is a base change matrix,
- S are standard generators of $\langle X \rangle^{\mathscr{L}}$,
- $\mathfrak{S}$ is an MSLP from $X^{\mathscr{L}}$ to the standard generators S and
- $N' \in \mathbb{N}$ where $N - N'$ is the number of random selections that were used

function ConstructiveRecognitionSL2(G, N)

1 **repeat**
2 $a \leftarrow$ PseudoRandom(G) **AND** $N \leftarrow N-1$ // Step 1
3 **until** a **has order** $q-1$ **OR** $N \leq 0$
4 $(v,w) \leftarrow$ eigenvectors of a in $\mathbb{F}_q^2$ **AND** $(\iota, \iota^{-1}) \leftarrow$ the corresponding eigenvalues in $\mathbb{F}_q$ // Step 2
5 $c \leftarrow$ PseudoRandom(G) **AND** $b \leftarrow a^c$ // Step 3
6 **repeat** // Explained in Remark 5.28
7 $c_1 \leftarrow$ PseudoRandom(G) **AND** $i \leftarrow$ for loop condition if possible // Step 4
8 $N \leftarrow N-1$
9 **until** $b^i c_1$ **fixes** $\langle v \rangle$ **OR** $N \leq 0$
10 $a_2 \leftarrow [a, b^i c_1]$ // Step 5
11 **repeat**
12 $c_2 \leftarrow$ PseudoRandom(G) **AND** $j \leftarrow$ for loop condition if possible // Step 6
13 $N \leftarrow N-1$
14 **until** w **is an eigenvector for** $b^j c_2$ **and** $a_2 \leftarrow [a, b^j c_2]$ **is not the identity** **OR** $N \leq 0$
15 $\mathscr{L} \leftarrow (v,w)$ **AND** $a, a_1, a_2 \leftarrow a^{\mathscr{L}^{-1}}, a_1^{\mathscr{L}^{-1}}, a_2^{\mathscr{L}^{-1}}$ // Step 7
16 **if** $N \leq 0$ **then**
17 **return** fail
18 **return** $(\mathscr{L}, (a, a_1, a_2), \mathfrak{S}$ an MSLP for these steps, $N)$

Theorem 5.27

Algorithm ConstructiveRecognitionSL2 [Alg. 20] terminates using at most N random selections and works correctly.

Proof. [28]. □

Remark 5.28

1) Algorithm ConstructiveRecognitionSL2 [Alg. 20] requires the availability of a discrete logarithm oracle in $\mathbb{F}_q$ and, hence, the constructive recognition algorithms of this thesis require this oracle as well. The algorithms used for constructive recognition of base case groups as outlined in Table 3.4 can be replaced by other constructive recognition algorithms for base

case groups. That means if an algorithm to solve constructive recognition of $\mathrm{SL}(2,q)$ without the discrete logarithm becomes available, then that algorithm can be replaced.

2) The complexity of Algorithm ConstructiveRecognitionSL2 [Alg. 20] is polynomial in $\log(q)$ given a discrete logarithm oracle and a prime factorisation of $q-1$.

3) For further discussions about constructive recognition of $\mathrm{SL}(2,q)$ and the use of the discrete logarithm, see [59]. ◀

Example 5.29

We continue Example 5.20. As explained and proven in Section 5.1.4 we can describe stingray elements by their action on their stingray bodies and, therefore, as lower-dimensional matrices. Hence, we extract the non-trivial blocks of the matrices a_9 and a_{10} of Example 5.20 and continue with the matrices

$$g_1 := \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \quad \text{and} \quad g_2 := \begin{pmatrix} 4 & 1 \\ 4 & 0 \end{pmatrix}$$

and apply Algorithm ConstructiveRecognitionSL2 [Alg. 20] to $\langle g_1, g_2 \rangle$. By random search we identify

$$a := \begin{pmatrix} 4 & 4 \\ 2 & 1 \end{pmatrix} \in \langle g_1, g_2 \rangle$$

which has order $4 = 5-1 = q-1$. In the second step we compute the eigenvalues 3 and 2 as well as the eigenvectors $(3,1)$ and $(4,1)$ of a, respectively. Using the base change matrix $\mathscr{L}_4 := ((3,1),(4,1))$ we obtain

$$a = \begin{pmatrix} 3 & 0 \\ 0 & 2 \end{pmatrix}^{\mathscr{L}_4}.$$

Again by random search we compute

$$b := \begin{pmatrix} 4 & 1 \\ 3 & 1 \end{pmatrix} \quad \text{and} \quad c_1 := \begin{pmatrix} 4 & 2 \\ 4 & 1 \end{pmatrix}$$

and, hence,

$$b^0 c_1 := \begin{pmatrix} 2 & 0 \\ 1 & 3 \end{pmatrix}^{\mathscr{L}_4}.$$

The product $b^0 c_1$ fixes $\langle (3,1) \rangle$ and, therefore, already has a form similar to a transvection. In general

we can use the discrete logarithm to compute a suitable i in step 4. By taking the commutator $[a, b^0 c_1]$ the vector $(3,1)$ lies in the 1-eigenspace and, thus,

$$[a, b^0 c_1] := \begin{pmatrix} 1 & 0 \\ 4 & 1 \end{pmatrix}^{\mathscr{L}_4}$$

which is a transvection. Analogously we can compute $j = 1$ and

$$c_2 := \begin{pmatrix} 4 & 1 \\ 0 & 4 \end{pmatrix}^{\mathscr{L}_4}$$

such that

$$bc_2 := \begin{pmatrix} 4 & 2 \\ 0 & 4 \end{pmatrix}^{\mathscr{L}_4}.$$

Overall, the following matrices are written as words in g_1, g_2, namely

$$a = \begin{pmatrix} 3 & 0 \\ 0 & 2 \end{pmatrix}^{\mathscr{L}_4}, \quad a_1 := [a, b^0 c_1] = \begin{pmatrix} 1 & 0 \\ 4 & 1 \end{pmatrix}^{\mathscr{L}_4} \quad \text{and} \quad a_2 := [a, b^1 c_2] = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}^{\mathscr{L}_4}.$$

◀

Remark 5.30

Algorithm ConstructiveRecognitionSL2 [Alg. 20] returns three matrices (a, a_1, a_2) and a base change matrix $\mathscr{L}$. The three matrices (a, a_1, a_2) are given as elements of $G^{\mathscr{L}^{-1}}$ where a is a diagonal matrix with a primitive element at position $(1,1)$ while a_1 is a transvection with a non-zero entry at position $(2,1)$ and a_2 is a transvection with a non-zero entry at position $(1,2)$.

This generating set is not the same as the one described in Definition 5.1. Therefore, we must perform some additional computations. ◀

The generating set of Definition 5.1 can be computed using the output of Algorithm ConstructiveRecognitionSL2 [Alg. 20] by using Algorithm StandardGeneratingSetSL2 [Alg. 21]. Algorithm StandardGeneratingSetSL2 [Alg. 21] is a slight variation of Algorithm `CompleteSL2Basis`

implemented by O'Brien in Magma [16] at Magma/package/Group/GrpMat/CompTree/GrpMat/sl2q/natural.m in version Magma V2.27-5 and proceeds as follows.

We start by choosing a primitive element $\omega \in \mathbb{F}_q$ and compute the diagonal matrix $\mathrm{diag}(\omega, \omega^{-1})$ as a word in (a, a_1, a_2). This is done by computing $k \in \mathbb{N}$ such that $a^k = \mathrm{diag}(\omega, \omega^{-1})$. This process can be seen as normalising the primitive element of a at position $(1,1)$ to a desired primitive element. This is an optional step which also involves the discrete logarithm. Since Algorithm ConstructiveRecognitionSL2 [Alg. 20] already utilises the discrete logarithm, a second application of the discrete logarithm does not affect the asymptotic complexity. In practice and for large fields the run-time can be improved by skipping the normalisation process of the primitive element at position $(1,1)$ of a.

From now on, we say that $\omega \in \mathbb{F}_q$ is the primitive element at position $(1,1)$ of a independent of whether the normalisation step has been carried out or not. We continue by writing the inverse of the entry $(2,1)$ of a_1 and the inverse of the entry $(1,2)$ of a_2 as an $\mathbb{F}_p$ linear combination of $\{\omega^0, \omega^2, \ldots, \omega^{2f-2}\}$. Afterwards we compute a specific product of a and a_1 using this linear combination respectively of a and a_2 to write the transvections $E_{2,1}(1)$ respectively $E_{1,2}(1)$ as words in (a, a_1, a_2). Then we compute lists A'_D and A'_U containing the transvections $A'_D = \{E_{2,1}(\omega^0), E_{2,1}(\omega^2), \ldots, E_{2,1}(\omega^{2f-2}\}$ and $A'_U = \{E_{1,2}(\omega^0), E_{1,2}(\omega^2), \ldots, E_{1,2}(\omega^{2f-2}\}$. The set $\{\omega^0, \omega^2, \ldots, \omega^{2f-2}\}$ is an $\mathbb{F}_p$ basis of $\mathbb{F}_q$ but for the standard generators of $\mathrm{SL}(2,q)$ of Definition 5.1 we use the $\mathbb{F}_p$ basis $\{\omega^0, \omega^1, \ldots, \omega^{f-1}\}$. Therefore, we write ω^k for $0 \leq k \leq f-1$ as a linear combination in $\{\omega^0, \omega^2, \ldots, \omega^{2f-2}\}$. Afterwards we evaluate this linear combination in $\{E_{2,1}(\omega^0), E_{2,1}(\omega^2), \ldots, E_{2,1}(\omega^{2f-2}\}$ to obtain the transvection $E_{2,1}(\omega^k)$ and evaluate this linear combination in $\{E_{1,2}(\omega^0), E_{1,2}(\omega^2), \ldots, E_{1,2}(\omega^{2f-2})\}$ to obtain the transvection $E_{1,2}(\omega^k)$. Lastly, we use the transvections $E_{1,2}(1)$ and $E_{2,1}(1)$ to write the permutation matrix $z_1 \in \mathrm{SL}(2,q)$ as a word in (a, a_1, a_2).

These computations are summarised in Algorithm StandardGeneratingSetSL2 [Alg. 21].

Algorithm 21: StandardGeneratingSetSL2

Input: ▶ (a, a_1, a_2) the output of Algorithm ConstructiveRecognitionSL2 [Alg. 20]

Output: ▶ An MSLP $\mathcal{S}$ from (a, a_1, a_2) to standard generators of Definition 5.1

function StandardGeneratingSetSL2((a, a_1, a_2))

Choose ω a primitive element of $\mathbb{F}_q$

Find k such that $a[1,1]^k = \omega$ (Note that this involves the discrete logarithm)

$a_0 \leftarrow a^k$ **AND** $a_4 \leftarrow I_2$ **AND** $a_5 \leftarrow I_2$

Write $(a_1[2,1])^{-1} = \iota_0\omega^0 + \iota_1\omega^2 + \iota_2\omega^4 + \ldots + \iota_{f-1}\omega^{2f-2}$ for $\iota_0, \ldots, \iota_{f-1} \in \mathbb{F}_p$

Write $(a_2[1,2])^{-1} = \jmath_0\omega^0 + \jmath_1\omega^2 + \jmath_2\omega^4 + \ldots + \jmath_{f-1}\omega^{2f-2}$ for $\jmath_0, \ldots, \jmath_{f-1} \in \mathbb{F}_p$

for $i \in [0, \ldots, f-1]$ **do**

  $c \leftarrow a_0^{-i}$ **AND** $a_4 \leftarrow a_4((a_1)^c)^{\iota_i}$ **AND** $a_5 \leftarrow a_5((a_2)^c)^{\jmath_i}$

$A'_u \leftarrow [a_5]$ **AND** $A'_d \leftarrow [a_4]$

for $i \in [0, \ldots, f-1]$ **do**

  Append(A'_u, $a_5^{(a_0^i)}$)

  Append(A'_d, $a_4^{(a_0^i)}$)

$A_u \leftarrow [\,]$ **AND** $A_d \leftarrow [\,]$

for $i \in [1, \ldots, f]$ **do**

  Write $\omega_i = \iota_0\omega^0 + \iota_1\omega^2 + \iota_2\omega^4 + \ldots + \iota_{f-1}\omega^{2f-2}$ for $\iota_0, \ldots, \iota_{f-1} \in \mathbb{F}_p$

  Append(A_u, $A'_u[1]^{\iota_0}A'_u[2]^{\iota_1}\ldots A'_u[f]^{\iota_{f-1}}$)

  Append(A_d, $A'_d[1]^{\iota_0}A'_d[2]^{\iota_1}\ldots A'_d[f]^{\iota_{f-1}}$)

$\mathcal{S} \leftarrow$ MSLP from (a, a_1, a_2) to $(A_u, A_d, (A_u[1])^{-1}A_d[1](A_u[1])^{-1})$ using the steps above

return $\mathcal{S}$

Theorem 5.31

Algorithm StandardGeneratingSetSL2 [Alg. 21] terminates using at most N random selections and works correctly.

Proof. The algorithm terminates clearly. The correctness follows by easy computations. □

Remark 5.32

The output of Algorithm StandardGeneratingSetSL2 [Alg. 21] can be interpreted as follows. The set A_u corresponds to upper transvections, namely the transvections $E_{1,2}(\omega_i)$ for $1 \leq i \leq f$ of Definition 5.1, the set A_d corresponds to lower transvections, namely the transvections $E_{2,1}(\omega_i)$ for $1 \leq i \leq d$ of Definition 5.1, and $(A_u[1])^{-1}A_d[1](A_u[1])^{-1}$ corresponds to z_1 of Definition 5.1. Note that z_2 is trivial if $d = 2$. ◀

Example 5.33

In Example 5.29 we computed matrices a, a_1 and a_2 as words in $g_1^{(\mathscr{L}_4)^{-1}}$ and $g_2^{(\mathscr{L}_4)^{-1}}$ where

$$a = \begin{pmatrix} 3 & 0 \\ 0 & 2 \end{pmatrix}, \quad a_1 = \begin{pmatrix} 1 & 0 \\ 4 & 1 \end{pmatrix} \quad \text{and} \quad a_2 = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}.$$

In the following all matrices are given as elements of $\langle g_1, g_2 \rangle^{(\mathscr{L}_4)^{-1}}$ as we do not require an additional base change matrix. Applying Algorithm StandardGeneratingSetSL2 [Alg. 21] to (a, a_1, a_2) returns the standard generators of Definition 5.1 as follows. We choose 2 as the primitive element of $\mathbb{F}_5$. Solving the discrete logarithm, i.e. computing k such that $3^k = 2$, returns $k = 3$ and

$$a_0 := a^3 = \begin{pmatrix} 3^3 & 0 \\ 0 & 2^3 \end{pmatrix} = \begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix}.$$

For the upper transvections nothing is to do as $\tilde{a}_2 := a_2 = E_{12}(2^0)$. For the lower transvections we set $(a_1[2,1])^{-1} = (-1)^{-1} = -1 = -1 \cdot 1 = -1 \cdot 2^0$, i.e. $\iota_0 = -1$. Then

$$c = a_0^{-0} = I_2 \quad \text{and} \quad \tilde{a}_1 := I_2(a_1^c)^{\iota_0} = a_1^{\iota_0} = a_1^{-1} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}.$$

This concludes the computation of the transvections of Definition 5.1. Lastly we compute z_1 by

$$(A_u[1])^{-1}A_d[1](A_u[1])^{-1} = \tilde{a}_2^{-1}\tilde{a}_1\tilde{a}_2^{-1} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}.$$

◀

5.3 GoingUp algorithm

In this section we assume that we have found a stingray embedded subgroup H of $G^{\mathscr{L}}$ for a known base change matrix $\mathscr{L} \in \mathrm{GL}(d,q)$, i.e. $H \leq \langle X^{\mathscr{L}} \rangle = G^{\mathscr{L}} = \mathrm{SL}(d,q)$, with $H \cong \mathrm{SL}(2,q)$ and an MSLP from X to the standard generators of H. Starting from this setting, we describe an algorithm to compute standard generators of G. Similarly to the GoingDown algorithm we present a GoingUp step which is being used repeatedly. In this thesis we present two versions of a GoingUp step, both of which are randomised algorithms. The GoingUp algorithm presented in this section builds upon a strategy proposed in a draft by Ákos Seress and Max Neunhöffer.

The GoingUp step of this section uses linear algebra and relies on completely original ideas. On the one hand, every computation of this solution can be performed extremely fast in the natural representation of classical groups which results in a very efficient GoingUp step. On the other hand, the computations require that the given representation of a classical group is the natural representation and the GoingUp step of this section cannot easily be modified for non-natural representations. Using the GoingUp step repeatedly yields an ascending recognition chain

$$H = H_{(0)} \leq H_{(1)} \ldots \leq H_{(\ell-1)} \leq H_{(\ell)} = G.$$

as described in Definition 3.9.

The second solution for a GoingUp step of this thesis is based on ideas of [59]. Since this solution is applicable with minor modifications to all classical groups, the concepts and algorithms are presented in a separate chapter, see Chapter 9.

Comparing the GoingUp step of this section, in the following called the *solution based on linear algebra*, and the GoingUp step in Chapter 9, in the following called the *solution based on involutions*, we can observe that, practically speaking, the solution based on linear algebra performs much faster than the solution based on involutions when applied to a classical group in its natural representation. However, this comes at the price that the MSLPs for the standard generators of G are longer using the solution based on linear algebra instead of the solution based on involutions and that the solution based on linear algebra cannot be used in non-natural settings.

For the remainder of this thesis, we denote the GoingUp step of the solution based on linear algebra as GoingUp step, while the GoingUp step of the solution based on involutions is discussed in Chapter 9.

5.3.1 Overview of the GoingUp step

We start this section by stating the main theorem and giving a description of the GoingUp step in detail. The theorem is proved by the correctness of the presented GoingUp step of this section. The algorithm consists of seven phases. One of these phases is randomised while the others are deterministic. Since one phase is randomised, the GoingUp step overall is also randomised. The GoingUp step algorithm is presented as Algorithm GoingUpStep [Alg. 27] in Section 5.3.5. In Section 5.3.6 we call the GoingUp step repeatedly to construct standard generators of the input group G which results in the final Algorithm GoingUp [Alg. 28].

We start with our hypothesis on the setting of this section and then state the main theorem. Afterwards, the theorem is divided into seven phases which we investigate in more detail and prove their correctness in the course of this section.

Hypothesis 5.34

For the remainder of this section we assume $\langle X\rangle = G = \mathrm{SL}(d,q)$ containing a stingray embedded subgroup $H \leq \langle X^{\mathscr{L}}\rangle = G^{\mathscr{L}}$ with $H \cong \mathrm{SL}(n,q)$ for $n < d$ and for a known base change matrix $\mathscr{L} \in \mathrm{GL}(d,q)$. Moreover, standard generators Y_n of H are given as words in X. Let $V = \mathbb{F}_q^d$ and suppose that $\mathscr{B} = (v_1,\ldots,v_d)$ is a basis for V and let $V_n = \langle v_1,\ldots,v_n\rangle$ and $F_{d-n} = \langle v_{n+1},\ldots,v_d\rangle$ (cf. Definition 2.7). We assume that H acts on V_n as $\mathrm{SL}(n,q)$ and that H fixes F_{d-n} point-wise. Recall that $(\omega_1,\ldots,\omega_f)$ is an $\mathbb{F}_p$-basis for $\mathbb{F}_q$.

The main theorem summarising GoingUp step is the following.

Theorem 5.35

Let $X \subseteq \mathrm{SL}(d,q)$ such that $\langle X\rangle = G = \mathrm{SL}(d,q)$, let $2 \leq n < d$ with $n = 2$ or n odd and let $\mathscr{L} \in \mathrm{GL}(d,q)$ be a base change matrix. Let $Y_n^{\mathscr{L}}$ be a set of standard generators for the subgroup $\mathrm{SL}(n,q)$ stingray embedded into $G^{\mathscr{L}}$. Furthermore, let $\mathfrak{S}$ be an SLP from X to Y_n and let $n' \coloneqq \min\{2n-1, d\}$.

Then there is an algorithm that computes a base change matrix $\mathscr{L}' \in \mathrm{GL}(d,q)$ together with an SLP $\mathfrak{S}'$ from X to a set $Y_{n'}$, which is a set of standard generators for $\mathrm{SL}(n',q)$ and $\langle Y_{n'}^{\mathscr{L}'} \rangle$ is stingray embedded in $G^{\mathscr{L}'}$.

We prove Theorem 5.35 by stating an algorithm. The algorithm consists of seven phases, called SL1) to SL7), which are discussed and proven in the remainder of this section.

Remark 5.36

The general idea of the algorithm is the following. Let $H = \mathrm{SL}(V_n) \cong \mathrm{SL}(n,q)$ be stingray embedded in $G^{\mathscr{L}}$ and $G = \mathrm{SL}(d,q)$. Given standard generators Y_n for H, we construct an element $g \in G^{\mathscr{L}}$ with the following properties:

$(C1)$ $\dim(V_n + V_n g) = n'$,

$(C2)$ if $n' < d$, then $\dim(\mathrm{Fix}(H) + \mathrm{Fix}(g)) = \dim(F_{d-n} + \mathrm{Fix}(g)) = d$.

Note that H acts on V_n as $\mathrm{SL}(V_n) \cong \mathrm{SL}(n,q)$ and similarly H^g acts on $V_n g$ as $\mathrm{SL}(V_n g) \cong \mathrm{SL}(n,q)$. By Lemma 2.5 $\dim(V_n \cap V_n g) = \dim(V_n) + \dim(V_n g) - \dim(V_n + V_n g) = 2n - n' = \max\{1, 2n-d\}$. In Section 5.3.3 we formulate an additional property $(C3)$ which cannot be defined at this point. If g does not satisfy $(C3)$, then we construct a new element $g \in G^{\mathscr{L}}$. In the remainder of this section we prove that this setup allows us to choose a basis for the n'-dimensional subspace $V_{n'} = V_n + V_n g$ of V in such a way that we can use g-conjugates of certain transvections in H to assemble standard generators for $\mathrm{SL}(n',q)$ with respect to the new basis. Consequently, we conclude that $\langle H, H^g \rangle$ is indeed isomorphic to $\mathrm{SL}(n',q)$. If $n' \leq d$, then we have roughly doubled the degree from n to $2n-1$ using g. ◀

Definition 5.37

Assume the setting as described in Hypothesis 5.34 and let $g \in G^{\mathscr{L}}$.

1) If g satisfies $(C1)$ and $(C2)$ of Remark 5.36, then g is a *weak doubling element* with respect to H.
2) If g is a weak doubling element with respect to H and additionally fixes v_n, then g is a *doubling element* with respect to H.

If the context is clear, then a weak doubling element and doubling element with respect to H are only denoted by a weak doubling element and doubling element.

Remark 5.38

The next sections can be summarised as follows.

1) **Contruction of a doubling element:** Construct an element $g \in G^{\mathscr{L}}$ satisfying the properties $(C1)$ and $(C2)$ and fixing v_n, i.e. a doubling element as in Definition 5.37. This is achieved by random selection of elements of $G^{\mathscr{L}}$ and discussed in Section 5.3.2.
2) **Contruction of a new base change matrix:** Construct a new base change matrix $\mathscr{L}'$ such that $\langle H, H^g \rangle^{\mathscr{L}'}$ is stingray embedded in $G^{\mathscr{L}\mathscr{L}'}$ which is possible because of the properties $(C1)$ and $(C2)$. The computation of $\mathscr{L}'$ is deterministic and discussed in Section 5.3.3. In that section we also formulate an additional property $(C3)$ which can only be given after constructing $\mathscr{L}'$. If g does not satisfy $(C3)$, then we start over from 1).
3) **Contruction of transvections and standard generators:** If 2) is successful, then we can conclude that $\langle H, H^g \rangle^{\mathscr{L}'} \cong \mathrm{SL}(n', q)$. Hence, we construct transvections and standard generators for $\langle H, H^g \rangle^{\mathscr{L}'}$. Note that we only need to compute permutation matrices corresponding to the n'- and $(n'-1)$-cycles of Definition 5.1 as the transvections of the standard generators of H are also the transvections of the standard generators of $\langle H, H^g \rangle^{\mathscr{L}'}$. The computations performed in 3) are deterministic and discussed in Section 5.3.4. ◀

In each section the task described in Remark 5.38 is divided into more phases which are discussed in detail in the corresponding section and labelled as SL1) to SL7). Lastly, in Section 5.3.5 the phases SL1) to SL7) are combined into a single algorithm for the GoingUp step. Proving its correctness yields the proof for Theorem 5.35. In Section 5.3.5 we also discuss two examples of the GoingUp step. When first reading through the GoingUp step some parts may not be clear directly, but one can always return to these two examples to compare the theoretical results with their applications.

5.3.2 Construction of a dimension doubling element

The goal in this section is the construction of a doubling element, i.e. an element $g \in G^{\mathscr{L}}$ satisfying the properties $(C1)$ and $(C2)$ and fixing v_n, leading to Algorithm ComputeDoublingElement [Alg. 23]. Recall from Hypothesis 5.34 the setting for this section. We give a condensed version of what is achieved.

Remark 5.39

SL1) Construct an element $t \in H$ which has a fixed space of dimension $d-n+1$. This can be done easily as we have standard generators Y_n for H.

SL2) Compute random elements $a \in G^{\mathscr{L}}$ until $\tilde{g} := t^a$ satisfies the two conditions $(C1)$ and $(C2)$ of Remark 5.36, i.e. until $\tilde{g}$ is a weak doubling element.

SL3) Find a conjugate $g \in G^{\mathscr{L}}$ of $\tilde{g}$ which also satisfies the two conditions $(C1)$ and $(C2)$ and additionally fixes v_n, i.e. a doubling element. ◀

Note that the GoingUp algorithm is randomised as in SL2) we search for random $a \in G^{\mathscr{L}}$ such that $\tilde{g} := t^a$ satisfies the two properties $(C1)$ and $(C2)$ of Remark 5.36. We do not analyse the proportion of usable elements in this chapter and only prove the correctness. Complexity results are given in Chapter 10.

The following lemma shows how SL1) is carried out in the solution of this thesis. Note that there are multiple other choices for $t \in H$ having a fixed space of dimension $d-n+1$. Recall the permutation matrices z_1 and z_2 from Definition 5.1 which are permutation matrices corresponding to n and $n-1$ cycles, respectively.

Lemma 5.40

An element $t \in H \leq G^{\mathscr{L}}$ which has a fixed space of dimension $d-n+1$ is given by

$$t := \begin{cases} E_{1,2}(1), & \text{if } n = 2, \\ z_1, & \text{if } n > 2 \text{ and } p \text{ is even}, \\ z_2, & \text{if } n > 2 \text{ and } p \text{ is odd}. \end{cases}$$

Proof. If $n = 2$, then we take $t = E_{1,2}(1)$. Its fixed space is $\langle v_2, \ldots, v_d \rangle$ and thus has dimension $d-1 = d-2+1 = d-n+1$. If $n > 2$, then n is odd by assumption in Theorem 5.35. If the characteristic p is odd, then we take for t an $(n-1)$-cycle z_2 which has fixed space is $\langle v_1, v_{n+1}, v_{n+2}, \ldots, v_d \rangle$. If $p = 2$, then we select an n-cycle z_1 which has fixed space $\langle v_1 + v_2 + \ldots + v_n, v_{n+1}, \ldots, v_d \rangle$. □

The element t is constructed as an MSLP in the given standard generators Y_n of $H \cong \mathrm{SL}(n,q)$, but

can also be written explicitly as a matrix of $\langle X\rangle^{\mathcal{L}}$ at a cost of $\mathcal{O}(d^2)$.

Now that t has a fixed space of dimension $d-n+1$, we compute a random $a \in G^{\mathcal{L}}$ and check whether $\tilde{g} = t^a$ satisfies the properties $(C1)$ and $(C2)$ of Remark 5.36. We repeat this until $\tilde{g}$ has the required properties.

Remark 5.41

Verifying $(C1)$ and $(C2)$ of Remark 5.36 is one of the few places in GoingUp step where we must multiply $(d \times d)$-matrices, since a is created as a matrix in the same input basis as X, and we must construct the matrix of a and $\tilde{g}$ as an element of $\langle X\rangle^{\mathcal{L}}$. This is necessary to check $(C1)$ and $(C2)$.◀

Lemma 5.42

Let t be as in Lemma 5.40 and let $\tilde{g} := t^a$ be a weak doubling element.

1) $\dim(V_n \cap \mathrm{Fix}(\tilde{g})) \geq 1$ and if $n' < d$, then $\dim(V_n \cap \mathrm{Fix}(\tilde{g})) = 1$.
2) $V_{n'}$ is invariant under the action of $\tilde{g}$.
3) If $n' < d$, then $\dim(F_{d-n} \cap \mathrm{Fix}(\tilde{g})) = d - n'$.

Proof. In the following we use Lemma 2.5.

1) Since $\tilde{g}$ and t are conjugate we know that $\dim(\mathrm{Fix}(\tilde{g})) = \dim(\mathrm{Fix}(t)) = d-n+1$. Note that $\dim(V_n + \mathrm{Fix}(\tilde{g})) \leq \dim(V) = d$. Hence,

$$\begin{aligned}\dim(V_n \cap \mathrm{Fix}(\tilde{g})) &= \dim(V_n) + \dim(\mathrm{Fix}(\tilde{g})) - \dim(V_n + \mathrm{Fix}(\tilde{g}))\\ &= n + (d-n+1) - \dim(V_n + \mathrm{Fix}(\tilde{g}))\\ &= d + 1 - \dim(V_n + \mathrm{Fix}(\tilde{g})) \geq 1.\end{aligned}$$

Now let $n' < d$, i.e. $n' = 2n-1$. Then using $(C1)$ of Remark 5.36 it follows that

$$\begin{aligned}\dim(V_n \cap V_n\tilde{g}) &= \dim(V_n) + \dim(V_n\tilde{g}) - \dim(V_n + V_n\tilde{g})\\ &= n + n - n'\\ &= n + n - (2n-1) = 1.\end{aligned}$$

Notice that $V_n \cap \mathrm{Fix}(\tilde{g}) \subseteq V_n\tilde{g}$ and thus

$$\dim(V_n \cap \mathrm{Fix}(\tilde{g})) \leq \dim(V_n \cap V_n\tilde{g}) = 1.$$

Since $\dim(V_n \cap \mathrm{Fix}(\tilde{g})) \geq 1$, the result follows.

2) If $n' = d$, then the statement is clear. Let us assume that $n' = 2n-1 < d$. We construct a basis of $V_{n'}$ which shows that $V_{n'}$ is invariant under the action of $\tilde{g}$. A basis of $V_{n'}$ consists of n' elements since $\dim(V_{n'}) = n'$ by $(C1)$. Notice that using 1) it follows that

$$\begin{aligned}\dim(V_n + \mathrm{Fix}(\tilde{g})) &= \dim(V_n) + \dim(\mathrm{Fix}(\tilde{g})) - \dim(V_n \cap \mathrm{Fix}(\tilde{g})) \\ &= n + (d-n+1) - 1 = d.\end{aligned}$$

Therefore, $V_n + \mathrm{Fix}(\tilde{g}) = V$. Since $V_n \leq V_{n'}$, it follows that $V_{n'} + \mathrm{Fix}(\tilde{g}) = V$ which implies that

$$\begin{aligned}\dim(V_{n'} \cap \mathrm{Fix}(\tilde{g})) &= \dim(V_{n'}) + \dim(\mathrm{Fix}(\tilde{g})) - \dim(V_{n'} + \mathrm{Fix}(\tilde{g})) \\ &= (2n-1) + (d-n+1) - d = n.\end{aligned}$$

Since $\dim(V_n \cap (V_{n'} \cap \mathrm{Fix}(\tilde{g}))) = \dim(V_n \cap \mathrm{Fix}(\tilde{g})) = 1$, it follows that

$$\begin{aligned}\dim(V_n + (V_{n'} \cap \mathrm{Fix}(\tilde{g}))) &= \dim(V_n) + \dim(V_{n'} \cap \mathrm{Fix}(\tilde{g})) - \dim(V_n \cap (V_{n'} \cap \mathrm{Fix}(\tilde{g}))) \\ &= n + n - 1 = 2n - 1 = n'.\end{aligned}$$

Since $V_n + (V_{n'} \cap \mathrm{Fix}(\tilde{g})) \leq V_{n'}$, it follows that $V_n + (V_{n'} \cap \mathrm{Fix}(\tilde{g})) = V_{n'}$. Therefore, we can choose a basis of $V_{n'}$ as follows:

- Choose a non-zero vector $v_1 \in V_n \cap \mathrm{Fix}(\tilde{g})$.
- Select $n-1$ vectors $v_2, \ldots, v_n \in V_n$ to extend this to a basis of V_n.
- Choose $n-1$ vectors from $V_{n'} \cap \mathrm{Fix}(\tilde{g})$ to extend this to a basis of $V_{n'}$.

Clearly this basis is invariant under the action of $\tilde{g}$ as either $v_i\tilde{g} = v_i$ or $v_i\tilde{g} \in V_n\tilde{g} \leq V_{n'}$.

3) If $n' < d$, then $\dim(F_{d-n} + \mathrm{Fix}(\tilde{g})) = d$ by (C2). Hence, as claimed, we obtain

$$\begin{aligned}\dim(F_{d-n} \cap \mathrm{Fix}(\tilde{g})) &= \dim(F_{d-n}) + \dim(\mathrm{Fix}(\tilde{g})) - \dim(F_{d-n} + \mathrm{Fix}(\tilde{g})) \\ &= (d-n) + (d-n+1) - d \\ &= d - 2n + 1 = d - n'.\end{aligned}$$

□

Remark 5.43

If $n' = d < 2n-1$, then it is possible that $\dim(V_n \cap \mathrm{Fix}(\tilde{g})) > 1$ which is important in Remark 5.51.

◀

Corollary 5.44

Let $\tilde{g}$ be a weak doubling element and V_n as in Hypothesis 5.34. Then $V_n + V_n\tilde{g} = V_n + V_n\tilde{g}^{-1}$.

Proof. Note that $V_n + V_n\tilde{g} = V_{n'} \overset{3) \text{ of } 5.42}{=} V_{n'}\tilde{g}^{-1} = (V_n + V_n\tilde{g})\tilde{g}^{-1} = V_n\tilde{g}^{-1} + V_n$. □

Remark 5.45

Note that there is one more condition (C3) which can only be formulated in Section 5.3.3. If that condition is not satisfied, then we restart from SL2) and try another element a. ◀

Algorithm 22: ComputeWeakDoublingElement

Input:
- ▶ $\langle X \rangle = G \leq \mathrm{SL}(d,q)$
- ▶ A base change matrix $\mathscr{L} \in \mathrm{GL}(d,q)$
- ▶ $\mathrm{SL}(n,q) \cong \langle Y_n \rangle = H \leq G^{\mathscr{L}}$ stingray embedded and constructively recognised
- ▶ $N \in \mathbb{N}$

Output: **fail OR** $(c, \mathfrak{S}', N')$ **where**
- ▶ $\tilde{g} \in G^{\mathscr{L}}$ is a weak doubling element,
- ▶ $\mathfrak{S}$ is an MSLP from $X \cup Y_n$ to $\tilde{g}$ and
- ▶ $N' \in \mathbb{N}$ where $N - N'$ is the number of random selections that were used

```
function ComputeWeakDoublingElement(G, 𝒢, H, N)
    Choose t ∈ H as described in Lemma 5.40                    // SL1)
    repeat                                                     // Start of SL2)
        N ← N − 1
        if N < 0 then
            return fail
        g̃ ← t^a for random a ∈ G^𝒢
    until g̃ satisfies (C1) and (C2)
    𝔖 ← MSLP from X ∪ Y_n to g̃
    return (g̃, 𝔖, N)
```

Our next goal is to construct a conjugate g of $\tilde{g}$ such that g is a doubling element. We seek an element $L \in H$ with $v_n L = v$, where $0 \neq v \in V_n \cap \mathrm{Fix}(\tilde{g})$, and write L as a word in the standard generators Y_n and finally compute $g := L\tilde{g}L^{-1}$. Lemma 5.46 proves the existence of an element $L \in H$ with the described properties, Lemma 5.47 shows that g is a doubling element and lastly we describe how L can be found. We start by proving that $L \in H \cong \mathrm{SL}(n,q)$ exists.

Lemma 5.46

$\mathrm{SL}(d,q)$ acts doubly transitive on the 1-dimensional subspaces of $\mathbb{F}_q^d$.

Proof. [91, Theorem 4.1] □

Next we show that $g := L\tilde{g}L^{-1}$ is a doubling element.

Lemma 5.47

Let t, G and H be as in Lemma 5.40 and $\tilde{g} := t^a$ a weak doubling element for a random element $a \in G^{\mathcal{G}}$. Let $L \in H$ such that $v_n L = v$ where $0 \neq v \in V_n \cap \mathrm{Fix}(\tilde{g})$. Then

1) $L\tilde{g}L^{-1}$ satisfies (C1) and (C2).
2) $v_n \in V_n \cap \mathrm{Fix}(L\tilde{g}L^{-1})$.

Proof. 1) Since $L \in H$, it follows that $V_n L = V_n$ and so

$$\dim(V_n + V_n L\tilde{g}L^{-1}) = \dim(V_n L + V_n L\tilde{g}) = \dim(V_n + V_n\tilde{g}) = n'$$

and $(C1)$ holds. Moreover, $LF_{d-n}L^{-1} = F_{d-n}$ (notice $L \in H$) and so

$$\dim(\mathrm{Fix}(L\tilde{g}L^{-1}) + F_{d-n}) = \dim(\mathrm{Fix}(L\tilde{g}L^{-1}) + LF_{d-n}L^{-1}) = \dim(\mathrm{Fix}(\tilde{g}) + F_{d-n}) = d$$

and $(C2)$ holds.

2) We have $v_n \in V_n$ and $v_n L\tilde{g}L^{-1} = v\tilde{g}L^{-1} = vL^{-1} = v_n$ since $0 \neq v \in V_n \cap \mathrm{Fix}(\tilde{g})$. □

Remark 5.48

We proceed as follows to find L satisfying the hypothesis of Lemma 5.47:

1) Let $0 \neq v \in V_n \cap \mathrm{Fix}(\tilde{g})$ and $v = \sum_{j=1}^{n} \lambda_j v_j$ for $\lambda_j \in \mathbb{F}$. This is possible since $v \in V_n$ and $(v_1, \ldots, v_n)$ is a basis of V_n.
2) If $\lambda_n \neq 0$, then we normalise the first entry of v by multiplying it by λ_n^{-1}, allowing us, without loss of generality, to assume $\lambda_n = 1$. Through this normalisation L can be written as a product of the elements $E_{n,j}(\omega_i)$ for $1 \leq i \leq f$ and $1 \leq j \leq n-1$. The elements $E_{n,j}(\omega_i)$ are expressed in the standard generators Y_n of $H \cong \mathrm{SL}(n,q)$, as shown in Lemma 5.3.
3) If $\lambda_n = 0$, then we select a non-zero λ_j and find a $L' \in H$ such that $v_n L' = vE_{j,n}(1)$, following the same procedure outlined in 2). Finally, we define $L := L'E_{j,n}(1)^{-1}$. ◀

Algorithm 23: ComputeDoublingElement

Input:
- ▶ $\langle X \rangle = G \leq \mathrm{SL}(d,q)$
- ▶ A base change matrix $\mathscr{L} \in \mathrm{GL}(d,q)$
- ▶ $\mathrm{SL}(n,q) \cong \langle Y_n \rangle = H \leq G^{\mathscr{L}}$ stingray embedded and constructively recognised
- ▶ $N \in \mathbb{N}$

Output: fail **OR** $(g, \mathfrak{S}, N')$ **where**
- ▶ $g \in G^{\mathscr{L}}$ is a doubling element,
- ▶ $\mathfrak{S}$ is an MSLP from $X \cup Y_n$ to g and
- ▶ $N' \in \mathbb{N}$ where $N - N'$ is the number of random selections that were used

```
function ComputeDoublingElement(G, 𝒟, H, N)
    (g̃, 𝔖₁, N) ← ComputeWeakDoublingElement(G, 𝒟, H, N)
    if g̃ = fail then
        return fail
    L ← as described in Remark 5.48 AND 𝔖₂ ← MSLP from Y_n to L        // SL3)
    g ← L g̃ L^{-1} AND 𝔖 ← Compose 𝔖₁ and 𝔖₂
    return (g, 𝔖, N)
```

Remark 5.49

Notice that the matrices $E_{1,2}(\omega_i)$ and $E_{2,1}(\omega_i)$ are given as elements of $\langle X \rangle^{\mathscr{D}}$ which contains the stingray embedded subgroup H isomorphic to $\mathrm{SL}(n,q)$ and standard generators Y_n can be written as words in $X^{\mathscr{D}}$ as described in Hypothesis 5.34. ◀

5.3.3 Construction of a new base change matrix

The goal of this section is the construction of a new base change matrix $\mathscr{D}'$ such that $\langle H, H^g \rangle^{\mathscr{D}'}$ is stingray embedded in $G^{\mathscr{D}\mathscr{D}'}$. All computations in this section are deterministic. Recall from Hypothesis 5.34 the setting for this section and that $g \in G^{\mathscr{D}}$ is a doubling element, i.e. satisfies $(C1)$ and $(C2)$ and fixes v_n as described in Section 5.3.2. This section only covers one phase as described in the next remark.

Remark 5.50

SL4) Compute a base change matrix $\mathscr{D}'$ such that $\langle H, H^g \rangle^{\mathscr{D}'}$ is stingray embedded in $G^{\mathscr{D}\mathscr{D}'}$. ◀

The goal of SL4) is to compute a base change matrix $\mathscr{D}' \in \mathrm{GL}(d,q)$ such that $\langle H, H^g \rangle^{\mathscr{D}'}$ is stingray embedded in $G^{\mathscr{D}\mathscr{D}'}$ and if $\langle H, H^g \rangle \cong \mathrm{SL}(n',q)$, then standard generators of $\langle H, H^g \rangle^{\mathscr{D}'}$ can be constructed using the steps SL5) to SL7) described in Section 5.3.4. Note that we formulate an additional condition $(C3)$ in this section which can only be defined after the construction of $\mathscr{D}'$. If g does not satisfy $(C3)$, then we must choose a new random element $a \in G^{\mathscr{D}}$ and restart from phase SL2). If g satisfies $(C3)$, then we can conclude in Section 5.3.4 that $\langle H, H^g \rangle \cong \mathrm{SL}(n',q)$ and construct standard generators of $\langle H, H^g \rangle$.

Remark 5.51

The base change matrix $\mathscr{L}'$ is computed by constructing a specific basis of $\mathbb{F}_q^d$. Let $\pi\colon V \to F_{d-n}$ be the projection map to F_{d-n} of the decomposition $V = V_n \oplus F_{d-n}$.

1) The first n vectors of the new basis are equal to the vectors in the old basis, i.e. $v_i' := v_i$ for $1 \leq i \leq n$.
2) Notice that the vectors $\pi(v_1 g), \ldots, \pi(v_{n-1} g)$ are linear independent if $n' < d$ using the following argument: Let $a_i \in \mathbb{F}_q$ such that $0 = \sum_{i=1}^{n-1} a_i \pi(v_i g) = \pi(\sum_{i=1}^{n-1} a_i v_i g)$. Thus, $\sum_{i=1}^{n-1} a_i v_i g \in V_n$ and $\sum_{i=1}^{n-1} a_i v_i g = (\sum_{i=1}^{n-1} a_i v_i) g \in V_n g$. Note that $V_n \cap V_n g = \langle v_n \rangle$ and that $(\sum_{i=1}^{n-1} a_i v_i) g \in V_n \cap V_n g$. Hence, λv_n is a linear combination of $(v_1 g, \ldots, v_{n-1} g)$ such that $\lambda v_n = \sum_{i=1}^{n-1} a_i v_i g$ which is equivalent to $\lambda v_n = \lambda v_n g^{-1} = \sum_{i=1}^{n-1} a_i v_i$. This is only possible if $\lambda = a_1 = \ldots = a_{n-1} = 0$ as $(v_1, \ldots, v_n)$ is a basis. With a similar argument it can be shown that $\pi(v_1 g), \ldots, \pi(v_{n-1} g)$ contains a linear independent subset of size $d - n$ if $d = n'$.

   The vectors $v_{n+1}', \ldots, v_{n'}'$ of the new basis are chosen as a linearly independent subset of the vectors $\pi(v_1 g), \ldots, \pi(v_{n-1} g)$. If $n' < d$, then we take all the vectors $\pi(v_1 g), \ldots, \pi(v_{n-1} g)$ and otherwise we choose a linearly independent subset.
3) In the case $n' < d$ we extend $(v_1', \ldots, v_{n'}')$ to a basis $\mathscr{B}' = (v_1', \ldots, v_d')$ of V by choosing a basis of $F_{d-n} \cap \mathrm{Fix}(g)$ which is possible by condition $(C2)$. ◀

The matrix $\mathscr{L}' \in \mathrm{GL}(d, q)$ is now chosen to be the base change matrix between $\mathscr{B}$ and $\mathscr{B}'$. Note that $(C1)$ and $(C2)$ ensure that $\langle H, H^g \rangle$ can be stingray embedded in $G^{\mathscr{L}}$ using Remark 5.51.

Remark 5.52

With respect to the new basis $\mathscr{B}'$ we can observe the following.

1) $V_n = \langle v_1', \ldots, v_n' \rangle$.
2) $V_n + V_n g = \langle v_1', \ldots, v_{n'}' \rangle$.
3) $\langle v_{n+1}', \ldots, v_d' \rangle \subseteq \mathrm{Fix}(H)$.
4) Note that 1) and 3) ensure that for our stingray embedded subgroup $H \cong \mathrm{SL}(n, q)$ in $G^{\mathscr{L}}$ also $H^{\mathscr{L}'}$ is a stingray embedded subgroup in $G^{\mathscr{L}\mathscr{L}'}$. ◀

From this point on, we consider all of our matrices as elements of $G^{\mathscr{L}\mathscr{L}'}$. For the standard generators Y_n of $H \cong \mathrm{SL}(n, q)$, no changes are necessary, see 3) of Remark 5.52. However, the matrix g must be conjugated by the base change matrix $\mathscr{L}'$ and we denote the resulting matrix by c, i.e. $c := g^{\mathscr{L}'}$.

At this point we formulate the last condition $(C3)$. If $(C3)$ is satisfied by c, then we can conclude that $\langle H, H^g \rangle \cong \mathrm{SL}(n', q)$ in Section 5.3.4. The necessity for condition $(C3)$ is discussed in more detail in Section 5.3.4.

Remark 5.53

The final condition on c is the following.

$(C3)$ The vectors v_j' and $v_j' c^{-\mathrm{Tr}}$ for $1 \leq j \leq n-1$ span as an $\mathbb{F}_q$-subspace the $\mathbb{F}_q$-subspace $\langle v_1', \ldots, v_{n-1}', v_{n+1}', \ldots, v_{n'}' \rangle$. ◀

Definition 5.54

Assume the setting as described in Hypothesis 5.34. Let $g \in G^{\mathscr{L}}$ be a doubling element and $c := g^{\mathscr{L}'}$, where $\mathscr{L}'$ is chosen as in Remark 5.51. If c additionally satisfies $(C3)$, then c is a *strong doubling element*.

Note that $(C3)$ can only be verified after computing the base change matrix $\mathscr{L}'$ which is also the reason why this condition is introduced at this point. If $(C3)$ is not satisfied by c, then we return to SL2) and try another random element $a \in G^{\mathscr{L}}$. Note that $\langle H, H^g \rangle^{\mathscr{L}'} = \langle H, H^c \rangle$. We give an algorithm to compute a strong doubling element $c \in G^{\mathscr{L}\mathscr{L}'}$ satisfying $(C1)$, $(C2)$ and $(C3)$ in pseudo-code called Algorithm ComputeStrongDoublingElement [Alg. 24] using Algorithm ComputeDoublingElement [Alg. 23] of Section 5.3.4.

Algorithm 24: ComputeStrongDoublingElement

Input:
- $\langle X \rangle = G \leq \mathrm{SL}(d, q)$
- A base change matrix $\mathscr{L} \in \mathrm{GL}(d, q)$
- $\mathrm{SL}(n, q) \cong \langle Y_n \rangle = H \leq G^{\mathscr{L}}$ stingray embedded and constructively recognised
- $N \in \mathbb{N}$

Output: fail **OR** $(c, \mathscr{L}', \mathfrak{S}, N')$ **where**
- $c \in G^{\mathscr{L}\mathscr{L}'}$ is a strong doubling element,
- $\mathscr{L}' \in \mathrm{GL}(d, q)$ is a base change matrix as in Remark 5.51,
- $\mathfrak{S}$ is an MSLP from $X \cup Y_n$ to c and
- $N' \in \mathbb{N}$ where $N - N'$ is the number of random selections that were used

```
function ComputeStrongDoublingElement(G, 𝒟, H, N)
    while N > 0 do
        (g, 𝔖, N) ← ComputeDoublingElement(G, 𝒟, H, N)        // Remark 5.7 and 5.8
        𝒟' ← as described in Remark 5.51 AND c ← g^{𝒟'}        // SL4)
        if the submatrix (c^{-1})_{i,j} for n+1 ≤ i ≤ n' and 1 ≤ j ≤ n−1 has full rank then
            return (c, 𝒟', 𝔖, N)
    return fail
```

5.3.4 Construction of transvections and standard generators

Given a strong doubling element $c \in G^{\mathcal{D}\mathcal{D}'}$ as described in Section 5.3.3 the goal of this section is to conclude that $\langle H, H^c\rangle \cong \mathrm{SL}(n', q)$ and the construction of transvections and standard generators for $\langle H, H^c\rangle$. All computations in this section are deterministic. Recall from Hypothesis 5.34 the setting of this section and that $c \in G^{\mathcal{D}\mathcal{D}'}$ satisfies $(C1)$, $(C2)$ and $(C3)$. Moreover, note that $\langle H, H^c\rangle$ is stingray embedded in $G^{\mathcal{D}\mathcal{D}'}$ by the construction of $\mathcal{D}' \in \mathrm{GL}(d, q)$ in Section 5.3.3. In this section the last phases SL5) to SL7) are described in the next remark.

Remark 5.55

SL5) Using c, construct transvections $E_{j,n}(\omega_i)$ for $n < j \leq n'$ for $\langle H, H^c\rangle \cong \mathrm{SL}(n', q)$.

SL6) Using c, construct transvections $E_{n,j}(\omega_i)$ for $n < j \leq n'$ for $\langle H, H^c\rangle \cong \mathrm{SL}(n', q)$.

SL7) Using the transvections of phases SL5) and SL6) construct standard generators for $\langle H, H^c\rangle \cong \mathrm{SL}(n', q)$ by assembling permutation matrices corresponding to n'- and $(n'-1)$-cycles as in Definition 5.1. ◀

In SL5) and SL6), the transvections $E_{j,n}(\omega_i)$ and $E_{n,j}(\omega_i)$ are conjugated by c and transformed by matrix multiplications into transvections of $\mathrm{SL}(n', q) \cong \langle H, H^c\rangle$ as stingray embedded elements of $G^{\mathcal{D}\mathcal{D}'}$. This is necessary to subsequently compute permutation matrices of $\mathrm{SL}(n', q)$ in SL7). For SL6) we need the transvections computed in SL5). Recall from Definition 2.29 the notation $T_{v,w}$ for transvections.

The purpose of SL5) is to construct the elements $E_{j,n}(\omega_i)$ for $1 \leq i \leq f$ and $n+1 \leq j \leq n'$ as elements of $G^{\mathcal{D}\mathcal{D}'}$. For this consider the transvections $T^c_{\omega_i v'_j, v'_n}$ for $1 \leq i \leq f$ and $1 \leq j \leq n-1$ and

the lemmas from Section 2.3.

For any $1 \leq i \leq f$, Lemma 2.32 states that the conjugate $T^c_{\omega_i v'_j, v'_n}$ corresponds to the transvection $T_{\omega_i v'_j c^{-\mathrm{Tr}}, v'_n}$, given that $v'_n c = v'_n$. Note that all the transvections $T^c_{\omega_i v'_j, v'_n}$ fix v'_n, as this is the case for c and the transvections $T_{\omega_i v'_j, v'_n}$. Consequently, $\langle v'_n \mid v'_j c^{-\mathrm{Tr}} \rangle = 0$ for $1 \leq j \leq n-1$.

Thus, for $1 \leq i \leq f$ and $1 \leq j \leq n-1$, the vectors $\omega_i v'_j$ and $\omega_i v'_j c^{-\mathrm{Tr}}$ are all orthogonal to v_n under the standard scalar product, see Definition 2.31. Moreover, we have MSLPs for T_{w, v'_n} for all vectors $w \in A := \{\omega_i v'_j \mid 1 \leq i \leq f, 1 \leq j \leq n-1\} \cup \{\omega_i v'_j c^{-\mathrm{Tr}} \mid 1 \leq i \leq f, 1 \leq j \leq n-1\}$. From $(C3)$, it follows that the vectors in A span the subspace $\langle v'_1, \ldots, v'_{n-1}, v'_{n+1}, \ldots, v'_{n'} \rangle$. Hence, utilising Lemmas 2.33 and 2.34, we express all $T_{\omega_i v'_j, v'_n}$ for $1 \leq i \leq f$ and $n+1 \leq j \leq n'$ as words of the elements T_{w, v'_n} for $w \in A$. This results in the successful construction of the elements $E_{j,n}(\omega_i)$ in $G^{\mathcal{L}\mathcal{L}'}$.

Remark 5.56

1.) Note that we must perform linear algebra computations over the prime field GF(p) here, but once again we use the results of the computations only to write an MSLP for these transvections.

2.) In our implementation we check the property $(C3)$ of c of Remark 5.36 already in Algorithm ComputeStrongDoublingElement [Alg. 24] to avoid unnecessary computations at this point. ◀

The idea of SL5) is to compute $E_{j,n}(\omega_i)^c$ for $1 \leq i \leq f$ and $1 \leq j \leq n-1$ and somehow to construct $E_{j,n}(\omega_i)$ for $1 \leq i \leq f$ and $n+1 \leq j \leq n'$ using the elements $\{E_{j,n}(\omega_i)^c \mid 1 \leq i \leq f, 1 \leq j \leq n-1\}$. To obtain a usable algorithm we start by computing $T^c_{\omega_i v'_j, v'_n}$ as elements of $G^{\mathcal{L}\mathcal{L}'}$, i.e. $E_{j,n}(\omega_i)^c$, which is given in Lemma 5.57.

Lemma 5.57

Let $n' = \min\{2n-1, d\}$, let $1 \leq j \leq n-1$ and let $c \in G^{\mathscr{L}\mathscr{L}'}$ be a strong doubling element. Let

$$y := \begin{pmatrix} 1 & 0 & \dots & 0 & \omega_i(c^{-1})_{1,j} & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 & \omega_i(c^{-1})_{2,j} & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & \omega_i(c^{-1})_{n-1,j} & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & \omega_i(c^{-1})_{n+1,j} & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & \omega_i(c^{-1})_{n',j} & 0 & \dots & 1 \end{pmatrix} \in \mathrm{SL}(n', q).$$

Then $E_{j,n}(\omega_i)^c = \mathrm{diag}(y, I_{d-n'}) \in G^{\mathscr{L}\mathscr{L}'}$.

Proof. If $k \in \{n'+1, \dots, d\}$, then

$$e_k E_{j,n}(\omega_i)^c = e_k c^{-1} E_{j,n}(\omega_i) c = e_k E_{j,n}(\omega_i) c = e_k c = e_k.$$

Moreover for $k \in \{1, \dots, n'\} - \{n\}$ we have

$$\begin{aligned} e_k c^{-1} E_{j,n}(\omega_i) c &= (c^{-1})_{k,-} E_{j,n}(\omega_i) c \\ &= ((c^{-1})_{k,-} + \omega_i (c^{-1})_{k,j} e_n) c \\ &= e_k c^{-1} c + \omega_i (c^{-1})_{k,j} e_n c = e_k + \omega_i (c^{-1})_{k,j} e_n \end{aligned}$$

and

$$e_n c^{-1} E_{j,n}(\omega_i) c = e_n E_{j,n}(\omega_i) c = e_n c = e_n.$$

□

Remark 5.58

Using the standard generators of $H \cong \mathrm{SL}(n, q)$, the matrix $E_{j,n}(\omega_i)^c$ can be multiplied by transvections of H resulting in row and column operations. Using row and column operations another element of $\langle H, H^c \rangle$ can be constructed where the entries in the n-th column of $E_{j,n}(\omega_i)^c$ above the

(n,n) entry are zero such that matrices of the form $\mathrm{diag}(y_{j,n,\omega_i}, I_{d-n'})$ are constructed where

$$y_{j,n,\omega_i} \coloneqq \begin{pmatrix} 1 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & \omega_i(c^{-1})_{n+1,j} & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & \omega_i(c^{-1})_{n',j} & 0 & \dots & 1 \end{pmatrix}.$$

In the following we denote the process of using row and column operations to construct another matrix with zeros at specific positions as "eliminating" these entries. Using $(C3)$ the column vectors $((c^{-1})_{n+1,i}, \dots, (c^{-1})_{n',i})$ for $1 \leq i \leq n-1$ of length $(n'-n)$ below the n-row entry of c^{-1} generate the full $f(n'-n)$-dimensional $\mathbb{F}_p$-vector space. Observe that the group $G_{n'} \coloneqq \langle \{ y_{j,n,\omega_i} \mid 1 \leq j \leq n-1, 1 \leq i \leq f \} \rangle$ is abelian and in fact isomorphic to $\mathbb{F}_q^{n'-n}$ via the isomorphism given by

$$\varphi \colon \mathbb{F}_q^{n-1} \to G_{n'},\ (\lambda_{n+1}, \dots, \lambda_{n'}) \mapsto \begin{pmatrix} 1 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & \lambda_{n+1} & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & \lambda_{n'} & 0 & \dots & 1 \end{pmatrix}.$$

That means that for elements

$$y_1 \coloneqq \begin{pmatrix} 1 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & (y_1)_{n+1} & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & (y_1)_{n'} & 0 & \dots & 1 \end{pmatrix}, \quad y_2 \coloneqq \begin{pmatrix} 1 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & (y_2)_{n+1} & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & (y_2)_{n'} & 0 & \dots & 1 \end{pmatrix}$$

the isomorphism φ yields

$$y_1^{\imath_1} y_2^{\imath_2} = \begin{pmatrix} 1 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & \imath_1(y_1)_{n+1} + \imath_2(y_2)_{n+1} & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & \imath_1(y_1)_{n'} + \imath_2(y_2)_{n'} & 0 & \dots & 1 \end{pmatrix}$$

where $\imath_1, \imath_2 \in \mathbb{F}_p$ represented as elements of $\{0, \dots, p-1\}$, i.e. linear combinations of the columns are computed. Note that our goal is the computation of the transvections $E_{j,n}(\omega_i)$ for $1 \leq i \leq f$

and $n+1 \leq j \leq n'$, e.g.

$$E_{n+1,n}(\omega_i) = \begin{pmatrix} 1 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & \omega_i & 1 & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & 0 & 0 & \dots & 1 \end{pmatrix}.$$

By multiplying the matrices y_{j,n,ω_i} appropriately, the standard basis of $\mathbb{F}_q^{n-1}$ as an $\mathbb{F}_p$-vector space, i.e. $\omega_1 e_1, \dots, \omega_f e_1, \dots, \omega_1 e_{n-1}, \dots, \omega_f e_{n-1}$, can be computed and thus the transvections $E_{j,n}(\omega_i)$ for $1 \leq i \leq f$ and $n+1 \leq j \leq n'$. ◀

We combine Lemma 5.57 and Remark 5.58 and obtain Algorithm ComputeVerticalTransvections [Alg. 25].

In SL6), the transvections $E_{n,j}(\omega_i)$ for $1 \leq i \leq f$ and $n+1 \leq j \leq n'$ as elements of $G^{\mathscr{L}\mathscr{L}'}$ should be computed. This is more complicated than computing the transvections $E_{j,n}(\omega_i)$ for $1 \leq i \leq f$ and $n+1 \leq j \leq n'$ in phase SL5) because $c^{-\mathrm{Tr}}$ does not necessarily fix the n-th basis vector anymore. The next lemma therefore carries out a computation similar to the one described in Lemma 5.57 and has the desired consequences. Recall that for a matrix $a \in \mathrm{GL}(d,q)$ the i-th row of a is denoted by $a_{i,\_}$ and the i-th column of a is denoted by $a_{\_,i}$.

Algorithm 25: ComputeVerticalTransvections

Input:
- ▶ $\mathrm{SL}(n,q) \cong \langle Y_n \rangle = H \leq \mathrm{SL}(d,q)$ stingray embedded and constructively recognised
- ▶ $c \in \mathrm{SL}(d,q)$ a strong doubling element

Output:
- ▶ $T_V := \{E_{j,n}(\omega_i) \mid 1 \leq i \leq f, n+1 \leq j \leq n'\} \subset \langle H, H^c \rangle$ transvections of $\mathrm{SL}(n',q)$
- ▶ An MSLP $\mathfrak{S}$ from $Y_n \cup \{c\}$ to T_V

function COMPUTEVERTICALTRANSVECTIONS(H, c)

```
// Function implements SL5) based on Lemma 5.57 and Remark 5.58
T̃_V ← [ ]
for λ ∈ {ω_1,...,ω_f} and j ∈ {2,...,n−1} do
    T ← E_{j,n}(λ)^c
    for k ∈ [1,...,n−1] do
        T ← E_{k,n}(−λ(c^{-1})_{k,j})T        // Note that E_{k,n}(−λ(c^{-1})_{k,j}) ∈ H
    ADD(T̃_V, T)
T_V ← [ ]
for λ ∈ {ω_1,...,ω_f} and j ∈ {n+1,...,n'} do
    E_{j,n}(λ) ← Multiply the matrices of T̃_V suitably
    ADD(T_V, E_{j,n}(λ))
𝔖 ← MSLP for the computations of T_V
return (T_V, 𝔖)
```

Lemma 5.59

Let $n' = \min\{2n-1, d\}$, let $1 \leq j \leq n-1$ and let $c \in G^{\mathscr{L}\mathscr{L}'}$ be a strong doubling element. Let $y \in \mathrm{SL}(n', q)$ with

$$y := I_{n'} + \mathrm{diag}(\omega_i(c^{-1})_{1,n}, \ldots, \omega_i(c^{-1})_{n-1,n}, \omega_i, \omega_i(c^{-1})_{n+1,n}, \ldots, \omega_i(c^{-1})_{n',n}) \cdot (c_{j,-}, \ldots, c_{j,-})^{\mathrm{Tr}}.$$

Then $E_{n,j}(\omega_i)^c = \mathrm{diag}(y, I_{d-n'}) \in G^{\mathscr{L}\mathscr{L}'}$.

Proof. If $k \in \{n'+1, \ldots, d\}$, then

$$e_k E_{n,j}(\omega_i)^c = e_k c^{-1} E_{n,j}(\omega_i) c = e_k E_{n,j}(\omega_i) c = e_k c = e_k.$$

For $k \in \{1, \ldots, n'\} - \{n\}$ we have

$$\begin{aligned} e_k c^{-1} E_{n,j}(\omega_i) c &= (c^{-1})_{k,-} E_{n,j}(\omega_i) c \\ &= ((c^{-1})_{k,-} + \omega_i (c^{-1})_{k,n} e_j) c \\ &= e_k c^{-1} c + \omega_i (c^{-1})_{k,n} e_j c = e_k + \omega_i (c^{-1})_{k,n} c_{j,-} \end{aligned}$$

and

$$e_n c^{-1} E_{n,j}(\omega_i) c = e_n E_{n,j}(\omega_i) c = (\omega_i e_j + e_n) c = \omega_i e_j c + e_n c = \omega_i c_{j,-} + e_n.$$

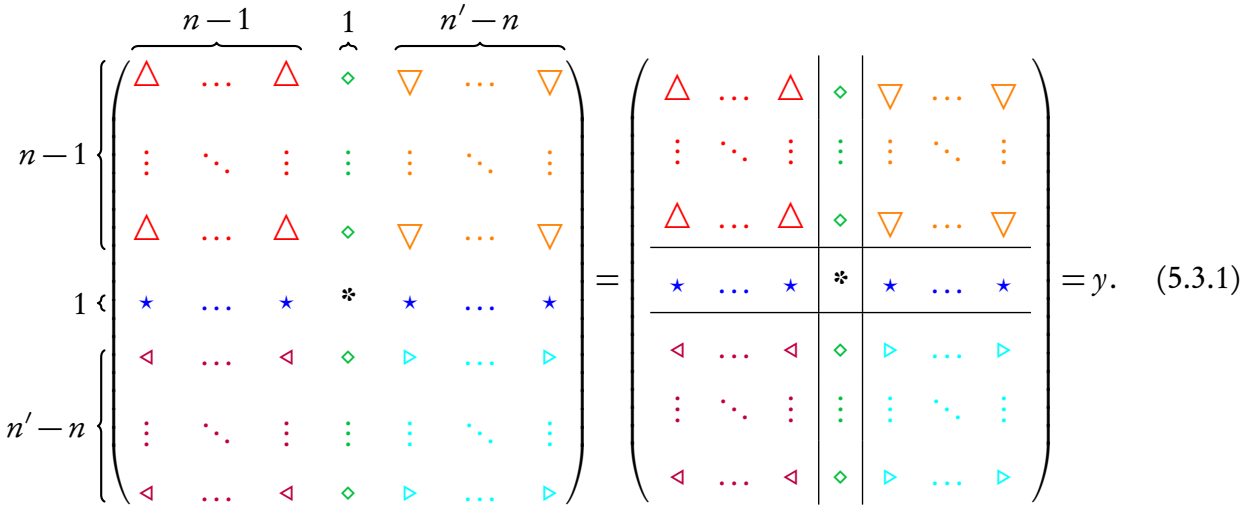
□

After conjugating $E_{n,j}(\omega_i)$ by c the result becomes slightly more complicated than the conjugation of $E_{j,n}(\omega_i)$ by c. Remark 5.60 displays how the transvections $E_{n,j}(1)$ for $n+1 \leq j \leq n'$ can still be computed starting from $E_{n,j}(\omega_i)^c$.

Remark 5.60

Let $1 \leq j \leq n-1$, let $n' = \min\{2n-1, d\}$ and let c and $\mathscr{L}'$ be as constructed in the previous phases. Moreover, we assume that SL5) has already been performed, i.e. that MSLPs for the transvections $E_{j,n}(\omega_i)$ for $1 \leq i \leq f$ and $n+1 \leq j \leq n'$ have been computed.

By Lemma 5.59 $E_{n,j}(1)^c = \mathrm{diag}(y, I_{d-n'})$ where $y \in \mathrm{SL}(n', q)$ and $1 \leq j \leq n-1$. As in SL5) the $n' \times n'$ top left block y of $E_{n,j}(1)^c$ should be transformed into the transvection $E_{n,n+j}(1)$. To illustrate the operations we are performing, we represent the $n' \times n'$ top left block y of $E_{n,j}(1)^c$ as in (5.3.1).



(5.3.1)

We are using $n'-n$ to deal with both cases for n', i.e. $n' = 2n-1 \leq d$ and $n' = d \neq 2n-1$ simultaneously. If $n' = 2n-1 \leq d$, then $n'-n = 2n-1-n = n-1$ and if $n' = d \neq 2n-1$, then $n'-n < n-1$. Remember that we have standard generators and, therefore, all transvections of $\mathrm{SL}(n,q)$ for the top left $n \times n$ block (consisting of red upper triangles, green diamonds, blue stars and black flower). We start by eliminating the red (upper-) and orange (lower-) triangle blocks by

adding the n-th row using row operations. Let $k \in \{1,\dots,n-1\}$ and i as in Lemma 5.59. Then

$$\begin{aligned}(e_k + \omega_i(c^{-1})_{k,n}c_{j,-}) + (-(c^{-1})_{k,n})(\omega_i c_{j,-} + e_n) &= e_k + \omega_i(c^{-1})_{k,n}c_{j,-} - (c^{-1})_{k,n}\omega_i c_{j,-} - (c^{-1})_{k,n}e_n \\ &= e_k - (c^{-1})_{k,n}e_n\end{aligned}$$

where $e_k + \omega_i(c^{-1})_{k,n}c_{j,-}$ is the k-th row of $E_{n,j}(1)^c$ and $\omega_i c_{j,-} + e_n$ is the n-th row of $E_{n,j}(1)^c$. Hence, the addition of these rows can be achieved by multiplying with $E_{k,n}(-(c^{-1})_{k,n})$ from the left. Note that $E_{k,n}(-(c^{-1})_{k,n}) \in \mathrm{SL}(n,q)$ and, therefore, we can write $E_{k,n}(-(c^{-1})_{k,n})$ as a word in Y_n. After these $n-1$ row operations, the matrix of (5.3.1) is transformed into the matrix of (5.3.2):

$$\left(\begin{array}{ccc|c|ccc} & & & \diamond & & & \\ & I_{n-1} & & \vdots & & 0 & \\ & & & \diamond & & & \\ \hline \star & \dots & \star & * & \star & \dots & \star \\ \hline \triangleleft & \dots & \triangleleft & \diamond & \triangleright & \dots & \triangleright \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \triangleleft & \dots & \triangleleft & \diamond & \triangleright & \dots & \triangleright \end{array}\right). \tag{5.3.2}$$

We proceed analogously with the rows below the n-th row using the same argument for $k \in \{n+1,\dots,n'\}$. Notice that this is only possible since we have already computed the transvections $E_{j,n}(\omega_i)$ for $1 \le i \le f$ and $n+1 \le j \le n'$ in SL5). After these additional $n-1$ row operations we transform (5.3.2) into

$$\left(\begin{array}{ccc|c|ccc} & & & \diamond & & & \\ & I_{n-1} & & \vdots & & 0 & \\ & & & \diamond & & & \\ \hline \star & \dots & \star & * & \star & \dots & \star \\ \hline & & & \diamond & & & \\ 0 & & & \vdots & & I_{n-1} & \\ & & & \diamond & & & \end{array}\right). \tag{5.3.3}$$

Notice that the green diamond entries are known, i.e. the entry at position (k,n) is $-(c^{-1})_{k,n}$, since

$(e_k+\omega_i(c^{-1})_{k,n}c_{j,-})+(-(c^{-1})_{k,n})(\omega_i c_{j,-}+e_n)=e_k-(c^{-1})_{k,n}e_n$. By adding the columns 1 to $n-1$ multiplying by the corresponding scalars to the n-th column, the green diamond entries in the n-th column above the n-th entry can be eliminated. This can be performed by multiplying the matrices $E_{k,n}((c^{-1})_{k,n})\in H$ from the right. The entry in position (n,n) is changed to 1 during these operations if this was not the case which is shown at the end of this remark. After these column operations (5.3.3) is transformed into

$$\left(\begin{array}{ccc|c|ccc} & & & 0 & & & \\ & I_{n-1} & & \vdots & & 0 & \\ & & & 0 & & & \\ \hline \star & \dots & \star & * & \star & \dots & \star \\ \hline & & & \diamond & & & \\ & 0 & & \vdots & & I_{n-1} & \\ & & & \diamond & & & \end{array}\right). \tag{5.3.4}$$

The rest is now clear. The rows 1 to $n-1$ are added to the n-th row in order to eliminate the entries to the left of the n-th entry in the n-th row. Moreover, the columns $n+1$ to n' are added to the n-th column to eliminate the entries below the n-th entry, resulting in the final matrix

$$\left(\begin{array}{ccc|c|ccc} & & & 0 & & & \\ & I_{n-1} & & \vdots & & 0 & \\ & & & 0 & & & \\ \hline 0 & \dots & 0 & 1 & \star & \dots & \star \\ \hline & & & 0 & & & \\ & 0 & & \vdots & & I_{n-1} & \\ & & & 0 & & & \end{array}\right). \tag{5.3.5}$$

The entry in Position (n,n) of (5.3.5) must be 1 since this is an upper triangular matrix which is also contained in SL. Note that the entries in position (n,n) in this matrix and matrix in (5.3.4) are the same which is why it already was 1 before these final row and column operations. Moreover,

the rows in blue are equal to $(c_{j,-})_{n+1,\ldots,n'}$ which are equal to $e_1,\ldots,e_{n-1}$ by choosing the basis $\mathscr{B}'$ of phase SL4) proactively. Overall, $E_{n,j}(1)^c$ is transformed into $E_{n,n+j}(1)$ using only transvections for which an MSLP is known. Therefore, an MSLP evaluating to $E_{n,n+j}(1)$ is constructed. ◀

We also give a pseudo-code performing SL6) in Algorithm ComputeHorizontalTransvections [Alg. 26] for computing the "horizontal" transvections $E_{n,j}(\omega_i)$ for $n < j \leq n'$ and $1 \leq i \leq f$.

Algorithm 26: ComputeHorizontalTransvections

Input: ▶ $\mathrm{SL}(n,q) \cong \langle Y_n \rangle = H \leq \mathrm{SL}(d,q)$ stingray embedded and constructively recognised
▶ $c \in \mathrm{SL}(d,q)$ a strong doubling element

Output: ▶ $T_H := \{E_{n,j}(1) \mid n+1 \leq j \leq n'\} \subset \langle H, H^c \rangle$ transvections of $\mathrm{SL}(n',q)$
▶ An MSLP $\mathfrak{S}$ from $Y_n \cup \{c\} \cup T_V$ to T_H

function ComputeHorizontalTransvections(H,c)

```
// Function implements SL6) based on Lemma 5.59 and Remark 5.60
T_H ← [ ]
for j ∈ {2,...,n−1} do
    T ← E_{n,j}(1)^c
    for k ∈ [1,...,n−1] do
        T ← E_{k,n}(−(c^{-1})_{k,n})T                    // As in (5.3.2) of Remark 5.60
    for k ∈ [n+1,...,n'] do
        T ← E_{k,n}(−(c^{-1})_{k,n})T                    // As in (5.3.3) of Remark 5.60
    for k ∈ [1,...,n−1] do
        T ← TE_{k,n}((c^{-1})_{k,n})                     // As in (5.3.4) of Remark 5.60
    for k ∈ [1,...,n−1] do
        T ← E_{n,k}(−(c^{-1})_{j,k})T                    // As in (5.3.5) of Remark 5.60
    for k ∈ [n+1,...,n'] do
        T ← TE_{k,n}((c^{-1})_{k,n})                     // As in (5.3.5) of Remark 5.60
    Add(T_H, E)
S ← MSLP for the computations of T_H
return (T_H, S)
```

In the last and final phase, the n'- and $(n'-1)$-cycles are now assembled. This can easily be realised with the transvections from SL5) and SL6) as described in Lemma 5.61.

Lemma 5.61

Let $n = 2$ or n be odd. The permutation matrices z_1', z_2' as in Definition 5.1 for SL(n', q) can be computed using the matrices of the set $X = \{z_1, z_2, E_{i,n}(1), E_{n,i}(1)\}$ for $n+1 \leq i \leq n'$.

Proof. Let n be odd. Transpositions can be easily computed as $E_{i,n}^{-1}(1)E_{n,i}(1)E_{i,n}^{-1}(1)$ is the permutation matrix which corresponds to $(n,i) \in S_{n'}$ for $n < i \leq n'$, where the entry in position (i,n) is equal to -1 and $E_{i,n}(1)E_{n,i}^{-1}(1)E_{i,n}(1)$ is the permutation matrix which corresponds to $(n,i) \in S_{n'}$ for $n < i \leq n'$, where the entry in position (n,i) is equal to -1. Moreover

$$\begin{aligned}&(n,n')\cdot(n,n'-1)\cdot(n,n'-2)\cdot\ldots\cdot(n,n+1)\\ =&(n,n',n'-1)\cdot(n,n'-2)\cdot\ldots\cdot(n,n+1)\\ =&(n,n',n'-1,\ldots,n+1).\end{aligned}$$

and

$$(n,n-1,\ldots,1)\cdot(n,n',n'-1,\ldots,n+1)=(n,n-1,\ldots,1,n',n'-1,\ldots,n+1)=(n',n'-1,\ldots,1),$$
$$(n,n-1,\ldots,2)\cdot(n,n',n'-1,\ldots,n+1)=(n,n-1,\ldots,2,n',n'-1,\ldots,n+1)=(n',n'-1,\ldots,2).$$

Due to the position of -1 in the transpositions, the matrices correspond to the standard generators of Definition 5.1. Now let $n = 2$. Then $E_{3,2}(1)E_{2,3}^{-1}(1)E_{3,2}(1)$ is a word for z_2 of SL$(3,q)$ and $E_{1,2}(1)E_{2,1}^{-1}(1)E_{1,2}(1)(E_{3,2}(1)E_{2,3}^{-1}(1)E_{3,2}(1))$ is a word for z_1 of SL$(3,q)$. □

5.3.5 GoingUp step

Finally all phases and subalgorithms of this section are combined into a single algorithm which can be used for a single GoingUp step as stated in Theorem 5.35. The seven phases of the previous sections are summarised in Remark 5.62. Recall that phases SL1) to SL3) are formulated in Remark 5.39 of Section 5.3.2, that phase SL4) is formulated in Remark 5.50 of Section 5.3.3 and that phases SL5) to SL7) are formulated in Remark 5.55 of Section 5.3.4.

Remark 5.62

Let $\langle X\rangle = G = \mathrm{SL}(d,q)$ contain a stingray embedded subgroup $H \leq \langle X^{\mathscr{L}}\rangle = G^{\mathscr{L}}$ with $H \cong \mathrm{SL}(n,q)$ for $n < d$ and for a known base change matrix $\mathscr{L} \in \mathrm{GL}(d,q)$. Moreover, standard generators Y_n of H are given as words in X. The following seven phases must be performed for Theorem 5.35 and, therefore, for one GoingUp step:

SL1) Construct an element $t \in H$ which has a fixed space of dimension $d-n+1$.

SL2) Compute random elements $a \in G^{\mathscr{L}}$ until $\tilde{g} := t^a$ is a weak doubling element.

SL3) Find a conjugate $g \in G^{\mathscr{L}}$ of $\tilde{g}$ which is a doubling element.

SL4) Compute a base change matrix $\mathscr{L}'$ such that $\langle H, H^g\rangle^{\mathscr{L}'}$ is stingray embedded in $G^{\mathscr{L}\mathscr{L}'}$. Set $c := g^{\mathscr{L}'}$ and verify whether c is a strong doubling element. Proceed if c is a strong doubling element.

SL5) Using c, construct transvections $E_{j,n}(\omega_i)$ for $n < j \leq n'$ for $\langle H, H^c\rangle \cong \mathrm{SL}(n',q)$.

SL6) Using c, construct transvections $E_{n,j}(\omega_i)$ for $n < j \leq n'$ for $\langle H, H^c\rangle \cong \mathrm{SL}(n',q)$.

SL7) Using the transvections of SL5) and SL6) construct standard generators for $\langle H, H^c\rangle \cong \mathrm{SL}(n',q)$ by assembling permutation matrices corresponding to n'- and $(n'-1)$-cycles as in Definition 5.1. ◀

Note that the condition $(C3)$ is tested by Algorithm ComputeStrongDoublingElement [Alg. 24] in SL4) and if c does not satisfy $(C3)$, then we return to SL2). An overall algorithm for one GoingUp step is given in pseudo-code in Algorithm GoingUpStep [Alg. 27].

Algorithm 27: GoingUpStep

Input:
- ▶ $\langle X\rangle = G \leq \mathrm{SL}(d,q)$
- ▶ A base change matrix $\mathscr{L} \in \mathrm{GL}(d,q)$
- ▶ $\mathrm{SL}(n,q) \cong \langle Y_n\rangle = H \leq G^{\mathscr{L}}$ stingray embedded and constructively recognised
- ▶ $N \in \mathbb{N}$

Output: **fail OR** $(Y_{n'}, \mathscr{L}', \mathfrak{S}, N')$ **where**
- ▶ $\mathrm{SL}(\min\{2n-1,d\},q) \cong \langle Y_{n'}\rangle = \tilde{H}$,
- ▶ $\mathscr{L}' \in \mathrm{GL}(d,q)$ is a base change matrix such that $\tilde{H}$ is stingray embedded in $G^{\mathscr{L}\mathscr{L}'}$,
- ▶ $\mathfrak{S}$ is an MSLP from $X \cup Y_n$ to the standard generators $Y_{n'}$ of $\tilde{H}$ and
- ▶ $N' \in \mathbb{N}$ where $N - N'$ is the number of random selections that were used

function GoingUpStep($G, \mathscr{L}, H, N$)

1 $(c, \mathscr{L}', \mathfrak{S}_1, N) \leftarrow$ ComputeStrongDoublingElement($G, \mathscr{L}, H, N$)
2 **if** $c =$ **fail** **then**
3 **return** fail
4 $T_V, \mathfrak{S}_2 \leftarrow$ ComputeVerticalTransvections(H, c)
5 $T_H, \mathfrak{S}_3 \leftarrow$ ComputeHorizontalTransvections(H, c)
6 Use T_V and T_H to construct z_1 and z_2 of SL(n', q) as an MSLP $\mathfrak{S}_4$ using Lemma 5.61 // SL7)
7 Compose $\mathfrak{S}_1, \mathfrak{S}_2, \mathfrak{S}_3, \mathfrak{S}_4$ into one MSLP $\mathfrak{S}$
8 **return** $(\langle H, H^c \rangle, \mathscr{L}', \mathfrak{S}, N)$

Theorem 5.63

Algorithm GoingUpStep [Alg. 27] terminates using at most N random selections and is correct.

Proof. The correctness is clear since the correctness of each phase has been proven in the preceding sections. Therefore, it is left to show that Algorithm GoingUpStep [Alg. 27] terminates. Note that most lines are deterministic and can be performed in finite time. The only critical line is Line 1, i.e. finding a suitable c, which is controlled by N to have only a finite number of tries. □

We provide two examples to illustrate the algorithm and its underlying ideas. We recommend revisiting these examples while delving into the detailed algorithm description to gain a deeper understanding of its workings and overall structure.

Example 5.64

We again continue Example 5.4. In the course of this chapter we computed a base change matrix $\mathscr{L}_4$ and a subgroup $G_3^\Xi \leq G^\Xi = \langle a^{\Xi_1}, a^{\Xi_2} \rangle$ with $G_3^\Xi \cong \mathrm{SL}(2, q)$ and $(G_3^\Xi)^{\mathscr{L}_4}$ is stingray embedded in $(G^\Xi)^{\mathscr{L}_4}$ such that we can express the standard generators of G_3^Ξ as words in a^{Ξ_1}, a^{Ξ_2}. Using the GoingUp step of this section we compute standard generators of a subgroup of G^Ξ isomorphic to SL$(3, q)$. Note that in our setting, $d = 16$, $n = 2$, $q = p = 5$, $n' = 3$ and until we have computed a new base change matrix, every matrix is given as an element of $(G^\Xi)^{\mathscr{L}_4}$. Moreover, we set $H_{(0)}^\Xi := (G_3^\Xi)^{\mathscr{L}_4}$. We now work through the seven phases given in Remark 5.62.

SL1) We need an element t which has a fixed space of dimension $d - n + 1 = 16 - 2 + 1 = 15$. We

may choose t to be a transvection of $H_{(0)}^{\Xi}$, namely

$$t := \left(\begin{array}{cc|c} 1 & 1 & 0 \\ 0 & 1 & 0 \\ \hline 0 & 0 & I_{14} \end{array}\right).$$

SL2) We compute random elements $a \in (G^{\Xi})^{\mathscr{L}_4}$ until $\tilde{g} := t^a$ satisfies the properties $(C1)$ and $(C2)$. In this example we choose the following two elements of GL(16, 5):

$$\underbrace{\begin{pmatrix}
1&4&4&4&4&4&1&2&0&1&1&0&1&3&2&0\\
3&2&0&1&0&4&0&0&4&4&4&2&1&1&4&1\\
3&2&1&2&3&0&4&1&1&0&2&4&2&3&3&0\\
2&2&3&3&0&0&1&0&2&2&0&2&4&2&1&1\\
2&0&4&2&1&3&4&0&0&4&3&0&0&4&1&2\\
4&4&4&1&4&1&1&4&2&2&0&0&1&3&4&1\\
2&2&4&3&1&0&3&0&1&0&3&1&2&4&0&0\\
1&1&0&0&2&4&4&0&4&4&4&3&0&2&1&0\\
0&4&2&2&3&2&3&4&2&3&1&3&1&2&4&3\\
0&0&0&3&2&0&3&3&3&0&1&1&3&1&4&4\\
1&2&2&1&1&2&4&4&0&3&1&2&2&4&0&4\\
3&4&3&1&2&3&3&1&3&1&3&0&4&1&4&4\\
2&3&2&1&1&2&4&4&2&2&3&2&3&1&0&0\\
2&0&3&4&0&4&1&4&3&4&4&1&1&1&1&0\\
1&0&3&3&1&4&0&1&4&0&1&2&1&0&4&2\\
3&3&4&0&1&4&2&3&2&3&1&1&1&0&3&4
\end{pmatrix}}_{:=a}
\underbrace{\begin{pmatrix}
0&1&1&3&1&1&2&2&0&3&2&3&1&3&3&1\\
1&0&4&2&4&4&3&3&0&2&3&2&4&2&2&4\\
3&2&3&1&2&2&4&4&0&1&4&1&2&1&1&2\\
2&3&3&0&3&3&1&1&0&4&1&4&3&4&4&3\\
3&2&2&1&3&2&4&4&0&1&4&1&2&1&1&2\\
3&2&2&1&2&3&4&4&0&1&4&1&2&1&1&2\\
1&4&4&2&4&4&4&3&0&2&3&2&4&2&2&4\\
2&3&3&4&3&3&1&2&0&4&1&4&3&4&4&3\\
1&4&4&2&4&4&3&3&1&2&3&2&4&2&2&4\\
3&2&2&1&2&2&4&4&0&2&4&1&2&1&1&2\\
0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0\\
3&2&2&1&2&2&4&4&0&1&4&2&2&1&1&2\\
2&3&3&4&3&3&1&1&0&4&1&4&4&4&4&3\\
4&1&1&3&1&1&2&2&0&3&2&3&1&4&3&1\\
4&1&1&3&1&1&2&2&0&3&2&3&1&3&4&1\\
3&2&2&1&2&2&4&4&0&1&4&1&2&1&1&3
\end{pmatrix}}_{:=\tilde{g}}.$$

Even though $\tilde{g}$ appears to be a random element of $\mathbb{F}_5^{16\times 16}$, this is not a case. We discussed in Lemma 5.42 that elements in $\mathrm{SL}(d,q)$ satisfying property $(C1)$ and $(C2)$ are rare. Checking that $\tilde{g}$ satisfies $(C1)$ and $(C2)$ can be done with just three applications of the Gaussian algorithm. In our example $V_2 = \langle e_1, e_2\rangle$ and $V_2\tilde{g} = \langle \tilde{g}_{1,-}, \tilde{g}_{2,-}\rangle$ where $\tilde{g}_{i,-}$ denotes the i-th row of $\tilde{g}$. Therefore, $\dim(V_2 + V_2\tilde{g})$ can be computed by applying the Gaussian algorithm to the matrix containing $e_1, e_2, \tilde{g}_{1,-}$ and $\tilde{g}_{2,-}$ as rows. To verify condition $(C2)$, note that $F_{14} = \langle e_3, \ldots, e_{16}\rangle$ and that $\mathrm{Fix}(\tilde{g})$ is just the null space of $\tilde{g} - I_{16}$. Then we can use the same computation as we did for $(C1)$.

SL3) We want a conjugate g of $\tilde{g}$ such that g satisfies $(C1)$ and $(C2)$ and fixes $v_2 = e_2$. First, we

compute $V_2 \cap V_2\tilde{g} = \langle e_1 + e_2 \rangle$ and note that e_2 is not contained in the fixed space of $\tilde{g}$. We write

$$L := \left(\begin{array}{cc|c} 1 & 0 & 0 \\ 1 & 1 & 0 \\ \hline 0 & 0 & I_{14} \end{array}\right) \in H^{\Xi}_{(0)}$$

as a word in the standard generators of $H^{\Xi}_{(0)}$. Note that in this case L is one of the standard generators we already computed. Moreover, $e_2 L = e_1 + e_2$. By setting $g := L\tilde{g}L^{-1}$ we have $V_2 \cap V_2 g = \langle e_2 \rangle$. We proved in Lemma 5.47 that our new element g still satisfies $(C1)$ and $(C2)$ and that $V_2 \cap V_2 g = \langle e_2 \rangle$.

SL4) We compute a new base change matrix $\mathscr{L}_8$. This is straightforward and explained in Section 5.3.3 and, thus, we only state the base change matrix $\mathscr{L}_8$

$$\mathscr{L}_8 := \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 3 & 1 & 1 & 2 & 2 & 0 & 3 & 2 & 3 & 1 & 3 & 3 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 4 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 4 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2
\end{pmatrix}.$$

From now on every matrix is given as an element of $(G^{\Xi})^{\mathscr{L}_4\mathscr{L}_8^{-1}}$. We have

$$g^{\mathscr{L}_8^{-1}} =: c = \left(\begin{array}{ccc|c} 4 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 2 & 3 & 0 \\ \hline 0 & 0 & 0 & I_{13} \end{array}\right).$$

SL5) We need to construct $E_{3,2}(1)$ and write it as a word in X and the standard generators of $H^{\Xi}_{(0)}$.

For this we conjugate transvections of $(H_{(0)}^{\Xi})^{\mathscr{L}_8^{-1}} = H_{(0)}^{\Xi} \cong \mathrm{SL}(2,5)$ with c. We have

$$E_{1,2}(1)^c = \left(\begin{array}{ccc|c} 1 & 3 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 4 & 1 & 0 \\ \hline 0 & 0 & 0 & I_{13} \end{array}\right).$$

Since we have standard generators for $H_{(0)}^{\Xi}$ we can write $E_{1,2}(1)^2 \in H_{(0)}^{\Xi}$ as a word in the standard generators and obtain

$$E_{1,2}(1)^c E_{1,2}(1)^2 = \left(\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 4 & 1 & 0 \\ \hline 0 & 0 & 0 & I_{13} \end{array}\right).$$

Moreover, we can normalise the entry at position $(3,1)$ such that

$$(E_{1,2}(1)^c E_{1,2}(2))^4 = (E_{1,2}(1)^c E_{1,2}(2))^{-1} = \left(\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ \hline 0 & 0 & 0 & I_{13} \end{array}\right) = E_{3,2}(1)$$

which completes SL5). In general, we must perform further computations in SL5) which we see in Example 5.65.

SL6) To obtain $E_{2,1}(1)$ we start by conjugating $E_{2,1}(1)$ by c such that

$$E_{2,1}(1)^c = \left(\begin{array}{ccc|c} 2 & 4 & 4 & 0 \\ 4 & 2 & 1 & 0 \\ 2 & 3 & 4 & 0 \\ \hline 0 & 0 & 0 & I_{13} \end{array}\right).$$

As for $E_{1,2}(1)^c$, we aim to replace this element by a transvection in $\langle H_{(0)}^{\Xi}, (H_{(0)}^{\Xi})^c \rangle \leq (G_3^{\Xi})^{\mathscr{L}_4 \mathscr{L}_8^{-1}}$ by multiplying group elements which can be written as words in X. For this we note that we can

multiply the given element by transvections. Since all transvections $E_{i,j}(\lambda)$ for $1 \leq i, j \leq 2$ with $i \neq j$ and $\lambda \in \mathbb{F}_q^*$ lie in $H_{(0)}^{\Xi}$, these elements can be written as words in X. Moreover, we wrote $E_{3,2}(1)$ as a word in X in SL5). We can view multiplying $E_{2,1}(1)^c$ by transvections as effecting elementary row and column operations. We start by adding the second row to the first such that

$$E_{1,2}(1)E_{2,1}(1)^c = \left(\begin{array}{ccc|c} 1 & 1 & 0 & 0 \\ 4 & 2 & 1 & 0 \\ 2 & 3 & 4 & 0 \\ \hline 0 & 0 & 0 & I_{13} \end{array}\right).$$

Since we already have computed the transvection $E_{3,2}(1)$ we can use this element to add the second row to the third row such that

$$E_{3,2}(1)^2E_{1,2}(1)E_{2,1}(1)^c = \left(\begin{array}{ccc|c} 1 & 1 & 0 & 0 \\ 4 & 2 & 1 & 0 \\ 0 & 2 & 1 & 0 \\ \hline 0 & 0 & 0 & I_{13} \end{array}\right).$$

Now we can clear the top left 2×2 block using transvections of $H_{(0)}^{\Xi} \cong SL(2,5)$ to obtain

$$E_{2,1}(1)E_{3,2}(1)^2E_{1,2}(1)E_{2,1}(1)^cE_{1,2}(1)^{-1} = \left(\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 3 & 1 & 0 \\ 0 & 2 & 1 & 0 \\ \hline 0 & 0 & 0 & I_{13} \end{array}\right).$$

Now the only thing left is to use the transvection $E_{3,2}(1)$ to clear the entry at position $(3,2)$

$$E_{2,1}(1)E_{3,2}(1)^2E_{1,2}(1)E_{2,1}(1)^cE_{1,2}(1)^{-1}E_{3,2}(1)^3 = \left(\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & I_{13} \end{array}\right) = E_{2,3}(1).$$

Similarly to SL1) we must do more computations in SL6) in general.

SL7) We finish this example by computing the permutation matrices of SL(3,5). We have

$$E_{3,2}(1)E_{2,3}(1)^{-1}E_{3,2}(1) = \left(\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & I_{13} \end{array}\right)$$

which corresponds to the permutation matrix z_2 of SL(3,5) and

$$\left(\begin{array}{ccc|c} 0 & 4 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & I_{13} \end{array}\right)\left(\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & I_{13} \end{array}\right) = \left(\begin{array}{ccc|c} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & I_{13} \end{array}\right)$$

which corresponds to the permutation matrix z_1 of SL(3,5). Now we have standard generators for a SL(3,5) stingray embedded subgroup of $(G^{\Xi})^{\mathscr{L}_4\mathscr{L}_8^{-1}}$ which we call $H^{\Xi}_{(1)} \cong \mathrm{SL}(3,5)$ in the following.◀

As a few important details are not visible in the first application of the GoingUp step, we also review a quick version of the second application. Here, we do not go into as much details as in Example 5.64 but instead focus on the differences.

Example 5.65

Up to this point we have computed words for standard generators of $H^{\Xi}_{(1)} \cong \mathrm{SL}(3,5)$ as a stingray embedded subgroup of $(G^{\Xi})^{\mathscr{L}_4\mathscr{L}_8^{-1}}$. By applying the GoingUp step a second time, we compute standard generators for $\mathrm{SL}(n',5)$ where $n' = 2n-1 = 5$. First, we need an element t which has a fixed space of dimension $d-n+1 = 16-3+1 = 14$. Instead of a transvection as in Example 5.64 we chose $t := z_2$. Trying random elements until we find a $\tilde{g}$ which satisfies $(C1)$ and $(C2)$,

replacing $\tilde{g}$ by a conjugate g such that it fixes $v_3 = e_3$ and performing a base change using

$$\mathscr{L}_9 := \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 3 & 0 & 1 & 0 & 1 & 0 & 1 & 4 & 1 & 4 & 2 & 3 & 1 \\
0 & 0 & 0 & 0 & 2 & 0 & 2 & 0 & 4 & 0 & 3 & 1 & 4 & 3 & 3 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 3 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 4 & 2 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 2 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 4 & 4 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 4 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0
\end{pmatrix}$$

yields the element

$$c = \begin{pmatrix}
3 & 1 & 3 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
3 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 4 & 1 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
3 & 1 & 0 & 1 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}.$$

From now on every matrix is given as an element of $(G^{\Xi})^{\mathscr{L}_4\mathscr{L}_8^{-1}\mathscr{L}_9^{-1}}$. In SL5) we again conjugate transvections of SL(3,5) with c such that

$$E_{1,3}(1)^c = \left(\begin{array}{ccccc|c}
1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 2 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 4 & 1 & 0 & 0 \\
0 & 0 & 3 & 0 & 1 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & I_{11}
\end{array}\right) \quad \text{and} \quad E_{2,3}(1)^c = \left(\begin{array}{ccccc|c}
1 & 0 & 2 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 4 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & I_{11}
\end{array}\right).$$

Using suitable $h_1, h_2 \in H^{\Xi}_{(1)}$ we compute

$$g_1 := E_{1,3}(1)^c h_1 = \left(\begin{array}{ccccc|c} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 4 & 1 & 0 & 0 \\ 0 & 0 & 3 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & I_{11} \end{array}\right) \quad \text{and} \quad g_2 := E_{2,3}(1)^c h_2 = \left(\begin{array}{ccccc|c} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 4 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & I_{11} \end{array}\right).$$

Note that we do not compute the transvections $E_{4,3}(1)$ and $E_{5,3}(1)$ directly as it was the case in Example 5.64. We use an additional trick here by extracting parts of a column in each matrix, namely those marked in orange to obtain $w_1 = (4,3) \in \mathbb{F}_5^2$ and $w_2 = (4,0) \in \mathbb{F}_5^2$. Since $\langle w_1, w_2 \rangle = \mathbb{F}_5^2$ we can express the standard basis of $\mathbb{F}_5^2$ as a linear combination in the basis (w_1, w_2) which is $e_1 = 4w_2$ and $e_2 = 2w_1 + 2w_2$. Hence,

$$E_{4,3}(1) = g_2^4 = \left(\begin{array}{ccccc|c} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & I_{11} \end{array}\right) \quad \text{and} \quad E_{5,3}(1) = g_1^2 g_2^2 = \left(\begin{array}{ccccc|c} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & I_{11} \end{array}\right).$$

SL6) and SL7) are similar as in Example 5.64 except that we must perform more computations. ◀

5.3.6 Combining GoingUp steps

We have demonstrated how we can compute standard generators for a stingray embedded special linear group of dimension nearly twice that of a given stingray embedded special linear group with standard generators. In this section Algorithm GoingUpStep [Alg. 27] is used to develop an algorithm which can be used to compute standard generators for $G \leq \mathrm{SL}(d,q)$ with $G \cong \mathrm{SL}(d,q)$, where $H \leq G$ with $H \cong \mathrm{SL}(2,q)$ are given and standard generators of H are known.

Algorithm 28: GoingUp

Input:
- $\langle X \rangle = G = \mathrm{SL}(d,q)$
- A base change matrix $\mathscr{L} \in \mathrm{GL}(d,q)$
- $\mathrm{SL}(2,q) \cong \langle Y_2 \rangle = H \leq G^{\mathscr{L}}$ stingray embedded and constructively recognised
- $N \in \mathbb{N}$

Output: **fail OR** $(\mathscr{L}', \mathfrak{S}, N')$ **where**
- $\mathscr{L}' \in \mathrm{GL}(d,q)$ is a base change matrix,
- $\mathfrak{S}$ is an MSLP from $X \cup Y_2$ to the standard generators of $G^{\mathscr{L}\mathscr{L}'}$ and
- $N' \in \mathbb{N}$ where $N - N'$ is the number of random selections that were used

function GoingUp$(G, \mathscr{L}, H, N)$

1 $n \leftarrow 2$ **AND** $\mathfrak{S} \leftarrow$ an MSLP from $X \cup Y_2$ to $X \cup Y_2$
2 **while** $n < d$ **do**
3 $n \leftarrow \min\{2 \cdot n - 1, d\}$ // Nearly double the dimension.
4 $(H, \mathscr{L}, \mathfrak{S}', N) \leftarrow$ GoingUpStep$(G, \mathscr{L}, H, N)$ // Remark 5.7 and 5.8
5 $\mathfrak{S} \leftarrow$ Compose $\mathfrak{S}$ and $\mathfrak{S}'$
6 **return** $(\mathscr{L}, \mathfrak{S}, N)$

Theorem 5.66

Algorithm GoingUp [Alg. 28] terminates using at most N random selections and is correct.

Proof. Follows immediately from Theorem 5.63. □

5.3.7 GoingUp with lower-dimensional matrices

Similar to the results in Section 5.1.4 we aim to improve the performance of the GoingUp algorithm using linear algebra. The key idea is to reuse the descending recognition chain of special linear groups

$$\mathrm{SL}(2,q) = H = U_k \leq U_{k-1} \cong \mathrm{SL}(4,q) \leq U_{k-2} \cong \mathrm{SL}(d_{k-2},q) \leq \ldots \leq U_1 \cong \mathrm{SL}(d_1,q) \leq U_0 = G$$

where $d_i \leq 4 \cdot \log(d_{i-1})$ of the GoingDown algorithm. Instead of representing the elements of H as elements of G, i.e. $d \times d$ matrices we can represent them as 4×4 matrices and as a subgroup of U_{k-1}.

In this setting we can also use Algorithm GoingUpStep [Alg. 27] to construct standard generators of a subgroup of U_{k-1} isomorphic to $\mathrm{SL}(3,q)$. In the next step we can represent the generators of the subgroup of U_{k-1} isomorphic to $\mathrm{SL}(3,q)$ as elements of U_{k-2} and again call Algorithm GoingUpStep [Alg. 27] on $\mathrm{SL}(3,q)$ and U_{k-2}.

Using this idea we can work with lower-dimensional matrices.

Chapter 6

Symplectic group

This chapter details an efficient realisation of the strategy outlined in Chapter 3 for the symplectic group $\mathrm{Sp}(d,q)$ in its natural representation for $d \geq 8$ and q odd. If $d = 2$, then $\mathrm{Sp}(2,q) = \mathrm{SL}(2,q)$ and we refer to Section 5.2 which provides a detailed description of [27]. If $d = 4$, then $\mathrm{Sp}(4,q)$ is a base case group and we refer to [20], and if $d = 6$, then we also refer to [20].

This chapter is structured identically as the chapter for the special linear groups, see Chapter 5. Since the main ideas for constructive recognition of classical groups are as outlined in Chapter 3 and we discussed a detailed description of the algorithms in Chapter 5 for special linear groups we focus mostly on the differences between the algorithms for special linear groups and symplectic groups and do not provide a complete explanation of the GoingUp step for symplectic groups.

Remark 6.1

In symplectic groups we have to deal with two different cases depending on the characteristic of the underlying field $\mathbb{F}_q$. If q is odd, then we can proceed in a similar way as we do for special linear groups. The differences and additional steps between the GoingDown and GoingUp algorithm for symplectic groups in odd characteristic and the GoingDown and GoingUp algorithm for special linear groups are described and explained in Section 6.1 and Section 6.3. If q is even, then we encounter the following situation. The goal of the GoingDown algorithm is to compute a full

descending recognition chain

$$\mathrm{CL}(d_{k+1},q) \cong U_{k+1} \leq \mathrm{CL}(d_k,q) \cong U_k \leq \mathrm{CL}(d_{k-1},q) \cong U_{k-1} \leq \ldots \leq \mathrm{CL}(d_1,q) \cong U_1 \leq U_0 = G$$

where $\mathrm{CL}(d_{k+1},q)$ is a base case group and $\mathrm{CL}(d_k,q)$ is a terminal group. It is important that each group of this chain has the same type as the input group G. For example if we are dealing with symplectic groups in odd characteristic, then repeated calls to the GoingDown basic step for symplectic groups yield a descending recognition chain

$$\mathrm{Sp}(d_k,q) \cong U_k \leq \mathrm{Sp}(d_{k-1},q) \cong U_{k-1} \leq \ldots \leq \mathrm{Sp}(d_1,q) \cong U_1 \leq U_0 = G = \mathrm{Sp}(d,q).$$

Note that self-reciprocal ppd-stingray elements, which we compute in the GoingDown basic step of symplectic and orthogonal groups, always have even degree by Definition 4.19. Thus, d_i is even for all $1 \leq i \leq k$ which we also conclude in Section 6.1.

For q even there exists an isomorphism between $\mathrm{Sp}(d,q)$ and $\mathrm{O}^\circ(d+1,q)$ [91, Theorem 11.9]. Hence, we are indirectly dealing with two different types of classical groups as input group. This has the effect that for q even the groups of the descending recognition chain computed by the GoingDown basic step for symplectic groups have a different type than the input group $G = \mathrm{Sp}(d,q)$ leading to the following chain

$$\Omega^{\pm}(d_k,q) \cong U_k \leq \Omega^{\pm}(d_{k-1},q) \cong U_{k-1} \leq \ldots \leq \Omega^{\pm}(d_1,q) \cong U_1 \leq U_0 = G = \mathrm{Sp}(d,q).$$

That U_i is isomorphic to $\Omega^{\pm}(d_i,q)$ follows from the fact that d_i is even and that U_i is contained in an orthogonal group which is unavoidable using stingray elements and proven in [40] or can be generalised from [81, Lemma 3.3]. That U_i is isomorphic to $\Omega^{\pm}(d_i,q)$ is a problem for the overall constructive recognition algorithm for symplectic groups in even characteristic since the GoingUp algorithm assumes that the base case group has the same type as the input group. If this is not the case, as for symplectic groups in even characteristic, then the GoingUp step conjugates a group $\Omega(n,q) \cong H \leq \mathrm{Sp}(d,q)$ with an element $g \in \mathrm{Sp}(d,q)$. But, unfortunately, the group $\langle H, H^g \rangle$ is not isomorphic to a classical group and the GoingUp step fails. Therefore, $\mathrm{Sp}(d,q)$ for q even is

not considered in this thesis. Recall that $\mathrm{Sp}(d,q)$ and $\mathrm{O}^\circ(d+1,q)$ are isomorphic for q even [91, Theorem 11.9]. Since $\mathrm{O}^\circ(d+1,q)$ is reducible $\mathrm{O}^\circ(d+1,q)$ is identified as $\mathrm{Sp}(d,q)$ and a constructive recognition algorithms for $\mathrm{Sp}(d,q)$ is used for $\mathrm{O}^\circ(d+1,q)$. As we exclude $\mathrm{Sp}(d,q)$ for q even in this thesis we also exclude $\mathrm{O}^\circ(d+1,q)$ for q even. ◀

Nevertheless, the GoingUp algorithm for symplectic groups which is presented and proven in this chapter is applicable for even and odd characteristic but as we cannot compute a stingray embedded symplectic base case group in even characteristic the requirements for the GoingUp step are not satisfied.

From this point on we are only dealing with symplectic groups in odd characteristic if not mentioned otherwise. This chapter is structured as Chapter 5 for special linear groups. After the introduction, we define a set of standard generators for symplectic groups, prove that the elements of this set indeed generate symplectic groups and provide rules to write specific transvections as words in these generators. In Section 6.1 we describe a GoingDown algorithm for symplectic groups and prove its correctness. The GoingDown basic step differs only in one aspect compared to the GoingDown basic step for special linear groups which is that we compute two self-reciprocal stingray elements instead of two stingray elements, see Definition 4.19 for the definition of self-reciprocal stingray elements. In Section 6.2 we briefly discuss the constructive recognition algorithms for the symplectic base case groups. We do not dive as deep into the details of the presented algorithms as for the special linear group and instead only provide a reference and a reasoning why the methods are applicable. In Section 6.3 we present an algorithm for the GoingUp step which requires only a few modifications compared to the GoingUp step for special linear groups, see Section 5.3.

The notations for the remainder of this Chapter are as follows. The input group is denoted by G and G is a symplectic group in its natural representation. We use U and H as subgroups of G where U is used in the GoingDown algorithm and H is used in the GoingUp algorithm. Depending on the characteristic p of the underlying field, U and H are either symplectic groups, if p is odd, or orthogonal groups if p is even.

Before we introduce standard generators of symplectic groups a notation for specific elements of

symplectic groups is given in the next definition. As in Chapter 2 $q = p^f$ denotes a prime power, $(\omega_1,\dots,\omega_f)$ an $\mathbb{F}_p$-basis of $\mathbb{F}_q$ and d,n are natural numbers with $n \le d$. Let $V = \mathbb{F}_q^d$ with basis $\{b_1,\dots,b_d\}$. The matrices $I_{i,j}(\iota)$ satisfy $(I_{i,j})_{ij} = 1$ and all other entries of $I_{i,j}$ are equal to 0 as defined in Section 2.1.

Definition 6.2

Let d be even. For $i,j \in \{1,\dots,d\}$, with $j \neq i$ and $\lambda \in \mathbb{F}_q - \{0\}$ we set

$$E_{i,j}^{\mathrm{Sp}}(\lambda) := \begin{cases} I_d + I_{i,j}(\lambda), & \text{if } i+j = d+1, \\ I_d + I_{i,j}(\lambda) - I_{d-j+1,d-i+1}(\lambda), & \text{if } i \in \{2,\dots,\frac{d}{2}\}, j < i \text{ or } j \in \{\frac{d}{2}+1,\dots,d-1\}, j < i \\ & \text{or } j \in \{2,\dots,\frac{d}{2}\}, i < j \text{ or } i \in \{\frac{d}{2}+1,\dots,d-1\}, i < j, \\ I_d + I_{i,j}(\lambda) + I_{d-j+1,d-i+1}(\lambda), & \text{otherwise.} \end{cases}$$

Definition 6.3

Let d be even and q an odd prime power. Let $S^{\mathrm{Sp}} \subset \mathrm{SL}(d,q)$. Then S^{Sp} is a set of *standard generators* for $\mathrm{Sp}(d,q)$ if S^{Sp} is conjugate to the following set consisting of $3f+3$ elements:

- $E_{1,2}^{\mathrm{Sp}}(\omega_i)$ for $1 \le i \le f$,
- $E_{2,1}^{\mathrm{Sp}}(\omega_i)$ for $1 \le i \le f$,
- $E_{1,n}^{\mathrm{Sp}}(\omega_i)$ for $1 \le i \le f$,
- a permutation matrix z_1^{Sp} corresponding to the permutation $(1,2,\dots,\frac{d}{2})(\frac{d}{2}+1,d,d-1,\dots,\frac{d}{2}+2)$,
- a permutation matrix z_2^{Sp} corresponding to the permutation $(1,2)(d-1,d)$ and
- a permutation matrix z_3^{Sp} corresponding to the permutation $(1,d)$ with the entry $(z_3^{\mathrm{Sp}})_{d,1}$ changed to -1.

Lemma 6.4

Let d be even and q an odd prime power. Every element $E_{i,j}^{\mathrm{Sp}}(\lambda)$ can be written in terms of the standard generators of Definition 6.3 as follows:

$$E_{1,2}^{\mathrm{Sp}}(\iota) = \prod_{j=1}^{f} E_{1,2}^{\mathrm{Sp}}(\omega_j)^{\lambda_j}, \quad E_{2,1}^{\mathrm{Sp}}(\iota) = \prod_{j=1}^{f} E_{2,1}^{\mathrm{Sp}}(\omega_j)^{\lambda_j} \quad \text{and} \quad E_{1,n}^{\mathrm{Sp}}(\iota) = \prod_{j=1}^{f} E_{1,n}^{\mathrm{Sp}}(\omega_j)^{\lambda_j}$$

where $\imath = \sum_{j=1}^{f} \lambda_j \omega_j$ for $\lambda_j \in \mathbb{F}_p$. Moreover,

$$\begin{aligned}
E^{\mathrm{Sp}}_{1,i}(\imath)^{z_1^{\mathrm{Sp}} z_2^{\mathrm{Sp}}} &= E^{\mathrm{Sp}}_{1,i+1}(\imath) && \text{for } 2 \le i \le \frac{d}{2}-1,\\
E^{\mathrm{Sp}}_{i,j}(\imath)^{z_1^{\mathrm{Sp}}} &= E^{\mathrm{Sp}}_{i+1,j+1}(\imath) && \text{for } 2 \le j \le \frac{d}{2}-1 \text{ and } 1 \le i \le \frac{d}{2}-2,\\
E^{\mathrm{Sp}}_{1,\frac{d}{2}}(\imath)^{z_3^{\mathrm{Sp}}} &= E^{\mathrm{Sp}}_{\frac{d}{2}+1,1}(\imath) &&\\
E^{\mathrm{Sp}}_{i,1}(\imath)^{(z_1^{\mathrm{Sp}} z_2^{\mathrm{Sp}})^{-1}} &= E^{\mathrm{Sp}}_{i+1,1}(\imath) && \text{for } \frac{d}{2}+1 \le i \le d-2,\\
E^{\mathrm{Sp}}_{i,j}(\imath)^{z_1^{\mathrm{Sp}}} &= E^{\mathrm{Sp}}_{i-1,j+1}(\imath) && \text{for } 1 \le j \le \frac{d}{2}-2 \text{ and } 1 \le i \le \frac{d}{2}-1,\\
E^{\mathrm{Sp}}_{n-i+1,i}(\imath)^{z_1^{\mathrm{Sp}}} &= E^{\mathrm{Sp}}_{n-i+2,i-1}(\imath) && \text{for } \frac{d}{2}+2 \le i \le n
\end{aligned}$$

and analogously starting with $E^{\mathrm{Sp}}_{2,1}(\imath)$.

Proof. The proof is similar to the proof of Lemma 5.3. □

Lemma 6.5

Let d be even and q an odd prime power. Then $\mathrm{Sp}(d,q)$ is generated by the standard generators of Definition 6.3.

Proof. This follows immediately as the standard generators of Definition 6.3 contain the DLLO standard generators [59] for symplectic groups. □

For the remainder of this chapter let d be even, q an odd prime power and $G := \mathrm{Sp}(d,q)$ in its natural representation.

6.1 GoingDown algorithm

In this chapter we discuss the GoingDown algorithm for symplectic groups. As for special linear groups and as outlined in Chapter 3 the GoingDown algorithm for symplectic groups uses a GoingDown basic step which is discussed in Section 6.1.1 and a final step which is discussed in Section 6.1.3. In Section 6.1.2 the GoingDown basic step is used repeatedly for the computation of

a descending recognition chain and afterwards the final step is used for the computation of the full descending recognition chain.

Overall the GoingDown basic step for symplectic groups is similar to the GoingDown basic step for special linear groups in Section 5.1. The goal of the GoingDown basic step for symplectic groups is the same, which is to compute a descending recognition chain as in Definition 3.4, i.e.

$$\mathrm{Sp}(8,q) \cong U_k \leq \mathrm{Sp}(d_{k-1},q) \cong U_{k-1} \leq \ldots \leq \mathrm{Sp}(d_1,q) \cong U_1 \leq U_0 = G$$

where $d_i \leq 4\lceil \log(d_{i-1}) \rceil$. Recall from Definition 3.6 that a terminal group is a subgroup of G of the same type and of smallest degree such that it can be reached in the descending recognition chain using stingray elements. For special linear groups the terminal group is given by $\mathrm{SL}(4,q)$ while $\mathrm{Sp}(8,q)$ is the terminal group in symplectic groups as described in Table 3.2 in Chapter 3.

There is only one slight difference between the GoingDown basic step for special linear groups and symplectic groups which is the stingray element itself. Recall from Section 4.1 the definition of self-reciprocal stingray elements given in Definition 4.19. While we use stingray elements for special linear groups, we choose self-reciprocal stingray elements for symplectic and orthogonal groups as described in [75]. Note that we discussed Algorithm FindSelfReciprocalStingrayElement [Alg. 7] for randomised computations of self-reciprocal stingray elements which is a slight variation of Algorithm FindStingrayElement [Alg. 5].

In the final step we briefly discuss two algorithms from the literature to compute a stingray embedded $\mathrm{Sp}(4,q)$ in $\mathrm{Sp}(8,q)$. One of these two algorithms is the constructive recognition from Leedham-Green and O'Brien [59] and the other algorithm is for black box constructive recognition of symplectic groups by Brooksbank [20].

6.1.1 GoingDown basic step

In this section we present an algorithm for the GoingDown basic step in symplectic and orthogonal groups. As outlined in Section 6.1 the main idea of the GoingDown basic step for symplectic groups is similar to Algorithm GoingDownBasicStepSL [Alg. 13] which is discussed in Section 5.1.1. Instead of general stingray elements we compute self-reciprocal stingray elements by random procedures in the GoingDown basic step for symplectic groups. A definition of self-reciprocal stingray

elements and how these elements can be computed, has been discussed in Chapter 4. An algorithm for a randomised computation of self-reciprocal stingray elements is given by Algorithm FINDSELFRECIPROCALSTINGRAYELEMENT [Alg. 7].

An algorithm for the GOINGDOWN basic step for symplectic groups is given by Algorithm GOINGDOWNBASICSTEPRECIPROCAL [Alg. 29].

Remark 6.6

Recall from Remark 4.28 that we use self-reciprocal stingray elements for the GOINGDOWN basic step in symplectic and orthogonal groups. Therefore, Algorithm GOINGDOWNBASICSTEPRECIPROCAL [Alg. 29] presented below is designed to be a GOINGDOWN basic step algorithm for both symplectic and orthogonal groups. Note that we call a naming algorithm in Line 9 of Algorithm GOINGDOWNBASICSTEPRECIPROCAL [Alg. 29]. If the input group G is isomorphic to a symplectic group, then we check $\langle s_1, s_2\rangle \cong \mathrm{Sp}(d_2, q)$ and if the input group G is isomorphic to an orthogonal group, then we check $\langle s_1, s_2\rangle \cong \Omega(d_2, q)$. Note that the function body of Algorithm GOINGDOWNBASICSTEPRECIPROCAL [Alg. 29] is otherwise identical to the function body of Algorithm GOINGDOWNBASICSTEPSL [Alg. 13] except that we use Algorithm FINDSELFRECIPROCALSTINGRAYELEMENT [Alg. 7] instead of Algorithm FINDSTINGRAYELEMENT [Alg. 5]. ◀

Algorithm 29: GOINGDOWNBASICSTEPRECIPROCAL

Input:
- ▶ $d_1 \in \mathbb{N}$ with $d_1 > 8$
- ▶ $\langle X\rangle = G \leq \mathrm{GL}(d, q)$ with $G \cong \mathrm{Sp}(d_1, q)$ and q odd or $G \cong \Omega(d_1, q)$
- ▶ Φ a form preserved by G
- ▶ $N \in \mathbb{N}$

Output: **fail OR** $(d_2, U, \mathfrak{S}, N')$ **where**
- ▶ $d_2 \in \mathbb{N}$ with $8 \leq d_2 \leq 4\lceil \log(d_1)\rceil$,
- ▶ $U \leq G$ with $U \cong \mathrm{Sp}(d_2, q)$ if $G \cong \mathrm{Sp}(d_1, q)$ and $U \cong \Omega(d_2, q)$ otherwise,
- ▶ $\mathfrak{S}$ is an MSLP from X to generators of U and
- ▶ $N' \in \mathbb{N}$ where $N - N'$ is the number of random selections that were used

function GoingDownBasicStepReciprocal(d_1, G, Φ, N)

```
while N > 0 do                                                          // Remark 5.7
    (s_1, 𝔖_1, N) ← FindSelfReciprocalStingrayElement(G, d_1, N)        // Remark 5.8
    W_{s_1} ← ComputeStingrayBody(s_1)
    repeat
        (s_2, 𝔖_2, N) ← FindSelfReciprocalStingrayElement(G, d_1, N)    // Remark 5.8
        W_{s_2} ← ComputeStingrayBody(s_2)
    until IsStingrayDuo((s_1, s_2), Φ)
    d_2 ← dim(W_{s_1}) + dim(W_{s_2})
    if ⟨s_1, s_2⟩ ≅ Sp(d_2, q) (resp. ⟨s_1, s_2⟩ ≅ Ω(d_2, q)) then   // Using a naming algorithm, see Section 1.1.7
        𝔖 ← an MSLP from X to (s_1, s_2) using 𝔖_1 and 𝔖_2
        return (d_2, ⟨s_1, s_2⟩, 𝔖, N)
return fail
```

Theorem 6.7

Algorithm GoingDownBasicStepReciprocal [Alg. 29] terminates using at most N random selections and works correctly.

Proof. The proof is similar to the proof of Algorithm GoingDownBasicStepSL [Alg. 13]. It is clear that the algorithm terminates. If the algorithm does not return fail, then the output must be isomorphic to $\mathrm{Sp}(d_2, q)$ and $8 \leq d_2 \leq 4\lceil \log(d_1) \rceil$. □

Remark 6.8

Since symplectic groups are defined by forms defined on the underlying vector space, we must verify an additional property of the stingray pairs to ensure that the (classical) group generated by stingray pairs is non-degenerate as in Definition 4.22. By our assumption G is a symplectic group in its natural representation and, thus, we can compute the underlying symplectic form and represent the form by its Gram-matrix. Recall from Lemma 4.21 that if $s \in \mathrm{Sp}(d, q)$ is a ppd-stingray element, then W_s is non-degenerate, where W_s is the stingray body of s. Therefore, given a stingray pair (s_1, s_2) of ppd-stingray elements Algorithm IsStingrayDuo [Alg. 10] verifies that $W_{s_1} \cap W_{s_2} = \{0\}$

and that $W_{s_1} \oplus W_{s_2}$ is non-degenerate to ensure that (s_1, s_2) is a stingray duo. Note that we use the underlying form Φ of the input group G to verify that $W_{s_1} \oplus W_{s_2}$ is non-degenerate by restricting Φ to $W_{s_1} \oplus W_{s_2}$ and verifying that the restriction has full rank. ◀

6.1.2 Combining GoingDown basic steps

Similarly to the GoingDown algorithm for special linear groups the GoingDown basic step for symplectic groups is repeatedly called until a terminal group as in Definition 3.6 is computed. An algorithm implementing this is given by Algorithm GoingDownToDim8Symplectic [Alg. 30].

Algorithm 30: GoingDownToDim8Symplectic

Input:
- ▶ $d \in \mathbb{N}$ with $d \geq 8$
- ▶ $\langle X \rangle = G = \mathrm{Sp}(d, q)$ with q odd
- ▶ Φ a form preserved by G
- ▶ $N \in \mathbb{N}$

Output: fail **OR** $(U, \mathfrak{S}, N')$ **where**
- ▶ $U \leq G$ with $U \cong \mathrm{Sp}(8, q)$,
- ▶ $\mathfrak{S}$ is an MSLP from X to generators of U and
- ▶ $N' \in \mathbb{N}$ where $N - N'$ is the number of random selections that were used

function GoingDownToDim8Symplectic(d, G, Φ, N)

```
1  U ← G AND dim ← d AND 𝔖 ← an MSLP from X to X
2  while dim > 8 do
3      (dim, U, 𝔖′, N) ← GoingDownBasicStepReciprocal(dim, U, Φ, N)   // Remark 5.7 and 5.8
4      𝔖 ← Composition of 𝔖 and 𝔖′
5  return (U, 𝔖, N)
```

Theorem 6.9

Algorithm GoingDownToDim8Symplectic [Alg. 30] terminates using at most N random selections and works correctly.

Proof. Analogously to the proof of Theorem 5.13. □

6.1.3 Final step of the GoingDown algorithm

In this chapter we assume that a stingray embedded subgroup $U \leq G = \mathrm{Sp}(d,q)$ with $U \cong \mathrm{Sp}(8,q)$ is computed and that standard generators of U can be written as words in the generators X of G. Note that U is a terminal group as $U \cong \mathrm{Sp}(8,q)$ and that U can be computed using Algorithm GoingDownToDim8Symplectic [Alg. 30]. The goal of this chapter is to identify a subgroup $U_0 \leq U$ with $U_0 \cong \mathrm{Sp}(4,q)$ and U_0 is stingray embedded in U.

As for special linear groups it is not possible to find a symplectic base case group, i.e. $\mathrm{Sp}(d,q)$ for $d \leq 4$, using the GoingDown basic step of symplectic groups. Therefore, we rely on alternative algorithms and use methods from the DLLO constructive recognition algorithm [59] as well as a method from Brooksbank [20]. This is done in two steps as follows:

1) Call Algorithm GoingDownFinalStepCL [Alg. 16] from the DLLO constructive recognition algorithm on U to compute $\tilde{U} \leq U$ with $\tilde{U} \cong \mathrm{Sp}(6,q)$ and $\tilde{U}$ is stingray embedded in U. The algorithm involves involutions as in Definition 5.17 and is similar to Algorithm GoingDownFinalStepSL [Alg. 15].

2) Apply a method from Brooksbank [20, Section 5.1] to $\tilde{U}$ to compute $U_0 \leq \tilde{U}$ with $U_0 \cong \mathrm{Sp}(4,q)$ and U_0 is stingray embedded in $\tilde{U}$.

As $\tilde{U}$ is stingray embedded in U and U_0 is stingray embedded in $\tilde{U}$ it is clear that U_0 must also be stingray embedded in U. Note that there are more possibilities to identify U_0, e.g. by applying the DLLO constructive recognition algorithm [59] on U and afterwards writing down generators for U_0. For this thesis no comparisons between different approaches have been performed as the running time is negligible in contrast to the running time of the entire GoingDown and GoingUp algorithm.

We are not dealing with the details of 1) and 2) in this thesis as both methods are well-known but instead give an overview of the used methods. Algorithm GoingDownFinalStepCL [Alg. 16] is used for 1) by calling `GoingDownFinalStepCL`$(G,4,N)$. For 2) we use a method from [20, Section 5.1] which is based on ppd-elements, see Definition 4.8, and given in Algorithm GoingDownSp6To4 [Alg. 31].

Algorithm 31: GoingDownSp6To4

Input: ▶ $\langle X \rangle = G = \mathrm{Sp}(6,q)$ with q odd

▶ $N \in \mathbb{N}$

Output: **fail OR** $(\mathscr{L}, U, \mathfrak{S}, N')$ **where**

▶ $U \leq G$ with $U \cong \mathrm{Sp}(4,q)$,

▶ $\mathscr{L} \in \mathrm{GL}(d,q)$ is a base change matrix such that $U^{\mathscr{L}}$ is stingray embedded in $G^{\mathscr{L}}$,

▶ $\mathfrak{S}$ is an MSLP from X to generators of U and

▶ $N' \in \mathbb{N}$ where $N - N'$ is the number of random selections that were used

function GoingDownSp6To4(G, N)

```
repeat
    g̃ ← PseudoRandom(G) AND N ← N − 1
until g̃ is a ppd-element of a specified order given in [20, Section 5.1] or N ≤ 0
if N ≤ 0 then
    return fail
g ← g̃^(q^((d−2)/2)+1)
repeat
    if q ≤ 5 then
        g1, g2, g3 ← PseudoRandom(G) AND N ← N − 1
        U ← ⟨g, g^g1, g^g2, g^g3⟩
    else
        g1 ← PseudoRandom(G) AND N ← N − 1
        U ← ⟨g, g^g1⟩
until Sp(4,q) ≅ U OR N ≤ 0
if N ≤ 0 then
    return fail
L ← base change matrix to embed U standard AND S ← MSLP to the generators of U
return (U, L, S, N)
```

We are not discussing details or correctness of these algorithms and refer to the publications [20] and [59]. Using these algorithms we know that there are efficient methods to construct a stingray embedded $\mathrm{Sp}(4,q)$ in $\mathrm{Sp}(8,q)$.

6.2 BaseCase algorithm

In this section we assume that $H \leq G := \mathrm{Sp}(d,q)$ has been computed successfully, where $H \cong \mathrm{Sp}(4,q)$ and H is stingray embedded in G. This can be done using the algorithms described in 6.1. Since $H \cong \mathrm{Sp}(4,q)$ we have found a symplectic base case group as in Definition 3.2 for which efficient constructive recognition algorithms are known, see e.g. [20]. For the implementation of the BaseCase algorithm for symplectic groups, we utilise the constructive recognition algorithm for $H \cong \mathrm{Sp}(4,q)$ given in [20]. We are not discussing the details of the algorithm in [20] but instead state the main theorem about its complexity.

Theorem 6.10: [20]

Let $H = \langle X \rangle = \mathrm{Sp}(4,q)$. Then there is a Las Vegas algorithm to recognise H constructively. The complexity of the algorithm is $\mathcal{O}(\mathscr{E} + \zeta \log(q))$ where ζ is the complexity to construct a (nearly) uniformly distributed random element of H as an MSLP in X and $\mathscr{E}$ is the complexity of constructively recognising (a homomorphic image of) $\mathrm{SL}(2,q)$.

Remark 6.11

Since the constructive recognition algorithm for $\mathrm{Sp}(4,q)$ given in Theorem 6.10 uses a constructive recognition algorithm for $\mathrm{SL}(2,q)$ which currently needs a discrete logarithm oracle as in Definition 1.7, the BaseCase algorithm for symplectic groups also requires a discrete logarithm oracle. ◀

6.3 GoingUp algorithm

This section describes the GoingUp algorithm for symplectic groups. Let $\langle X \rangle = G = \mathrm{Sp}(d,q)$ for q even or odd. Recall the definition of an ascending recognition chain of G outlined in Chapter 3

$$H_{(0)}^{\mathscr{L}_0^{-1}} \leq H_{(1)}^{\mathscr{L}_1^{-1}} \leq \ldots \leq H_{(\ell-1)}^{\mathscr{L}_{\ell-1}^{-1}} \leq H_{(\ell)}^{\mathscr{L}_\ell^{-1}} = G^{\mathscr{L}_\ell^{-1}}$$

where $H_{(i)} \cong \mathrm{Sp}(d_i,q)$ stingray embedded in G and $d_0 = 4 < d_1 < \ldots < d_\ell = d$. The group $H_{(0)}$ of the ascending recognition chain with $H_{(0)} \cong \mathrm{Sp}(4,q)$ can be computed for q odd using the GoingDown and BaseCase algorithm for symplectic groups of Section 6.1 and Section 6.2 such that $\mathrm{Sp}(4,q) \cong H \leq G$ stingray embedded is computed. Moreover, standard generators of $H_{(0)}$ can be

written as words in X. As for special linear groups, we apply a GoingUp step on $H_{(0)}$ to compute the next group of the ascending recognition chain $H_{(1)}$. Additionally, by the GoingUp step we can write standard generators of $H_{(1)}$ as words in X. By applying the GoingUp step repeatedly this yields an ascending recognition chain. Therefore, we assume in this section that $H \leq G$ is stingray embedded with $H \cong \mathrm{Sp}(n,q)$ is given and that standard generators of H can be written as words in X, i.e. $H = H_{(i)}$ for some i.

Section 6.3.1 introduces a GoingUp step for symplectic groups and Section 6.3.2 uses the GoingUp step for symplectic groups repeatedly to compute standard generators of G.

6.3.1 GoingUp step

In this section we describe the GoingUp step for symplectic groups. The solution and key ideas are similar as to the GoingUp step of special linear groups, see Section 5.3. Therefore, we mainly focus on the differences between the GoingUp steps for symplectic groups and special linear groups. The details of the GoingUp step for special linear groups are not discussed and repeated in this section and instead we refer to Section 5.3. The results of this section are valid for q even and odd. Our hypothesis for the remainder of this section is the following.

Hypothesis 6.12

Let q be even or odd and $d \in \mathbb{N}$ be even and $\langle X \rangle = G = \mathrm{Sp}(d,q)$ containing a stingray embedded subgroup $H \leq \langle X^{\mathscr{L}} \rangle = G^{\mathscr{L}}$ with $H \cong \mathrm{Sp}(n,q)$ for $n < d$ and for a known base change matrix $\mathscr{L} \in \mathrm{GL}(d,q)$. Note, that n must be even. Moreover, standard generators Y_n of H are given as words in X. Let $V = \mathbb{F}_q^d$ and suppose that $\mathscr{B} = (v_1, \ldots, v_d)$ is a basis for V and let $V_n = \langle v_1, \ldots, v_n \rangle$ and $F_{d-n} = \langle v_{n+1}, \ldots, v_d \rangle$ (cf. Definition 2.7). We assume that H acts naturally on V_n as $\mathrm{Sp}(n,q)$ and that H fixes F_{d-n} point-wise. Recall that $(\omega_1, \ldots, \omega_f)$ is an $\mathbb{F}_p$-basis for $\mathbb{F}_q$.

We start this section by stating the main theorem which is similar to Theorem 5.35 for special linear groups.

Theorem 6.13

Let q be even or odd. Let $X \subseteq \mathrm{SL}(d,q)$ such that $\langle X \rangle = G = \mathrm{Sp}(d,q)$ with $4 \leq n < d$ and n,d even and let $\mathscr{L} \in \mathrm{GL}(d,q)$ be a base change matrix. Let $Y_n^{\mathscr{L}}$ be a set of standard generators for the subgroup $\mathrm{Sp}(n,q)$ stingray embedded into $G^{\mathscr{L}}$. Furthermore, let $\mathfrak{S}$ be an SLP from X to Y_n and let $n' := \min\{2n-2, d\}$.

Then there is an algorithm that computes a base change matrix $\mathscr{L}' \in \mathrm{GL}(d,q)$ together with an SLP $\mathfrak{S}'$ from X to a set $Y_{n'}$, which is a set of standard generators for $\mathrm{Sp}(n',q)$ and $\langle Y_{n'}^{\mathscr{L}'} \rangle$ is stingray embedded in $G^{\mathscr{L}'}$.

Recall that the definition of a weak doubling element in special linear groups requires the element to satisfy two properties $(C1)$ and $(C2)$ of Remark 5.36 which are repeated in the next remark.

Remark 6.14

Let $g \in G^{\mathscr{L}}$. Then g is a weak doubling element with respect to H if

$(C1)$ $\dim(V_n + V_n g) = n'$ and

$(C2)$ if $n' < d$, then $\dim(F_{d-n} + \mathrm{Fix}(g)) = d$. ◀

Note that we defined $n' := \min\{2n-2, d\}$ for symplectic groups while $n' := \min\{2n-1, d\}$ for special linear groups as indicated in Table 3.5 of Chapter 3. One reason for this is that $2n-1$ is odd. The other reason for this is as follows: The main aim of one GoingUp step is to compute the permutation matrices for $\mathrm{SL}(n',q)$ or $\mathrm{Sp}(n',q)$. The standard generators of special linear groups, see Definition 5.1, contain permutation matrices corresponding to an n- and an $(n-1)$-cycle. Therefore, computing a permutation matrix of larger degree can be achieved by multiplying two permutation matrices of two copies of SL intersecting in a subspace of dimension 1. The standard generators of symplectic groups, see Definition 6.3, contain a permutation matrix z_1^{Sp} corresponding to a permutation of cycle type $(\frac{n}{2})^2$. Therefore, multiplying two permutation matrices which correspond to the standard generator z_1^{Sp} in two different copies of $\mathrm{Sp}(n,q)$ will only yield the required permutation matrix, if the subspaces on which the two copies of $\mathrm{Sp}(n,q)$ act intersect in a space of dimension at least 2.

As for special linear group we start by computing a weak doubling element but are actually interested in finding a strong doubling element. In special linear groups this is achieved by replacing a weak doubling element $\tilde{g} \in G^{\mathscr{L}}$ by a conjugate $g \in G^{\mathscr{L}}$ yielding a doubling element element and

after applying a base change matrix $\mathscr{L}' \in \mathrm{GL}(d,q)$ to $c := g^{\mathscr{L}'}$ we verify the last condition $(C3)$ given in Remark 5.53. Recall that $\langle v'_1,\ldots,v'_d\rangle = \mathscr{B}^{\mathscr{L}'}$. We repeat the last condition $(C3)$ in the next remark.

Remark 6.15

$(C3)$ The vectors v'_j and $v'_j c^{-\mathrm{Tr}}$ for $1 \le j \le n-1$ span as an $\mathbb{F}_q$-subspace the $\mathbb{F}_q$-subspace $\langle v'_1,\ldots,v'_{n-1},v'_{n+1},\ldots,v'_{n'}\rangle$. ◀

Similar to the special linear group we prove the correctness of Theorem 6.13 by stating an algorithm and proving the correctness of this algorithm. Moreover, the algorithm also consists of seven phases as for special linear groups, see Remark 5.62.

Remark 6.16

Let $\langle X\rangle = G = \mathrm{Sp}(d,q)$ contain a stingray embedded subgroup $H \le \langle X^{\mathscr{L}}\rangle = G^{\mathscr{L}}$ with $H \cong \mathrm{Sp}(n,q)$ for $n < d$, n even and for a known base change matrix $\mathscr{L} \in \mathrm{GL}(d,q)$. Moreover, standard generators Y_n of H are given as words in X. The following seven phases must be performed for Theorem 6.13 and, therefore, for one GoingUp step of symplectic groups:

Sp1) Construct an element $t \in H$ which has a fixed space of dimension $d-n+2$.

Sp2) Choose random elements $a \in G^{\mathscr{L}}$ until $\tilde{g} := t^a$ satisfies $(C1)$ and $(C2)$.

Sp3) Find a conjugate $g \in G^{\mathscr{L}}$ of $\tilde{g}$ which satisfies $(C1)$ and $(C2)$ and additionally fixes v_1 and v_n.

Sp4) Compute base change matrices $\mathscr{L}'$ and $\mathscr{L}''$ such that $\langle H, H^g\rangle^{\mathscr{L}'\mathscr{L}''}$ is stingray embedded in $G^{\mathscr{L}\mathscr{L}'}$ and preserves the standard form. Set $c := g^{\mathscr{L}'\mathscr{L}''}$ and verify whether c satisfies $(C3)$ of Remark 6.15. Proceed if c satisfies $(C3)$.

Sp5) Using c, construct transvections $E^{\mathrm{Sp}}_{j,n}(\omega_i)$ for $n < j \le n'$ for $\langle H, H^c\rangle \cong \mathrm{Sp}(n',q)$.

Sp6) Using c, construct transvections $E^{\mathrm{Sp}}_{n,j}(\omega_i)$ for $n < j \le n'$ for $\langle H, H^c\rangle \cong \mathrm{Sp}(n',q)$.

Sp7) Using the transvections of Sp5) and Sp6) construct standard generators for $\langle H, H^c\rangle \cong \mathrm{Sp}(n',q)$ by assembling a permutation matrix z_1^{Sp} corresponding to a permutation of cycle type $(\frac{n'}{2})^2$ as in Definition 6.3. ◀

In the following we discuss the phases in more detail which shows that many results of the GoingUp step for special linear groups also hold for symplectic group even though if $n' < d$, then $V_n \cap V_n c$ is 2-dimensional for symplectic groups and $V_n \cap V_n c$ is 1-dimensional for special linear groups.

In Sp1) we construct an element $t \in H$ which has a fixed space of dimension $d-n+2$. This can be achieved without further computations for symplectic groups as such an element is contained in the standard generators Y_n of H.

Lemma 6.17

Recall from our Hypothesis 6.12 that $H \leq \langle X^{\mathscr{L}} \rangle = G^{\mathscr{L}}$ with $H \cong \mathrm{Sp}(n,q)$ for $n < d$ and for a known base change matrix $\mathscr{L} \in \mathrm{GL}(d,q)$. An element of $t \in H \leq G^{\mathscr{L}}$ which has a fixed space of dimension $d-n+2$ is given by $t := z_1^{\mathrm{Sp}}$.

Proof. The fixed space of z_1^{Sp} is given by $\langle v_1+v_2+\ldots+v_{\frac{n}{2}}, v_{\frac{n}{2}+1}+\ldots+v_n, v_{n+1}, \ldots, v_d \rangle$. □

In Sp2) we choose random elements $a \in G^{\mathscr{L}}$ and check whether $\tilde{g} := t^a$ satisfies $(C1)$ and $(C2)$ of Remark 6.14. If $\tilde{g}$ satisfies $(C1)$ and $(C2)$, then $\tilde{g}$ has similar properties as in special linear groups summarised in Lemma 5.42. Here we state a version of this result for symplectic groups.

Lemma 6.18

Recall from our Hypothesis 6.12 that $H \leq \langle X^{\mathscr{L}} \rangle = G^{\mathscr{L}}$ with $H \cong \mathrm{Sp}(n,q)$ for $n < d$ and for a known base change matrix $\mathscr{L} \in \mathrm{GL}(d,q)$. Let $t \in H \leq G^{\mathscr{L}}$ be as in Lemma 6.17 and for random $a \in G$ let $\tilde{g} = t^a$ be a weak doubling element, i.e. an element which satisfies $(C1)$ and $(C2)$.

1) $\dim(V_n \cap \mathrm{Fix}(\tilde{g})) \geq 2$ and $\dim(V_n \cap \mathrm{Fix}(\tilde{g})) = 2$ if $n' < d$.
2) $V_{n'} := V_n + V_n\tilde{g}$ is invariant under the action of $\tilde{g}$.
3) If $n' < d$, then $\dim(F_{d-n} \cap \mathrm{Fix}(\tilde{g})) = d-n'$.

Proof. Analogously to the proof of Lemma 5.42. □

If $n' < d$, then V_n and $V_n\tilde{g}$ intersect in a 2-dimensional subspace instead of an 1-dimensional subspace as in the case of special linear groups. This is necessary for the success of the GoingUp step of symplectic groups as our goal is to construct a double cycle z_1^{Sp} of H^c which multiplied with the double cycle z_1^{Sp} of H results in the double cycle z_1^{Sp} of a stingray embedded subgroup of G isomorphic to $\mathrm{Sp}(n',q)$. Note that the product of the double cycle z_1^{Sp} of H^c and the double cycle z_1^{Sp} of H can only result in a standard generator of $\mathrm{Sp}(n',q)$ if V_n and $V_n\tilde{g}$ intersect in a subspace of dimension at least 2.

The aim of Sp3) is to find a conjugate $g \in G^{\mathscr{L}}$ of $\tilde{g}$ which satisfies $(C1)$ and $(C2)$ and additionally fixes v_1 and v_n. As $\dim(V_n \cap \mathrm{Fix}(\tilde{g})) \geq 2$ using 1) of Lemma 6.18 we can assume that we find two linearly independent vectors w_1, w_2 such that $\langle w_1, w_2 \rangle < V_n \cap \mathrm{Fix}(\tilde{g})$. We can find elements $L_1, L_2 \in H$ with $v_1 L_1 = w_1$, $v_n L_2 = w_2 L_1^{-1}$ and $v_1 L_2 = v_1$, express $L_1 L_2$ in terms of the standard generators Y_n and then compute the conjugate $g := L_2 L_1 \tilde{g} (L_2 L_1)^{-1}$. The following lemma shows that g is an element satisfying $(C1)$ and $(C2)$ and additionally fixes v_1 and v_n. Afterwards we describe how L_1 and L_2 can be found and written as words in Y_n.

Lemma 6.19

Recall from our Hypothesis 6.12 that $H \leq \langle X^{\mathscr{L}} \rangle = G^{\mathscr{L}}$ with $H \cong \mathrm{Sp}(n,q)$ for $n < d$ and for a known base change matrix $\mathscr{L} \in \mathrm{GL}(d,q)$. Moreover, $\mathscr{B} = (v_1, \ldots, v_d)$ is a basis for V. Let $t \in H$ be as in Lemma 6.17. For $a \in G^{\mathscr{L}}$ assume that $\tilde{g} = t^a$ is a weak doubling element, i.e. a satisfies $(C1)$ and $(C2)$. Let $L_1, L_2 \in H$ such that $v_1 L_1 = w_1$, $v_n L_2 = w_2 L_1^{-1}$ and $v_1 L_2 = v_1$ where $\langle w_1, w_2 \rangle < V_n \cap \mathrm{Fix}(\tilde{g})$ and $\dim(\langle w_1, w_2 \rangle) = 2$. Then

1) $L_2 L_1 \tilde{g} (L_2 L_1)^{-1}$ satisfies $(C1)$ and $(C2)$ and
2) $v_1, v_n \in V_n \cap \mathrm{Fix}(L_2 L_1 \tilde{g} (L_2 L_1)^{-1})$.

Proof. Note that if $L_i v = v$, then also $L_i^{-1} v = v$ for $i = 1, 2$.

1) We have $L_2 L_1 \in H$. Hence, the claim follows with the same proof as for Lemma 5.47 1).

2) As $v_1, v_n \in V_n$ and since $0 \neq w_1 \in V_n \cap \mathrm{Fix}(\tilde{g})$

$$v_1 L_2 L_1 \tilde{g} (L_2 L_1)^{-1} = v_1 L_1 \tilde{g} (L_2 L_1)^{-1} = w_1 \tilde{g} (L_2 L_1)^{-1} = w_1 L_1^{-1} L_2^{-1} = v_1 L_2^{-1} = v_1$$

shows that $v_1 \in V_n \cap \mathrm{Fix}(L_2 L_1 \tilde{g} (L_2 L_1)^{-1})$. Moreover, since $0 \neq w_2 \in V_n \cap \mathrm{Fix}(\tilde{g})$

$$v_n L_2 L_1 \tilde{g} (L_2 L_1)^{-1} = w_2 L_1^{-1} L_1 \tilde{g} (L_2 L_1)^{-1} = w_2 \tilde{g} (L_2 L_1)^{-1} = (w_2 L_1^{-1}) L_2^{-1} = v_n$$

shows that $v_n \in V_n \cap \mathrm{Fix}(L_2 L_1 \tilde{g} (L_2 L_1)^{-1})$. □

We proceed as follows to compute L_1 and L_2.

Remark 6.20

If $(w_1)_1 \neq 0$ and $(w_2 L_1^{-1})_n \neq 0$, then we assume w.l.o.g. that $w_1 = (1, w_{1,2}, w_{1,3}, \ldots, w_{1,n-1}, 0)$ and $w_2 L_1^{-1} = (0, w_{2,2}, w_{2,3}, \ldots, w_{2,n-1}, 1)$. If this is not the case, we simply go back to Sp2). We can now directly write down L_1 and L_2 for the vectors w_1, w_2 as in Lemma 6.19 as follows:

$$L_1 := \begin{pmatrix} 1 & w_{1,2} & w_{1,3} & \ldots & w_{1,n-1} & 0 \\ 0 & 1 & 0 & \ldots & 0 & w_{1,n-1} \\ 0 & 0 & 1 & \ldots & 0 & w_{1,n-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & 1 & -w_{1,2} \\ 0 & 0 & 0 & \ldots & 0 & 1 \end{pmatrix}, \quad L_2 := \begin{pmatrix} 1 & 0 & 0 & \ldots & 0 & 0 \\ -w_{2,n-1} & 1 & 0 & \ldots & 0 & 0 \\ -w_{2,n-2} & 0 & 1 & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ w_{2,2} & 0 & 0 & \ldots & 1 & 0 \\ 0 & w_{2,2} & w_{2,3} & \ldots & w_{2,n-1} & 1 \end{pmatrix}$$

◀

Remark 6.21

1) Note that $\mathrm{Sp}(n,q)$ acts on the 2-dimensional subspaces of $\mathbb{F}_q^n$ and has two orbits namely the hyperbolic pairs, i.e. $w_1, w_2 \in \mathbb{F}_q^n$ linear independent with $\langle w_1 \mid w_2 \rangle \neq 0$, and pairs which are orthogonal, i.e. $w_1, w_2 \in \mathbb{F}_q^n$ linear independent with $\langle w_1 \mid w_2 \rangle = 0$, see [91, Theorem 8.2]. The $\mathrm{Sp}(n,q)$ orbit of hyperbolic pairs is larger than the orbit of orthogonal pairs and, hence, with high probability a basis of a 2-dimensional subspace of $V_n \cap \mathrm{Fix}(c)$ is a hyperbolic pair. Since $\mathrm{Sp}(n,q)$ acts transitively on the hyperbolic pairs and since v_1 and v_n are a hyperbolic pair L_1 and L_2 of Lemma 6.19 are contained in $\mathrm{Sp}(n,q)$ with a high probability.
2) Clearly L_1 and L_2 satisfy the requirements of Lemma 6.19. ◀

The goal of Sp4) is to compute a new base change matrix $\mathscr{L}' \in \mathrm{GL}(d,q)$ from $\mathscr{B}$ to a basis $\mathscr{B}'$ such that $\langle H, H^g \rangle^{\mathscr{L}'}$ is stingray embedded in $G^{\mathscr{L}\mathscr{L}'}$.

Remark 6.22

This can be achieved similarly to the corresponding goal in Remark 5.51 except that we choose fewer vectors of $V_n g$ as $\dim(V_n \cap \mathrm{Fix}(g)) = 2$. Recall that $\pi \colon V \to F_{d-n}$ is the projection map to F_{d-n} of the decomposition $V = V_n \oplus F_{d-n}$. We define the basis $\mathscr{B}'$ as follows:

- The first n vectors of the new basis are equal to the vectors in the old basis, i.e. $v_i' := v_i$ for $1 \leq i \leq n$.

- The basis vectors $v'_{n+1},\dots,v'_{n'}$ are chosen as a linearly independent subset of the vectors $\pi(v_2 g),\dots,\pi(v_{n-1}g)$. If $n' < d$, then we take all the vectors $\pi(v_2 g),\dots,\pi(v_{n-1}g)$ and otherwise we choose a linearly independent subset.
- In the case $n' < d$, that is, $n' = 2n-2$, we extend $(v'_1,\dots,v'_{n'})$ to a basis $\mathscr{B}' = (v'_1,\dots,v'_d)$ of V by choosing a basis of $F_{d-n} \cap \mathrm{Fix}(g)$, which is possible by condition $(C2)$. ◀

Let $\mathscr{L}' \in \mathrm{GL}(d,q)$ be the base change matrix from $\mathscr{B}$ to $\mathscr{B}'$ and from this point on, we assume that all of our matrices are given as elements of $G^{\mathscr{L}\mathscr{L}'}$. For the standard generators Y_n of $\mathrm{Sp}(n,q)$, there is nothing to do, while the matrix of g must be conjugated by the base change $\mathscr{L}'$.

Now we must deal with an additional problem which does not occur in special linear groups. One advantageous property of special linear groups is that we can ignore underlying forms but this is not the case for the other classical groups as for the symplectic group. Hence, we perform an additional base change before we continue as for special linear groups.

Remark 6.23

Let $H_B \coloneqq \langle H, H^{g^{\mathscr{L}'}} \rangle$. Note that H_B acts on $\langle v'_1,\dots,v'_{n'}\rangle$ and fixes $\langle v'_{n'+1},\dots,v'_d\rangle$. We compute the underlying form of H_B on $\langle v'_1,\dots,v'_{n'}\rangle$ and perform a base change $\mathscr{L}''$ such that $H_B^{\mathscr{L}''}$ respects the standard form on $\langle v'_1,\dots,v'_{n'}\rangle^{\mathscr{L}''}$. If this is not possible, then we know that H_B is not isomorphic to $\mathrm{Sp}(n',q)$ and we return to Sp2). From this point on, we assume that all of our matrices are expressed as elements of $G^{\mathscr{L}\mathscr{L}'\mathscr{L}''}$. Note, that the base change has no effect on the standard generators of H as the underlying form of H on V_n is already the standard form. ◀

Remark 6.24

Let $d \in \mathbb{N}$ and define $J_d \coloneqq (J_d)_{i,j\in\{1,\dots,d\}} \in \mathrm{GL}(d,q)$ with

$$(J_d)_{i,j} = \begin{cases} 1, & \text{if } i+j = d+1, \\ 0, & \text{otherwise.} \end{cases}$$

J_n is the *back-diagonal matrix*. Moreover, for $d \in \mathbb{N}$ even we define

$$P_d := \begin{pmatrix} 0 & J_{\frac{d}{2}} \\ -J_{\frac{d}{2}} & 0 \end{pmatrix}.$$

Note that P_d is the standard form preserved by $\mathrm{Sp}(d,q)$ as in Remark 2.28. By Remark 6.23 $H_B^{\mathscr{L}''}$ preserves the form $\mathrm{diag}(P_{n'}, I_{d-n'})$. In the implementation of the GoingUp step in GAP [37], see Chapter 11, a base change matrix $\mathscr{L}''' \in \mathrm{GL}(d,q)$ is chosen such that $H_B^{\mathscr{L}'''}$ preserves a different form. Let

$$b := \begin{pmatrix} P_n & 0 \\ 0 & P_{n'-n} \end{pmatrix} \in \mathrm{GL}(n',q).$$

Given that $H_B^{\mathscr{L}''}$ preserves the form $\mathrm{diag}(P_{n'}, I_{d-n'})$ it follows that there exists a base change matrix $\mathscr{L}''' \in \mathrm{GL}(d,q)$ such that $H_B^{\mathscr{L}'''}$ preserves the form $\mathrm{diag}(b, I_{d-n'})$. For consistency with the implementation we assume for the remainder of this section that the form $\mathrm{diag}(b, I_{d-n'})$ is preserved by $H_B^{\mathscr{L}''}$. Note that this assumption does not alter the results, but rather affects how the matrices are presented in the following. ◀

As for special linear groups, we denote c by $g^{\mathscr{L}'\mathscr{L}''}$. Moreover, we can now verify the last condition:

(C3) The vectors $\omega_i v_j''$ and $\omega_i v_j'' c^{-\mathrm{Tr}}$ for $1 \leq i \leq f$ and $2 \leq j \leq n-1$ span the subspace $\langle v_2'', \ldots, v_{n-1}'', v_{n+1}'', \ldots, v_{n'}'' \rangle$.

If (C3) is not satisfied by c, then we return to Sp2) and try another random element $a \in G^{\mathscr{L}}$.

We give an algorithm to compute $c \in G^{\mathscr{L}\mathscr{L}'\mathscr{L}''}$ in pseudo-code by Algorithm ComputeCSp [Alg. 32].

Algorithm 32: ComputeCSp

Input:
- $\langle X \rangle = G \leq \mathrm{Sp}(d,q)$
- A base change matrix $\mathscr{L} \in \mathrm{GL}(d,q)$
- $\mathrm{Sp}(n,q) \cong \langle Y_n \rangle = H \leq G^{\mathscr{L}}$ stingray embedded and constructively recognised
- $N \in \mathbb{N}$

Output: **fail OR** $(c, \mathscr{L}', \mathfrak{S}, N')$ **where**
- $c \in G^{\mathscr{L}\mathscr{L}'}$ is an element satisfying $(C1)$, $(C2)$ and $(C3)$,
- $\mathscr{L}' \in \mathrm{GL}(d,q)$ is a base change matrix,
- $\mathfrak{S}$ is an MSLP from $X \cup Y_n$ to c and
- $N' \in \mathbb{N}$ where $N - N'$ is the number of random selections that were used

function ComputeCSp$(G, \mathscr{L}, H, N)$

$t \in H$ as described in Lemma 6.17 // Sp1)
$\tilde{g} \leftarrow t^a$ for random $a \in G^{\mathscr{L}}$ // Start of Sp2)
while $N > 0$ **do**
  while $\tilde{g}$ **does not satisfy** $(C1)$ **and** $(C2)$ **and** (w_1, w_2) **is not a hyperbolic pair for** $\langle w_1, w_2 \rangle < V_n \cap \mathrm{Fix}(g)$ **do**
    $N \leftarrow N - 1$
    if $N \leq 0$ **then**
      return fail
    $\tilde{g} \leftarrow t^a$ for random $a \in G^{\mathscr{L}}$
  $\mathfrak{S}_1 \leftarrow$ MSLP from $X \cup Y_n$ to $\tilde{g}$
  $L_1 \leftarrow$ as described in Remark 6.20 **AND** $\mathfrak{S}_2 \leftarrow$ MSLP from Y_n to L_1 // Sp3)
  $L_2 \leftarrow$ as described in Remark 6.20 **AND** $\mathfrak{S}_3 \leftarrow$ MSLP from Y_n to L_2 // Sp3)
  $g \leftarrow L_2 L_1 \tilde{g} L_1^{-1} L_2^{-1}$ **AND** $\mathscr{L}' \leftarrow$ as described in Remark 6.22 // Sp4)
  $\mathscr{L}'' \leftarrow$ base change matrix to standard form of $\langle H, H^{g^{\mathscr{L}'}} \rangle$ **AND** $c \leftarrow g^{\mathscr{L}'\mathscr{L}''}$
  if the submatrix $(c^{-1})_{i,j}$ **for** $n+1 \leq i \leq n'$ **and** $2 \leq j \leq n-1$ **has full rank then**
    $\mathfrak{S} \leftarrow$ Compose $\mathfrak{S}_1$, $\mathfrak{S}_2$ and $\mathfrak{S}_3$ into one MSLP
    return $(c, \mathscr{L}'\mathscr{L}'', \mathfrak{S}, N)$
  else
    $\tilde{g} \leftarrow I_d$ // Element of H which does not satisfy $(C1)$ and $(C2)$
return fail

Remark 6.25

Computing the preserved form of a group and a base change matrix to the standard form involves an application of the meataxe [79] which has complexity $\mathcal{O}((n')^3)$ measured in field operations of $\mathbb{F}_q$. For more information see [13]. ◀

Lemma 6.26

Recall from our Hypothesis 6.12 that $H \leq \langle X^{\mathscr{L}} \rangle = G^{\mathscr{L}}$ with $H \cong \mathrm{Sp}(n,q)$ for $n < d$ and for a known base change matrix $\mathscr{L} \in \mathrm{GL}(d,q)$. Let $\mathscr{L}', \mathscr{L}'' \in \mathrm{GL}(d,q)$ be base change matrices as in Remark 6.22, 6.23 and 6.24. Let c be as constructed in ComputeCSp [Alg. 32] and

$$y := \left(\begin{array}{c|ccc|c|ccc} 1 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\ \hline 0 & * & \dots & * & 0 & * & \dots & * \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & * & \dots & * & 0 & * & \dots & * \\ \hline 0 & 0 & \dots & 0 & 1 & 0 & \dots & 0 \\ \hline 0 & * & \dots & * & 0 & * & \dots & * \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & * & \dots & * & 0 & * & \dots & * \end{array}\right) \in \mathrm{Sp}(n',q).$$

Then $c = \mathrm{diag}(y, I_{d-n'}) \in G^{\mathscr{L}\mathscr{L}'\mathscr{L}''}$.

Proof. It is clear that $v_1 c = v_1$ and $v_n c = v_n$. So it remains to show that $v_1 c^{\mathrm{Tr}} = v_1$ and that $v_n c^{\mathrm{Tr}} = v_n$. Note that (v_1, v_n) is a hyperbolic pair. Let b be the Gram-matrix of the underlying form. Since $c \in G = \mathrm{Sp}(d,q)$ we have for all $v, w \in V$ that

$$\langle wc \mid vc \rangle = wcb(vc)^{\mathrm{Tr}} = wcbc^{\mathrm{Tr}}v^{\mathrm{Tr}} = wbv^{\mathrm{Tr}} = \langle w \mid v \rangle$$

Hence, using that $bv_1^{\mathrm{Tr}} = v_n^{\mathrm{Tr}}$,

$$v_n w^{\mathrm{Tr}} = v_1 b w^{\mathrm{Tr}} = \langle v_1 \mid w \rangle = \langle v_1 c \mid wc \rangle = \langle v_1 \mid wc \rangle = v_1 bc^{\mathrm{Tr}} w^{\mathrm{Tr}} = v_n c^{\mathrm{Tr}} w^{\mathrm{Tr}}.$$

Therefore, $v_n c^{\mathrm{Tr}} = v_n$. With the same argumentation it follows that $v_1 c^{\mathrm{Tr}} = v_1$. □

Remark 6.27

With the computations so far we know that $H_B^{\mathscr{L}''}$ is stingray embedded in $G^{\mathscr{L}\mathscr{L}'\mathscr{L}''}$, that $H_B^{\mathscr{L}''}$ is isomorphic to $\mathrm{Sp}(n',q)$ in its natural representation and that $H_B^{\mathscr{L}''}$ respects the standard form. ◀

Now we can continue as for special linear groups and conjugate transvections of H with c.

Lemma 6.28

Let $n' = \min\{2n-2, d\}$, and let c and $\mathscr{L}'\mathscr{L}''$ be as constructed in the previous phases. Let $2 \leq j \leq n-1$, let $\imath = 1$ if $j \leq \frac{n}{2}$ and $\imath = -1$ if $j > \frac{n}{2}$ and

$$y := \begin{pmatrix} 1 & \imath\omega_i c_{n-j+1,2} & \dots & \imath\omega_i c_{n-j+1,n-1} & 0 & \imath\omega_i c_{n-j+1,n+1} & \dots & \imath\omega_i c_{n-j+1,n'} \\ 0 & 1 & \dots & 0 & \omega_i(c^{-1})_{2,j} & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & \omega_i(c^{-1})_{n-1,j} & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & \omega_i(c^{-1})_{n+1,j} & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & \omega_i(c^{-1})_{n',j} & 0 & \dots & 1 \end{pmatrix}.$$

Then $y \in \mathrm{Sp}(n',q)$ and $E_{j,n}^{\mathrm{Sp}}(\omega_i)^c = \mathrm{diag}(y, I_{d-n'})$.

Proof. Recall that $E_{j,n}^{\mathrm{Sp}}(\omega_i) = I_d + I_{j,n}(\omega_i) + \imath I_{1,n-j+1}(\omega_i)$. If $k \in \{n'+1, \dots, d\}$, then

$$e_k E_{j,n}^{\mathrm{Sp}}(\omega_i)^c = e_k c^{-1} E_{j,n}^{\mathrm{Sp}}(\omega_i) c = e_k E_{j,n}^{\mathrm{Sp}}(\omega_i) c = e_k c = e_k.$$

Moreover for $k \in \{1, \dots, n'\} \setminus \{1, n\}$

$$\begin{aligned} e_k c^{-1} E_{j,n}^{\mathrm{Sp}}(\omega_i) c = (c^{-1})_{k,-} E_{j,n}^{\mathrm{Sp}}(\omega_i) c &= (c^{-1})_{k,-}(I_d + I_{j,n}(\omega_i) + \imath I_{1,n-j+1}(\omega_i))c \\ &= ((c^{-1})_{k,-} + \omega_i(c^{-1})_{k,j} e_n + \imath\omega_i(c^{-1})_{k,1} e_{n-j+1})c \\ &\overset{6.26}{=} ((c^{-1})_{k,-} + \omega_i(c^{-1})_{k,j} e_n)c \\ &= e_k c^{-1} c + \omega_i(c^{-1})_{k,j} e_n c = e_k + \omega_i(c^{-1})_{k,j} e_n \end{aligned}$$

and

$$\begin{aligned} e_1 c^{-1} E^{\mathrm{Sp}}_{j,n}(\omega_i)c = e_1 E^{\mathrm{Sp}}_{j,n}(\omega_i)c &= e_1(I_d + I_{j,n}(\omega_i) + \imath I_{1,n-j+1}(\omega_i))c \\ &= (e_1 + e_1 I_{j,n}(\omega_i) + \imath e_1 I_{1,n-j+1}(\omega_i))c \\ &= (e_1 + \imath\omega_i e_{n-j+1})c = e_1 c + \imath\omega_i e_{n-j+1}c = e_1 + \imath\omega_i c_{n-j+1,-} \end{aligned}$$

as well as

$$e_n c^{-1} E^{\mathrm{Sp}}_{j,n}(\omega_i)c = e_n E^{\mathrm{Sp}}_{j,n}(\omega_i)c = e_n c = e_n.$$

□

Remark 6.29

Similarly to Remark 5.58 for special linear groups, the top left $n \times n$ block of $E^{\mathrm{Sp}}_{j,n}(\omega_i)^c$ is contained in $H \cong \mathrm{Sp}(n,q)$, hence we can express an element $h \in H$ as a word in the standard generators Y_n of H such that

$$hE^{\mathrm{Sp}}_{j,n}(\omega_i)^c = \begin{pmatrix} 1 & 0 & \dots & 0 & 0 & \omega_i(c^{-1})_{n+1,j} & \dots & -\omega_i(c^{-1})_{n',j} \\ 0 & 1 & \dots & 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & \omega_i(c^{-1})_{n+1,j} & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & \omega_i(c^{-1})_{n',j} & 0 & \dots & 1 \end{pmatrix}.$$

By $(C3)$ the column vectors of length $(n-2)$ below the (n,n) entry generate the full $f(n'-n)$ dimensional $\mathbb{F}_p$-vector space $\mathbb{F}_q^{n'-n}$, i.e. by multiplication the transvections $E^{\mathrm{Sp}}_{j,n}(\omega_i)$ for $1 \le i \le f$ and $n+1 \le j \le n'$ can be computed as in Remark 5.58. ◀

We combine Lemma 5.57 and Remark 6.29 into Algorithm ComputeVerticalTransvections [Alg. 25] for computing the vertical transvections $E^{\mathrm{Sp}}_{j,n}(\omega_i)$ for $1 \le i \le f$ and $n+1 \le j \le n'$.

Algorithm 33: COMPUTEVERTICALTRANSVECTIONSSP

Input: ▶ $\mathrm{Sp}(n,q) \cong \langle Y_n \rangle = H \leq \mathrm{Sp}(d,q)$ stingray embedded and constructively recognised
▶ $c \in \mathrm{Sp}(d,q)$ satisfying $(C1)$, $(C2)$ and $(C3)$

Output: ▶ $T_V := \{E^{\mathrm{Sp}}_{j,n}(\omega_i) \mid 1 \leq i \leq f, n+1 \leq j \leq n'\} \subset \langle H, H^c \rangle$ transvections of $\mathrm{Sp}(n',q)$
▶ An MSLP $\mathfrak{S}$ from $Y_n \cup \{c\}$ to T_V

function COMPUTEVERTICALTRANSVECTIONSSP(H, c)

```
// Sp5)
1  T̃_V ← [ ]
2  for λ ∈ {ω_1, ..., ω_f} and j ∈ {2, ..., n−1} do
3      T ← E^Sp_{j,n}(λ)^c
       // Do row operations
4      for k ∈ [2, ..., n−1] do
5          T ← E^Sp_{k,n}(−λ(c^{-1})_{k,j})T        // Note that E^Sp_{k,n}(−λ(c^{-1})_{k,j}) ∈ H
6      ι ← T_{1,n} AND T ← E^Sp_{1,n}(−ι)T          // Note that E^Sp_{1,n}(−ι) ∈ H
7      ADD(T̃_V, T)
8  T_V ← [ ]
9  for λ ∈ {ω_1, ..., ω_f} and j ∈ {n+1, ..., n'} do
10     E^Sp_{j,n}(λ) ← Multiply the matrices of T̃_V suitably
11     ι ← E^Sp_{j,n}(λ)_{1,n} AND E^Sp_{j,n}(λ) ← E^Sp_{1,n}(−ι)E^Sp_{j,n}(λ)
12     ADD(T_V, E_{j,n}(λ))
13 𝔖 ← MSLP for the computations of T_V
14 return (T_V, 𝔖)
```

In Sp6), the transvections $E^{\mathrm{Sp}}_{n,j}(\omega_i)$ for $1 \leq i \leq f$ and $n+1 \leq j \leq n'$ are computed. In contrast to special linear groups, this is easier for symplectic groups as c^{Tr} also fixes v_1 and v_n.

Lemma 6.30

Let $n' = \min\{2n-2, d\}$, and let c and $\mathscr{L}'\mathscr{L}''$ be as constructed in the previous phases. Let $2 \leq j \leq n-1$, let $\iota = 1$ if $j \leq \frac{n}{2}$ and $\iota = -1$ if $j > \frac{n}{2}$ and

$$y \coloneqq \begin{pmatrix} 1 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\ \imath\omega_i(c^{-1})_{2,n-j+1} & 1 & \dots & 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \imath\omega_i(c^{-1})_{n-1,n-j+1} & 0 & \dots & 1 & 0 & 0 & \dots & 0 \\ 0 & \omega_i c_{j,2} & \dots & \omega_i c_{j,n-1} & 1 & \omega_i c_{j,n+1} & \dots & \omega_i c_{j,n'} \\ \imath\omega_i(c^{-1})_{n+1,n-j+1} & 0 & \dots & 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \imath\omega_i(c^{-1})_{n',n-j+1} & 0 & \dots & 0 & 0 & 0 & \dots & 1 \end{pmatrix}.$$

Then $y \in \mathrm{Sp}(n',q)$ and $E^{\mathrm{Sp}}_{n,j}(\omega_i)^c = \mathrm{diag}(y, I_{d-n'})$.

Proof. We have that $E^{\mathrm{Sp}}_{n,j}(\omega_i) = I_d + I_{n,j}(\omega_i) + \imath I_{n-j+1,1}(\omega_i)$. If $k \in \{n'+1,\dots,d\}$, then it is clear that $e_k E^{\mathrm{Sp}}_{j,n}(\omega_i)^c = e_k$. For $k \in \{1,\dots,n'\} \setminus \{1,n\}$ we have

$$\begin{aligned} e_k c^{-1} E^{\mathrm{Sp}}_{n,j}(\omega_i)c = (c^{-1})_{k,-} E^{\mathrm{Sp}}_{n,j}(\omega_i)c &= (c^{-1}_{k,-})(I_d + I_{n,j}(\omega_i) + \imath I_{n-j+1,1}(\omega_i))c \\ &= ((c^{-1})_{k,-} + \omega_i(c^{-1})_{k,n} e_j + \imath\omega_i(c^{-1})_{k,n-j+1} e_1)c \\ &\overset{6.26}{=} ((c^{-1})_{k,-} + \imath\omega_i(c^{-1})_{k,n-j+1} e_1)c \\ &= e_k c^{-1} c + \imath\omega_i(c^{-1})_{k,n-j+1} e_1 c = e_k + \imath\omega_i(c^{-1})_{k,n-j+1} e_1 \end{aligned}$$

and

$$\begin{aligned} e_n c^{-1} E^{\mathrm{Sp}}_{n,j}(\omega_i)c = e_n E^{\mathrm{Sp}}_{j,n}(\omega_i)c &= e_n(I_d + I_{n,j}(\omega_i) + \imath I_{n-j+1,1}(\omega_i))c \\ &= (e_n + e_n I_{n,j}(\omega_i) + \imath e_n I_{n-j+1,1}(\omega_i))c \\ &= (e_n + \omega_i e_j)c = e_1 c + \omega_i e_j c = e_1 + \omega_i c_{j,-} \end{aligned}$$

as well as

$$e_1 c^{-1} E^{\mathrm{Sp}}_{n,j}(\omega_i)c = e_1 E^{\mathrm{Sp}}_{n,j}(\omega_i)c = e_1 c = e_1.$$

□

Now we continue analogously as for Sp5) in Remark 6.29.

Remark 6.31

In contrast to the special linear group we do not need the transvections of Sp5) to compute the desired transvections of Sp6). Therefore, one could also swap Sp5) and Sp6) for symplectic groups but to minimise the differences between the special linear group and the other classical groups we retain the order. ◀

We also provide pseudo-code for performing Sp6) using Algorithm ComputeHorizontalTransvectionsSp [Alg. 34].

Algorithm 34: ComputeHorizontalTransvectionsSp

Input: ▶ $\mathrm{Sp}(n,q) \cong \langle Y_n \rangle = H \leq \mathrm{Sp}(d,q)$ stingray embedded and constructively recognised
▶ $c \in \mathrm{Sp}(d,q)$ satisfying $(C1)$, $(C2)$ and $(C3)$

Output: ▶ $T_H := \{E^{\mathrm{Sp}}_{n,j}(1) \mid n+1 \leq j \leq n'\} \subset \langle H, H^c \rangle$ transvections of $\mathrm{Sp}(n',q)$
▶ An MSLP $\mathfrak{S}$ from $Y_n \cup \{c\}$ to T_H

function ComputeHorizontalTransvectionsSp(H,c)

// Sp6)

1 $\tilde{T_H} \leftarrow [\,]$

2 **for** $\lambda \in \{\omega_1,\ldots,\omega_f\}$ **and** $j \in \{2,\ldots,n-1\}$ **do**

3 $T \leftarrow E^{\mathrm{Sp}}_{n,j}(\lambda)^c$

// Do row operations

4 **for** $k \in [2,\ldots,n-1]$ **do**

5 $T \leftarrow T E^{\mathrm{Sp}}_{n,k}(-\lambda c_{j,k})$ // Note that $E^{\mathrm{Sp}}_{n,k}(-\lambda c_{j,k}) \in H$

6 $\imath \leftarrow T_{n,1}$ **AND** $T \leftarrow T E^{\mathrm{Sp}}_{n,1}(-\imath)$ // Note that $E^{\mathrm{Sp}}_{n,1}(-\imath) \in H$

7 Add$(\tilde{T_H}, T)$

8 $T_H \leftarrow [\,]$

9 **for** $\lambda \in \{\omega_1,\ldots,\omega_f\}$ **and** $j \in \{n+1,\ldots,n'\}$ **do**

10 $E^{\mathrm{Sp}}_{n,j}(\lambda) \leftarrow$ Multiply the matrices of $\tilde{T_H}$ suitably

11 $\imath \leftarrow E^{\mathrm{Sp}}_{n,j}(\lambda)_{n,1}$ **AND** $E^{\mathrm{Sp}}_{j,n}(\lambda) \leftarrow E^{\mathrm{Sp}}_{n,j}(\lambda) E^{\mathrm{Sp}}_{n,1}(-\imath)$

12 Add$(T_V, E_{n,j}(\lambda))$

13 $\mathfrak{S} \leftarrow$ MSLP for the computations of T_H

14 **return** $(T_H, \mathfrak{S})$

Finally, we have enough symplectic transvections to compute the element $z_1^{\mathrm{Sp}} \in \mathrm{Sp}(n',q)$ in $\langle H, H^c\rangle$ which automatically proves that $\langle H, H^c\rangle \cong \mathrm{Sp}(n',q)$.

Lemma 6.32

Let n be even. The permutation matrix $(z_1^{\mathrm{Sp}})'$ of $\mathrm{Sp}(n',q)$ can be computed using the matrices of the set $X = \{z_1^{\mathrm{Sp}}, E_{i,n}^{\mathrm{Sp}}(1), E_{n,i}^{\mathrm{Sp}}(1)\}$ for $n+1 \leq i \leq n'$.

Proof. Double transpositions can be computed via $(E_{i,n}^{\mathrm{Sp}}(1))^{-1} E_{n,i}^{\mathrm{Sp}}(1)(E_{i,n}^{\mathrm{Sp}}(1))^{-1}$ which is the permutation matrix corresponding to $(1,i)(n,n'-i+n+1) \in S_{n'}$ for $n < i \leq \frac{n'}{2}$ where the entries in position $(i,1)$ and $(n'-i+n+1,n)$ are equal to -1 and $E_{i,n}^{\mathrm{Sp}}(1)(E_{n,i}^{\mathrm{Sp}})^{-1}(1)E_{i,n}^{\mathrm{Sp}}(1)$ is the permutation matrix which corresponds to $(1,i)(n,n'-i+n+1) \in S_{n'}$ for $n < i \leq \frac{n'}{2}$ where the entries in position $(1,i)$ and $(n,n'-i+n)$ is equal to -1. Moreover,

$$\begin{aligned}
&\big(1,n+1\big)\big(n,n'\big)\cdot\big(1,n+2\big)\big(n,n'-1\big)\cdot\ldots\cdot\big(1,n+\frac{n'}{2}\big)\big(n,n'-\frac{n'}{2}\big)\\
=&\big(1,n+1,n+2\big)\big(n,n',n'-1\big)\cdot\ldots\cdot\big(1,n+\frac{n'}{2}\big)\big(n,n'-\frac{n'}{2}\big)\\
=&\big(1,n+1,\ldots,\frac{n'}{2}\big)\big(n,n'\ldots,n'-\frac{n'}{2}\big)
\end{aligned}$$

and

$$\begin{aligned}
&\big(1,\ldots,\frac{n}{2}\big)\big(\frac{n}{2}+1,n,\ldots,\frac{n}{2}+2\big)\cdot\big(1,n+1,\ldots,\frac{n'}{2}\big)\big(n,n'\ldots,n'-\frac{n'}{2}\big)\\
=&\big(1,\ldots,\frac{n}{2},n+1,\ldots,\frac{n'}{2}\big)\big(\frac{n}{2}+1,n'\ldots,n'-\frac{n'}{2},n,\ldots,\frac{n}{2}+2\big).
\end{aligned}$$

Because of the position of -1 in the transpositions, the matrices correspond to the standard generators of Definition 5.1 after a final base change. □

The GoingUp step of this section is given in pseudo-code using Algorithm GoingUpStepSp [Alg. 35].

Algorithm 35: GoingUpStepSp

Input:
- $\langle X \rangle = G \leq \mathrm{Sp}(d,q)$
- A base change matrix $\mathscr{L} \in \mathrm{GL}(d,q)$
- $\mathrm{Sp}(n,q) \cong \langle Y_n \rangle = H \leq G^{\mathscr{L}}$ stingray embedded and constructively recognised
- $N \in \mathbb{N}$

Output: **fail OR** $(Y_{n'}, \mathscr{L}', \mathfrak{S}, N')$ **where**
- $\mathrm{Sp}(\min\{2n-2, d\}, q) \cong \langle Y_{n'} \rangle = \tilde{H}$,
- $\mathscr{L}' \in \mathrm{GL}(d,q)$ is a base change matrix such that $\tilde{H}$ is stingray embedded in $G^{\mathscr{L}\mathscr{L}'}$,
- $\mathfrak{S}$ is an MSLP from $X \cup Y_n$ to the standard generators $Y_{n'}$ of $\tilde{H}$ and
- $N' \in \mathbb{N}$ where $N - N'$ is the number of random selections that were used

function GoingUpStepSp$(G, \mathscr{L}, H, N)$

1 $(c, \mathscr{L}', \mathfrak{S}_1, N) \leftarrow$ ComputeCSp$(G, \mathscr{L}, H, N)$
2 **if** $c =$ **fail then**
3 **return fail**
4 $T_V, \mathfrak{S}_2 \leftarrow$ ComputeVerticalTransvectionsSp(H, c)
5 $T_H, \mathfrak{S}_3 \leftarrow$ ComputeHorizontalTransvectionsSp(H, c)
6 Use T_V and T_H to construct z_1^{Sp} of $\mathrm{Sp}(n', q)$ as an MSLP $\mathfrak{S}_4$ using Lemma 6.32 // Sp7)
7 Compose $\mathfrak{S}_1, \mathfrak{S}_2, \mathfrak{S}_3, \mathfrak{S}_4$ into one MSLP $\mathfrak{S}$
8 **return** $(\langle H, H^c \rangle, \mathscr{L}', \mathfrak{S}, N)$

Theorem 6.33

Algorithm GoingUpStepSp [Alg. 35] terminates using at most N random selections and works correctly.

Proof. Follows with the same argumentation as in Theorem 5.63. □

6.3.2 Combining GoingUp steps

As for special linear groups, the GoingUp step for symplectic groups is called repeatedly until standard generators of the input group $G = \mathrm{Sp}(d,q)$ are constructed.

Algorithm 36: GoingUpSp

Input:
- $\langle X \rangle = G = \mathrm{Sp}(d,q)$
- A base change matrix $\mathscr{L} \in \mathrm{GL}(d,q)$
- $\mathrm{Sp}(4,q) \cong \langle Y_4 \rangle = H \leq G^{\mathscr{L}}$ stingray embedded and constructively recognised
- $N \in \mathbb{N}$

Output: **fail OR** $(\mathscr{L}', \mathfrak{S}, N')$ **where**
- $\mathscr{L}' \in \mathrm{GL}(d,q)$ is a base change matrix,
- $\mathfrak{S}$ is an MSLP from $X \cup Y_4$ to the standard generators of $G^{\mathscr{L}\mathscr{L}'}$ and
- $N' \in \mathbb{N}$ where $N - N'$ is the number of random selections that were used

function GoingUpSp$(G, \mathscr{L}, H, N)$

1 $n \leftarrow 4$ **AND** $\mathfrak{S} \leftarrow$ an MSLP from $X \cup Y_4$ to $X \cup Y_4$
2 **while** $n < d$ **do**
3 $n \leftarrow \min\{2 \cdot n - 2, d\}$ // Nearly double the dimension.
4 $(H, \mathscr{L}, \mathfrak{S}', N) \leftarrow$ GoingUpStepSp$(G, \mathscr{L}, H, N)$ // Remark 5.7 and 5.8
5 $\mathfrak{S} \leftarrow$ Compose $\mathfrak{S}$ and $\mathfrak{S}'$
6 **return** $(\mathscr{L}, \mathfrak{S}, N)$

Theorem 6.34

Algorithm GoingUpSp [Alg. 36] terminates using at most N random selections and works correctly.

Proof. Follows immediately from Theorem 6.33. □

Chapter 7

Special unitary group

In this chapter we describe an efficient constructive recognition algorithm for special unitary groups $\mathrm{SU}(d,q)$ in their natural representation for $d \geq 6$ if q is odd and $d \geq 10$ if q is even as outlined in Chapter 3. If q is odd and $d = 5$, then we refer to [21] and if q is even and $d \in \{5,\ldots,9\}$, then we refer to [32]. If $d \leq 4$, then we also refer to [21].

The ideas of this chapter are similar to the ones introduced for special linear groups, see Chapter 5, and symplectic groups, see Chapter 6, which is why we do not repeat the whole underlying theory in this chapter but instead highlight the differences. The results are valid for all characteristics, i.e. for q even and odd. Recall that in unitary groups if $g \in \mathrm{SU}(d,q)$, then $g \in \mathrm{GL}(d,q^2)$, and that $\mathbb{F}_{q^2}$ admits a field automorphism $\overline{\ }$ of order 2.

This chapter is structured similarly to Chapter 5 for special linear groups and Chapter 6 for symplectic groups. Section 7.1 deals with the GoingDown algorithm for unitary groups. The GoingDown basic step for unitary groups is identical to the GoingDown basic step for special linear groups. The final step also relies on algorithms from the DLLO algorithm [32, 59], see Section 5.1.3. Recall from Chapter 3 that the terminal group for unitary groups differs depending on whether q is odd or q is even as displayed in Table 3.1. If q is even, then we compute a stingray embedded $\mathrm{SU}(10,q)$ as a terminal group using the GoingDown basic step while we compute a stingray embedded $\mathrm{SU}(6,q)$ if q is odd. How we handle this distinction of q is discussed in more detail in Section 7.1.

In Section 7.2 we cite an algorithm which can be used for constructive recognition of $\mathrm{SU}(4,q)$, i.e.

an algorithm for handling unitary base case groups. We do not discuss the details of the constructive recognition algorithm for $\mathrm{SU}(4,q)$ and instead refer to the publication [21].

In Section 7.3 the GoingUp step for unitary groups is presented. The main approach is very similar to the GoingUp algorithm for symplectic groups and we discuss the differences in Section 7.3.

Before we introduce our standard generators of unitary groups a notation for specific elements of unitary groups is given in the next definition.

Definition 7.1

Let $d \in \mathbb{N}$ be even. For $i,j \in \{1,\dots,d\}$, $\lambda \in \mathbb{F}_{q^2}\backslash\{0\}$ and $j \neq i$ we set

$$E^{\mathrm{SU}}_{i,j}(\lambda) := \begin{cases} I_d + I_{i,j}(\lambda), & \text{if } i+j = d+1, \\ I_d + I_{i,j}(\lambda) - I_{d-j+1,d-i+1}(\overline{\lambda}), & \text{otherwise.} \end{cases}$$

Let $d \in \mathbb{N}$ be odd. For $i,j \in \{1,\dots,d\}$, $\lambda \in \mathbb{F}_{q^2}\backslash\{0\}$ and $j \neq i$ we set

$$E^{\mathrm{SU}}_{i,j}(\lambda) := \begin{cases} I_d + I_{i,j}(\lambda), & \text{if } i+j = d+1, \\ I_d + I_{i,j}(\lambda) - I_{d-j+1,d-i+1}(\overline{\lambda}), & \text{if } i+j \neq d+1 \text{ and } i,j \neq \frac{d+1}{2} \\ I_d + I_{i,j}(\lambda) - I_{d-j+1,d-i+1}(\overline{\lambda}) + I_{n-j+1,j}(\lambda_2), & \text{if } i = \frac{d+1}{2} \\ I_d + I_{i,j}(\lambda) - I_{d-j+1,d-i+1}(\overline{\lambda}) + I_{i,n-i+1}(\lambda_2), & \text{otherwise.} \end{cases}$$

For d odd $\lambda_2 \in \mathbb{F}_{q^2} - \{0\}$ has to be chosen such that $\lambda\overline{\lambda} + \lambda_2 + \overline{\lambda_2} = 0$.

Remark 7.2

1) There always exists $\lambda_2 \in \mathbb{F}_{q^2} - \{0\}$ such that $\lambda\overline{\lambda} + \lambda_2 + \overline{\lambda_2} = 0$ as required in Definition 7.1.
2) The matrix $I_d + E_{i,j}(\lambda)$ for $i+j = d+1$ is an element of $\mathrm{SU}(d,q)$ if and only if $\lambda + \overline{\lambda} = 0$. Therefore, we have to be careful during computations to not accidentally use elements which are not contained in $\mathrm{SU}(d,q)$. ◀

Definition 7.3

Let $S^{\mathrm{SU}} \subseteq \mathrm{SL}(d,q)$. If q is odd, then let $\omega \in \mathbb{F}_{q^2}$ be a primitive element and let $\alpha := \omega^{(q+1)/2}$. Then S^{SU} is a set of *standard generators* for $\mathrm{SU}(d,q)$ if S^{SU} is conjugate to the following set consisting of $5f+5$ elements if d is even and $5f+6$ elements if d is odd:

- $E^{\mathrm{SU}}_{1,2}(\omega_i)$ for $1 \leq i \leq 2f$,
- $E^{\mathrm{SU}}_{2,1}(\omega_i)$ for $1 \leq i \leq 2f$,
- $E^{\mathrm{SU}}_{1,n}(\alpha^{-q}(\omega^{q+1})^i)$ for $0 \leq i \leq f-1$,
- a permutation matrix z_1^{SU} corresponding to the permutation $(1,2,\ldots,\frac{d}{2})(\frac{d}{2}+1,d,d-1,\ldots,\frac{d}{2}+2)$ if d is even and $(1,2,\ldots,\frac{d-1}{2})(\frac{d+1}{2}+1,d,d-1,\ldots,\frac{d+1}{2}+2)$ if d is odd,
- a permutation matrix z_2^{SU} corresponding to the permutation $(1,2)(d-1,d)$,
- a permutation matrix z_3^{SU} corresponding to the permutation $(1,d)$ with the entry $(z_3^{\mathrm{SU}})_{1,d}$ changed to α and entry $(z_3^{\mathrm{SU}})_{d,1}$ changed to α^{-q},
- a diagonal matrix z_4^{SU} corresponding to $\mathrm{diag}(\omega^{q+1},1,\ldots,1,\omega^{-(q+1)})$,
- a diagonal matrix z_5^{SU} corresponding to $\mathrm{diag}(\omega,\omega^{-1},1,\ldots,1,\omega^q,\omega^{-q})$ if d is even and $\mathrm{diag}(\omega^{-q},1,\ldots,1,\omega^q)+(\omega^{q-1}-1)I_{\frac{n+1}{2},\frac{n+1}{2}}$ if d is odd and
- $E^{\mathrm{SU}}_{1,\frac{n+1}{2}}(1)$ if d is odd.

Lemma 7.4

Every element $E^{\mathrm{SU}}_{i,j}(\lambda)$ can be written in terms of the standard generators of Definition 7.3.

Proof. The standard generators of Definition 7.3 contain the DLLO standard generators [32, 59] and, therefore, generate $\mathrm{SU}(n,q)$. Hence, the elements $E^{\mathrm{SU}}_{i,j}(\lambda)$ can be written in terms of the standard generators of Definition 7.3. □

Lemma 7.5

$\mathrm{SU}(d,q)$ is generated by the standard generators from Definition 7.3.

Proof. This follows immediately as the DLLO standard generators [32, 59] are contained in the set of standard generators from Definition 7.3. □

7.1 GoingDown algorithm

In this section we describe the GoingDown algorithm for unitary groups $\langle X \rangle = G = \mathrm{SU}(d,q)$ in their natural representation. The algorithms of this chapter are valid for all characteristics but require that $d \geq 6$ if q is odd and $d \geq 10$ if q is even. Note that if $d = 4$, then the input group G is a base case group, which is handled by a separate base case algorithm by the StandardGenerators algorithm as outlined in Chapter 3. The output of the GoingDown algorithm for unitary groups is a base change matrix $\mathscr{L} \in \mathrm{GL}(d,q^2)$, a stingray embedded subgroup $\mathrm{SU}(4,q) \cong U$ of $G^{\mathscr{L}}$ and an MSLP $\mathfrak{S}$ from $X^{\mathscr{L}}$ to generators of U.

The GoingDown algorithm consists of two subalgorithms as presented in Chapter 3. First, a GoingDown basic step is repeatedly used to compute a stingray embedded terminal group of G. Second, an additional algorithm is called on the terminal group to construct a stingray embedded base case group. The process of computing a stingray embedded base case group in a terminal group is the final step.

For unitary groups we can use the same methods as for the GoingDown basic step for special linear groups described in Section 5.1.1. The modifications of the GoingDown basic step for special linear groups to unitary groups are outlined in Remark 7.6.

Remark 7.6

Recall the GoingDown basic step for special linear groups given in pseudo-code as Algorithm GoingDownBasicStepSL [Alg. 13] in Section 5.1.1. For readability of Chapter 5 the Algorithm GoingDownBasicStepSL [Alg. 13] is only provided for special linear groups but can easily be modified to be applicable for special linear groups and special unitary groups as it is done in Algorithm GoingDownBasicStepReciprocal [Alg. 29] for symplectic and orthogonal groups. By changing the input and output of Algorithm GoingDownBasicStepSL [Alg. 13] we introduce a new algorithm GoingDownBasicStep [Alg. 37] which is usable in special linear groups and unitary groups. Note that the function body of Algorithm GoingDownBasicStep [Alg. 37] is the same as of Algorithm GoingDownBasicStepSL [Alg. 13] except that we call a naming algorithm for special linear group on $\langle s_1, s_2 \rangle$ if the input group G is isomorphic to a special linear group and a naming algorithm for unitary group if the input group G is isomorphic to a unitary group in Line 9. ◀

Algorithm 37: GoingDownBasicStep

Input:
- $d_1 \in \mathbb{N}$ with $d_1 > 4$
- $\langle X \rangle = G \leq \mathrm{GL}(d, q^2)$ with $G \cong \mathrm{SL}(d_1, q)$ or $G \cong \mathrm{SU}(d_1, q)$ and $d_1 > 6$
- Φ a form preserved by G (omitted if $G \cong \mathrm{SL}(d_1, q)$)
- $N \in \mathbb{N}$

Output: **fail OR** $(d_2, U, \mathfrak{S}, N')$ **where**
- $d_2 \in \mathbb{N}$ with $4 \leq d_2 \leq 4\lceil \log(d_1) \rceil$,
- $U \leq G$ with $U \cong \mathrm{SL}(d_2, q)$ if $G \cong \mathrm{SL}(d_1, q)$ and $U \cong \mathrm{SU}(d_2, q)$ otherwise,
- $\mathfrak{S}$ is an MSLP from X to generators of U and
- $N' \in \mathbb{N}$ where $N - N'$ is the number of random selections that were used

function GoingDownBasicStep(d_1, G, Φ, N)

```
while N > 0 do                                                          // Remark 5.7
    (s_1, 𝔖_1, N) ← FindStingrayElement(G, d_1, N)                      // Remark 5.8
    W_{s_1} ← ComputeStingrayBody(s_1)
    repeat
        (s_2, 𝔖_2, N) ← FindStingrayElement(G, d_1, N)                  // Remark 5.8
        W_{s_2} ← ComputeStingrayBody(s_2)
    until IsStingrayDuo((s_1, s_2), Φ)
    d_2 ← dim(W_{s_1}) + dim(W_{s_2})
    if ⟨s_1, s_2⟩ ≅ SL(d_2, q) (resp. ⟨s_1, s_2⟩ ≅ SU(d_2, q)) then // Using a naming algorithm, see Section 1.1.7
        𝔖 ← an MSLP from X to (s_1, s_2) using 𝔖_1 and 𝔖_2
        return (d_2, ⟨s_1, s_2⟩, 𝔖, N)
return fail
```

Using Algorithm GoingDownBasicStep [Alg. 37] we introduce an algorithm in pseudo-code which uses the GoingDown basic step for unitary groups repeatedly until a terminal group is computed as Algorithm GoingDownToDim6Or10SU [Alg. 38]. Note that Algorithm GoingDownToDim6Or10SU [Alg. 38] computes a stingray embedded $\mathrm{SU}(6, q)$ if q is odd and $\mathrm{SU}(10, q)$ if q is even.

Algorithm 38: GoingDownToDim6Or10SU

Input:
- ▶ $d \in \mathbb{N}$ with $d \geq 6$ if q odd and $d \geq 10$ if q is even
- ▶ $\langle X \rangle = G = \mathrm{SU}(d, q)$
- ▶ Φ a unitary form preserved by G
- ▶ $N \in \mathbb{N}$

Output: **fail OR** $(U, \mathfrak{S}, N')$ **where**
- ▶ $U \leq G$ with $U \cong \mathrm{SU}(6, q)$ if q is odd and $U \cong \mathrm{SU}(10, q)$ if q is even,
- ▶ $\mathfrak{S}$ is an MSLP from X to generators of U and
- ▶ $N' \in \mathbb{N}$ where $N - N'$ is the number of random selections that were used

function GoingDownToDim6Or10SU(d, G, Φ, N)

```
1  if q is even then
2      e ← 10
3  else
4      e ← 6
5  U ← G AND dim ← d AND 𝔖 ← an MSLP from X to X
6  while dim > e do
7      (dim, U, 𝔖′, N) ← GoingDownBasicStep(dim, U, Φ, N)        // Remark 5.7 and 5.8
8      𝔖 ← Composition of 𝔖 and 𝔖′
9  return (U, 𝔖, N)
```

Theorem 7.7

Algorithm GoingDownToDim6Or10SU [Alg. 38] terminates and works correctly.

Proof. Analogously to the proof of Theorem 5.13. □

Lastly we deal with the final step of the GoingDown algorithm for unitary groups which aims to compute a stingray embedded base case group in the terminal group. Note that the terminal group for unitary groups depends on q, i.e. Algorithm GoingDownToDim6Or10SU [Alg. 38] outputs a subgroup U of G with $U \cong \mathrm{SU}(6, q)$ if q is odd and $U \cong \mathrm{SU}(10, q)$ if q is even. In the final step for unitary groups we use methods from the DLLO algorithm [32, 59] by calling Algorithm GoingDownFinalStepCL [Alg. 16] which is described in Section 5.1.3. Note that Algorithm

GoingDownFinalStepCL [Alg. 16] can be used for both settings, i.e. $U \cong \mathrm{SU}(6,q)$ for q odd and $U \cong \mathrm{SU}(10,q)$ for q even, by calling `GoingDownFinalStepCL`$(U,4,N)$, see Remark 5.21. Using these methods a stingray embedded $\mathrm{SU}(4,q)$ in U can be computed.

An overall GoingDown algorithm for unitary groups is not discussed but can easily be derived from the GoingDown algorithm for special linear groups in Section 5.1.5 using the subalgorithms introduced in this section.

7.2 BaseCase algorithm

In this section we assume that a stingray embedded subgroup $\langle X \rangle = H \leq G = \mathrm{SU}(d,q)$ has been computed and that $H \cong \mathrm{SU}(4,q)$ which can be achieved using the results and GoingDown algorithm for unitary groups of Section 7.1. Since $H \cong \mathrm{SU}(4,q)$ we have a unitary base case group as in Definition 3.2. Therefore, there is an efficient constructive recognition algorithm for H. The algorithm presented in [21] is used in the implementation of the algorithms of this thesis. As displayed in Figure 3.1 constructive recognition of $\mathrm{SU}(4,q)$ is based on constructive recognition of $\mathrm{SU}(3,q)$ which relies on constructive recognition of $\mathrm{SL}(2,q)$. Therefore, constructive recognition of $\mathrm{SU}(4,q)$ also involves the usage of a discrete logarithm oracle as in Definition 1.6. As for symplectic and orthogonal groups we are not diving into the details of [21] and only state the main theorem about the complexity.

Theorem 7.8: [21]

Assume the availability of oracles which constructively recognise $\mathrm{SL}(2,q)$ and compute discrete logarithms in cyclic groups of order $q \pm 1$. There is an

$$\mathcal{O}(d^2 \log(d)(d \log^4(q) + \mathcal{E} \log q + \zeta))$$

time Las Vegas black-box algorithm which constructively recognises any $\mathrm{SU}(d,q)$ where $d \geq 3$. Here, ζ is the complexity to construct a (nearly) uniformly distributed random element of H as an MSLP in X and $\mathcal{E}$ is the complexity of constructively recognising (a homomorphic image of) $\mathrm{SL}(2,q)$.

Proof. [21]. □

Corollary 7.9

Assume the availability of oracles which constructively recognise $\mathrm{SL}(2,q)$ and compute discrete logarithms in cyclic groups of order $q \pm 1$. There is an

$$\mathcal{O}(\zeta + \mathcal{E} \log q + \log^4(q))$$

time Las Vegas black-box algorithm which constructively recognises $\mathrm{SU}(4,q)$. Here, ζ is the complexity to construct a (nearly) uniformly distributed random element of H as an MSLP in X and $\mathcal{E}$ is the complexity of constructively recognising (a homomorphic image of) $\mathrm{SL}(2,q)$.

7.3 GoingUp algorithm

In this section we describe the GoingUp algorithm for unitary groups. The algorithm is similar to the GoingUp algorithm for symplectic groups, see Section 6.3. Thus, we only highlight and discuss the differences between the GoingUp algorithm for symplectic and unitary groups. Our hypothesis for the remainder of this section is the following.

Hypothesis 7.10

Let $d \in \mathbb{N}$ and $\langle X \rangle = G = \mathrm{SU}(d,q)$ containing a stingray embedded subgroup $H \leq \langle X^{\mathcal{L}} \rangle = G^{\mathcal{L}}$ with $H \cong \mathrm{SU}(n,q)$ for $n < d$, n even and for a known base change matrix $\mathcal{L} \in \mathrm{GL}(d,q^2)$. Moreover, standard generators Y_n of H are given as words in X. Let $V = \mathbb{F}_q^d$ and suppose that $\mathcal{B} = (v_1, \ldots, v_d)$ is a basis for V and let $V_n = \langle v_1, \ldots, v_n \rangle$ and $F_{d-n} = \langle v_{n+1}, \ldots, v_d \rangle$ (cf. Definition 2.7). We assume that H acts on V_n as $\mathrm{SU}(n,q)$ and that H fixes F_{d-n} point-wise. Recall that $(\omega_1, \ldots, \omega_f)$ is an $\mathbb{F}_p$-basis for $\mathbb{F}_q$.

Theorem 7.11

Let $X \subseteq \mathrm{SU}(d,q)$ such that $\langle X \rangle = G = \mathrm{SU}(d,q)$ with $4 \leq n < d$ and n even and let $\mathcal{L} \in \mathrm{GL}(d,q^2)$ be a base change matrix. Let $Y_n^{\mathcal{L}}$ be a set of standard generators for the subgroup $\mathrm{SU}(n,q)$ stingray embedded into $G^{\mathcal{L}}$. Furthermore, let $\mathfrak{S}$ be an SLP from X to Y_n and let $n' := \min\{2n-2, d\}$.

Then there is an algorithm that computes a base change matrix $\mathscr{L}' \in \mathrm{GL}(d, q^2)$ together with an SLP $\mathfrak{S}'$ from X to a set $Y_{n'}$, which is a set of standard generators for $\mathrm{SU}(n', q)$ and $\langle Y_{n'}^{\mathscr{L}'} \rangle$ is stingray embedded in $G^{\mathscr{L}'}$.

Recall that the definition of a strong doubling element in special linear and symplectic groups requires the element to satisfy three conditions $(C1)$, $(C2)$ and $(C3)$ of Remark 5.36 and 5.53. Let $\tilde{g} \in G^{\mathscr{L}}$. Then $\tilde{g}$ is a strong doubling element with respect to H if

$(C1)$ $\dim(V_n + V_n\tilde{g}) = n'$.

$(C2)$ If $n' < d$, then $\dim(F_{d-n} + \mathrm{Fix}(\tilde{g})) = d$.

$(C3)$ $\tilde{g}$ fixes v_1 and v_n. Let $(v_1'', \ldots, v_{n'}'') := \mathscr{B}^{\mathscr{L}'\mathscr{L}''}$ and $c := \tilde{g}^{\mathscr{L}'\mathscr{L}''}$ for base change matrices $\mathscr{L}', \mathscr{L}''$ as computed in Remark 6.22 and Remark 6.23. The vectors $\omega_i v_j''$ and $\omega_i v_j'' c^{-\mathrm{Tr}}$ for $1 \leq i \leq f$ and $2 \leq j \leq n-1$ span the subspace $\langle v_2'', \ldots, v_{n-1}'', v_{n+1}'', \ldots, v_{n'}'' \rangle$.

Moreover, recall the seven phases of the GoingUp step for symplectic groups:

Sp1) Construct an element $t \in H$ which has a fixed space of dimension $d - n + 2$.

Sp2) Choose random elements $a \in G^{\mathscr{L}}$ until $\tilde{g} := t^a$ satisfies $(C1)$ and $(C2)$.

Sp3) Find a conjugate $g \in G^{\mathscr{L}}$ of $\tilde{g}$ which satisfies $(C1)$ and $(C2)$ and additionally fixes v_1 and v_n.

Sp4) Compute base change matrices $\mathscr{L}'$ and $\mathscr{L}''$ such that $\langle H, H^g \rangle^{\mathscr{L}'\mathscr{L}''}$ is stingray embedded in $G^{\mathscr{L}\mathscr{L}'}$ and preserves the standard form. Set $c := g^{\mathscr{L}'\mathscr{L}''}$ and verify whether c satisfies $(C3)$ of Remark 6.15.

Sp5) Using c, construct transvections $E_{j,n}^{\mathrm{Sp}}(\omega_i)$ for $n < j \leq n'$ for $\langle H, H^c \rangle \cong \mathrm{Sp}(n', q)$.

Sp6) Using c, construct transvections $E_{n,j}^{\mathrm{Sp}}(\omega_i)$ for $n < j \leq n'$ for $\langle H, H^c \rangle \cong \mathrm{Sp}(n', q)$.

Sp7) Using the transvections of Sp5) and Sp6) construct standard generators for $\langle H, H^c \rangle \cong \mathrm{Sp}(n', q)$ by assembling a permutation matrix z_1^{Sp} corresponding to a permutation of cycle type $(\frac{n'}{2})^2$ as in Definition 6.3.

The phases applied to the unitary group are denoted by SU1) to SU7). A summary of the seven phases for unitary groups is given in Remark 7.16 and the complete GoingUp algorithm for unitary

groups is given by Algorithm GoingUpSU [Alg. 39].

The phases SU1), SU2) and SU4) can be performed the same way as the phases Sp1), Sp2) and Sp4). For the phases SU5), SU6) and SU7) there are minor differences compared to the phases Sp5), Sp6) and Sp7), while phase SU3) has to be redesigned from scratch. We start by discussing how phase SU3) can be achieved, which we do in Remark 7.13, and afterwards discuss the slight differences in the phases SU5), SU6) and SU7) compared to the phases Sp5), Sp6) and Sp7). A solution of the phases SU5), SU6) is given in Remark 7.14 and a solution of the phase SU7) in Lemma 7.15.

In phase Sp3) a weak doubling element $\tilde{g}$ is given and we aim to compute a conjugate of $\tilde{g}$ which is a doubling element, i.e. an element which satisfies $(C1)$ and $(C2)$ and additionally fixes v_1 and v_n. In phase Sp3) we constructed matrices $L_1, L_2 \in \mathrm{Sp}(n,q)$ of the form

$$L_1 := \begin{pmatrix} 1 & w_{1,2} & w_{1,3} & \dots & w_{1,n-1} & 0 \\ 0 & 1 & 0 & \dots & 0 & w_{1,n-1} \\ 0 & 0 & 1 & \dots & 0 & w_{1,n-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & -w_{1,2} \\ 0 & 0 & 0 & \dots & 0 & 1 \end{pmatrix}, \quad L_2 := \begin{pmatrix} 1 & 0 & 0 & \dots & 0 & 0 \\ -w_{2,n-1} & 1 & 0 & \dots & 0 & 0 \\ -w_{2,n-2} & 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ w_{2,2} & 0 & 0 & \dots & 1 & 0 \\ 0 & w_{2,2} & w_{2,3} & \dots & w_{2,n-1} & 1 \end{pmatrix}$$

for two vectors $w_1 = (1, w_{1,2}, w_{1,3}, \dots, w_{1,n-1}, 0)$ and $w_2 = (0, w_{2,2}, w_{2,3}, \dots, w_{2,n-1}, 1)$ such that the element $(L_1L_2)^{-1}\tilde{g}(L_1L_2)$ is a doubling element. Note that in symplectic groups $L_1, L_2 \in H \cong \mathrm{Sp}(n,q)$ for all vectors $w_1 = (1, w_{1,2}, w_{1,3}, \dots, w_{1,n-1}, 0)$ and $w_2 = (0, w_{2,2}, w_{2,3}, \dots, w_{2,n-1}, 1)$. Recall that in unitary groups $E^{\mathrm{SU}}_{1,n}(\lambda)$ and $E^{\mathrm{SU}}_{n,1}(\lambda)$ is an element of $\mathrm{SU}(n,q)$ if and only if $\lambda + \overline{\lambda} = 0$, see Remark 7.2. Thus, for $(1, \nu_2, \dots, \nu_{n-1}, 0) \in \mathbb{F}^n_{q^2}$ the matrix

$$\begin{pmatrix} 1 & \nu_2 & \nu_3 & \dots & \nu_{n-1} & 0 \\ 0 & 1 & 0 & \dots & 0 & -\overline{\nu_{n-1}} \\ 0 & 0 & 1 & \dots & 0 & -\overline{\nu_{n-2}} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & -\overline{\nu_2} \\ 0 & 0 & 0 & \dots & 0 & 1 \end{pmatrix} \in \mathrm{SL}(n, q^2)$$

is not always contained in $\mathrm{SU}(n,q)$. Hence, we cannot proceed for unitary groups as for symplectic groups. For phase SU3) we start by observing the action of unitary groups on 1-dimensional subspaces.

Lemma 7.12: [91, Theorem 10.12]

If V is an $\mathbb{F}$-vector space with Witt index 1, then $\mathrm{PSU}(V)$ is a faithful doubly transitive group on the isotropic points of V. If V is an $\mathbb{F}$-vector space with Witt index at least 2, then $\mathrm{PSU}(V)$ is transitively on the hyperbolic pairs of V.

Proof. [91, Theorem 10.12]. □

Based on Lemma 7.12 we proceed as in the next remark to compute $L_1, L_2 \in H \cong \mathrm{SU}(n,q)$ such that $(L_1L_2)^{-1}\tilde{g}(L_1L_2)$ satisfies $(C1)$ and $(C2)$ and additionally fixes v_1 and v_n.

Remark 7.13: SU3)

We assume that $\langle w_1, w_2\rangle \leq V_n \cap \mathrm{Fix}(c)$ where (w_1, w_2) is a hyperbolic pair, w_1 has a non-zero entry at position 1 and w_2 has a non-zero entry at position n. If this is not satisfied, then we return to choosing random elements $c_1 \in G^{\mathscr{L}}$. We normalise the entry at position 1 of w_1 and the entry at position n of w_n. Moreover, by linear combinations of w_1 and w_2 we may assume that

$$w_1 = (1, \lambda_2, \ldots, \lambda_{n-1}, 0) \qquad \text{and} \qquad w_2 = (0, \iota_2, \ldots, \iota_{n-1}, 1)$$

as elements of $\mathbb{F}_{q^2}^n$. Note that this process can fail if w_1 and w_2 both have the same entry in position 1 and n but in this case we also restart with choosing another random element $c_1 \in G^{\mathscr{L}}$. Next we introduce for $(1, \nu_1, \ldots, \nu_{n-1}) = \nu \in \mathbb{F}_{q^2}^n$ the notation

$$T_\nu := \begin{pmatrix} 1 & \nu_1 & \nu_2 & \ldots & \nu_{n-2} & \nu_{n-1} \\ 0 & 1 & 0 & \ldots & 0 & -\overline{\nu_{n-2}} \\ 0 & 0 & 1 & \ldots & 0 & -\overline{\nu_{n-3}} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & 1 & -\overline{\nu_1} \\ 0 & 0 & 0 & \ldots & 0 & 1 \end{pmatrix} \in \mathrm{SL}(n, q^2).$$

Note that in contrast to symplectic groups T_ν is not contained in $\mathrm{SU}(n,q)$ for all $(1,\nu_1,\dots,\nu_{n-1}) \in \mathbb{F}_{q^2}^n$ which is why the construction of $L_1, L_2 \in H$ requires more effort. For $\beta \in \mathbb{F}_{q^2}$ we now compute

$$\nu := w_1 + \beta w_2 = (1, \lambda_2 + \beta\imath_2, \dots, \lambda_{n-1} + \beta\imath_{n-1}, \beta).$$

We aim to prove two things. Firstly, that there always exists an element $\beta \in \mathbb{F}_{q^2}$ such that $T_{w_1+\beta w_2} \in \mathrm{SU}(n,q)$. Secondly, we seek to develop an efficient method for computing $\beta \in \mathbb{F}_{q^2}$ such that $T_{w_1+\beta w_2} \in \mathrm{SU}(n,q)$. Since we describe a constructive and efficient method for computing $\beta \in \mathbb{F}_{q^2}$ with $T_{w_1+\beta w_2} \in \mathrm{SU}(n,q)$, this also proves the existence. Note that n is even and recall from Remark 2.28 that for

$$b := \begin{pmatrix} 0 & & 1 \\ & \iddots & \\ 1 & & 0 \end{pmatrix} \in \mathrm{GL}(n,q^2)$$

follows $\mathrm{SU}(n,q) = \{a \in \mathrm{SL}(n,q^2) \mid aba^* = b\}$ where $a^* = \overline{(a^{\mathrm{Tr}})} = (\overline{a})^{\mathrm{Tr}}$. Hence, we aim to compute $\beta \in \mathbb{F}_{q^2}$ such that

$$\begin{aligned}
& b = T_{w_1+\beta w_2} b T_{w_1+\beta w_2}^* \\
\Leftrightarrow\ & 0 = \beta + (\lambda_{n-1} + \beta\imath_{n-1})\overline{(\lambda_2 + \beta\imath_2)} + \dots + (\lambda_2 + \beta\imath_2)\overline{(\lambda_{n-1} + \beta\imath_{n-1})} + \overline{\beta} \\
\Leftrightarrow\ & 0 = \beta + \overline{\beta} + (\lambda_{n-1} + \beta\imath_{n-1})(\overline{\lambda_2} + \overline{\beta}\,\overline{\imath_2}) + (\lambda_2 + \beta\imath_2)(\overline{\lambda_{n-1}} + \overline{\beta}\,\overline{\imath_{n-1}}) \\
\Leftrightarrow\ & 0 = \beta\overline{\beta}\underbrace{(\imath_{n-1}\overline{\imath_2} + \dots + \imath_2\overline{\imath_{n-1}})}_{=:\gamma_1} + \beta\underbrace{(1 + \imath_{n-1}\overline{\lambda_2} + \dots + \imath_2\overline{\lambda_{n-1}})}_{=:\gamma_2} + \overline{\beta}\underbrace{(1 + \lambda_{n-1}\overline{\imath_2} + \dots + \lambda_2\overline{\imath_{n-1}})}_{=:\gamma_3} + \\
& \qquad \underbrace{(\lambda_{n-1}\overline{\lambda_2} + \dots + \lambda_2\overline{\lambda_{n-1}})}_{=:\gamma_4} \\
\Leftrightarrow\ & 0 = \beta\overline{\beta}\gamma_1 + \beta\gamma_2 + \overline{\beta}\gamma_3 + \gamma_4.
\end{aligned}$$

Note that $\gamma_1 \in \mathbb{F}_0 := \mathrm{Fix}(\mathbb{F}_{q^2})$ since

$$\overline{\gamma_1} = \overline{\imath_{n-1}\overline{\imath_2} + \dots + \imath_2\overline{\imath_{n-1}}} = \overline{\imath_{n-1}}\imath_2 + \dots + \overline{\imath_2}\imath_{n-1} = \gamma_1.$$

Therefore, we solve the equation $\gamma_0\overline{\gamma_0} = \gamma_1$ for some $\gamma_0 \in \mathbb{F}_{q^2}$ which can be done efficiently using

Hilbert 90 [62, Theorem 3 and 4]. We set $\beta_0 \coloneqq \beta\gamma_0$ yielding

$$\begin{aligned} 0 &= \beta\overline{\beta}\gamma_1 + \beta\gamma_2 + \overline{\beta}\gamma_3 + \gamma_4 \\ \Leftrightarrow 0 &= \beta\overline{\beta}\gamma_0\overline{\gamma_0} + \beta\gamma_0\gamma_0^{-1}\gamma_2 + \overline{\beta}\overline{\gamma_0}\overline{\gamma_0^{-1}}\gamma_3 + \gamma_4 \\ \Leftrightarrow 0 &= \beta_0\overline{\beta_0} + \beta_0\frac{\gamma_2}{\gamma_0} + \overline{\beta_0}\frac{\gamma_3}{\overline{\gamma_0}} + \gamma_4. \end{aligned}$$

Observe that

$$\overline{\gamma_2} = \overline{1 + \iota_{n-1}\overline{\lambda_2} + \ldots + \iota_2\overline{\lambda_{n-1}}} = 1 + \overline{\iota_{n-1}}\lambda_2 + \ldots + \overline{\iota_2}\lambda_{n-1} = \gamma_3$$

yielding

$$\begin{aligned} 0 &= \beta_0\overline{\beta_0} + \beta_0\frac{\gamma_2}{\gamma_0} + \overline{\beta_0}\frac{\gamma_3}{\overline{\gamma_0}} + \gamma_4 \\ \Leftrightarrow 0 &= \beta_0\overline{\beta_0} + \beta_0\frac{\gamma_2}{\gamma_0} + \overline{\beta_0}\frac{\overline{\gamma_2}}{\overline{\gamma_0}} + \gamma_4 \\ \Leftrightarrow 0 &= \beta_0\overline{\beta_0} + \beta_0\frac{\gamma_2}{\gamma_0} + \overline{\beta_0}\overline{\left(\frac{\gamma_2}{\gamma_0}\right)} + \gamma_4. \end{aligned}$$

We set $\tilde{\gamma} \coloneqq \frac{\gamma_2}{\gamma_0}$ such that

$$\begin{aligned} 0 &= \beta_0\overline{\beta_0} + \beta_0\frac{\gamma_2}{\gamma_0} + \overline{\beta_0}\overline{\left(\frac{\gamma_2}{\gamma_0}\right)} + \gamma_4 \\ \Leftrightarrow 0 &= \beta_0\overline{\beta_0} + \beta_0\tilde{\gamma} + \overline{\beta_0}\overline{\tilde{\gamma}} + \gamma_4 \\ \Leftrightarrow 0 &= \beta_0\overline{\beta_0} + \beta_0\tilde{\gamma} + \overline{\beta_0}\overline{\tilde{\gamma}} + \tilde{\gamma}\overline{\tilde{\gamma}} - \tilde{\gamma}\overline{\tilde{\gamma}} + \gamma_4 \\ \Leftrightarrow 0 &= (\beta_0 + \overline{\tilde{\gamma}})(\overline{\beta_0} + \tilde{\gamma}) - \tilde{\gamma}\overline{\tilde{\gamma}} + \gamma_4 \\ \Leftrightarrow 0 &= (\beta_0 + \overline{\tilde{\gamma}})\overline{(\beta_0 + \tilde{\gamma})} - \tilde{\gamma}\overline{\tilde{\gamma}} + \gamma_4. \end{aligned}$$

Hence, we have to solve $(\beta_0 + \overline{\tilde{\gamma}})\overline{(\beta_0 + \tilde{\gamma})} = \tilde{\gamma}\overline{\tilde{\gamma}} - \gamma_4$. Note that $\tilde{\gamma}\overline{\tilde{\gamma}} \in \mathbb{F}_0$ as $\overline{\tilde{\gamma}\overline{\tilde{\gamma}}} = \overline{\tilde{\gamma}}\tilde{\gamma}$ and $\gamma_4 \in \mathbb{F}_0$ as $\overline{\gamma_4} = \gamma_4$ and, therefore, $\tilde{\gamma}\overline{\tilde{\gamma}} - \gamma_4 \in \mathbb{F}_0$. Again using Hilbert 90 [62, Theorem 3 and 4] we can efficiently compute $x \in \mathbb{F}_{q^2}$ such that $x\overline{x} = \tilde{\gamma}\overline{\tilde{\gamma}} - \gamma_4$. Finally, we can substitute the solution to compute a suitable β as

$$x = \beta_0 + \overline{\tilde{\gamma}} \Leftrightarrow \beta_0 = x - \overline{\tilde{\gamma}} \Leftrightarrow \beta\gamma_0 = x - \overline{\tilde{\gamma}} \Leftrightarrow \beta = \frac{x - \overline{\tilde{\gamma}}}{\gamma_0}.$$

Overall, we proceed as follows to compute $\beta \in \mathbb{F}_{q^2}$ such that $T_{w_1+\beta w_2} \in \mathrm{SU}(n,q)$ starting from $w_1 = (1, \lambda_2, \ldots, \lambda_{n-1}, 0)$ and $w_2 = (0, \iota_2, \ldots, \iota_{n-1}, 1)$:

1) Compute $\gamma_1, \gamma_2, \gamma_3, \gamma_4$ based on w_1 and w_2.
2) Solve γ_0 with $\gamma_0\overline{\gamma_0} = \gamma_1$ using Hilbert 90.
3) Set $\tilde{\gamma} := \frac{\gamma_2}{\gamma_0}$.
4) Solve $x\overline{x} = \tilde{\gamma}\overline{\tilde{\gamma}} - \gamma_4$ using Hilbert 90.
5) A solution is now given by $\beta = \frac{x - \overline{\tilde{\gamma}}}{\gamma_0}$.

Using these computations, we compute $\beta \in \mathbb{F}_{q^2}$ such that $T_{w_1+\beta w_2} \in \mathrm{SU}(n,q)$ and set $L_1 := T_{w_1+\beta w_2}$. This concludes the computation of L_1.

For the computation of L_2 we first replace w_2 by $w_3 := w_2(L_1)^{-1}$. Note that all the entries of w_2 and w_3 are the same except the entry at position n. If w_3 has a zero at position n, then we return to computing $c_1 \in G^{\mathscr{L}}$. Without loss of generality we assume that $\langle w_1, w_2 \rangle \leq V_n \cap \mathrm{Fix}(c)$ where

$$w_1 = (1, 0, \ldots, 0) \qquad \text{and} \qquad w_2 = (0, \imath_2, \ldots, \imath_{n-1}, 1).$$

Similarly to the computation of L_1 we define for $(\nu_1, \ldots, \nu_{n-1}, 1) = \nu \in \mathbb{F}_{q^2}^n$ the notation

$${}_\nu T := \begin{pmatrix} 1 & 0 & 0 & \ldots & 0 & 0 \\ -\overline{\nu_{n-1}} & 1 & 0 & \ldots & 0 & 0 \\ -\overline{\nu_{n-2}} & 0 & 1 & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ -\overline{\nu_2} & 0 & 0 & \ldots & 1 & 0 \\ \nu_1 & \nu_2 & \nu_3 & \ldots & \nu_{n-1} & 1 \end{pmatrix} \in \mathrm{SL}(n, q^2).$$

Note that ${}_\nu T$ is not contained in $\mathrm{SU}(n,q)$ for all $(\nu_1, \ldots, \nu_{n-1}, 1) = \nu \in \mathbb{F}_{q^2}^n$ which is why we have to perform a few more computations again. As for L_1 we aim to compute $\beta \in \mathbb{F}_{q^2}$ such that for

$$\nu := \beta w_1 + w_2 = (\beta, \imath_2, \ldots, \imath_{n-1}, 1)$$

holds ${}_{\beta w_1+w_2}T \in \mathrm{SU}(n,q)$. Using the Gram-matrix b yields

$$b = {}_{\beta w_1+w_2}T\, b\, {}_{\beta w_1+w_2}T^* \Leftrightarrow 0 = \beta + \overline{\beta} + \underbrace{\underbrace{\imath_1\overline{\imath_{n-1}}}_{\in\mathbb{F}_0} + \underbrace{\imath_2\overline{\imath_{n-2}}}_{\in\mathbb{F}_0} + \ldots + \underbrace{\imath_{n-1}\overline{\imath_1}}_{\in\mathbb{F}_0}}_{=:\gamma_0\in\mathbb{F}_0}.$$

Since $\gamma_0 \in \mathbb{F}_0$ we compute $\gamma \in \mathbb{F}_{q^2}$ with $\gamma\overline{\gamma} = \gamma_0$ using Hilbert 90 [62, Theorem 3 and 4] resulting in $0 = \beta + \overline{\beta} + \gamma\overline{\gamma}$. A solution of $0 = \beta + \overline{\beta} + \gamma\overline{\gamma}$ is given by $\beta := \frac{-\gamma\overline{\gamma}}{2}$ if $\mathrm{char}(\mathbb{F}) \neq 2$ and otherwise a solution is given by the entry at position $(n,1)$ of $E^{\mathrm{SU}}_{n,\frac{n+1}{2}}(\gamma) \in \mathrm{SU}(n,q)$ which can be computed efficiently and concludes the computation of L_2. ◀

We continue this section by discussing the minor differences between the phases SU5), SU6) and SU7) and the phases Sp5), Sp6) and Sp7).

Remark 7.14: SU5) and SU6) [a minor modification of Sp5) and Sp6)]

Recall from Lemma 6.28 that the conjugation of symplectic transvections $E^{\mathrm{Sp}}_{j,n}(\omega_i)$ for $2 \leq j \leq n-1$ with a strong doubling element c yields an element of the form $\mathrm{diag}(a, I_{d-n'})$ with

$$a := \begin{pmatrix}
1 & \imath\omega_i c_{n-j+1,2} & \ldots & \imath\omega_i c_{n-j+1,n-1} & 0 & \imath\omega_i c_{n-j+1,n+1} & \ldots & \imath\omega_i c_{n-j+1,n'} \\
0 & 1 & \ldots & 0 & \omega_i(c^{-1})_{2,j} & 0 & \ldots & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \ldots & 1 & \omega_i(c^{-1})_{n-1,j} & 0 & \ldots & 0 \\
0 & 0 & \ldots & 0 & 1 & 0 & \ldots & 0 \\
0 & 0 & \ldots & 0 & \omega_i(c^{-1})_{n+1,j} & 1 & \ldots & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \ldots & 0 & \omega_i(c^{-1})_{n',j} & 0 & \ldots & 1
\end{pmatrix}.$$

Afterwards we compute an element $h \in H \cong \mathrm{Sp}(n,q)$ such that

$$hE_{j,n}^{\mathrm{Sp}}(\omega_i)^c = \begin{pmatrix} 1 & 0 & \dots & 0 & 0 & \omega_i(c^{-1})_{n+1,j} & \dots & -\omega_i(c^{-1})_{n',j} \\ 0 & 1 & \dots & 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & \omega_i(c^{-1})_{n+1,j} & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & \omega_i(c^{-1})_{n',j} & 0 & \dots & 1 \end{pmatrix}.$$

The conjugation of unitary transvections $E_{j,n}^{\mathrm{SU}}(\omega_i) \in H \cong \mathrm{SU}(n,q)$ for $2 \leq j \leq n-1$ with a strong doubling element c also yields an element of the form $\mathrm{diag}(a, I_{d-n'})$ with

$$a \coloneqq \begin{pmatrix} 1 & -\overline{\omega_i(c^{-1})_{n-1,j}} & \dots & -\overline{\omega_i(c^{-1})_{2,j}} & 0 & -\overline{\omega_i(c^{-1})_{n+1,j}} & \dots & -\overline{\omega_i(c^{-1})_{n',j}} \\ 0 & 1 & \dots & 0 & \omega_i(c^{-1})_{2,j} & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & \omega_i(c^{-1})_{n-1,j} & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & \omega_i(c^{-1})_{n+1,j} & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & \omega_i(c^{-1})_{n',j} & 0 & \dots & 1 \end{pmatrix}.$$

Recall that $E_{1,n}^{\mathrm{SU}}(\lambda)$ is an element of $\mathrm{SU}(n,q)$ if and only if $\lambda + \overline{\lambda} = 0$, see Remark 7.2. Thus, we can

only ensure to compute $h \in H \cong \mathrm{SU}(n,q)$ such that

$$hE_{j,n}^{\mathrm{SU}}(\omega_i)^c = \begin{pmatrix} 1 & 0 & \dots & 0 & * & -\overline{\omega_i(c^{-1})_{n+1,j}} & \dots & -\overline{\omega_i(c^{-1})_{n',j}} \\ 0 & 1 & \dots & 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & \omega_i(c^{-1})_{n+1,j} & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & \omega_i(c^{-1})_{n',j} & 0 & \dots & 1 \end{pmatrix}.$$

The non-zero entry $(hE_{j,n}^{\mathrm{SU}}(\omega_i)^c)_{1,n}$ has no impact on further computations and we simply have to eliminate this entry lastly. The same minor modification holds for Sp6). ◀

The last minor difference between the algorithm for unitary and symplectic groups can be identified in phase SU7). Note that in symplectic groups $\mathrm{Sp}(d,q)$ the degree d has to be even while for unitary groups $\mathrm{SU}(d,q)$ the degree d can also be odd. Thus, in unitary groups we additionally have to consider the case that d is odd which requires additional computations for all standard generators given in Definition 7.3.

Lemma 7.15: SU7)

Let n be even and n' be odd, i.e. $n' = d$. Then the standard generators of Definition 7.3 for $\mathrm{Sp}(n',q)$ can be computed using the elements of the set $A := Y_n \cup \{E_{j,n}^{\mathrm{SU}}(\omega_i), E_{n,j}^{\mathrm{SU}}(\omega_i) \mid n+1 \le j \le n', 1 \le i \le f\}$.

Proof. All standard generators except z_1^{SU} and z_5^{SU} are already contained in A. Note that $E_{1,\frac{n+1}{2}}^{\mathrm{SU}}(1) \in \{E_{j,n}^{\mathrm{SU}}(\omega_i), E_{n,j}^{\mathrm{SU}}(\omega_i) \mid \text{for } n+1 \le j \le n', 1 \le i \le f\}$. The standard generator z_1^{SU} can be computed as for symplectic groups in Lemma 6.32. Thus, it is only left to compute z_5^{SU} which can easily be done since we have all unitary transvections including $E_{1,\frac{n+1}{2}}^{\mathrm{SU}}(1) \in \{E_{j,n}^{\mathrm{SU}}(\omega_i), E_{n,j}^{\mathrm{SU}}(\omega_i) \mid \text{for } n+1 \le j \le n', 1 \le i \le f\}$. □

We finish this chapter by summarising the seven phases and the GoingUp algorithm for unitary groups.

Remark 7.16

Recall from our Hypothesis 7.10 that $H \leq \langle X^{\mathscr{L}} \rangle = G^{\mathscr{L}}$ is stingray embedded with $H \cong \mathrm{SU}(n,q)$ for $n < d$ and for a known base change matrix $\mathscr{L} \in \mathrm{GL}(d,q^2)$.

SU1) Construct an element $t \in H$ which has a fixed space of dimension $d-n+2$. This phase can be done as Sp1) for symplectic groups, see Lemma 6.17.

SU2) Choose random elements $a \in G^{\mathscr{L}}$ until $\tilde{g} := t^a$ satisfies $(C1)$ and $(C2)$. This phase can be done as Sp2) for symplectic groups.

SU3) Find a conjugate $g \in G^{\mathscr{L}}$ of $\tilde{g}$ which satisfies $(C1)$ and $(C2)$ and additionally fixes v_1 and v_n. A solution for unitary groups is given in Remark 7.13.

SU4) Compute base change matrices $\mathscr{L}'$ and $\mathscr{L}''$ such that $\langle H, H^g \rangle^{\mathscr{L}'\mathscr{L}''}$ is stingray embedded in $G^{\mathscr{L}\mathscr{L}'}$ and preserves the standard form. Set $c := g^{\mathscr{L}'\mathscr{L}''}$ and verify whether c satisfies $(C3)$. This phase can be done as Sp4) for symplectic groups, see Remark 6.22, 6.23 and 6.24. Proceed if c satisfies $(C3)$.

SU5) Using c, construct transvections $E^{\mathrm{SU}}_{j,n}(\omega_i)$ for $n < j \leq n'$ for $\langle H, H^c \rangle \cong \mathrm{SU}(n',q)$. This phase can be done as Sp5) for symplectic groups with a minor modification, see Remark 7.14.

SU6) Using c, construct transvections $E^{\mathrm{SU}}_{n,j}(\omega_i)$ for $n < j \leq n'$ for $\langle H, H^c \rangle \cong \mathrm{SU}(n',q)$. This phase can be done as Sp6) for symplectic groups with a minor modification, see Remark 7.14.

SU7) Using the transvections of SU5) and SU6) construct standard generators for $\langle H, H^c \rangle \cong \mathrm{SU}(n',q)$ by assembling a permutation matrix z_1^{SU} corresponding to a permutation of cycle type $(\frac{n'}{2})^2$ as in Definition 7.3. A solution for unitary groups is given in Lemma 7.15.

Note that we do not present an algorithm in pseudo-code for the GoingUp step for unitary groups but such an algorithm can easily be derived from this remark and Section 6.3. ◀

Theorem 7.17

Algorithm GoingUpSU [Alg. 39] terminates using at most N random selections and works correctly.

Proof. This can be proven as for symplectic groups, see Theorem 6.34. □

Algorithm 39: GoingUpSU

Input:
- $\langle X \rangle = G = \mathrm{SU}(d,q)$
- A base change matrix $\mathscr{L} \in \mathrm{GL}(d,q^2)$
- $\mathrm{SU}(4,q) \cong \langle Y_4 \rangle = H \leq G^{\mathscr{L}}$ stingray embedded and constructively recognised
- $N \in \mathbb{N}$

Output: **fail OR** $(\mathscr{L}', \mathfrak{S}, N')$ **where**
- $\mathscr{L}' \in \mathrm{GL}(d,q^2)$ is a base change matrix,
- $\mathfrak{S}$ is an MSLP from $X \cup Y_4$ to the standard generators of $G^{\mathscr{L}\mathscr{L}'}$ and
- $N' \in \mathbb{N}$ where $N - N'$ is the number of random selections that were used

function GoingUpSU$(G, \mathscr{L}, H, N)$

1 $n \leftarrow 4$ **AND** $\mathfrak{S} \leftarrow$ an MSLP from $X \cup Y_4$ to $X \cup Y_4$
2 **while** $n < d$ **do**
3 $n \leftarrow \min\{2 \cdot n - 2, d\}$ // Nearly double the dimension.
4 $(H, \mathscr{L}, \mathfrak{S}', N) \leftarrow$ GoingUpStepSU$(G, \mathscr{L}, H, N)$ // Remark 5.7 and 5.8 and 7.16
5 $\mathfrak{S} \leftarrow$ Compose $\mathfrak{S}$ and $\mathfrak{S}'$
6 **return** $(\mathscr{L}, \mathfrak{S}, N)$

Chapter 8

Orthogonal group

This chapter details an efficient realisation of the strategy outlined in Chapter 3 for orthogonal groups $\Omega(d,q)$ in their natural representation for $d \geq 8$ and all characteristics except $\Omega^\circ(d,q)$ for d odd and q even. If $d \leq 6$, then we refer to [22], and if $d = 7$, then we refer to [32, 59].

Recall that there are three types of (quasi-)simple orthogonal groups namely $\Omega^+(d,q)$, $\Omega^\circ(d,q)$ and $\Omega^-(d,q)$. The fact that there are three different types influences the success of the GoingDown and GoingUp algorithm for orthogonal groups which is explained in more detail in Section 8.1 and Section 8.3.

Section 8.1 describes the GoingDown algorithm for orthogonal groups. The GoingDown algorithm uses a GoingDown basic step and a final step as outlined in Chapter 3. The GoingDown basic step for orthogonal groups is identical to the GoingDown basic step for symplectic groups which is described in Section 6.1.1. Note that Section 6.1 only deals with the GoingDown algorithm for symplectic groups in odd characteristic but the algorithms described in Section 6.1 are also applicable for orthogonal groups in even characteristic. For the GoingDown final step of orthogonal groups we use methods from the DLLO algorithm [32, 59] to compute a stingray embedded base case group $\Omega(6,q)$. For the GoingUp algorithm we additionally require that the base case group $\Omega(6,q)$ is of plus type. Therefore, if the base case group $\Omega(6,q)$ is of minus type, then we repeat the final step.

In Section 8.2 we cite an algorithm which can be used for constructive recognition of $\Omega^+(6,q)$, i.e. the orthogonal base case group. As for symplectic and unitary groups we do not dive into the details

of the constructive recognition algorithm for $\Omega^+(6,q)$ and instead refer to the publication [22].

In Section 8.3 the GoingUp basic step for orthogonal groups is presented. There are a few differences and special situations compared to the other classical groups which we address in Section 8.3 but the overall solution is similar to the algorithm for symplectic groups, see Section 6.3, and for unitary groups, see Section 7.3.

As for the other classical groups we introduce a notation for specific elements of orthogonal groups. Note that we avoid orthogonal groups of minus type in all algorithms as much as possible which is why we only need to define specific elements for orthogonal groups of plus and circle type.

Definition 8.1

Let d be even. For $i,j \in \{1,\dots,d\}$, $\lambda \in \mathbb{F}_q \setminus \{0\}$ and $j \neq i$ with $i+j \neq d+1$ we set

$$E^{\mathrm{O}}_{i,j}(\lambda) := I_d + I_{i,j}(\lambda) - I_{d-j+1,d-i+1}(\lambda).$$

For d and q odd, $\lambda \in \mathbb{F}_q - \{0\}$ and $i,j \in \{1,\dots,d\}$ with $i+j \neq d+1$ and $j \neq i$ we set

$$E^{\mathrm{O}}_{i,j}(\lambda) := \begin{cases} I_d + I_{i,j}(\lambda) - I_{d-j+1,d-i+1}(\lambda), & \text{if } i,j \neq \frac{d+1}{2}, \\ I_d + I_{i,j}(\lambda) + E_{d-j+1,d-i+1}(2\lambda) + E_{n-j+1,j}(\lambda^2), & \text{if } i = \frac{d+1}{2} \text{ and} \\ I_d + I_{i,j}(\lambda) + I_{d-j+1,d-i+1}(\frac{\lambda}{2}) + E_{i,n-i+1}((\frac{\lambda}{2})^2), & \text{otherwise.} \end{cases}$$

Remark 8.2

Note that for q even and d odd $E^{\mathrm{O}}_{i,j}(\lambda)$ is not defined if $i = \frac{d+1}{2}$ or $j = \frac{d+1}{2}$. ◀

Based on the matrices of Definition 8.1 we present standard generators for orthogonal groups in odd characteristic.

Definition 8.3

Let $S^{\mathrm{O}} \subseteq \Omega(d,q)$. If q is odd, then let $\gamma \in \mathbb{F}_{q^2}$ be a primitive element, let $\alpha := \gamma^{(q+1)/2}$ and $\omega := \alpha^2$. Then S^{O} is a set of *standard generators* for $\Omega(d,q)$ if S^{O} is conjugate to the following set consisting of $2f+6$ elements if $\Omega(d,q)$ is of plus type, $2f+5$ elements if $\Omega(d,q)$ is of circle type and $2f+5$ elements if $\Omega(d,q)$ is of minus type:

- $E^{\mathrm{O}}_{1,2}(\omega_i)$ for $1 \leq i \leq f$,
- $E^{\mathrm{O}}_{2,1}(\omega_i)$ for $1 \leq i \leq f$,

- a permutation matrix z_1^{O}. If $\Omega(d,q)$ is of plus type, then z_1^{O} corresponds to the permutation $(1,d-1)(2,d)$ with entries $(z_1^{\mathrm{O}})_{1,d-1} = (z_1^{\mathrm{O}})_{d,2} = -1$. If $\Omega(d,q)$ is of circle or minus type, then z_1^{O} corresponds to the permutation $(1,d)$ with the entries $(z_1^{\mathrm{O}})_{1,d} = (z_1^{\mathrm{O}})_{d,1} = -1$.
- a transvection z_2^{O} with $z_2^{\mathrm{O}} := E_{2,d}^{\mathrm{O}}(1)$ if $\Omega(d,q)$ is of plus type, $z_2^{\mathrm{O}} := E_{1,\frac{d+1}{2}}^{\mathrm{O}}(1)$ if $\Omega(d,q)$ is of circle type and $z_2^{\mathrm{O}} := I_d + I_{1,\frac{d}{2}}(1) + I_{1,d}(1) + I_{\frac{d}{2},1}(2)$ if $\Omega(d,q)$ is of minus type.
- a diagonal matrix z_3^{O} with $z_3^{\mathrm{O}} := \mathrm{diag}(\omega,\omega,1,\ldots,1,\omega^{-1},\omega^{-1})$ if $\Omega(d,q)$ is of plus type and $z_3^{\mathrm{O}} := \mathrm{diag}(\omega^2,1,\ldots,1,\omega^{-2})$ if $\Omega(d,q)$ is of circle type. If $\Omega(d,q)$ is of minus type, then let $A := \frac{1}{2}(\gamma^{q-1} + \gamma^{-q+1})$, $B := \frac{1}{2}\alpha(\gamma^{q-1} - \gamma^{-q+1})$ and $C := \frac{1}{2}\alpha^{-1}(\gamma^{q-1} - \gamma^{-q+1})$ and set $z_3^{\mathrm{O}} := \mathrm{diag}(\omega,1,\ldots,1,A,A,1,\ldots,1,\omega^{-1}) + I_{\frac{d}{2},\frac{d}{2}+1}(C) + I_{\frac{d}{2}+1,\frac{d}{2}}(B)$.
- a permutation matrix z_4^{O}. If $\Omega(d,q)$ is of plus type, then z_4^{O} corresponds to the permutation $(1,2,\ldots,\frac{d}{2})(\frac{d}{2}+1,d,d-1,\ldots,\frac{d}{2}+2)$ and if $\frac{d}{2}$ is even, then $(z_4^{\mathrm{O}})_{\frac{d}{2},1} = (z_4^{\mathrm{O}})_{\frac{d}{2}+1,d} = -1$. If $\Omega(d,q)$ is of circle type, then z_4^{O} corresponds to the permutation $(1,2,\ldots,\frac{d-1}{2})(\frac{d+1}{2}+1,d,d-1,\ldots,\frac{d+1}{2}+2)$ and if $\frac{d-1}{2}$ is even, then $(z_4^{\mathrm{O}})_{\frac{d+1}{2}-1,1} = (z_4^{\mathrm{O}})_{\frac{d+1}{2}+1,d} = -1$. If $\Omega(d,q)$ is of minus type, then z_4^{O} corresponds to the permutation $(1,2,\ldots,\frac{d}{2}-1)(\frac{d}{2}+2,d,d-1,\ldots,\frac{d}{2}+3)$ and if $\frac{d}{2}-1$ is even, then $(z_4^{\mathrm{O}})_{\frac{d}{2}-1,1} = (z_4^{\mathrm{O}})_{\frac{d}{2}+2,d} = -1$.
- a permutation matrix z_5^{O} which corresponds to the permutation $(1,2)(d-1,d)$ with $(z_5^{\mathrm{O}})_{2,1} = (z_5^{\mathrm{O}})_{d-1,d} = -1$.
- If $\Omega(d,q)$ is of plus type, then we additionally require the diagonal matrix z_6^{O} with $z_6^{\mathrm{O}} := \mathrm{diag}(\omega,\omega^{-1},1,\ldots,1,\omega,\omega^{-1})$.

Remark 8.4

If q is even, then we have to replace some of the standard generators of Definition 8.3 as follows. For orthogonal groups of plus type there is nothing to do. For orthogonal groups of minus type we have to replace the generators z_2^{O} and z_3^{O}. As in Definition 8.3 let $\gamma \in \mathbb{F}_{q^2}$ be a primitive element and $\eta = \gamma + \gamma^q$. Then we set $z_2^{\mathrm{O}} := I_d + I_{1,\frac{d}{2}}(1) + I_{1,d}(1) + I_{\frac{d}{2},1}(\eta)$. Moreover, we set $z_3^{\mathrm{O}} := \mathrm{diag}(\omega,1,\ldots,1,1,C,1,\ldots,1,\omega^{-1}) + I_{\frac{d}{2},\frac{d}{2}+1}(A) + I_{\frac{d}{2}+1,\frac{d}{2}}(B)$ where $A := \gamma^{-1} + \gamma^{-q}$, $B := \gamma + \gamma^q$ and $C := \gamma^{-q+1} + \gamma^{q-1} + 1$. ◀

Note that the standard generators of Definition 8.3 for orthogonal groups contain the DLLO standard generators. For more details about the DLLO standard generators see [32, 59].

Lemma 8.5

Let $\Omega(d,q)$ be of plus or minus type. Then every element $E^{\mathrm{O}}_{i,j}(\lambda)$ can be written in terms of the standard generators of Definition 8.3.

Proof. The standard generators of Definition 8.3 contain the DLLO standard generators [32, 59] and, therefore, generate $\Omega(d,q)$. Hence, the elements $E^{\mathrm{O}}_{i,j}(\lambda)$ can be written in terms of the standard generators of Definition 8.3. □

Lemma 8.6

$\Omega(d,q)$ is generated by the standard generators of Definition 8.3.

Proof. This follows immediately as the DLLO standard generators [32, 59] are contained in the set of standard generators from Definition 8.3. □

Remark 8.7

In this chapter we present algorithms for constructive recognition of $\Omega(d,q)$ for $d \geq 8$ in its natural representation except $\Omega^{\circ}(d,q)$ for d odd and q even. Note that the algorithms are also valid for the orthogonal groups $\mathrm{O}(d,q)$ and $\mathrm{SO}(d,q)$ in their natural representation except that a few more standard generators must be computed in the BaseCase algorithm. The GoingDown and GoingUp algorithm can be applied analogously for $\mathrm{O}(d,q)$ and $\mathrm{SO}(d,q)$. ◀

8.1 GoingDown algorithm

In this section let $d \in \mathbb{N}$ with $d \geq 8$ and $\langle X \rangle = G = \Omega(d,q)$, except for d odd and q even. Recall from Remark 2.18 that we use $\Omega(d,q)$ if a result is independent of the type of the underlying orthogonal form. The goal of this section is the description of a GoingDown algorithm for the computation of the full descending recognition chain for orthogonal groups, i.e.

$$\Omega^{\pm}(6,q) \cong U_k \leq U_{k-1} \cong \Omega^{\pm}(8,q) \leq \ldots \leq U_1 \cong \Omega(d_1,q) \leq U_0 = G = \Omega(d,q),$$

where $d_i \leq 4\lceil \log(d_{i-1}) \rceil$ for $2 \leq i \leq k-2$. As outlined in Chapter 3 a GoingDown basic step is repeatedly used to compute the descending recognition chain

$$U_{k-1} \cong \Omega^{\pm}(8,q) \leq \ldots \leq U_1 \cong \Omega(d_1,q) \leq U_0 = G = \Omega(d,q),$$

and afterwards a final step algorithm is used to compute the last subgroup $\Omega(6,q)^{\pm} \cong U_k \leq U_{k-1}$. For orthogonal groups we use the same GoingDown basic step algorithm as for symplectic groups, i.e. Algorithm GoingDownBasicStepReciprocal [Alg. 29]. Note that we only described algorithms for symplectic groups in odd characteristic in Chapter 6 but Algorithm GoingDownBasicStepReciprocal [Alg. 29] is also applicable in even characteristics and, therefore, for orthogonal groups $\Omega(d,q)$ with q even and q odd.

For orthogonal groups we set an additional condition on the full descending recognition chain which is that the base case group is an orthogonal group of plus type which means $\Omega^{+}(6,q) \cong U_k$. Applying the final step algorithm to the terminal group $U_{k-1} \cong \Omega^{\pm}(8,q)$ yields a stingray embedded subgroup $U_k \leq U_{k-1}$ with either $U_k \cong \Omega^{+}(6,q)$ or $U_k \cong \Omega^{-}(6,q)$. If $U_k \cong \Omega^{-}(6,q)$, then we call the final step algorithm with input $U_{k-1} \cong \Omega^{\pm}(8,q)$ again.

As we use the same GoingDown basic step for orthogonal groups as for symplectic groups, we waive to recall a description of the GoingDown basic step and refer to Section 6.1.1. Nevertheless, we use Algorithm GoingDownBasicStepReciprocal [Alg. 29] to provide a pseudo-code algorithm for the computation of the descending recognition chain for orthogonal groups.

Algorithm 40: GoingDownToDim8Orthogonal

Input:
- $d \in \mathbb{N}$ with $d \geq 8$
- $\langle X \rangle = G = \Omega(d,q)$
- Q a quadratic form preserved by G and the polar form Φ of Q
- $N \in \mathbb{N}$

Output: **fail OR** $(U, \mathfrak{S}, N')$ **where**
- $U \leq G$ with $U \cong \Omega^{\pm}(8,q)$,
- $\mathfrak{S}$ is an MSLP from X to generators of U and
- $N' \in \mathbb{N}$ where $N - N'$ is the number of random selections that were used

function GoingDownToDim8Orthogonal($d, G, (Q, \Phi), N$)

$U \leftarrow G$ **AND** dim $\leftarrow d$ **AND** $\mathfrak{S} \leftarrow$ an MSLP from X to X
while dim > 8 **do**
  $(\dim, U, \mathfrak{S}', N) \leftarrow$ GoingDownBasicStepReciprocal(dim, U, Φ, N) // Remark 5.7 and 5.8
  $\mathfrak{S} \leftarrow$ Composition of $\mathfrak{S}$ and $\mathfrak{S}'$
return $(U, \mathfrak{S}, N)$

Theorem 8.8

Algorithm GoingDownToDim8Orthogonal [Alg. 40] terminates and works correctly.

Proof. Analogously to the proof of Theorem 5.13. □

Algorithm 41: GoingDownOrthogonal

Input:
- $\langle X \rangle = G = \Omega(d, q)$
- Q a quadratic form preserved by G and the polar form Φ of Q
- $N \in \mathbb{N}$

Output: **fail OR** $(U, \mathscr{L}, \mathfrak{S})$ **where**
- $U \leq G$ with $U \cong \Omega^+(6, q)$,
- $\mathscr{L} \in \mathrm{GL}(d, q)$ is a base change matrix such that $U^{\mathscr{L}}$ is stingray embedded in $G^{\mathscr{L}}$ and
- $\mathfrak{S}$ is an MSLP from X to generators of U

function GoingDownOrthogonal($G, (Q, \Phi), N$)

$(U, \mathfrak{S}_1, N) \leftarrow$ GoingDownToDim8Orthogonal($d, G, (Q, \Phi), N$) // Remark 5.7 and 5.8
$\mathscr{L}_1 \leftarrow$ ComputeBaseChangeMatrixForStingrayEmbedding(U, G)
$U \leftarrow$ InducedActionRepresentationGroup(U)
repeat
  $(U', \mathscr{L}_2, \mathfrak{S}_2, N) \leftarrow$ GoingDownFinalStepCL($U, 6, N$) // Remark 5.7 and 5.8
until $U' \cong \Omega^+(6, q)$
$\mathfrak{S} \leftarrow$ Composition of $\mathfrak{S}_1$ and $\mathfrak{S}_2$ **AND** $\mathscr{L} \leftarrow \mathscr{L}_1 \operatorname{diag}(\mathscr{L}_2, I_{d-8})$
return $(\langle \operatorname{diag}(g_1, I_{d-8}), \operatorname{diag}(g_2, I_{d-8}) \rangle, \mathscr{L}, \mathfrak{S})$ // Let $\langle g_1, g_2 \rangle = U'$

For the final step in orthogonal groups we use algorithms from the DLLO algorithm [32, 59] as

we did for special linear groups, symplectic groups and unitary groups. Again we do not provide a pseudo-code algorithm for the final step and refer to Section 5.1.3 and Section 6.1.3. We state the complete GoingDown algorithm for orthogonal groups by Algorithm GoingDownOrthogonal [Alg. 41] in pseudo-code.

Theorem 8.9

Algorithm GoingDownOrthogonal [Alg. 41] terminates and works correctly.

Proof. Analogously to the proof of Theorem 5.26. □

8.2 BaseCase algorithm

This chapter deals with constructive recognition of $H \cong \Omega(6,q)$ where H is a stingray embedded subgroup of $G = \Omega(d,q)$. Even though H could be an orthogonal group of plus or minus type we only focus on the plus type as the GoingDown algorithm of Section 8.1 only outputs a subgroup of G isomorphic to $\Omega^+(6,q)$. Since $H \cong \Omega(6,q)$, the group H is a base case group as in Definition 3.2 and, therefore, efficient constructive recognition algorithms for H are known. In the implementation of the algorithms of this thesis the algorithm used is presented in [22]. As for symplectic and unitary base case groups constructive recognition of H is based on constructive recognition of $\mathrm{SL}(2,q)$ which is why the algorithm of [22] is randomised and involves the discrete logarithm oracle. We are not discussing details of [22] and only state the main theorem about the complexity.

Theorem 8.10: [22]

There is an $\mathcal{O}(\log(q)(\log^2(q)+\mathscr{E}\log(q)+\zeta))$-time Las Vegas algorithm which constructively recognises H, with probability greater than $3/4$, when given $\langle X\rangle = H \cong \Omega^+(6,q)$ and having available an constructive recognition algorithm for $\mathrm{SL}(2,q)$ and a discrete log oracle. Here, ζ is the complexity to construct a (nearly) uniformly distributed random element of H as an MSLP in X and $\mathscr{E}$ is the complexity of constructively recognising (a homomorphic image of) $\mathrm{SL}(2,q)$.

Proof. [22]. □

8.3 GoingUp algorithm

In this section we discuss the GoingUp algorithm for orthogonal groups. As for the other classical groups a GoingUp step is repeatedly applied. Thus, the GoingUp algorithm can be implemented with minor modifications as the GoingUp algorithm for symplectic groups, see Algorithm GoingUpSp [Alg. 36] and Algorithm GoingUpOmega [Alg. 42]. The GoingUp step for orthogonal groups is similar to the GoingUp step for symplectic and unitary groups and outlined in Chapter 3. Hence, we focus on the differences between the GoingUp step for orthogonal groups and the GoingUp step for symplectic and unitary groups in this section. Our hypothesis for the remainder of this section is the following.

Hypothesis 8.11

Let $d \in \mathbb{N}$ and $\langle X \rangle = G = \Omega(d,q)$, except for d odd and q even, containing a stingray embedded subgroup $H \leq \langle X^{\mathscr{L}} \rangle = G^{\mathscr{L}}$ with $H \cong \Omega^+(n,q)$ for $n < d$ and for a known base change matrix $\mathscr{L} \in \mathrm{GL}(d,q)$. Note that n must be even. Moreover, standard generators Y_n of H are given as words in X. Let $V = \mathbb{F}_q^d$ and suppose that $\mathscr{B} = (v_1, \ldots, v_d)$ is a basis for V and let $V_n = \langle v_1, \ldots, v_n \rangle$ and $F_{d-n} = \langle v_{n+1}, \ldots, v_d \rangle$ (cf. Definition 2.7). We assume that H acts on V_n as $\Omega^+(n,q)$ and that H fixes F_{d-n} point-wise. Recall that $(\omega_1, \ldots, \omega_f)$ is an $\mathbb{F}_p$-basis for $\mathbb{F}_q$.

We start this section by stating the main theorem for orthogonal groups.

Theorem 8.12

Let $X \subseteq \mathrm{SL}(d,q)$ such that $\langle X \rangle = G = \Omega(d,q)$, except for d odd and q even, with $6 \leq n < d$ and n even and let $\mathscr{L} \in \mathrm{GL}(d,q)$ be a base change matrix. Let $Y_n^{\mathscr{L}}$ be a set of standard generators for the subgroup $\Omega^+(n,q)$ stingray embedded into $G^{\mathscr{L}}$. Furthermore, let $\mathfrak{S}$ be an SLP from X to Y_n and let $n' := \min\{2n-4, d\}$.

Then there is an algorithm that computes a base change matrix $\mathscr{L}' \in \mathrm{GL}(d,q)$ together with an SLP $\mathfrak{S}'$ from X to a set $Y_{n'}$. If $n' < d$, then $Y_{n'}$ is a set of standard generators for $\Omega^+(n',q)$ and $\langle Y_{n'}^{\mathscr{L}'} \rangle$ is stingray embedded in $G^{\mathscr{L}'}$. If $n' = d$, then $Y_{n'}$ is a set of standard generators for $G^{\mathscr{L}'}$.

As in Section 7.3 we restate the conditions for strong doubling element and the phases of a GoingUp step. Afterwards, we discuss which phases require further discussion for orthogonal groups. Recall

that the definition of a strong doubling element in special linear, symplectic and unitary groups requires the element to satisfy three conditions $(C1)$, $(C2)$ and $(C3)$ of Remark 5.36 and 5.53. Let $\tilde{g} \in G^{\mathscr{L}}$. Then $\tilde{g}$ is a strong doubling element with respect to H if $\tilde{g}$ fixes v_1 and v_n and the following three conditions hold

$(C1)$ $\dim(V_n + V_n\tilde{g}) = n'$.

$(C2)$ If $n' < d$, then $\dim(F_{d-n} + \mathrm{Fix}(\tilde{g})) = d$.

$(C3)$ Let $(v_1'', \ldots, v_{n'}'') := \mathscr{B}^{\mathscr{L}'\mathscr{L}''}$ and $c := \tilde{g}^{\mathscr{L}'\mathscr{L}''}$ for base change matrices $\mathscr{L}', \mathscr{L}''$ as computed in Remark 6.22 and Remark 6.23. The vectors $\omega_i v_j''$ and $\omega_i v_j'' c^{-\mathrm{Tr}}$ for $1 \le i \le f$ and $2 \le j \le n-1$ span the subspace $\langle v_2'', \ldots, v_{n-1}'', v_{n+1}'', \ldots, v_{n'}'' \rangle$.

Moreover, recall the seven phases of the GoingUp step for symplectic groups:

Sp1) Construct an element $t \in H$ which has a fixed space of dimension $d - n + 2$.

Sp2) Choose random elements $a \in G^{\mathscr{L}}$ until $\tilde{g} := t^a$ satisfies $(C1)$ and $(C2)$.

Sp3) Find a conjugate $g \in G^{\mathscr{L}}$ of $\tilde{g}$ which satisfies $(C1)$ and $(C2)$ and additionally fixes v_1 and v_n.

Sp4) Compute base change matrices $\mathscr{L}'$ and $\mathscr{L}''$ such that $\langle H, H^g \rangle^{\mathscr{L}'\mathscr{L}''}$ is stingray embedded in $G^{\mathscr{L}\mathscr{L}'}$ and preserves the standard form. Set $c := g^{\mathscr{L}'\mathscr{L}''}$ and verify whether c satisfies $(C3)$ of Remark 6.15.

Sp5) Using c, construct transvections $E^{\mathrm{Sp}}_{j,n}(\omega_i)$ for $n < j \le n'$ for $\langle H, H^c \rangle \cong \mathrm{Sp}(n', q)$.

Sp6) Using c, construct transvections $E^{\mathrm{Sp}}_{n,j}(\omega_i)$ for $n < j \le n'$ for $\langle H, H^c \rangle \cong \mathrm{Sp}(n', q)$.

Sp7) Using the transvections of Sp5) and Sp6) construct standard generators for $\langle H, H^c \rangle \cong \mathrm{Sp}(n', q)$ by assembling a permutation matrix z_1^{Sp} corresponding to a permutation of cycle type $(\frac{n'}{2})^2$ as in Definition 6.3.

The phases applied to the orthogonal group are denoted by O1) to O7). A summary of the seven phases for orthogonal groups is given in Remark 8.21 and the complete GoingUp algorithm for orthogonal groups is given by Algorithm GoingUpOmega [Alg. 42].

The phases O1), O2) can be performed the same way as the phases Sp1), Sp2). Note that for phase O1) we have to write $t \in H$ as a word in Y_n which has a fixed space of dimension $d - n + 4$, e.g.

$z_4^{\mathrm{O}} z_5^{\mathrm{O}}$. For the phases O4), O5), O6) and O7) there are minor differences compared to the phases Sp4), Sp5), Sp6) and Sp7) while phase O3) requires more changes and a different approach. We continue this section by discussing how phase O3) can be solved which is done in Remark 8.17. Afterwards we describe how the phases O4), O5), O6) and O7) can be carried out. A solution for phase O4) is given in Remark 8.18, a solution for the phases O5), O6) is given in Remark 8.19 and a solution for phase O7) is given in Lemma 8.20.

For orthogonal groups Theorem 8.12 states that $\dim(V_n + V_n g) = \min\{2n-4, d\}$ instead of $\min\{2n-2, d\}$ as it is the case for symplectic and unitary groups. Equivalent to $\dim(V_n + V_n g) = \min\{2n-4, d\}$ is that $\dim(V_n \cap \mathrm{Fix}\, g) \geq 4$, see Lemma 5.42 1) with a slightly modified proof. For O3) we require that $\dim(V_n \cap \mathrm{Fix}\, g) \geq 4$ which is discussed in detail in the following.

We start by observing the action of orthogonal groups on 1-dimensional subspaces. Recall that a non-zero vector $v \in V$ is singular if $Q(v) = 0$.

Lemma 8.13: [91, Lemma 11.29]

Suppose the dimension of an $\mathbb{F}$-vector space V is at least 5 and the Witt index of V is at least 2. Then for all singular points w_1, w_2 and w_3 such that $w_1, w_2 \in \langle w_3 \rangle^\perp \backslash \{w_3\}$ there is an element $g \in \Omega(V)$ which fixes w_3 and takes w_1 to w_2.

Proof. [91, Lemma 11.29]. □

Now we have to ensure, that we can find singular points satisfying the requirements of Lemma 8.13. By our assumption $\dim(V_n) \geq 6$, $\dim(V_n \cap V_n \tilde{g}) \geq 4$ and v_1 and v_n are singular vectors, see [91, Lemma 7.3]. Moreover, $v_1 \in \langle v_n \rangle^\perp$ and $v_n \in \langle v_1 \rangle^\perp$.

Lemma 8.14: [91, Lemma 11.2]

If $\mathbb{F}$ is finite and $\dim(V) \geq 3$, then $V \backslash \{0\}$ contains a singular vector.

Proof. [91, Lemma 11.2]. □

Lemma 8.15

$V_n \cap V_n \tilde{g}$ contains at least two linear independent and singular vectors.

Proof. Using Lemma 8.14 we know that $V_n \cap V_n\tilde{g}$ contains at least one singular vector $w_1 \neq 0$. Since, $\dim((V_n \cap V_n\tilde{g})\backslash\langle w_1\rangle) = 3$, we can use Lemma 8.14 again to identify a second singular element $w_2 \in (V_n \cap V_n\tilde{g})\backslash\langle w_1\rangle$. Clearly, w_1 and w_2 are linear independent. □

Remark 8.16

Note that the proofs of Lemma 8.14 and 8.15 are constructive, i.e. they provide an algorithm for computing two linear independent singular vectors $w_1, w_2 \in V_n \cap V_n\tilde{g}$. ◀

Our next aim is to show how we can use Lemma 8.13 to map two linear independent singular vectors $w_1, w_2 \in V_n \cap V_n\tilde{g}$ to v_1 and v_n.

Remark 8.17: O3)

By our assumption given in Theorem 8.12, $\dim(V_n) \geq 6$ and by the computations so far we know $\dim(V_n \cap V_n\tilde{g}) \geq 4$. Moreover, v_1 and v_n are singular vectors, $v_1 \in \langle v_n\rangle^\perp$ and $v_n \in \langle v_1\rangle^\perp$. Our aim is to compute two singular vectors $w_1, w_2 \in V_n \cap V_n\tilde{g}$ with $w_1 \in \langle v_n\rangle^\perp\backslash\{v_n\}$ and $w_2 \in \langle v_1\rangle^\perp\backslash\{v_1\}$. If $v_1 \in V_n \cap V_n\tilde{g}$ and $v_n \in V_n \cap V_n\tilde{g}$, then there is nothing to do. If $v_1 \in V_n \cap V_n\tilde{g}$ and $v_n \notin V_n \cap V_n\tilde{g}$ or vice verse, then we can proceed as in the case that $v_1, v_n \notin V_n \cap V_n\tilde{g}$. Thus, we assume in the following that $v_1, v_n \notin V_n \cap V_n\tilde{g}$.

The orthogonal complement of $\langle v_n\rangle^\perp$ in V_n can be computed in $\mathcal{O}(n^3)$. Moreover, by Lemma 2.14 $\dim(\langle v_n\rangle^\perp) \geq n-1$ and, thus, $\dim(\langle v_n\rangle^\perp \cap (V_n \cap V_n\tilde{g})) \geq 3$. By Lemma 8.14 $W := \langle v_n\rangle^\perp \cap (V_n \cap V_n\tilde{g})$ contains a singular vector. The proof of Lemma 8.14 is constructive and, hence, we continue as in the proof of [91, Lemma 11.2]. Suppose first that $\mathrm{char}(\mathbb{F}_q) = 2$. We choose $0 \neq u_1 \in W$, compute the orthogonal complement $\langle u_1\rangle^\perp$ and choose $u_2 \in W \cap \langle u_1\rangle^\perp\backslash\{u_1\}$. Since $\mathrm{char}(\mathbb{F}_q) = 2$, every element of $\mathbb{F}_q$ is a square and, thus, there are $x, y \in \mathbb{F}_q$ such that $xu_1 + yu_2 \neq 0$ and $Q(xu_1 + yu_2) = x^2Q(u_1) + y^2Q(u_2) = 0$. Suppose now that the characteristic of $\mathbb{F}_q$ is odd. Then we choose non-zero vectors $u_1, u_2, u_3 \in W$ with $u_2 \in \langle u_1\rangle^\perp$ and $u_3 \in \langle u_1, u_2\rangle^\perp$. We suppose that u_1, u_2, u_3 are non-singular. Using [91, Lemma 11.1] we can find $x, y \in \mathbb{F}_q$ such that $x^2Q(u_1) + y^2Q(u_2) = -Q(u_3)$ and, thus, $xu_1 + yu_2 + u_3$ is singular.

Hence, we can identify a singular vector $w_1 \in W$. Moreover, we assume that the first entry of w_1 is non-zero such that we can rescale the entry in the first position to 1 and, thus, $w_1 = (1, w_{1,2}, w_{1,3}, \ldots, w_{1,n})$. Overall, $v_1, w_1 \in \langle v_n\rangle^\perp\backslash\{v_n\}$ and the vectors v_1, w_1, v_n are singular. By

Lemma 8.13 there is an element $L_1 \in H$ which maps v_1 to w_1 and fixes v_n such as

$$L_1 := \begin{pmatrix} 1 & w_{1,2} & w_{1,3} & \dots & w_{1,n-1} & w_{1,n} \\ 0 & 1 & 0 & \dots & 0 & -w_{1,n-1} \\ 0 & 0 & 1 & \dots & 0 & -w_{1,n-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & -w_{1,2} \\ 0 & 0 & 0 & \dots & 0 & 1 \end{pmatrix} \in \Omega^+(n,q).$$

Again we continue as in the proof of [91, Lemma 11.2] to identify a singular vector $w_2 \in V_n \cap V_n(L_1 \tilde{g} L_1^{-1})$ such that $w_2 \in \langle v_1 \rangle^\perp \setminus \{v_1\}$. Note that $\dim(\langle v_1 \rangle^\perp \cap (V_n \cap V_n(L_1 \tilde{g} L_1^{-1}))) \geq 3$. Analogously to w_1, we assume that the last entry of w_2 is non-zero such that we rescale the entry in the last position to 1 and, thus, $w_2 = (w_{2,1}, w_{2,2}, \dots, w_{2,n-1}, 1)$. Overall, $v_n, w_2 \in \langle v_1 \rangle^\perp \setminus \{v_1\}$ and the vectors v_1, w_2, v_n are singular. By Lemma 8.13 there is an element $L_2 \in H$ which maps v_n to w_2 and fixes v_1 such as

$$L_2 := \begin{pmatrix} 1 & 0 & 0 & \dots & 0 & 0 \\ -w_{2,n-1} & 1 & 0 & \dots & 0 & 0 \\ -w_{2,n-2} & 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ -w_{2,2} & 0 & 0 & \dots & 1 & 0 \\ w_{2,1} & w_{2,2} & w_{2,3} & \dots & w_{2,n-1} & 1 \end{pmatrix} \in \Omega^+(n,q).$$

◀

Remark 8.18: O4) [a minor modification of Sp4)]

Sp4) aims to compute a base change matrix $\mathscr{L}' \in \mathrm{GL}(d,q)$ such that $\langle H, H^g \rangle^{\mathscr{L}'}$ is stingray embedded in $G^{\mathscr{L}\mathscr{L}'}$ and, if it is possible, then a second base change matrix $\mathscr{L}'' \in \mathrm{GL}(d,q)$ such that the top left $(n' \times n')$-block of $\langle H, H^g \rangle^{\mathscr{L}'\mathscr{L}''}$ preserves the standard form. Note that if the computation of $\mathscr{L}''$ fails, then we can already conclude that $\langle H, H^g \rangle^{\mathscr{L}'}$ is not isomorphic to $\mathrm{Sp}(n',q)$ and, thus, we return to phase Sp2).

In orthogonal groups we proceed as described above and as in Remark 6.22 and 6.23 except that

for the computation of the second base change matrix $\mathscr{L}'' \in \mathrm{GL}(d,q)$ we compute the preserved quadratic form of $\langle H, H^g \rangle^{\mathscr{L}'}$ and a base change matrix $\mathscr{L}'' \in \mathrm{GL}(d,q)$ such that the standard quadratic form is preserved. This also ensures the correctness of this phase for even characteristic. Moreover, if $n' < d$, then we only proceed if $\langle H, H^g \rangle^{\mathscr{L}'\mathscr{L}''}$ preserves the standard quadratic form of an orthogonal group of plus type. If $\langle H, H^g \rangle^{\mathscr{L}'\mathscr{L}''}$ preserves the standard quadratic form of an orthogonal group of minus type, then we also return to phase O2). ◀

Remark 8.19: O5) and O6) [a minor modification of Sp5) and Sp6)]

The phases O5) and O6) can be performed in a similar way as the steps Sp5) and Sp6). The results of Lemma 6.28 and Lemma 6.30 are also valid for orthogonal groups with $\iota = 1$ in all cases and the proof can be done analogously. We also continue as in Remark 6.29 except that we never try to eliminate the entry at position $(1,n)$ as matrices of the form $I_n + I_{1,n}(\lambda)$ are not contained in $\Omega^+(n,q)$. Note that this has no consequences on the further computations of phases O5) and O6) as $\langle H, H^g \rangle^{\mathscr{L}'\mathscr{L}''}$ preserves the standard quadratic form by Remark 8.18 and, thus, the entry at position $(1,n)$ automatically becomes zero. ◀

Lemma 8.20: O7)

Let n be even. Then the standard generators of Definition 8.3 for $\Omega(n',q)$ can be computed using the elements of the set $A := Y_n \cup \{E^{\mathrm{O}}_{j,n}(\omega_i), E^{\mathrm{O}}_{n,j}(\omega_i) \mid n+1 \le j \le n', 1 \le i \le f\}$.

Proof. If $n' < d$, then $\langle H, H^g \rangle^{\mathscr{L}'\mathscr{L}''} \cong \Omega^+(n',q)$ and we only need to assemble the standard generator z^{O}_4 which can be done as in Lemma 6.32. If $n' = d$ and $G = \Omega^+(d,q)$, then we can also proceed as in Lemma 6.32. If $n' = d$ and G is of circle or minus type, then we can proceed as in Lemma 6.32 to assemble the standard generator z^{O}_4. Afterwards, we can compute all Siegel transformations and, thus, the remaining standard generators. □

We finish this chapter by summarising the seven phases and the GoingUp algorithm for orthogonal groups.

Remark 8.21

Recall from our Hypothesis 8.11 that $H \leq \langle X^{\mathscr{L}} \rangle = G^{\mathscr{L}}$ is stingray embedded with $H \cong \Omega^+(n,q)$ for $n < d$ and for a known base change matrix $\mathscr{L} \in \mathrm{GL}(d,q)$.

O1) Construct an element $t \in H$ which has a fixed space of dimension $d-n+4$.

O2) Choose random elements $a \in G^{\mathscr{L}}$ until $\tilde{g} := t^a$ satisfies $(C1)$ and $(C2)$. This phase can be done as Sp2) for symplectic groups.

O3) Find a conjugate $g \in G^{\mathscr{L}}$ of $\tilde{g}$ which satisfies $(C1)$ and $(C2)$ and additionally fixes v_1 and v_n. A solution for orthogonal groups is given in Remark 8.17.

O4) Compute a base change matrix $\mathscr{L}'$ such that $\langle H, H^g \rangle^{\mathscr{L}'}$ is stingray embedded in $G^{\mathscr{L}\mathscr{L}'}$. Set $c := g^{\mathscr{L}'}$ and verify whether c satisfies $(C3)$. This phase can be done as Sp4) for symplectic groups, see Remark 6.22, 6.23 and 6.24 except that we use quadratic forms and if $n' < d$, then additionally $\langle H, H^g \rangle^{\mathscr{L}'} \cong \Omega^+(n',q)$, see Remark 8.18. Proceed if c satisfies $(C3)$.

O5) Using c, construct transvections $E^{\mathrm{O}}_{j,n}(\omega_i)$ for $n < j \leq n'$ for $\langle H, H^c \rangle \cong \Omega(n',q)$. This phase can be done as Sp5) for symplectic groups with a minor modification, see Remark 8.19.

O6) Using c, construct transvections $E^{\mathrm{O}}_{n,j}(\omega_i)$ for $n < j \leq n'$ for $\langle H, H^c \rangle \cong \Omega(n',q)$. This phase can be done as Sp6) for symplectic groups with a minor modification, see Remark 8.19.

O7) Using the transvections of O5) and O6) construct standard generators for $\langle H, H^c \rangle \cong \Omega(n',q)$ by assembling standard generators as in Definition 7.3. A solution for orthogonal groups is given in Lemma 8.20.

Note that we do not present an algorithm in pseudo-code for the GoingUp step for orthogonal groups but such an algorithm can easily be derived from this remark and Section 6.3. ◀

Algorithm 42: GoingUpOmega

Input:
- $\langle X \rangle = G = \Omega(d,q)$
- A base change matrix $\mathscr{L} \in \mathrm{GL}(d,q)$
- $\Omega^+(6,q) \cong \langle Y_6 \rangle = H \leq G^{\mathscr{L}}$ stingray embedded and constructively recognised
- $N \in \mathbb{N}$

Output: **fail OR** $(\mathscr{L}', \mathfrak{S}, N')$ **where**
- $\mathscr{L}' \in \mathrm{GL}(d,q)$ is a base change matrix,
- $\mathfrak{S}$ is an MSLP from $X \cup Y_6$ to the standard generators of $G^{\mathscr{L}\mathscr{L}'}$ and
- $N' \in \mathbb{N}$ where $N - N'$ is the number of random selections that were used

function GoingUpOmega($G, \mathscr{L}, H, N$)

1 $n \leftarrow 6$ **AND** $\mathfrak{S} \leftarrow$ an MSLP from $X \cup Y_6$ to $X \cup Y_6$
2 **while** $n < d$ **do**
3 $n \leftarrow \min\{2 \cdot n - 4, d\}$ // Nearly double the dimension.
4 $(H, \mathscr{L}, \mathfrak{S}', N) \leftarrow$ GoingUpStepOmega$(G, \mathscr{L}, H, N)$ // Remark 5.7 and 5.8 and 8.21
5 $\mathfrak{S} \leftarrow$ Compose $\mathfrak{S}$ and $\mathfrak{S}'$
6 **return** $(\mathscr{L}, \mathfrak{S}, N)$

Theorem 8.22

Algorithm GoingUpOmega [Alg. 42] terminates using at most N random selections and works correctly.

Proof. This can be proven as for symplectic groups, see Theorem 6.34. □

Chapter 9

GoingUp using involutions

Even though the GoingUp step presented in this thesis is very fast and efficient in practice for all classical groups, the fact that the algorithm is only applicable to such groups in their natural representations is a disadvantage. Therefore, we introduce and prove the correctness of an alternative GoingUp step based on involutions for all classical groups in odd characteristic in this chapter which in practice appears to be usable in gray box settings. In this chapter we prove that the alternative GoingUp step is also correct in all classical groups of odd characteristic in their natural representation but we do not show the correctness in gray box situations. In the following the GoingUp step introduced in Chapter 5 to Chapter 8 is called GoingUp *algorithm based on linear algebra* and the algorithm of this chapter is called GoingUp *algorithm based on involutions*. Involutions can be used for constructive recognition of classical groups in gray and black box situations as shown in [33]. Moreover, the length of the output MSLPs produced by the GoingUp algorithm based on linear algebra is larger than the length of the MSLPs produced by the GoingUp algorithm based on involutions. Note that the GoingUp algorithm based on involutions requires q to be odd which we assume for the remainder of this chapter.

We start this chapter by describing the framework used in the Leedham-Green and O'Brien (LGO) algorithm [59] and discuss how this setting differs from the framework after using the GoingDown and BaseCase algorithm of this thesis. Let $G = \mathrm{CL}(d,q)$. The main idea of the LGO constructive recognition algorithm consists of the computation of an involution $i \in G$ and the centraliser of the

involution in $C_G(i)$ which has the following form after a suitable base change.

$$C_G(i) = \left(\begin{array}{c|c} U_1 & 0 \\ \hline 0 & U_2 \end{array}\right) \leq G,$$

where $U_1 = \mathrm{CL}(d_1, q)$ and $U_2 = \mathrm{CL}(d_2, q)$ are classical groups of the same type as G and $d_1 + d_2 = d$. Afterwards, the LGO algorithm is applied to U_1 and U_2 recursively, resulting in a binary tree structure as in Figure 9.1.

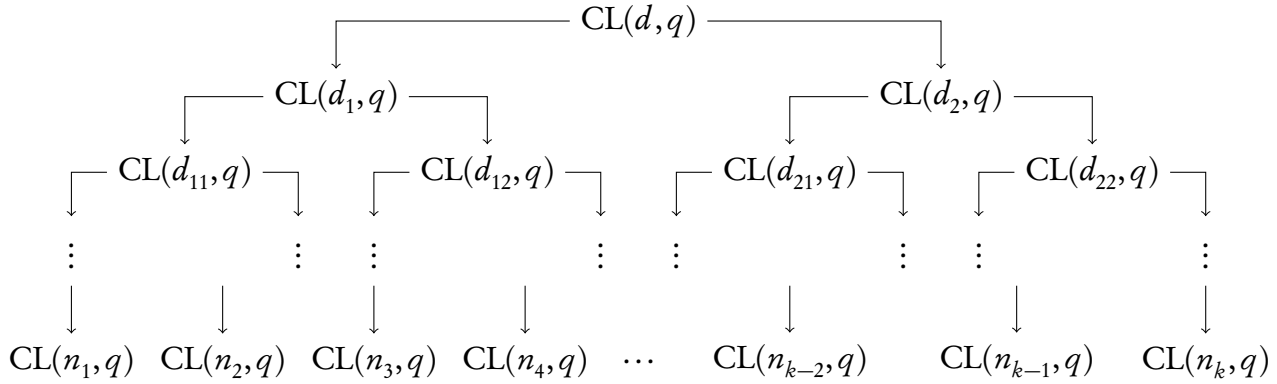


Figure 9.1: Simplified graphical visualisation of the LGO algorithm.

In Figure 9.1 a constructive recognition algorithm is applied on the groups $\mathrm{CL}(n_1, q), \ldots, \mathrm{CL}(n_k, q)$ as these groups are base case groups as described in 3.2. Except at leaf groups of Figure 9.1 we can assume that we have the setting

$$\left(\begin{array}{c|c} U_1 & 0 \\ \hline 0 & U_2 \end{array}\right) \leq G$$

where $U_1 = \mathrm{CL}(d_1, q)$, $U_2 = \mathrm{CL}(d_2, q)$ and $d_1 + d_2 = d$ and by recursion we can assume that standard generators of U_1 and U_2 are known. In this situation a *glueing* algorithm is applied to G, U_1 and U_2 to compute standard generators of G. The solution of the LGO constructive recognition algorithm is reminiscent of a paradigm known as "divide-and-conquer" in the area of computer science.

Remark 9.1

After describing the main idea of the LGO algorithm we summarise the details of the framework for the *glueing* algorithm. We require

- a group $G = \mathrm{CL}(d, q)$,
- $U_1, U_2 \leq G$ with $U_1 \cong \mathrm{CL}(d_1, q)$, $U_2 \cong \mathrm{CL}(d_2, q)$ all of the same type as G and $d_1 + d_2 = d$,

- U_1 and U_2 are constructively recognised and
- U_1 and U_2 commute, i.e. there is a base change such that

$$\left(\begin{array}{c|c} U_1 & 0 \\ \hline 0 & U_2 \end{array}\right) \leq G.$$

◀

After summarising some details of the LGO algorithm we now deal with the setting using the GoingDown and BaseCase algorithm of this thesis. The GoingDown algorithm constructs a descending recognition chain which is visualised in Figure 9.2 similar to Figure 9.1 for the LGO algorithm.

$$\begin{array}{c} \mathrm{CL}(d,q) \\ \downarrow \\ \mathrm{CL}(d_1,q) \\ \downarrow \\ \vdots \\ \downarrow \\ \mathrm{CL}(d_{k-2},q) \\ \downarrow \\ \mathrm{CL}(d_{k-1},q) \\ \downarrow \\ \mathrm{CL}(d_k,q) \end{array}$$

Figure 9.2: Simplified graphical visualisation of the GoingDown algorithm.

In Figure 9.2 a vertical version of a full descending recognition chain is given. Comparing Figure 9.1 and Figure 9.2 we notice that after having applied the GoingDown algorithm of this thesis, a less useful structure compared to the LGO algorithm is obtained. However, the trade-off is that we reach a base case group much quicker and that we only need to recognise one base case group constructively.

The goal of this chapter is to use the *glueing* algorithm of the LGO algorithm as an GoingUp step. Since the setting after the GoingDown algorithm is different compared to the setting of the LGO algorithm we must perform additional computations such that the LGO *glueing* algorithm becomes applicable. From the setting of the LGO algorithm as described in Remark 9.1 we have

$U_1 \cong \mathrm{CL}(d_1, q)$ as we only constructed the standard generators of one subgroup of G. Hence, we are missing

- U_2 with $U_2 \cong \mathrm{CL}(d_2, q)$ such that U_1 and U_2 commute,
- an MSLP which evaluates from the generators of G to the standard generators of U_2 and
- a stingray embedded group $\tilde{G} \leq G$ such that $U_1 \times U_2 \leq \tilde{G}$ and $\tilde{G} \cong \mathrm{CL}(d_1 + d_2, q)$.

We proceed as follows to find the missing groups and MSLPs evaluating to generators of these groups. We assume that $\mathrm{CL}(n, q) \cong H \leq \langle X \rangle = G = \mathrm{CL}(d, q)$ and that standard generators of H can be written as words in X. The group H takes on the role of U_1. First, we compute a group $H_B \leq G$ with $H_B \cong \mathrm{CL}(2n, q)$ and $H \leq H_B$. Secondly, we compute an element $u \in H_B$ such that H and H^u commute. The group H^u takes on the role of U_2. Moreover, standard generators of H^u can be written as words in X as H^u is conjugate to H. We summarise this in Table 9.1.

Requirements of LGO *glueing* algorithm	Computed in this chapter
$\mathrm{CL}(d_1, q) \cong U_1 \leq G$	$\mathrm{CL}(n, q) \cong H \leq H_B$
U_1 is constructively recognised	H is constructively recognised
$\mathrm{CL}(d_2, q) \cong U_2 \leq G$	$\mathrm{CL}(n, q) \cong H^u \leq H_B$
U_2 is constructively recognised	H^u is constructively recognised
$G \cong \mathrm{CL}(d_1 + d_2, q)$	$H_B \cong \mathrm{CL}(2n, q)$
U_1 and U_2 commute	H and H^u commute

Table 9.1: Comparison between the requirements of the LGO *glueing* algorithm and the computations of this chapter.

The first section of this chapter deals with the computations of H_B and $u \in H_B$ which are needed for the LGO *glueing* algorithm. These computations and the LGO *glueing* algorithm are then combined in a single algorithm which is used as the GoingUp step based on involutions. The second section uses the GoingUp step based on involutions iteratively to construct standard generators of the input group similarly to Algorithm GoingUp [Alg. 28] based on linear algebra in Section 5.3.6. The GoingUp step of this chapter uses methods from [32] and [59].

9.1 Overview of the GoingUp step

In this section we present a GoingUp step based on involutions and the *glueing* algorithm of the LGO algorithm. For the remainder of this section let $\mathrm{char}(\mathbb{F}) \neq 2$. We start this section by stating

the setting and the main theorem. Similarly to the GoingUp algorithm based on linear algebra we prove the correctness of the GoingUp step based on involutions by describing and proving the correctness of an algorithm. The hypothesis of this section is similar to Hypothesis 5.34 of the GoingUp step based on linear algebra except that we deal with all classical groups at the same time by using $\mathrm{CL}(d,q)$ instead of $\mathrm{SL}(d,q)$, $\mathrm{Sp}(d,q)$, $\mathrm{SU}(d,q)$ or $\Omega(d,q)$.

Hypothesis 9.2

Let $H \leq \langle X^{\mathscr{L}} \rangle = G^{\mathscr{L}}$ where $G = \mathrm{CL}(d,q)$ and $H \cong \mathrm{CL}(n,q)$ for n even, $2n \leq d$ and H is stingray embedded in $G^{\mathscr{L}}$ for a known base change matrix $\mathscr{L} \in \mathrm{GL}(d,q)$. Moreover, standard generators Y_n of H are given as words in X and let $V = \mathbb{F}_q^d$. Let $\mathscr{B} = (v_1,\ldots,v_d)$ be a basis of V, let $V_n = \langle v_1,\ldots,v_n \rangle$ and $F_{d-n} = \langle v_{n+1},\ldots,v_d \rangle$. We assume that H acts on V_n as $\mathrm{CL}(n,q)$ and that H fixes F_{d-n} point-wise. Additionally, if CL is an orthogonal group, then we assume $\mathrm{CL}(n,q) = \Omega^+(n,q)$.

The main theorem for the GoingUp step based on involutions is the following.

Theorem 9.3

Let $G = \mathrm{CL}(d,q)$ and $X \subseteq G$ such that $\langle X \rangle = G$. Let $2 \leq n < d$ with n even and $2n \leq d$ and let $\mathscr{L} \in \mathrm{GL}(d,q)$ be a base change matrix. Let $Y_n^{\mathscr{L}}$ be a set of standard generators for $H \cong \mathrm{CL}(n,q)$ stingray embedded into $G^{\mathscr{L}}$ of the same type as G. Furthermore, let $\mathfrak{S}$ be an SLP from X to Y_n. Then there is an algorithm that computes a base change matrix $\mathscr{L}' \in \mathrm{GL}(d,q)$ together with an SLP $\mathfrak{S}'$ from X to a set Y_{2n}, which is a set of standard generators for $H_B \cong \mathrm{CL}(2n,q)$ of the same type as G and $\langle Y_{2n}^{\mathscr{L}'} \rangle$ is stingray embedded in $G^{\mathscr{L}'}$.

Similarly to the GoingUp step based on linear algebra, the algorithm we describe for proving Theorem 9.3 consists of seven phases called I1) to I7). Before stating the phases we discuss the idea of GoingUp step based on involutions in the next remark.

Remark 9.4

Let $\langle X \rangle = G = \mathrm{CL}(d,q)$ be a classical group in its natural representation such that d is greater or equal to the value given in Table 3.1 for each classical group. For the GoingUp algorithm based on involutions, we assume that standard generators of a stingray embedded subgroup $H \leq G$ with

$H \cong \mathrm{CL}(n,q)$ of the same type as G can be written as words in X. Using the standard generators of H we construct an element $g \in G^{\mathscr{L}}$ with the following three properties:

$(C1)$ $\dim(V_n + V_n g) = 2n$,

$(C2)$ $\dim(F_{d-n} + \mathrm{Fix}(g)) = d$ and

$(C3)$ $H_B := \langle H, H^g \rangle \cong \mathrm{CL}(2n,q)$ is of the same type as G.

Using $(C1)$ and $(C2)$ we prove that $H_B \cong \mathrm{CL}(2n,q)$ is stingray embedded in G using a suitable base change matrix. In this setting we compute an element $u \in H_B$ and the element u can be used to compute $H_2 := H^u \leq H_B$. In the course of this chapter we prove that H and H^u commute. Therefore, we can call a *glueing* algorithm of [59] on H, H_2 and H_B to construct standard generators for H_B. Starting from a subgroup of G isomorphic to a base case group this yields an ascending chain. Note that we are doubling the degree from n to $2n$ using g and u. ◀

Definition 9.5

Assume the setting as described in Hypothesis 9.2.

1) An element $g \in G^{\mathscr{L}}$ satisfying $(C1)$, $(C2)$ and $(C3)$ is an *honest doubling element*.
2) Let $H_B \leq G^{\mathscr{L}}$ with $H_B \cong \mathrm{CL}(2n,q)$ and H_B is stingray embedded in $G^{\mathscr{L}'}$ for some base change matrix $\mathscr{L}' \in \mathrm{GL}(d,q)$. Let $i \in H_B$ be an involution, i.e. $i^2 = I_d$. Moreover, we assume that the (-1)-eigenspace of i is V_n. An element $u \in H_B$ is a *swapping element* if there is a subspace W of the 1-eigenspace of i such that $W^u = V_n$ and $V_n^u = W$. Note that $W \cap V_n = \{0\}$ since W is a subspace of the 1-eigenspace of i.

Remark 9.6

The next sections can roughly be summarised as follows.

1) Construct an element $g \in G^{\mathscr{L}}$ satisfying the properties $(C1)$, $(C2)$ and $(C3)$, i.e. an honest doubling element as in Definition 9.5. This is achieved by random selection of elements of $G^{\mathscr{L}}$ and discussed in Section 9.2.
2) Construct a new base change matrix $\mathscr{L}'$ such that $\langle H, H^g \rangle^{\mathscr{L}'}$ is stingray embedded in $G^{\mathscr{L}\mathscr{L}'}$ which is possible because of the properties $(C1)$ and $(C2)$. The computation of $\mathscr{L}'$ is deterministic and discussed in Section 9.3.
3) Construct a swapping element $u \in \langle H, H^g \rangle$. This step involves Bray's algorithm [18] and is,

therefore, randomised. The computation of u is discussed in Section 9.4.

4) After 3) we use the LGO *glueing* algorithm from [59] to construct standard generators for $\langle H, H^g \rangle$. Since the LGO *glueing* algorithm is randomised, this step is also randomised. Note that we do not discuss the *glueing* algorithm from the LGO algorithm in this thesis and instead refer to [59]. ◀

In each of the following sections the task described in Remark 9.6 is divided into more phases which are discussed in detail leading to the seven phases I1) to I7). Finally, in Section 9.5 the phases I1) to I7) are combined into a single algorithm for the GoingUp step based on involutions which proves Theorem 9.3.

9.2 Construction of a dimension doubling element

In this section we compute an honest doubling element, i.e. an element $g \in G^{\mathscr{L}}$ satisfying $(C1)$, $(C2)$ and $(C3)$. The results of this section finally lead to Algorithm ComputeHonestDoublingElement [Alg. 43]. Recall our setting from Hypothesis 9.2. We start with a brief summary of the phases of this section.

Remark 9.7

I1) Construct an element $t \in H$ which has a fixed space of dimension $d-n$. This can be easily achieved as we have standard generators Y_n for H.

I2) Choose random elements $a \in G^{\mathscr{L}}$ until $g := t^a$ satisfies the three conditions $(C1)$, $(C2)$ and $(C3)$ of Remark 9.4, i.e. until g is an honest doubling element. If g is an honest doubling element, then set $H_B := \langle H, H^g \rangle$. ◀

Note that the computation of g is a randomised process as we compute random elements $a \in G^{\mathscr{L}}$ and test whether t^a satisfies $(C1)$, $(C2)$ and $(C3)$ of Remark 9.4. We do not analyse the probability that t^a is an honest doubling element in this thesis and refer to future publications.

We start with the computation of an element $t \in H$ which has a fixed space of dimension $d-n$, i.e. how phase I1) can be carried out. This is easily done with the standard generators Y_n of H already found. The construction of an element $t \in H$ with fixed space of dimension $d-n$ is displayed

exemplarily for the special linear groups in the next lemma. Note that the construction of such an element can easily be done in special linear groups and symplectic groups since these groups contain ppd$(d,q;n)$-elements which have a fixed space of dimension $d-n$ [74, 80]. For unitary and orthogonal groups we choose ppd$(d,q;e_1)$- and ppd$(d,q;e_2)$-elements such that $e_1+e_2=n$ and continue with both these elements. Note that we can find such elements, see [74, 80].

Lemma 9.8

Set

$$t := \begin{cases} (E_{1,2}(1)^{-1})^{z_1}E_{1,2}(1), & \text{if } n=2, \\ E_{1,2}(1)z_1, & \text{if } n>2 \text{ and } p \text{ is even}, \\ z_1, & \text{if } n>2 \text{ and } p \text{ is odd}. \end{cases}$$

Then t has a fixed space of dimension $d-n$.

Proof. The fixed space is in all cases $\langle e_{n+1},\dots,e_d\rangle$ and, therefore, has dimension $d-n$. □

As in the GoingUp approach based on linear algebra, the element $t\in H$ is constructed as an MSLP in the given standard generators Y_n of $H\cong \mathrm{CL}(n,q)$, but can at the same time be written down explicitly as an element of $G^{\mathscr{L}}$ at a cost of $\mathcal{O}(d^2)$.

Now that we have an element $t\in H$ with fixed space of dimension $d-n$, we select random elements $a\in G^{\mathscr{L}}$ and test whether $g:=t^a$ satisfies the three conditions $(C1)$, $(C2)$ and $(C3)$ of Remark 9.4. We repeat this until g has the required properties. If g is an honest doubling element, then we set $H_B:=\langle H,H^g\rangle$ and $V_{2n}=V_n+V_ng$. The next lemma summarises some properties of g.

Lemma 9.9

Let t be as in Lemma 9.8 and for random $a\in G^{\mathscr{L}}$ let $g:=t^a$ be an honest doubling element. Then

1) $\dim(V_n\cap V_ng)=0$,
2) $\dim(V_n\cap \mathrm{Fix}(g))=0$,
3) V_{2n} is invariant under the action of g and
4) $\dim(F_{d-n}\cap \mathrm{Fix}(g))=d-2n$.

Proof. The proof is similar to the proof of Lemma 5.42 but added for completeness of this chapter.

1) $\dim(V_n \cap V_n g) = \dim(V_n) + \dim(V_n g) - \dim(V_n + V_n g) = n + n - 2n = 0$.

2) Let $v \in V_n \cap \mathrm{Fix}(g)$. Then $v \in V_n g$ and $v \in V_n g \cap \mathrm{Fix}(g)$ which implies $V_n \cap \mathrm{Fix}(g) \subseteq V_n \cap V_n g$. Since $\dim(V_n \cap V_n g) = 0$, the claim follows.

3) We construct a basis of V_{2n} which shows clearly that V_{2n} is invariant under the action of g. A basis of V_{2n} consists of $2n$ elements since $\dim(V_{2n}) = 2n$ by $(C1)$. First notice that

$$\begin{aligned}\dim(V_n + \mathrm{Fix}(g)) &= \dim(V_n) + \dim(\mathrm{Fix}(g)) - \dim(V_n \cap \mathrm{Fix}(g)) \\ &= n + (d - n) = d\end{aligned}$$

by 2) and, therefore, $V_n + \mathrm{Fix}(g) = V$. Since $V_n \leq V_{2n}$, it follows that $V_{2n} + \mathrm{Fix}(c) = V$ which implies that

$$\begin{aligned}\dim(V_{2n} \cap \mathrm{Fix}(g)) &= \dim(V_{2n}) + \dim(\mathrm{Fix}(g)) - \dim(V_{2n} + \mathrm{Fix}(g)) \\ &= 2n + d - n - d = n.\end{aligned}$$

Since $\dim(V_n \cap (V_{2n} \cap \mathrm{Fix}(g)) = \dim(V_n \cap \mathrm{Fix}(g)) = 0$,

$$\begin{aligned}\dim(V_n + (V_{2n} \cap \mathrm{Fix}(g))) &= \dim(V_n) + \dim(V_{2n} \cap \mathrm{Fix}(g)) - \dim(V_n \cap (V_{2n} \cap \mathrm{Fix}(g))) \\ &= n + n - 0 = 2n\end{aligned}$$

and, since $V_n + (V_{2n} \cap \mathrm{Fix}(g)) \leq V_{2n}$, it follows that $V_n + (V_{2n} \cap \mathrm{Fix}(g)) = V_{2n}$. Therefore, we can choose a basis of V_{2n} as follows:

- Select n vectors $(v_1, \ldots, v_n) \in V_n^n$ as a basis of V_n.
- Choose n vectors from $V_{2n} \cap \mathrm{Fix}(g)$ to extend this to a basis of V_{2n}.

Clearly this basis is invariant under the action of g.

4) Lastly,

$$\begin{aligned}\dim(F_{d-n} \cap \mathrm{Fix}(g)) &= \dim(F_{d-n}) + \dim(\mathrm{Fix}(g)) - \dim(F_{d-n} + \mathrm{Fix}(g)) \\ &= (d - n) + (d - n) - d = 2d - 2n - d = d - 2n.\end{aligned}$$

□

Remark 9.10

We proceed as follows to test whether $g := t^a$ is an honest doubling element:

1) Using Gaussian elimination, verify that $\dim(V_n + V_n g) = 2n$ and $\dim(F_{d-n} + \mathrm{Fix}(g)) = d$ as outlined in Example 5.64.
2) Compute the induced actions of the generators $Y_n \cup Y_n^g$ of H_B on $V_{2n} = V_n + V_n g$ using Algorithm InducedActionRepresentationGroup [Alg. 12] which generate a subgroup of $\mathrm{SL}(2n,q)$ denoted by $\widetilde{H_B} \leq \mathrm{SL}(2n,q)$. Afterwards, we call a naming algorithm on $\widetilde{H_B}$ to verify that $\widetilde{H_B} = \mathrm{CL}(2n,q)$ and, therefore, that $H_B \cong \mathrm{CL}(2n,q)$.

Note that a naming algorithm only has to be called if 1) is satisfied. ◀

We finish this section by stating a pseudo-code algorithm for computing honest doubling elements.

Algorithm 43: ComputeHonestDoublingElement

Input:
- ▶ $\langle X \rangle = G \leq \mathrm{CL}(d,q)$
- ▶ A base change matrix $\mathscr{L} \in \mathrm{GL}(d,q)$
- ▶ $\mathrm{CL}(n,q) \cong \langle Y_n \rangle = H \leq G^{\mathscr{L}}$ stingray embedded and constructively recognised
- ▶ $N \in \mathbb{N}$

Output: fail **OR** $(g, \mathfrak{S}', N')$ **where**
- ▶ $g \in G$ is an element satisfying $(C1)$, $(C2)$ and $(C3)$, i.e. an honest doubling element,
- ▶ $\mathfrak{S}'$ is an MSLP from $X \cup Y_n$ to g and
- ▶ $N' \in \mathbb{N}$ where $N - N'$ is the number of random selections that were used

function ComputeHonestDoublingElement$(G, \mathscr{L}, H, N)$

```
1  Choose t ∈ H as described in Lemma 9.8                    // I1)
2  repeat                                                    // Start of I2)
3      N ← N − 1
4      if N < 0 then
5          return fail
6      g ← t^a for random a ∈ G^L
7  until g does not satisfy (C1) and (C2) and (C3)
8  S' ← MSLP from X ∪ Y_n to g
9  return (g, S', N)
```

9.3 Construction of a new base change matrix

Let $g \in G^{\mathscr{L}}$ be an honest doubling element, let $H_B \coloneqq \langle H, H^g \rangle$ and $V_{2n} = V_n + V_n g$. The aim of this section is to compute a stingray embedded H_B in $G^{\mathscr{L}}$. This is carried out by the next phase I3) which computes a new base change matrix $\mathscr{L}' \in \mathrm{GL}(d,q)$ such that $H_B^{\mathscr{L}'}$ is stingray embedded in $G^{\mathscr{L}\mathscr{L}'}$ which is also summarised in the next remark.

Remark 9.11

I3) Compute a base change matrix $\mathscr{L}' \in \mathrm{GL}(d,q)$ such that $H_B^{\mathscr{L}'}$ is stingray embedded in $G^{\mathscr{L}\mathscr{L}'}$. ◀

Similarly to SL4) of the GoingUp step based on linear algebra for special linear groups, we compute $\mathscr{L}' \in \mathrm{GL}(d,q)$ as a base change matrix from the basis $\mathscr{B}$ to another basis $\mathscr{B}'$ given in the next remark. Note that $(C1)$ and $(C2)$ ensure that $\mathscr{B}'$ can be constructed as in Remark 9.12.

Remark 9.12

Recall $\pi\colon V \to F_{d-n}$ the projection map to the second summand of $V = V_n \oplus F_{d-n}$ of Remark 5.51. The basis can be computed as follows:

1) Add $v_1, \ldots, v_n$ to $\mathscr{B}'$ as a basis of V_n.
2) Add $\pi(v_1 g), \ldots, \pi(v_n g)$ to $\mathscr{B}'$ as a basis of $V_n g$. Note that $\pi(v_1 g), \ldots, \pi(v_n g)$ are linearly independent. Let $\iota_1, \ldots, \iota_n \in \mathbb{F}_q$ such that $\sum_{i=1}^n \iota_i \pi(v_i g) = 0$. Then $\pi(\sum_{i=1}^n \iota_i v_i g) = 0$ and hence $\sum_{i=1}^n \iota_i v_i g = 0$ or $\sum_{i=1}^n \iota_i v_i g \in V_n$. Since $V_n \cap V_n g = \{0\}$ it is clear that $\sum_{i=1}^n \iota_i v_i g = 0$. This implies $\iota_1 = \ldots = \iota_n = 0$ as $v_1 c, \ldots, v_n c$ are linearly independent.
3) Extend $\mathscr{B}'$ to a basis of V by taking $d-2n$ basis vectors of $\dim(F_{d-n} \cap \mathrm{Fix}(g))$. ◀

The matrix $\mathscr{L}' \in \mathrm{GL}(d,q)$ is now chosen to be the base change matrix between $\mathscr{B}$ and $\mathscr{B}'$ of Remark 9.12.

Remark 9.13

A generating set of $H_B^{\mathscr{L}'}$ is given by $\{h_1, h_2, \ldots, h_k, h_1^g, h_2^g, \ldots, h_k^g\}^{\mathscr{L}'}$ where $H = \langle h_1, \ldots h_k \rangle$. The elements $\{h_1, h_2, \ldots, h_k, h_1^g, h_2^g, \ldots, h_k^g\}^{\mathscr{L}'}$ have the following form

$$h_i^{\mathscr{L}'} = \left(\begin{array}{cc|c} \mathrm{CL}(n,q) & 0 & \\ \hline 0 & 1 & 0 \\ \hline 0 & & I_{d-2n} \end{array}\right) \quad \text{and} \quad (h_i^g)^{\mathscr{L}'} = \left(\begin{array}{cc|c} * & * & \\ \hline 0 & \mathrm{CL}(n,q) & 0 \\ \hline 0 & & I_{d-2n} \end{array}\right).$$

◀

The elements of $H_B^{\mathscr{L}'}$ are, therefore, block-diagonal matrices or, to be more precise, stingray elements where the bottom right $(d-2n)\times(d-2n)$-matrix is the identity. We can extract the top left non-trivial $2n\times 2n$ blocks of the stingray embedded group $H_B^{\mathscr{L}'}$ in $G^{\mathscr{L}'}$ using InducedActionRepresentationGroup [Alg. 12] yielding a subgroup $\widetilde{H_B}\leq \mathrm{SL}(2n,q)$ with $\widetilde{H_B}=\mathrm{CL}(2n,q)$ of the same type as G.

9.4 Construction of a swapping element and standard generators

The goal of this section is to compute a swapping element as in Definition 9.5. Recall our setting from Hypothesis 9.2 and that we have a stingray embedded subgroup $H_B^{\mathscr{L}'}$ in $G^{\mathscr{L}\mathscr{L}'}$, where $H_B \coloneqq \langle H, H^g\rangle$ for an honest doubling element $g\in G^{\mathscr{L}}$. The honest doubling element g can be computed using the results of Section 9.2 and the base change matrix $\mathscr{L}'\in \mathrm{GL}(d,q)$ using the results of Section 9.3. We start this section by introducing new notations.

Hypothesis 9.14

Since $H_B^{\mathscr{L}'}$ is stingray embedded in $G^{\mathscr{L}\mathscr{L}'}$ we know that the elements of $h\in H_B^{\mathscr{L}}$ have a block structure, namely

$$\left(\begin{array}{c|c} h & 0 \\ \hline 0 & I_{d-2n}\end{array}\right)\in H_B^{\mathscr{L}}\leq G^{\mathscr{L}\mathscr{L}'}$$

which is also described in Remark 9.13. We now extract the top left $2n\times 2n$ blocks of the elements of $H_B^{\mathscr{L}}$ as elements of $\mathrm{SL}(2n,q)$ and define thereby subgroups of $\mathrm{SL}(2n,q)$. For this, we define the isomorphism $\mathfrak{x}\colon H_B^{\mathscr{L}'}\to \mathrm{SL}(2n,q)$, $a\mapsto \hat{a}$ where $\hat{a}$ is the top left $2n\times 2n$ block of a. Using $\mathfrak{x}$ we set $\hat{H}\coloneqq \mathfrak{x}(H)$ and $\hat{H}_B\coloneqq \mathfrak{x}(H_B)$.

We continue this section by stating the remaining phases.

Remark 9.15

I4) Let $m=\mathrm{diag}(-1,\ldots,-1)\in \mathrm{CL}(n,q)$ and $\hat{i}=\mathrm{diag}(m,-1m)\in \mathrm{SL}(2n,q)$. Write $i\coloneqq \mathfrak{x}^{-1}(\hat{i})=\mathrm{diag}(\hat{i},I_{d-2n})\in H_B^{\mathscr{L}'}$ as a word in Y_n. Note that this is possible since the standard generators Y_n of H are known and that m is an element of $\mathrm{CL}(n,q)$ as n is even by assumption.

I5) Compute the projective centraliser $PC_{\hat{H}_B}(\hat{i})$ using Bray's algorithm [18] and identify in $PC_{\hat{H}_B}(\hat{i})$ a swapping element $\hat{u}$ which interchanges the 1- and (-1)-eigenspaces of $\hat{i}$. Afterwards, we set

$u := \mathfrak{x}^{-1}(\hat{u})$.

I6) Compute the conjugate $H_2 := H^u$ and a new base change matrix $\mathscr{L}'' \in \mathrm{GL}(d,q)$ such that $H^{\mathscr{L}''}$ and $H_2^{\mathscr{L}''}$ are disjoint blocks in $H_B^{\mathscr{L}'\mathscr{L}''}$.

I7) Call the LGO *glueing* algorithm on H, H_2 and H_B to assemble standard generators for H_B. ◀

We start with I4) and show how this phase can be carried out.

Remark 9.16

Note that $i = \mathrm{diag}(m, I_{d-n}) \in H_B^{\mathscr{L}'}$ has the following form

$$i = \left(\begin{array}{cc|c} -1 \cdot I_n & 0 & \\ \hline 0 & 1 \cdot I_n & 0 \\ \hline & 0 & I_{d-2n} \end{array}\right).$$

◀

Lemma 9.17

We have $i \in H_B^{\mathscr{L}'}$ where $i = \mathrm{diag}(m, I_{d-n})$. Moreover, i can be written as a word in terms of the generators of H_B.

Proof. Clearly $i \in H^{\mathscr{L}'} \leq H_B^{\mathscr{L}'} = \langle H, H^g \rangle^{\mathscr{L}}$ and since the standard generators of $H = H^{\mathscr{L}'}$ are known we can use a rewriting procedure as in Section 1.1.10 to construct a word which evaluates to $\mathrm{diag}(i, I_{d-n})$. □

Note that one important advantage of the solution of this chapter compared to the LGO algorithm is that we do not have to search for a suitable involution by random selection of elements of H_B but rather construct the involution i as a word in H.

The goal of I5) is to compute the projective centraliser $PC_{\hat{H}_B}(\hat{i})$ which is defined as follows.

Definition 9.18: [59, p. 837]

Let $g \in G \leq \mathrm{GL}(d,q)$, let $\overline{G}$ denote $G/G \cap Z$ where Z denotes the centre of $\mathrm{GL}(d,q)$, and let $\overline{g}$ denote the image of g in $\overline{G}$. The *projective centraliser* $PC_G(g)$ of $g \in G$ is the preimage in G of $C_{\overline{G}}(\overline{g})$.

The next lemma summaries important properties of $PC_{\hat{H}_B}(\hat{i})$

Lemma 9.19

Let n be even and $\hat{H} \cong \mathrm{CL}(n,q)$ a stingray embedded subgroup of $\hat{H}_B$ with $\hat{H}_B = \mathrm{CL}(2n,q)$. Let $\hat{i} = \mathrm{diag}(m,-1m) \in \hat{H}$.

1) The projective centraliser $PC_{\hat{H}_B}(\hat{i})$ can be computed using Bray's algorithm [18] which requires $\mathcal{O}(n(\zeta + n^3\log(n) + n^2\log(n)\log(\log(n))\log(q)))$ finite field operations.
2) $PC_{\hat{H}_B}(\hat{i})$ contains an element $\hat{u}$ which swaps the 1- and (-1)-eigenspace of $\hat{i}$.

Proof. 1) [59, Theorem 12.3].

2) First note that the (-1)-eigenspace of $\hat{i}$ is given by the first n basis vectors of $\mathcal{B}'$ and the 1-eigenspace of $\hat{i}$ is given by the $n+1,\ldots,2n$ basis vectors of $\mathcal{B}'$. Moreover, note that the (-1)-eigenspace and the 1-eigenspace of $\hat{i}$ have the same dimension. The permutation matrix corresponding to $(1,2n)(2,2n-1)\ldots(n,n+1)$ is contained in $\mathrm{CL}(2n,q)$ and swaps the first n with the last n basis vectors. Therefore, there exists an element which swaps the 1- and (-1)-eigenspaces. Moreover, this permutation matrix is also contained in $PC_{\hat{H}_B}(\hat{i})$. □

Remark 9.20

Note that computing the projective centraliser is a known method for computing swapping elements and is used e.g. in the LGO algorithm [59]. Moreover, the method of constructing an involution in a stingray embedded subgroup and computing a second commutating subgroup using the centraliser of the involution is presented in [32]. Note that an element swapping the 1- and (-1)-eigenspaces of an involution is only contained in the projective centraliser of the involution if and only if the 1- and (-1)-eigenspaces have the same dimension. This is also the reason why we compute $PC_{\hat{H}_B}(\hat{i})$ instead of $PC_{H_B}(i)$ since i has an n-dimensional (-1)-eigenspace and an $(d-n)$-dimensional 1-eigenspace. ◀

Note that $\hat{u} \in \hat{H}_B$ and, therefore, we can compute $u := \mathfrak{r}^{-1}(\hat{u})$ as an element of $H_B^{\mathscr{L}'} \leq G^{\mathscr{L}\mathscr{L}'}$. For phase I6) we now set

- $H_2 = H^u$ and
- $\mathscr{B}'' = (v_1, \ldots, v_n, v_1 u, \ldots, v_n u)$ and extend $\mathscr{B}''$ to a basis of V by taking $d-2n$ basis vectors of $\dim(F_{d-n} \cap \mathrm{Fix}(g))$ as in 3) of Remark 9.12 for $\mathscr{B}'$.

Let $\mathscr{L}'' \in \mathrm{GL}(d,q)$ be the base change matrix from $\mathscr{B}$ to $\mathscr{B}''$.

Lemma 9.21

Let n be even and H, H_2 and $\mathscr{L}''$ as described above. Then all of the following hold.

1) $H_2 \cong \mathrm{CL}(n,q)$,
2) the standard generators of H and H_2 are known,
3) $\langle H, H_2 \rangle \cong \mathrm{CL}(n,q) \times \mathrm{CL}(n,q)$ and
4) $\langle H, H_2 \rangle^{\mathscr{L}''}$ is stingray embedded in $G^{\mathscr{L}\mathscr{L}''}$ as $\mathrm{CL}(n,q) \times \mathrm{CL}(n,q)$.

Proof. Most of the statements are clear or follow immediately by construction. Note that H_2 is a conjugate of H and, therefore, isomorphic to $\mathrm{CL}(n,q)$. Moreover, the standard generators of H_2 are the conjugates of the standard generators of H. The fact that H and H_2 commute follows by construction as H acts on the first n basis vectors of $\mathscr{B}''$ and fixes the rest while H_2 acts on the $(n+1)$-th up to $2n$-th basis vector of $\mathscr{B}''$ and fixes the remaining basis vectors since for any $h' \in H_2 = H^u$ and $i \in \{1, \ldots, n\}$

$$v_i u h' = v_i u u^{-1} h u = v_i h u \in V_n u.$$

□

In the last phase I7) we assemble standard generators for H_B as our setting is now identical to the setting from [59] before applying the LGO *glueing* algorithm. The LGO *glueing* algorithm is not discussed in this thesis and instead we refer to [59].

9.5 GoingUp step

Finally, we summarise all the phases introduced in Remark 9.7, 9.11 and 9.15 in Remark 9.22. Moreover, we provide a pseudo-code algorithm implementing the GoingUp step based on involutions of this chapter by Algorithm GoingUpWithInvolutionsStep [Alg. 44]. The proof of correctness of Algorithm GoingUpWithInvolutionsStep [Alg. 44] is given by Theorem 9.23 which thereby also proves Theorem 9.3. Recall Hypothesis 9.2 and the additional notation introduced in Hypothesis 9.14.

Remark 9.22

Let $G = \mathrm{CL}(d,q)$ and $X \subseteq G$ such that $\langle X \rangle = G$. Let $2 \leq n < d$ with n even and $2n \leq d$ and let $\mathscr{L} \in \mathrm{GL}(d,q)$ be a base change matrix. Let $Y_n^{\mathscr{L}}$ be a set of standard generators for $H \cong \mathrm{CL}(n,q)$ stingray embedded into $G^{\mathscr{L}}$ of the same type as G. Furthermore, let $\mathfrak{S}$ be an SLP from X to Y_n. Overall, the following phases must be performed for Theorem 9.3:

I1) Construct an element $t \in H$ which has a fixed space of dimension $d-n$. This can be achieved easily as we have standard generators Y_n for H.

I2) Choose random elements $a \in G^{\mathscr{L}}$ until $g := t^a$ satisfies the three conditions $(C1)$, $(C2)$ and $(C3)$ of Remark 9.4, i.e. until g is an honest doubling element. If g is an honest doubling element, then set $H_B := \langle H, H^g \rangle$.

I3) Compute a new base change matrix $\mathscr{L}' \in \mathrm{GL}(d,q)$ such that $H_B^{\mathscr{L}'}$ is stingray embedded in $G^{\mathscr{L}\mathscr{L}'}$.

I4) Let $m = \mathrm{diag}(-1,\ldots,-1) \in \mathrm{CL}(n,q)$ and $\hat{i} = \mathrm{diag}(m,-1m) \in \mathrm{SL}(2n,q)$. Write $i := \mathfrak{r}^{-1}(\hat{i}) = \mathrm{diag}(\hat{i}, I_{d-2n}) \in H_B^{\mathscr{L}'}$ as a word in Y_n. Note that this is possible since the standard generators Y_n of H are known and that m is an element of $\mathrm{CL}(n,q)$ as n is even by assumption.

I5) Compute the projective centraliser $PC_{\hat{H}_B}(\hat{i})$ using Bray's algorithm [18] and identify in $PC_{\hat{H}_B}(\hat{i})$ a swapping element $\hat{u}$ which interchanges the 1- and (-1)-eigenspaces of $\hat{i}$. Then set $u := \mathfrak{r}^{-1}(\hat{u})$.

I6) Compute the conjugate $H_2 := H^u$ and a new base change matrix $\mathscr{L}'' \in \mathrm{GL}(d,q)$ such that $H^{\mathscr{L}''}$ and $H_2^{\mathscr{L}''}$ are disjoint blocks in $H_B^{\mathscr{L}'\mathscr{L}''}$.

I7) Call the LGO *glueing* algorithm on H, H_2 and H_B to assemble standard generators for H_B. ◀

The GoingUp step based on involutions described in this chapter is given in pseudo code with

Algorithm GoingUpWithInvolutionsStep [Alg. 44].

Theorem 9.23

Algorithm GoingUpWithInvolutionsStep [Alg. 44] terminates using at most N random selections and works correctly.

Proof. The correctness follows immediately from the correctness of each phase. Therefore, it is left to prove that Algorithm GoingUpWithInvolutionsStep [Alg. 44] terminates. Every line is deterministic except the computation of g in Line 1, the computation of the projective centraliser in Line 7 and the LGO *glueing* algorithm in Line 11 which are randomised. Since each of these algorithm returns `fail` when $N \leq 0$ the claim follows and we perform at most N random selections. □

Remark 9.24

Note that the computation of g in Line 1 as well as two other phases of Algorithm GoingUpWithInvolutionsStep [Alg. 44] are randomised. The computation of the projective centraliser in Line 7 is randomised and the LGO *glueing* algorithm in Line 11 since the LGO *glueing* algorithm involves a call to the constructive recognition algorithm of $\mathrm{CL}(2,q)$. Since both Bray's algorithm (see [18]) and the LGO constructive recognition algorithm (see [59]) are well-known, we omitted additional details of proving that they terminate using a parameter N. ◀

Algorithm 44: GoingUpWithInvolutionsStep

Input:
- $\langle X \rangle = G \leq \mathrm{CL}(d,q)$
- A base change matrix $\mathscr{L} \in \mathrm{GL}(d,q)$
- $\mathrm{CL}(n,q) \cong \langle Y_n \rangle = H \leq G^{\mathscr{L}}$ stingray embedded and constructively recognised, $n \leq \frac{d}{2}$ even
- An MSLP $\mathfrak{S}$ from X to the standard generators Y_n of H
- $N \in \mathbb{N}$

Output: **fail OR** $(\mathscr{L}'', Y_{2n}, \mathfrak{S}', N')$ **where**
- $\mathscr{L}'' \in \mathrm{GL}(d,q)$ is a base change matrix,
- $\mathrm{CL}(2n,q) \cong \langle Y_{2n} \rangle = H_B \leq G^{\mathscr{L}\mathscr{L}''}$ where H_B is stingray embedded in $G^{\mathscr{L}\mathscr{L}''}$,
- $\mathfrak{S}'$ is an MSLP from $X \cup Y_n$ to the standard generators Y_{2n} of H_B and
- $N' \in \mathbb{N}$ where $N - N'$ is the number of random selections that were used

function GoingUpWithInvolutionsStep($G, \mathscr{L}, H, \mathfrak{S}, N$)

1 $(g, \mathfrak{S}_1, N) \leftarrow$ ComputeHonestDoublingElement($G, \mathscr{L}, H, N$) // I1), I2), Remark 5.7
2 **if** $g =$ **fail** **then**
3 **return** fail
4 $\mathscr{L}' \leftarrow$ as described in Remark 9.12 // I3)
5 $H_B \leftarrow \langle H, H^g \rangle^{\mathscr{L}'}$
6 $\mathfrak{S}_2 \leftarrow$ an MSLP for $i = \mathrm{diag}(m, I_{d-n}) \in H_B$ where $m = \mathrm{diag}(-1, \ldots, -1) \in \mathrm{CL}(n, q)$ // I4)
7 $(\hat{u}, N, \mathfrak{S}_3) \leftarrow$ ProjectiveCentraliserOfInvolution($\hat{H}_B, \hat{i}, N$) // I5), 5.8
8 $u \leftarrow \mathfrak{x}^{-1}(\hat{u})$ a swapping element
9 $H_2 \leftarrow H^u$ **AND** $\mathfrak{S}_4 \leftarrow$ an MSLP for H_2 // I6)
10 $\mathscr{L}'' \leftarrow$ a base change matrix from $(v_1, \ldots, v_n)$ to $(v_1, \ldots, v_n, v_1 u, \ldots, v_n u)$
11 $(\mathfrak{S}_5, N) \leftarrow$ MSLP to stand. gens. of H_B using GlueingLGO(H_B, H, H_2, N) // I7), 5.8
12 Compose $\mathfrak{S}, \mathfrak{S}_1, \mathfrak{S}_2, \mathfrak{S}_3, \mathfrak{S}_4, \mathfrak{S}_5$ into one MSLP $\mathfrak{S}'$
13 **return** $(\langle H, H^g \rangle^{\mathscr{L}''}, \mathscr{L}'', \mathfrak{S}', N)$

9.6 Combining GoingUp steps

In Section 9.5 we have proven how we can compute standard generators for a stingray embedded classical group doubling the dimension of a given stingray embedded classical group of the same type for which standard generators can be written as words. In this section Algorithm GoingUpWithInvolutionsStep [Alg. 44] is used repeatedly to design an algorithm which can be used to compute standard generators for $\langle X \rangle = G = \mathrm{CL}(d, q)$ when $H \leq G$ is a stingray embedded base case group of the same type and standard generators of H can be written as words in X.

Remark 9.25

In Algorithm GoingUpWithInvolutions [Alg. 45] k is chosen such that k is maximal with $k \leq d-2$ and $k \equiv 0 \pmod 4$. We require that k is even such that $k/2$ is an integer. We also require that $k/2$ is even because this is a requirement of Algorithm GoingUpWithInvolutionsStep [Alg. 44]. Moreover, $k \leq d-2$ because we cannot assemble standard generators for $\mathrm{CL}(d, q)$ using the LGO *glueing* algorithm on $\mathrm{CL}(d-1, q) \times \mathrm{CL}(1, q)$. Thus, we require H_2 to be isomorphic to $\mathrm{CL}(r, q)$ for $r \geq 2$ as the second factor. ◀

Algorithm 45: GoingUpWithInvolutions

Input:
- $\langle X \rangle = G = \mathrm{CL}(d,q)$
- A base change matrix $\mathscr{L} \in \mathrm{GL}(d,q)$
- $\mathrm{CL}(n',q) \cong \langle Y_{n'} \rangle = H \leq G^{\mathscr{L}}$ a base case group and stingray embedded in $G^{\mathscr{L}}$
- MSLP $\mathfrak{S}$ from X to the standard generators $Y_{n'}$ of H
- $N \in \mathbb{N}$

Output: fail **OR** $(\mathscr{L}', \mathfrak{S}', N')$ **where**
- $\mathscr{L}' \in \mathrm{GL}(d,q)$ is a base change matrix,
- $\mathfrak{S}'$ is an MSLP from $X \cup Y_{n'}$ to the standard generators of $G^{\mathscr{L}\mathscr{L}'}$ and
- $N' \in \mathbb{N}$ where $N - N'$ is the number of random selections that were used

function GoingUpWithInvolutions$(G, \mathscr{L}, H, \mathfrak{S}, N)$

$n \leftarrow n'$
while $n \leq \lfloor d/2 \rfloor$ **do**
  $n \leftarrow 2 \cdot n$ // Double the dimension.
  $(H, \mathscr{L}, \mathfrak{S}, N) \leftarrow$ GoingUpWithInvolutionsStep$(G, \mathscr{L}, H, \mathfrak{S}, N)$ // Remark 5.8 and 5.7
if $n = d$ **then**
  return $(\mathscr{L}, \mathfrak{S}, N)$
$k \leftarrow$ maximal with $k \leq d-2$ and $k \equiv 0 \pmod 4$ // Note $d-5 \leq k \leq d-2$
$\widetilde{H} \leq H \leftarrow$ stingray embedded with $\widetilde{H} \cong \mathrm{CL}(k/2, q)$
$\mathfrak{S} \leftarrow$ MSLP to standard generators of $\widetilde{H}$
$(H_1, \mathscr{L}, \mathfrak{S}_1, N) \leftarrow$ GoingUpWithInvolutionsStep$(G, \mathscr{L}, \widetilde{H}, \mathfrak{S}, N)$ // 5.8
$i \leftarrow \mathrm{diag}(-1 \cdot I_k, I_{d-k})$ **AND** $\mathfrak{S}_2 \leftarrow$ an MSLP to i
$(C, N) \leftarrow C(G^{\mathscr{L}}, i, N)'$ **AND** $\mathrm{CL}(d-k,q) \cong H_2 \leq G^{\mathscr{L}} \leftarrow$ AlgorithmLGO(C, N) // 5.8
$\mathfrak{S}_3 \leftarrow$ MSLP to standard generators of H_2 // Note $H_2 \cong \mathrm{CL}(r,q)$ for some $2 \leq r \leq 5$
$(\mathfrak{S}_4, N) \leftarrow$ MSLP to stand. gens. of G using GlueingLGO(G, H_1, H_2, N) // I7), 5.8
Compose $\mathfrak{S}, \mathfrak{S}_1, \mathfrak{S}_2, \mathfrak{S}_3, \mathfrak{S}_4$ into one MSLP $\mathfrak{S}'$
return $(\mathscr{L}, \mathfrak{S}', N)$

Theorem 9.26

Algorithm GoingUpWithInvolutions [Alg. 45] terminates using at most N random selections and works correctly.

Proof. That the algorithm terminates is clear since we prove in Theorem 9.23 that Algorithm GoingUpWithInvolutionsStep [Alg. 44] terminates and every other computation of Algorithm GoingUpWithInvolutions [Alg. 45] is either deterministic or a well-known algorithm which is proven to terminate using a probabilistic parameter.

For the correctness we start by assuming that d is a power of 2. In this case we can use Algorithm GoingUpWithInvolutionsStep [Alg. 44] until standard generators of G are computed and correctness immediately follows. From here on, we assume that d is not a power of 2 and that we can write standard generators of $H \leq G$ stingray embedded as words in X with $H \cong \mathrm{CL}(n,q)$ and $n > \lfloor d/2 \rfloor$. Since it is not possible to use Algorithm GoingUpWithInvolutionsStep [Alg. 44] again, we switch to the following strategy. The standard generators of H can be written as words in X and using this knowledge we can compute standard generators of $\widetilde{H} \leq H$ with $\widetilde{H} \cong \mathrm{CL}(k,q)$ for arbitrary $2 \leq k \leq n$ by reducing the length of the permutation matrices z_1 and z_2. We choose k in such a way that we can double the dimension again using Algorithm GoingUpWithInvolutionsStep [Alg. 44] on $\widetilde{H} \cong \mathrm{CL}(k/2,q)$ and that $d-k \in \{2,\ldots,5\}$. Therefore, by using Algorithm GoingUpWithInvolutionsStep [Alg. 44] on $\widetilde{H}$ we compute standard generators of a subgroup $H_1 \leq G$ with $H_1 \cong \mathrm{CL}(k,q)$. Since standard generators of H_1 are known we can write down an explicit involution i of G such that $C_G(i)' = \mathrm{CL}(k,q) \times \mathrm{CL}(d-k,q) = H_1 \times \mathrm{CL}(d-k,q) = H_1 \times H_2$. As $d-k \in \{2,\ldots,5\}$ is small we can easily compute standard generators of $H_2 \cong \mathrm{CL}(d-2k,q)$ and use the LGO *glueing* algorithm to compute standard generators for G. □

Chapter 10

Complexity analysis of algorithms

In this chapter we analyse the complexity of the StandardGenerators algorithm of Section 3.4. As proven in Chapter 5 for special linear groups, in Chapter 6 for symplectic groups, in Chapter 7 for unitary groups and in Chapter 8 for orthogonal groups, the StandardGenerators algorithm is a one-sided Monte Carlo algorithm using the subalgorithms GoingDown, BaseCase and GoingUp. All these three subalgorithms are randomised and, thus, a complexity analysis of the StandardGenerators algorithm requires a complexity analysis of these subalgorithms.

In Section 10.1 we compute the complexity of the GoingDown algorithm. This includes a probability analysis of computing a ppd-stingray element whose degree lies in a specific range by random procedure in Section 10.1.1, a probability analysis that a stingray pair is a stingray duo in Section 10.1.2 and a probability analysis that a stingray duo generates a classical group in Section 10.1.3. In Section 10.1.5 the probability results are used to compute an integer $N \in \mathbb{N}$ for a given $\epsilon \in (0,1)$ such that if the GoingDown algorithm computes at most N random elements, then the GoingDown algorithm succeeds with probability at least $1-\epsilon$. Finally, in Section 10.1.6 the complexity of the GoingDown algorithm is determined and proven.

In Section 10.2 we summarise complexity results of algorithms for constructive recognition of base case groups from the literature yielding an upper bound on the complexity of the BaseCase algorithm.

In Section 10.3 we state a conjecture on a probability aspect of the GoingUp algorithm which is not

analysed in this thesis. Assuming this conjecture holds, we compute the complexity of the GoingUp algorithm.

Lastly, in Section 10.4 a conjecture on the complexity of the StandardGenerators algorithm is proven, based on the results of Section 10.1, Section 10.2 and Section 10.3.

In the following let $\mathrm{CL}(d,q) \in \{\mathrm{SL}(d,q), \mathrm{Sp}(d,q), \mathrm{SU}(d,q), \Omega(d,q)\}$ denote one of the classical groups.

10.1 Complexity of the GoingDown algorithm

The GoingDown algorithm calls the randomised algorithm GoingDown basic step repeatedly. Therefore, the probability that the GoingDown algorithm succeeds depends on the probability that the GoingDown basic step succeeds.

Recall from Definition 4.22 that (s_1, s_2) is a stingray pair if $s_1, s_2 \in \mathrm{GL}(d,q)$ are stingray elements and that a stingray pair (s_1, s_2) is a stingray duo if $W_{s_1} \cap W_{s_2} = \{0\}$, where W_{s_i} is the stingray body of s_i for $i \in \{1,2\}$.

We start to analyse the GoingDown basic step which involves the study of the following probabilities:

1) The probability to find a pre-stingray candidate with ppd-stingray property by random selection of elements in $\mathrm{CL}(d,q)$.

2) The conditional probability that a stingray pair in $\mathrm{CL}(d,q)$ forms a stingray duo, i.e. the probability that their stingray bodies intersect trivially and, if $\mathrm{CL}(d,q) \neq \mathrm{SL}(d,q)$, then the sum of their stingray bodies is a non-degenerate subspace.

3) The probability that a stingray duo in $\mathrm{CL}(d,q)$ generates $\mathrm{CL}(d,q)$.

To increase the readability of this chapter we give a glossary of all probabilities and their notation used.

Symbol	Variables	Description
$P_{\mathfrak{s}}(k)$	$\lceil\log(d)\rceil \leq k \leq 2\lceil\log(d)\rceil$	Probability to find a pre-stingray candidate with ppd-stingray property and stingray factor of degree k
$P_{\text{stingray}}(d,q)$	q arbitrary, $d \geq$ as in Table 3.1	Probability to compute a ppd-stingray element of degree m with $\lceil\log(d)\rceil \leq m \leq 2\lceil\log(d)\rceil$
$P_{\text{pair}}(d,q)$	q arbitrary, $d \geq$ as in Table 3.1	Probability to compute two ppd-stingray elements of degree m_1, m_2 with $\lceil\log(d)\rceil \leq m_1, m_2 \leq 2\lceil\log(d)\rceil$ in exactly two random processes
$P_{\text{duo}}(d,q)$	q arbitrary, $d \geq$ as in Table 3.1	Conditional probability that a stingray pair forms a stingray duo
$P_{\text{gen}}(d,q)$	q arbitrary, $d \geq$ as in Table 3.1	Conditional probability that a stingray duo generates a classical group of the same type
$P_{\text{step}}(d,q)$	q arbitrary, $d \geq$ as in Table 3.1	Probability that a GOINGDOWN basic step with exactly two random selections is successful

Table 10.1: Glossary of all probabilities and their notation used in this chapter

10.1.1 Probability to find a stingray element

Recall Table 4.1 from Section 4.2 which is given again as Table 10.2. In Section 4.2 we summarised results from [75] on the proportion of pre-stingray candidates which power up to ppd-stingray elements $s \in \mathrm{CL}(d,q)$ where $\max\{3,\log(r)\} \leq k \leq r$ for $k := \dim(W_s)$ and r as in Table 4.1. In this section we extend these results to obtain better lower bounds for the probability to find a pre-stingray candidate with ppd-stingray property by random selection in a classical group with bounds on the dimension of the stingray body as required by the GOINGDOWN algorithm.

H	d	α	δ
$\mathrm{SL}(r,q)$	r	1	1
$\mathrm{SU}(r,q)$	r	1	2
$\mathrm{Sp}(2r,q)$	$2r$	2	1
$\mathrm{SO}^\circ(2r+1,q)$	$2r+1$	2	1
$\mathrm{SO}^\pm(2r,q)$	$2r$	2	1

Table 10.2: Groups and constants in Theorem 4.25 and 4.26 [75, Table 1].

In the following we define $P_{\mathfrak{s}}(k)$ to be the probability of finding in exactly one random selection in a classical group $G = \mathrm{CL}(d,q)$ an element that powers up to an element which has a $(d-\alpha k)$-dimensional 1-eigenspace and acts irreducibly on a complementary invariant subspace of dimension αk where α is as given in Table 4.1. Our first goal is to improve the lower bound for $P_{\mathfrak{s}}(k)$ given in Theorem 4.26 for k in the range $\lceil\log(d)\rceil \leq k \leq 2\lceil\log(d)\rceil$. Afterwards we prove lower bounds on

the success probability of one GoingDown basic step by summing the lower bounds for $P_{\mathfrak{s}}(k)$ for k in the range of $\lceil \log(d) \rceil \leq k \leq 2\lceil \log(d) \rceil$.

The results presented in this section are based on [75] and improved further. Note that Table 4.1 defines an integer r. In the proof of [75, Theorem 3.3] it is shown that for $k \geq \log(r)$

$$P_{\mathfrak{s}}(k) \geq \left(1 - \frac{1}{\alpha k}\right) \frac{b_k(r)}{\alpha} \tag{10.1.1}$$

where $b_k(r)$ denotes the proportion of elements in the symmetric group S_r with exactly one cycle of length k and all other cycle lengths not divisible by k. Note that [75, Theorem 3.3] requires $k \geq \log(d)$ but Inequality (10.1.1) remains valid for any $k \geq 2$. Moreover, it is shown in [75, Lemma 4.5] that if $k \geq 3$ and $\log(r) \leq k \leq r - k$, then

$$\frac{1}{3ek} \leq b_k(r) \leq \frac{5}{3k}.$$

For the GoingDown basic step we prove a sharper lower bound for $b_k(r)$.

Lemma 10.1

Let $m, r \in \mathbb{N}$ with $\log(r) \leq m \leq r - m$ or equivalently $\log(r) \leq m \leq \frac{r}{2}$. Let $b_m(r)$ denote the proportion of elements in the symmetric group S_r with exactly one cycle of length m and all other cycle lengths not divisible by m. Then $b_m(r) \geq \frac{1}{2em}\left(1 - \frac{1}{r-m}\right)$.

Proof. We adapt the proof of [75, Lemma 4.5]. Applying the inequality from the proof of [66, Lemma 4.2], which is derived from [15, Theorem 2.3(b)], yields

$$b_m(r) \geq \frac{1}{2m}\left(\frac{m}{r-m}\right)^{1/m}\left(1 - \frac{1}{r-m}\right)$$

and we only have to prove that

$$\left(\frac{m}{r-m}\right)^{1/m} \geq \frac{1}{e}.$$

This inequality is equivalent to showing that

$$e^m \geq \frac{r-m}{m}.$$

Since $e^m \geq e^{\log(r)} = r$ for $m \geq \log(r)$ and since $r \geq \frac{r-m}{m}$ the claim follows. □

Corollary 10.2

Let $\mathrm{CL}(d,q)$ be a classical group and let $k \in \mathbb{N}$ with $\lceil \log(d) \rceil \leq k \leq 2\lceil \log(d) \rceil$. If $\mathrm{CL}(d,q) = \mathrm{SL}(d,q)$ or $\mathrm{CL}(d,q) = \mathrm{SU}(d,q)$, then for $d \geq 12$

$$P_{\mathfrak{s}}(k) \geq \frac{1}{2e}\Big(1-\frac{1}{k}\Big)\Big(1-\frac{1}{d-k}\Big)\frac{1}{k} \geq C_1\Big(1-\frac{1}{d-k}\Big)\frac{1}{k}$$

for a constant C_1 and if $\mathrm{CL}(d,q) = \mathrm{Sp}(d,q)$ or $\mathrm{CL}(d,q) = \Omega(d,q)$, then for $d \geq 20$

$$P_{\mathfrak{s}}(k) \geq \frac{1}{4e}\Big(1-\frac{1}{2k}\Big)\Big(1-\frac{2}{d-2k-1}\Big)\frac{1}{k} \geq C_2\Big(1-\frac{2}{d-2k-1}\Big)\frac{1}{k}$$

for a constant C_2. In particular $C_1 \geq 0.123$ and $C_2 \geq 0.077$.

Proof. By equation (10.1.1)

$$P_{\mathfrak{s}}(k) \geq \Big(1-\frac{1}{\alpha k}\Big)\frac{b_k(r)}{\alpha}.$$

If $\mathrm{CL}(d,q) = \mathrm{SL}(d,q)$ or $\mathrm{CL}(d,q) = \mathrm{SU}(d,q)$, then $r = d$ and $\alpha = 1$. Note that $\log(d) \leq k \leq d-k$ is satisfied for all $\lceil \log(d) \rceil \leq k \leq 2\lceil \log(d) \rceil$ if $d \geq 12$. Thus, by Lemma 10.1 it follows that

$$P_{\mathfrak{s}}(k) \geq \Big(1-\frac{1}{\alpha k}\Big)\frac{1}{2e\alpha k}\Big(1-\frac{1}{r-k}\Big) = \frac{1}{2e}\Big(1-\frac{1}{k}\Big)\Big(1-\frac{1}{d-k}\Big)\frac{1}{k} \geq \frac{1}{2e}\Big(1-\frac{1}{\lceil \log(d) \rceil}\Big)\Big(1-\frac{1}{d-k}\Big)\frac{1}{k}.$$

If d is even and $\mathrm{CL}(d,q) = \mathrm{Sp}(d,q)$ or $\mathrm{CL}(d,q) = \Omega(d,q)$, then $r = \frac{d}{2}$ and $\alpha = 2$. Note that $\log(\frac{d}{2}) \leq k \leq \frac{d}{2} - k$ is satisfied for all $\lceil \log(d) \rceil \leq k \leq 2\lceil \log(d) \rceil$ if $d \geq 20$. If d is odd and $\mathrm{CL}(d,q) = \Omega(d,q)$, then $r = \frac{d-1}{2}$ and $\alpha = 2$. Note that $\log(\frac{d-1}{2}) \leq k \leq \frac{d-1}{2} - k$ is satisfied for all $\lceil \log(d) \rceil \leq k \leq 2\lceil \log(d) \rceil$ if $d \geq 21$. Thus, by Lemma 10.1

$$\begin{aligned} P_{\mathfrak{s}}(k) \geq \Big(1-\frac{1}{\alpha k}\Big)\frac{1}{2e\alpha k}\Big(1-\frac{1}{r-k}\Big) &= \Big(1-\frac{1}{2k}\Big)\frac{1}{4ek}\Big(1-\frac{2}{d-2k-1}\Big) \\ &\geq \Big(1-\frac{1}{2\lceil \log(d) \rceil}\Big)\frac{1}{4ek}\Big(1-\frac{2}{d-2k-1}\Big). \end{aligned}$$

Clearly $\frac{1}{2e}$ and $\frac{1}{4e}$ are constant and $\Big(1-\frac{1}{k}\Big)$ as well as $\Big(1-\frac{1}{2k}\Big)$ are increasing for ascending k. We set $C_1(d) := \frac{1}{2e}\Big(1-\frac{1}{\lceil \log(d) \rceil}\Big)$ and $C_2(d) := \frac{1}{4e}\Big(1-\frac{1}{2\lceil \log(d) \rceil}\Big)$. By noting that $C_1(d) \geq C_1(12) \geq 0.123$ and

$C_2(d) \geq C_2(20) \geq 0.077$ we chose $C_1 := C_1(12)$ and $C_2 := C_2(20)$ and the claim follows. □

Since we are interested in computing a stingray element $s \in \mathrm{CL}(d,q)$ with stingray body W_s of dimension k for any $k \leq 2\lceil \log(d) \rceil$, we require a lower bound for the sum of the probabilities $P_s(k)$ for $\lceil \log(d) \rceil \leq k \leq 2\lceil \log(d) \rceil$. Note that in practice we would also use a stingray element $s \in \mathrm{CL}(d,q)$ with stingray body of dimension k with $k < \lceil \log(d) \rceil$ but so far we do not have good lower bounds for $P_s(k)$ if $k < \lceil \log(d) \rceil$.

Definition 10.3

Let $\mathrm{CL}(d,q)$ be a classical group. We define

$$P_{\text{stingray}}(d,q) := \sum_{\lceil \log(d) \rceil \leq k \leq 2\lceil \log(d) \rceil} P_s(k).$$

Then $P_{\text{stingray}}(d,q)$ denotes the probability of finding a pre-stingray candidate with stingray factor of degree k in $\mathrm{CL}(d,q)$ in one random selection with $\lceil \log(d) \rceil \leq k \leq 2\lceil \log(d) \rceil$.

Theorem 10.4

Let $\mathrm{CL}(d,q)$ be a classical group. If $\mathrm{CL}(d,q) = \mathrm{SL}(d,q)$ or $\mathrm{CL}(d,q) = \mathrm{SU}(d,q)$ and $d \geq 12$, then there is a constant C_1 such that

$$\begin{aligned} P_{\text{stingray}}(d,q) &\geq \frac{\log(2)}{e} - \frac{1.32}{d} - \frac{1}{4e\lceil \log(d) \rceil} \geq C_1\Big(\frac{d-1}{d}\log(2) + \frac{1}{d}\log\big(1 - \frac{\lceil \log(d) \rceil}{d - \lceil \log(d) \rceil}\big)\Big) \\ &\geq \frac{C_1}{d}((d-1)\log(2) - 1) \end{aligned}$$

and if $\mathrm{CL}(d,q) = \mathrm{Sp}(d,q)$ or $\mathrm{CL}(d,q) = \Omega(d,q)$ and $d \geq 20$, then there is a constant C_2 such that

$$P_{\text{stingray}}(d,q) \geq C_2\Big(\frac{d-3}{d-1}\log(2) + \frac{2}{d-1}\log\big(1 - \frac{2\lceil \log(d) \rceil}{d - 2\lceil \log(d) \rceil - 1}\big)\Big).$$

If $\mathrm{CL}(d,q) = \mathrm{Sp}(d,q)$ or $\mathrm{CL}(d,q) = \Omega(d,q)$ and $d \geq 33$, then there is a constant C_2 such that

$$\begin{aligned} P_{\text{stingray}}(d,q) &\geq \frac{\log(2)}{4e} - \frac{2\log(2)}{4e(d-1)} - \frac{d-2}{4e(d-1)^2} - \frac{1}{16e\lceil \log(d) \rceil}\Big(\frac{d-2}{d-1}\Big)^2 \\ &\geq \frac{C_2}{d-1}((d-3)\log(2) - 1). \end{aligned}$$

Proof. We start with $\mathrm{CL}(d,q)=\mathrm{SL}(d,q)$ or $\mathrm{CL}(d,q)=\mathrm{SU}(d,q)$ and $d\geq 12$. Using Corollary 10.2

$$\begin{aligned}\sum_{\lceil\log(d)\rceil\leq k\leq 2\lceil\log(d)\rceil} P_{\mathfrak{s}}(k) &\geq \sum_{\lceil\log(d)\rceil\leq k\leq 2\lceil\log(d)\rceil} C_1\Big(1-\frac{1}{d-k}\Big)\frac{1}{k}\\ &= C_1 \sum_{\lceil\log(d)\rceil\leq k\leq 2\lceil\log(d)\rceil}\Big(1-\frac{1}{d-k}\Big)\frac{1}{k}.\end{aligned}$$

Note that a decreasing function satisfies $\sum_{i=r_1}^{r_2} f(i)\geq \int_{r_1}^{r_2+1} f(x)\mathrm{d}x\geq \int_{r_1}^{r_2} f(x)\mathrm{d}x$. Therefore,

$$\begin{aligned}\sum_{\lceil\log(d)\rceil\leq k\leq 2\lceil\log(d)\rceil} P_{\mathfrak{s}}(k) &\geq C_1 \sum_{\lceil\log(d)\rceil\leq k\leq 2\lceil\log(d)\rceil}\Big(1-\frac{1}{d-k}\Big)\frac{1}{k}\\ &\geq C_1\int_{\lceil\log(d)\rceil}^{2\lceil\log(d)\rceil}\Big(1-\frac{1}{d-x}\Big)\frac{1}{x}\mathrm{d}x\\ &= C_1\Big[\frac{(d-1)\log(x)+\log(d-x)}{d}\Big]_{\lceil\log(d)\rceil}^{2\lceil\log(d)\rceil}.\end{aligned}$$

Moreover,

$$\begin{aligned}&\Big[\frac{(d-1)\log(x)+\log(d-x)}{d}\Big]_{\lceil\log(d)\rceil}^{2\lceil\log(d)\rceil}\\ =&\frac{d-1}{d}\Big(\log(2\lceil\log(d)\rceil)-\log(\lceil\log(d)\rceil)\Big)+\frac{1}{d}\Big(\log(d-2\lceil\log(d)\rceil)-\log(d-\lceil\log(d)\rceil)\Big)\\ =&\frac{d-1}{d}\log\Big(\frac{2\lceil\log(d)\rceil}{\lceil\log(d)\rceil}\Big)+\frac{1}{d}\log\Big(\frac{d-2\lceil\log(d)\rceil}{d-\lceil\log(d)\rceil}\Big)\\ =&\frac{d-1}{d}\log(2)+\frac{1}{d}\log\Big(\frac{d-2\lceil\log(d)\rceil}{d-\lceil\log(d)\rceil}\Big)\\ =&\frac{d-1}{d}\log(2)+\frac{1}{d}\log\Big(1-\frac{\lceil\log(d)\rceil}{d-\lceil\log(d)\rceil}\Big)\end{aligned}$$

and, thus,

$$\begin{aligned}\sum_{\lceil\log(d)\rceil\leq k\leq 2\lceil\log(d)\rceil} P_{\mathfrak{s}}(k) &\geq C_1\Big[\frac{(d-1)\log(x)+\log(x-d)}{d}\Big]_{\lceil\log(d)\rceil}^{2\lceil\log(d)\rceil}\\ &= C_1\Big(\frac{d-1}{d}\log(2)+\frac{1}{d}\log\Big(1-\frac{\lceil\log(d)\rceil}{d-\lceil\log(d)\rceil}\Big)\Big).\end{aligned}$$

For $-1<x$ with $x\neq 0$ the logarithm satisfies $\log(1+x)>\frac{x}{1+x}$. By setting $u:=\lceil\log(d)\rceil$ and $x:=-\frac{u}{d-u}$ this yields

$$\frac{1}{d}\log(1-\frac{u}{d-u})\geq\frac{1}{d}\cdot\frac{-u/(d-u)}{1-u/(d-u)}=-\frac{u}{d(d-2u)}.$$

Moreover,

$$-\frac{u}{d(d-2u)} \geq -\frac{1}{d} \Leftrightarrow -\frac{u}{(d-2u)} \geq -1$$
$$\Leftrightarrow -u \geq 2u - d$$
$$\Leftrightarrow d \geq 3u = 3\lceil \log(d) \rceil$$

which holds for $d \geq 5$. Therefore,

$$\sum_{\lceil \log(d) \rceil \leq k \leq 2\lceil \log(d) \rceil} P_{\mathfrak{s}}(k) \geq \frac{C_1}{d}((d-1)\log(2) - 1).$$

This yields the last inequality if $\mathrm{CL}(d,q) = \mathrm{SL}(d,q)$ or $\mathrm{CL}(d,q) = \mathrm{SU}(d,q)$. For the sharper bound in the first inequality we have to deal with a more complicated integral as follows.

$$\sum_{\lceil \log(d) \rceil \leq k \leq 2\lceil \log(d) \rceil} P_{\mathfrak{s}}(k) \geq \sum_{\lceil \log(d) \rceil \leq k \leq 2\lceil \log(d) \rceil} \frac{1}{2e}\Big(1-\frac{1}{k}\Big)\Big(1-\frac{1}{d-k}\Big)\frac{1}{k}$$
$$= \frac{1}{2e} \sum_{\lceil \log(d) \rceil \leq k \leq 2\lceil \log(d) \rceil} \Big(1-\frac{1}{k}\Big)\Big(1-\frac{1}{d-k}\Big)\frac{1}{k}.$$

Note that each term is a decreasing function in k and, therefore,

$$\sum_{\lceil \log(d) \rceil \leq k \leq 2\lceil \log(d) \rceil} P_{\mathfrak{s}}(k) \geq \frac{1}{2e} \sum_{\lceil \log(d) \rceil \leq k \leq 2\lceil \log(d) \rceil} \Big(1-\frac{1}{k}\Big)\Big(1-\frac{1}{d-k}\Big)\frac{1}{k}$$
$$\geq \frac{1}{2e} \int_{\lceil \log(d) \rceil}^{2\lceil \log(d) \rceil} \Big(1-\frac{1}{x}\Big)\Big(1-\frac{1}{d-x}\Big)\frac{1}{x}\mathrm{d}x$$
$$= \frac{1}{2e}\Big[\frac{(d^2-d+1)x\log(x) + (d-1)(x\log(d-x)+d)}{d^2 x}\Big]_{\lceil \log(d) \rceil}^{2\lceil \log(d) \rceil}$$
$$= \frac{(d^2-d+1)\log(2)}{ed^2} - \frac{d-1}{4ed}\Big(\frac{2}{d} + \frac{1}{\lceil \log(d) \rceil}\Big)$$
$$\geq \frac{\log(2)}{e} - \frac{2\log(2)}{ed} - \frac{1}{4e\lceil \log(d) \rceil}$$

which yields the claim if $\mathrm{CL}(d,q) = \mathrm{SL}(d,q)$ or $\mathrm{CL}(d,q) = \mathrm{SU}(d,q)$ and $d \geq 12$. Now let $\mathrm{CL}(d,q) = \mathrm{Sp}(d,q)$ or $\mathrm{CL}(d,q) = \Omega(d,q)$ and $d \geq 20$. Using Corollary 10.2

$$\sum_{\lceil \log(d) \rceil \leq k \leq 2\lceil \log(d) \rceil} P_{\mathfrak{s}}(k) \geq \sum_{\lceil \log(d) \rceil \leq k \leq 2\lceil \log(d) \rceil} C_2\Big(1-\frac{2}{d-2k-1}\Big)\frac{1}{k}$$
$$= C_2 \sum_{\lceil \log(d) \rceil \leq k \leq 2\lceil \log(d) \rceil} \Big(1-\frac{2}{d-2k-1}\Big)\frac{1}{k}.$$

We continue similarly as for $\mathrm{CL}(d,q)=\mathrm{SL}(d,q)$ or $\mathrm{CL}(d,q)=\mathrm{SU}(d,q)$. Then

$$C_2 \sum_{\lceil \log(d)\rceil \le k \le 2\lceil \log(d)\rceil} \Big(1-\frac{2}{d-2k-1}\Big)\frac{1}{k} \ge C_2 \int_{\lceil \log(d)\rceil}^{2\lceil \log(d)\rceil} \Big(1-\frac{2}{d-2x-1}\Big)\frac{1}{x}dx$$
$$= C_2\Big[\frac{(d-3)\log(x)+2\log(d-2x-1)}{d-1}\Big]_{\lceil \log(d)\rceil}^{2\lceil \log(d)\rceil}$$

and

$$\begin{aligned}
&\Big[\frac{(d-3)\log(x)+2\log(d-2x-1)}{d-1}\Big]_{\lceil \log(d)\rceil}^{2\lceil \log(d)\rceil}\\
=&\frac{d-3}{d-1}\Big(\log(2\lceil \log(d)\rceil)-\log(\lceil \log(d)\rceil)\Big)+\frac{2}{d-1}\Big(\log(d-4\lceil \log(d)\rceil-1)-\log(d-2\lceil \log(d)\rceil-1)\Big)\\
=&\frac{d-3}{d-1}\log\Big(\frac{2\lceil \log(d)\rceil}{\lceil \log(d)\rceil}\Big)+\frac{2}{d-1}\log\Big(\frac{d-4\lceil \log(d)\rceil-1}{d-2\lceil \log(d)\rceil-1}\Big)\\
=&\frac{d-3}{d-1}\log(2)+\frac{2}{d-1}\log\Big(\frac{d-4\lceil \log(d)\rceil-1}{d-2\lceil \log(d)\rceil-1}\Big)\\
=&\frac{d-3}{d-1}\log(2)+\frac{2}{d-1}\log\Big(1-\frac{2\lceil \log(d)\rceil}{d-2\lceil \log(d)\rceil-1}\Big).
\end{aligned}$$

By setting $x \coloneqq -\frac{2\lceil \log(d)\rceil}{d-2\lceil \log(d)\rceil-1} > -1$ the logarithm satisfies $\log(1+x) > \frac{x}{1+x}$ and by setting $u \coloneqq \lceil \log(d)\rceil$ this yields

$$\frac{2}{d-1}\log(1-\frac{2u}{d-2u-1}) \ge \frac{2}{(d-1)}\cdot\frac{-2u/(d-2u-1)}{1-2u/(d-2u-1)} = -\frac{4u}{(d-1)(d-4u-1)}.$$

Lastly,

$$-\frac{4u}{(d-1)(d-4u-1)} \ge -\frac{1}{d-1} \Leftrightarrow -\frac{4u}{(d-4u-1)} \ge -1 \Leftrightarrow -4u \ge 4u+1-d$$
$$\Leftrightarrow d \ge 8u+1 = 8\lceil \log(d)\rceil+1$$

which holds for $d \ge 33$. Therefore,

$$\sum_{\lceil \log(d)\rceil \le k \le 2\lceil \log(d)\rceil} P_{\mathfrak{s}}(k) \ge \frac{C_2}{d-1}((d-3)\log(2)-1),$$

which yields the first inequality if $\mathrm{CL}(d,q)=\mathrm{Sp}(d,q)$ or $\mathrm{CL}(d,q)=\Omega(d,q)$ and $d \ge 33$. Using

Corollary 10.2 we observe the following for the sharper bounds

$$\begin{aligned}\sum_{\lceil\log(d)\rceil\le k\le 2\lceil\log(d)\rceil} P_{\text{s}}(k) &\ge \sum_{\lceil\log(d)\rceil\le k\le 2\lceil\log(d)\rceil} \frac{1}{4e}\Big(1-\frac{1}{2k}\Big)\Big(1-\frac{2}{d-2k-1}\Big)\frac{1}{k}\\ &= \frac{1}{4e}\sum_{\lceil\log(d)\rceil\le k\le 2\lceil\log(d)\rceil} \Big(1-\frac{1}{2k}\Big)\Big(1-\frac{2}{d-2k-1}\Big)\frac{1}{k}\end{aligned}$$

and, therefore,

$$\begin{aligned}&\sum_{\lceil\log(d)\rceil\le k\le 2\lceil\log(d)\rceil} P_{\text{s}}(k)\\ &\ge \frac{1}{4e}\sum_{\lceil\log(d)\rceil\le k\le 2\lceil\log(d)\rceil} \Big(1-\frac{1}{2k}\Big)\Big(1-\frac{2}{d-2k-1}\Big)\frac{1}{k}\\ &\ge \frac{1}{4e}\int_{\lceil\log(d)\rceil}^{2\lceil\log(d)\rceil} \Big(1-\frac{1}{2x}\Big)\Big(1-\frac{2}{d-2x-1}\Big)\frac{1}{x}\mathrm{d}x\\ &= \frac{1}{4e}\Big[\frac{2(d^2-4d+5)x\log(x)+d^2-4d+3+4(d-2)x\log(d-2x-1)}{2(d-1)^2x}\Big]_{\lceil\log(d)\rceil}^{2\lceil\log(d)\rceil}.\end{aligned}$$

Further computations result in

$$\begin{aligned}P_{\text{stingray}}(d,q) &\ge \frac{1}{4e}\Big[\frac{2(d^2-4d+5)x\log(x)+d^2-4d+3+4(d-2)x\log(d-2x-1)}{2(d-1)^2x}\Big]_{\lceil\log(d)\rceil}^{2\lceil\log(d)\rceil}\\ &\ge \frac{1}{4e}\Big(\frac{(d^2-4d+5)\log(2)-d+2}{(d-1)^2}-\frac{d^2-4d+3}{4(d-1)^2\lceil\log(d)\rceil}\Big)\\ &\ge \frac{\log(2)}{4e}-\frac{2\log(2)}{4e(d-1)}-\frac{d-2}{4e(d-1)^2}-\frac{1}{16e\lceil\log(d)\rceil}\Big(\frac{d-2}{d-1}\Big)^2\end{aligned}$$

which proves Theorem 10.4. □

Example 10.5

For $d \ge 100$ we have $\log(d) \ge \log(100) \approx 4.6$ and $\lceil\log(100)\rceil = 5$. Hence,

$$\sum_{\lceil\log(d)\rceil\le k\le 2\lceil\log(d)\rceil} P_{\text{s}}(k) \ge \frac{\log(2)}{e}-\frac{1.32}{100}-\frac{1}{20e} \approx 0.2234$$

which implies that in $\mathrm{SL}(d,q)$ for $d \ge 100$ at least one in five elements powers up to ppd-stingray element with stingray body of dimension k and $k \le 2\lceil\log(d)\rceil$. ◀

Corollary 10.6

Let $\epsilon \geq 0$, $N \in \mathbb{N}$ and $d \geq 33$. Then the probability of failing to find a stingray pair (s_1, s_2) of degrees m_1 and m_2 with $\lceil \log(d) \rceil \leq m_1, m_2 \leq 2\lceil \log(d) \rceil$ in $\mathrm{CL}(d,q)$ in maximal N random selections is at most ϵ if $N \geq \max\{\log(\epsilon^{-1}) + 2, 1 + \frac{1}{1-x}\}$ where x denotes the probability of finding a pre-stingray candidate in one random selection.

Proof. In the following let x denote the probability of finding a pre-stingray candidate in one random selection. We are interested in the probability of failing to find two stingray elements in at most N random selection, i.e. the computation fails if we only compute one or zero stingray elements. Therefore,

$$\begin{aligned}
&P[\text{failing to find two stingray elements in } N \text{ tries}] \\
=&P[\{\text{no stingray element found in } N \text{ tries}\} \cup \{\text{one stingray element found in } N \text{ tries}\}] \\
=&P[\text{no stingray element found in } N \text{ tries}] + P[\text{one stingray element found in } N \text{ tries}] \\
=&(1-x)^N + \binom{N}{1} x(1-x)^{N-1} \\
=&(1-x)^N + Nx(1-x)^{N-1}.
\end{aligned}$$

Our next goal is to choose N such that $(1-x)^N + Nx(1-x)^{N-1} < \epsilon$ leading to

$$(1-x)^N + Nx(1-x)^{N-1} = (1-x)^{N-1}(1-x+Nx) \leq (1-x)^{N-1}(N-1) \leq \epsilon$$

as $1-x+Nx \leq N-1 \Leftrightarrow 1 + \frac{1}{1-x} \leq N$ which is satisfied for N large enough. Now set $M \coloneqq N-1$ and $y \coloneqq (1-x)$ such that $(1-x)^{N-1}(N-1) = y^M M$ and note that $y^M M$ is closely related to the product logarithm function as follows

$$y^M M = e^{\log(y^M)} M = e^{M\log(y)} M = \epsilon \Leftrightarrow e^{M\log(y)} M \log(y) = \epsilon \log(y).$$

As $0 > \epsilon \log(y) \geq -\frac{1}{e}$ for $\epsilon \in (0,1)$ small enough, a solution for $e^{M\log(y)} M \log(y) = \epsilon \log(y)$ is given by the Lambert W function $M \log(y) = W_{-1}(\epsilon \log(y))$. As $y = (1-x) \in (0,1)$ we can write $y = \frac{1}{a}$ and, thus, $\epsilon \log(y) = -\epsilon \log(a) = -e^{-(-\log(\epsilon \log(a))-1)-1} = -e^{-u-1}$ where $u \coloneqq -\log(\epsilon \log(a)) - 1$. Using [26,

Theorem 1]

$$W_{-1}(\epsilon \log(y)) \leq -1 - \sqrt{2u} - \frac{2}{3}u.$$

Overall,

$$M \log(y) = W_{-1}(\epsilon \log(y)) \Leftrightarrow M = \frac{W_{-1}(\epsilon \log(y))}{\log(y)}$$

and, therefore, we choose $M \geq \frac{1+\sqrt{2u}+\frac{2}{3}u}{\log(a)}$. Moreover, $\sqrt{2u} \leq \frac{u}{3}$ for $u \geq 16$ and, thus, choose $M \geq \frac{1+u}{\log(a)}$ yielding $M \geq \frac{\log(\epsilon^{-1}\log(\frac{1}{1-x}))}{\log(\frac{1}{1-x})}$. Now the claim follows as

$$\frac{\log(\epsilon^{-1}\log(\frac{1}{1-x}))}{\log(\frac{1}{1-x})} = \frac{\log(\epsilon^{-1}) + \log(\log(\frac{1}{1-x}))}{\log(\frac{1}{1-x})} = \frac{\log(\epsilon^{-1})}{\log(\underbrace{\frac{1}{1-x}}_{>1})} + \underbrace{\frac{\log(\log(\frac{1}{1-x}))}{\log(\frac{1}{1-x})}}_{<1} \leq \log(\epsilon^{-1}) + 1 \leq M$$

and $\log(\epsilon^{-1}) + 1 \leq M = N - 1 \Leftrightarrow N \geq \log(\epsilon^{-1}) + 2$. □

Definition 10.7

Let $\mathrm{CL}(d,q)$ be a classical group. Let $P_{\text{pair}}(d,q)$ denote the probability that we obtain two ppd-stingray elements $s_1, s_2 \in \mathrm{CL}(d,q)$ with stingray bodies W_{s_i}, $\dim(W_{s_i}) = n_i$ and $\lceil \log(d) \rceil \leq n_i \leq 2\lceil \log(d) \rceil$ in two independent selections of random elements in $\mathrm{CL}(d,q)$.

Corollary 10.8

Let $\mathrm{CL}(d,q)$ be a classical group. If $\mathrm{CL}(d,q)$ is a special linear or unitary group, then

$$P_{\text{pair}}(d,q) = P_{\text{stingray}}(d,q)^2 \geq \left(\frac{\log(2)}{e} - \frac{1.32}{d} - \frac{1}{4e\lceil \log(d) \rceil}\right)^2$$

and if $\mathrm{CL}(d,q)$ is a symplectic or orthogonal group, then

$$P_{\text{pair}}(d,q) = P_{\text{stingray}}(d,q)^2 \geq \left(\frac{\log(2)}{4e} - \frac{2\log(2)}{4e(d-1)} - \frac{d-2}{4e(d-1)^2} - \frac{1}{16e\lceil \log(d) \rceil}\left(\frac{d-2}{d-1}\right)^2\right)^2.$$

10.1.2 Conditional probability that a stingray pair forms a stingray duo

In this section we compute lower bounds for the probability, that two ppd-stingray elements $s_1, s_2 \in G = \mathrm{CL}(d,q)$ form a stingray duo. Therefore, we assume in the entire section, that s_1, s_2

are ppd-stingray elements which are computed using the algorithms of Chapter 4. Moreover, let W_{s_1} and W_{s_2} be the stingray bodies of s_1 and s_2 and $\dim(W_{s_i}) = n_i$. The treatment presented in this section is based on [42].

Definition 10.9

Let $\mathrm{CL}(d,q)$ be a classical group and $s_1, s_2 \in \mathrm{CL}(d,q)$ ppd-stingray elements. Then (s_1, s_2) is a *ppd-stingray pair*. Moreover, (s_1, s_2) is a *ppd-stingray duo* if (s_1, s_2) is a stingray duo.

Our aim is to find a lower bound for the conditional probability that a ppd-stingray pair (s_1, s_2) of elements in $G \times G$ is a ppd-stingray duo namely to prove that

$$P((s_1, s_2) \text{ stingray duo} \mid (s_1, s_2) \text{ stingray pair}) \geq 1 - \frac{C_3}{q} \tag{10.1.2}$$

where C_3 is a positive constant.

Let $\mathscr{C}_i$ be the G-conjugacy class of s_i. Note that

$$P((s_1, s_2) \text{ stingray duo} \mid (s_1, s_2) \text{ stingray pair}) = \frac{|\{(s_1, s_2) \in \mathscr{C}_1 \times \mathscr{C}_2 \mid (s_1, s_2) \text{ is a stingray duo}\}|}{|\mathscr{C}_1 \times \mathscr{C}_2|}.$$

First, we observe the following connection.

Lemma 10.10

Let $G = \mathrm{CL}(d,q)$ be a classical group in its natural representation and let $s \in G$ be a stingray element with stingray body W_s and stingray tail E_s. Then W_s is the unique $\langle s \rangle$-invariant submodule of $\mathbb{F}_q^d$ on which s acts non-trivially and irreducibly.

Proof. [42, Lemma 3.7 (c)]. Note that [42, Lemma 3.7 (c)] is only proven for symplectic, unitary and orthogonal groups but the claim also holds for special linear groups with the same argument. □

The previous lemma suggests that it might be possible to count pairs of subspaces instead of pairs of elements. Note that we search for a stingray pair by seeking a pair $(\mathfrak{s}_1, \mathfrak{s}_2)$ of pre-stingray candidates with ppd-stingray property and raising each to some power. This yields a ppd-stingray pair $(s_1, s_2) \in G \times G$. Suppose that s_1 and s_2 lie in G-conjugacy classes $\mathscr{C}_1$ and $\mathscr{C}_2$, respectively. By Lemma

10.10 W_{s_i} is the unique $\langle s_i \rangle$-invariant subspace of $\mathbb{F}_q^d$ on which s_i acts non-trivially and irreducibly. First, assume CL = SL. Then $\mathcal{U}_i := \{W_g \mid g \in \mathcal{C}_i\}$ is a G-orbit of subspaces of dimension n_i of $\mathbb{F}_q^d$, see [42, p. 3.2.1], and the G-conjugacy class $\mathcal{C}_i$ is in one-to-one correspondence to the G-orbit $\mathcal{U}_i$. Second, let CL $\neq$ SL. Since s_i is a ppd-stingray element W_{s_i} is non-degenerate by Lemma 4.21. Hence, $\mathcal{U}_i := \{W_g \mid g \in \mathcal{C}_i\}$ is a G-orbit of non-degenerate subspaces of dimension n_i of $\mathbb{F}_q^d$ and the G-conjugacy class $\mathcal{C}_i$ is in one-to-one correspondence to the G-orbit $\mathcal{U}_i$. Next we show how $|\mathcal{C}_i|$ and $|\mathcal{U}_i|$ relate.

We follow [42, Section 3] but show the results for all classical groups. We start with [42, Lemma 3.11] which provides a formula for $|\mathcal{C}_i|$.

Lemma 10.11: [42, Lemma 3.11]

Let $\langle X \rangle = G = \mathrm{CL}(d,q)$ be a classical group and let $s \in G$ be a stingray element. Let $\mathcal{C}$ be the G-conjugacy class of s and $\mathcal{U} := \{W_g \mid g \in \mathcal{C}\}$. Let W_s be the stingray body and let $W := W_s$. Then

$$|\mathcal{C}| = |\mathcal{U}| \cdot |G_W : C_G(s)|,$$

where G_W is the stabiliser of W and $C_G(s)$ the centraliser of s. Moreover, there are precisely $\frac{|\mathcal{C}|}{|\mathcal{U}|}$ stingray elements $s' \in \mathcal{C}$ such that $W_{s'} = W$.

Proof. [42, Lemma 3.11]. Note that [42, Lemma 3.11] is only proven for symplectic, unitary and orthogonal groups but the claim also holds for special linear groups with the same argument. □

In the following we prove that using the one-to-one correspondence between the G-orbits of subspaces and G-conjugacy classes, the conditional probability given in equation (10.1.2) can be expressed using subspaces of $\mathbb{F}_q^d$ instead of stingray elements and their conjugacy classes. Therefore, this section is structured as follows. First, we show that the probability given in equation (10.1.2) can be formulated as a subspace problem of $\mathbb{F}_q^d$. Second, we compute lower bounds for the subspace proportions.

Definition 10.12: [42, Section 2.1]

Let $\langle X\rangle = G = \mathrm{CL}(d,q)$ be a classical group and let $(s_1,s_2)\in G$ be a ppd-stingray pair. Let $\mathscr{C}_i$ be the G-conjugacy class of s_i and $\mathscr{U}_i := \{W_g \mid g\in\mathscr{C}_i\}$.

1) Let $U_1\in\mathscr{U}_1$ and $U_2\in\mathscr{U}_2$. In $\mathrm{SL}(d,q)$ the pair (U_1,U_2) is a *subspace-duo* if $U_1\cap U_2=\{0\}$. In $\mathrm{Sp}(d,q)$, $\mathrm{SU}(d,q)$ and $\Omega(d,q)$ the pair (U_1,U_2) is a *subspace-duo* if $U_1\cap U_2=\{0\}$ and U_1,U_2 and $U_1\oplus U_2$ are non-degenerate.

2) We define

$$P(\mathscr{U}_1,\mathscr{U}_2) := \frac{|\text{subspace-duos in } \mathscr{U}_1\times\mathscr{U}_2|}{|\mathscr{U}_1|\cdot|\mathscr{U}_2|}.$$

Lemma 10.13

Let $\langle X\rangle = G = \mathrm{CL}(d,q)$ be a classical group and let $(s_1,s_2)\in G$ be a stingray pair. Let $\mathscr{C}_i$ be the G-conjugacy class of s_i and $\mathscr{U}_i := \{W_g \mid g\in\mathscr{C}_i\}$. Then

$$\frac{|\{(s_1,s_2)\in\mathscr{C}_1\times\mathscr{C}_2 \mid (s_1,s_2) \text{ is a stingray duo}\}|}{|\mathscr{C}_1\times\mathscr{C}_2|} = P(\mathscr{U}_1,\mathscr{U}_2).$$

Proof. [42, Lemma 3.12]. Note that [42, Lemma 3.12] is only proven for symplectic, unitary and orthogonal groups but the claim also holds for special linear groups with the same argument. □

Lemma 10.13 proves that instead of computing the proportion of ppd-stingray duos among the ppd-stingray pairs, we can compute the proportion of subspace-duos among the subspace conjugacy classes. Therefore, our next aim is to compute lower bounds for $P(\mathscr{U}_1,\mathscr{U}_2)$.

Lemma 10.14

Let $V=\mathbb{F}^d$ be a vector space over a finite field $\mathbb{F}$ and let $\mathscr{U}_i := \{W_g \mid g\in\mathscr{C}_i\}$ be a G-orbit of subspaces of dimension n_i of $\mathbb{F}_q^d$ for $i=1,2$ such that $n_1+n_2\le d$. Then

$$P(\mathscr{U}_1,\mathscr{U}_2) = \prod_{i=1}^{n_2}\frac{1-q^{-i}}{1-q^{-n-i}} \ge 1-\frac{3}{2q}.$$

Proof. [39, Lemma 2.1]. □

Lemma 10.14 only yields a lower bound for $P(\mathcal{U}_1, \mathcal{U}_2)$ of special linear groups. For symplectic, unitary and orthogonal group we additionally have to include that the subspaces and their sum are non-degenerate as in Definition 10.12 2). A lower bound is given in the next lemma.

Lemma 10.15

Let $V = \mathbb{F}^d$ be a vector space over a finite field $\mathbb{F}$ equipped with a non-degenerate symplectic, unitary or quadratic form and let $\mathcal{U}_i := \{W_g \mid g \in \mathcal{C}_i\}$ be a G-orbit of non-degenerate subspaces of dimension n_i of $\mathbb{F}_q^d$ for $i = 1,2$ such that $n_1 + n_2 \leq d$. Let $c = 3.125$. Then

$$P(\mathcal{U}_1, \mathcal{U}_2) \geq 1 - \frac{c}{q} \geq 0.$$

Proof. [42, Theorem 1.1]. □

In the following we use $P_{\text{duo}}(d,q)$ as the conditional probability that a ppd-stingray pair forms a ppd-stingray duo, i.e. $P_{\text{duo}}(d,q) = P((s_1,s_2) \text{ stingray duo} \mid (s_1,s_2) \text{ stingray pair})$.

Theorem 10.16

Let $\langle X \rangle = G = \text{CL}(d,q)$ be a classical group and let $(s_1,s_2) \in G \times G$ be a ppd-stingray pair. Then there exists a positive constant C_3 such that

$$P_{\text{duo}}(d,q) = P((s_1,s_2) \text{ stingray duo} \mid (s_1,s_2) \text{ stingray pair}) \geq 1 - \frac{C_3}{q}.$$

Proof. Suppose that s_1 and s_2 lie in G-conjugacy classes $\mathcal{C}_1$ and $\mathcal{C}_2$, respectively. By Lemma 10.10 W_{s_i} is the unique $\langle s_i \rangle$ invariant subspace of $\mathbb{F}_q^d$ on which s_i acts non-trivially and irreducibly. Therefore, there is a one-to-one correspondence between ppd-stingray elements and subspaces of $\mathbb{F}_q^d$. Let $\mathcal{U}_i := \{W_g \mid g \in \mathcal{C}_i\}$. By Lemma 10.13

$$\frac{|\{(s_1,s_2) \in \mathcal{C}_1 \times \mathcal{C}_2 \mid (s_1,s_2) \text{ is a stingray duo}\}|}{|\mathcal{C}_1 \times \mathcal{C}_2|} = P(\mathcal{U}_1, \mathcal{U}_2).$$

If CL = SL, then by Lemma 10.14

$$P(\mathcal{U}_1, \mathcal{U}_2) \geq 1 - \frac{3}{2q}$$

and if CL $\neq$ SL, then by Lemma 10.15

$$P(\mathscr{U}_1, \mathscr{U}_2) \geq 1 - \frac{3.125}{q} \geq 0.$$

Thus,

$$\begin{aligned} P((s_1,s_2) \text{ stingray duo} \mid (s_1,s_2) \text{ stingray pair}) &= \frac{|\{(s_1,s_2) \in \mathscr{C}_1 \times \mathscr{C}_2 \mid (s_1,s_2) \text{ is a stingray duo}\}|}{|\mathscr{C}_1 \times \mathscr{C}_2|} \\ &= P(\mathscr{U}_1, \mathscr{U}_2) \\ &\geq 1 - \frac{C_3}{q} \end{aligned}$$

for $C_3 \coloneqq \frac{3}{2}$. □

10.1.3 Conditional probability that a stingray duo generates a classical group

For an analysis of the GoingDown algorithms of this thesis it is essential to estimate the probability that a stingray duo generates a classical group. Let $s_1, s_2 \in G = \mathrm{CL}(d,q)$ with stingray bodies W_{s_1}, W_{s_2} and $n_i = \dim(W_{s_i})$ for $i \in \{1,2\}$ such that (s_1,s_2) is a stingray duo as in Definition 4.22. Then we denote by $P_{\text{gen}}(d,q)$ the conditional probability that a stingray duo of G generates G, i.e. $P_{\text{gen}}(d,q) = P(\langle s_1, s_2\rangle = G \mid (s_1,s_2) \text{ stingray duo of } G)$.

We summarise the results from [40].

Theorem 10.17: [40]

Let $d = n_1 + n_2$ with $1 \leq n_1 \leq n_2$. Let $q > 1$ be a prime power. Then there exists a positive constant C_4 such that

$$1 - q^{-1} - C_4 q^{-2} < P_{\text{gen}}(d,q) < 1 - q^{-1} - q^{-2} + 2q^{-3} - 2q^{-5}.$$

10.1.4 Probability of the GoingDown basic step

In this section we compute the probability that a GoingDown basic step of a classical group $G = \mathrm{CL}(d,q)$ is successful in at most N random selection. For this we combine the probability to find a stingray pair as discussed in Section 10.1.1, the conditional probability to have a stingray duo given a stingray pair as discussed in Section 10.1.2 and the conditional probability that a stingray

duo of G generates G. Let $P_{\text{step}}(d,q)$ be the probability that one GoingDown basic step succeeds with two random selections. Then we have

$$P_{\text{step}}(d,q) = P_{\text{pair}}(d,q)P_{\text{duo}}(d,q)P_{\text{gen}}(d,q)$$

Using the results of the previous sections we can give the following lower bounds for $P_{\text{step}}(d,q)$.

Theorem 10.18

Let $\mathrm{CL}(d,q)$ be a classical group. Then there exists positive constants C_3 and C_4 such that if $\mathrm{CL}(d,q) = \mathrm{SL}(d,q)$ or $\mathrm{CL}(d,q) = \mathrm{SU}(d,q)$ and $d \geq 12$, then one GoingDown basic step succeeds with two random selections with probability at least

$$P_{\text{step}}(d,q) \geq \left(\frac{\log(2)}{e} - \frac{1.32}{d} - \frac{1}{4e\lceil \log(d)\rceil}\right)^2 \left(1 - \frac{C_3}{q}\right)\left(1 - \frac{1}{q} - \frac{C_4}{q^2}\right).$$

If $\mathrm{CL}(d,q) = \mathrm{Sp}(d,q)$ or $\Omega(d,q) = \mathrm{SU}(d,q)$ and $d \geq 33$, then one GoingDown basic step succeeds with two random selections with probability at least

$$P_{\text{step}}(d,q) \geq \left(\frac{\log(2)}{4e} - \frac{2\log(2)}{4e(d-1)} - \frac{d-2}{4e(d-1)^2} - \frac{1}{16e\lceil \log(d)\rceil}\left(\frac{d-2}{d-1}\right)^2\right)^2 \left(1 - \frac{C_3}{q}\right)\left(1 - \frac{1}{q} - \frac{C_4}{q^2}\right).$$

10.1.5 Probability of the GoingDown algorithm

In Theorem 10.18 of Section 10.1.4 we computed the probability that a GoingDown basic step with exactly two selections of random elements is successful. Using these results we compute an upper bound for the required number of random selections to guarantee that the GoingDown algorithm without the GoingDown final step is successful with probability at least $1-\epsilon$ for a given $\epsilon \in (0,1)$.

Definition 10.19

Let k be a non-negative integer. We define

$$\log_i(k) := \begin{cases} k, & \text{if } i = 0, \\ \log(k), & \text{if } i = 1 \text{ and} \\ \log(\log_{i-1}(k)), & \text{if } i > 1. \end{cases}$$

Recall that the iterated logarithm $\log^*$ counts the number of times the logarithm function must be applied recursively until a non-negative real number is equal or less than 1, see Definition 2.62.

Theorem 10.20

Let $\mathrm{CL}(d,q)$ be a classical group and let $\epsilon \in (0,1)$. Let

$$N := \frac{2\log^*(d)\log(\epsilon)}{\log\left(1-\prod_{i=1}^{\log^*(d)} P_{\text{step}}(\log_i(d),q)\right)}.$$

If the GoingDown algorithm without the GoingDown final step selects at least N random elements, then the GoingDown algorithm without the GoingDown final step succeeds with probability at least $1-\epsilon$.

Proof. Using Definition 10.19

$$\begin{aligned}
P[\text{success after } N \text{ random selections}] &\geq P\left[\text{success after } \left\lfloor \frac{N}{2\log^*(d)} \right\rfloor \text{ runs}\right] \\
&= 1-P[\text{run not successful}]^{\left\lfloor \frac{N}{2\log^*(d)} \right\rfloor} \\
&= 1-\left(1-\prod_{i=1}^{\log^*(d)} P_{\text{step}}(\log_i(d),q)\right)^{\left\lfloor \frac{N}{2\log^*(d)} \right\rfloor} \\
&\geq 1-\epsilon.
\end{aligned}$$

This is now equivalent to

$$\begin{aligned}
& 1-\left(1-\prod_{i=1}^{\log^*(d)} P_{\text{step}}(\log_i(d),q)\right)^{\left\lfloor \frac{N}{2\log^*(d)} \right\rfloor} \geq 1-\epsilon \\
\Leftrightarrow\ & \epsilon > \left(1-\prod_{i=1}^{\log^*(d)} P_{\text{step}}(\log_i(d),q)\right)^{\left\lfloor \frac{N}{2\log^*(d)} \right\rfloor} \\
\Leftrightarrow\ & \log(\epsilon) > \left\lfloor \frac{N}{2\log^*(d)} \right\rfloor \log\left(1-\prod_{i=1}^{\log^*(d)} P_{\text{step}}(\log_i(d),q)\right) \\
\Leftrightarrow\ & \frac{\log(\epsilon)}{\log\left(1-\prod_{i=1}^{\log^*(d)} P_{\text{step}}(\log_i(d),q)\right)} < \left\lfloor \frac{N}{2\log^*(d)} \right\rfloor \leq \frac{N}{2\log^*(d)}
\end{aligned}$$

which proves the claim. □

Corollary 10.21

In the StandardGenerators algorithm a function $\text{MaximalRandomSelections}_{\text{CL}}(d,q,\epsilon)$ is used to compute for a given $\epsilon > 0$ three positive integers $N_1, N_2, N_3 \in \mathbb{N}$ such that if the GoingDown algorithm selects at most N_1 random elements, the BaseCase algorithm selects at most N_2 random elements and the GoingUp algorithm selects at most N_3 random elements, then the StandardGenerators algorithm succeeds with probability at least $1-\epsilon$. Using Theorem 10.20 we can set

$$N_1 := \frac{2\log^*(d)\log(\epsilon)}{\log\left(1-\prod_{i=1}^{\log^*(d)} P_{\text{step}}(\log_i(d),q)\right)}.$$

Lemma 10.22

Let $\text{CL}(d,q)$ be a classical group. If we use a naming algorithm for every GoingDown basic step to verify its success, then the expected number N of selected random elements for all GoingDown basic steps is

$$\mathbb{E}[N] := \sum_{i=0}^{\lceil \log^*(d)\rceil -1} \frac{2}{P_{\text{step}}(\log_i(d),q)}.$$

Proof. The proof is based on a well-known principle in probability theory. In the following let N_i be the number of random selections until the i-th GoingDown basic step is successful and let $P_i = P_{\text{step}}(\log_i(d),q)$. Then

$$\begin{aligned}
\mathbb{E}[N] = \mathbb{E}[N_1 + \ldots + N_{\lceil \log^*(d)\rceil -1}] &= \sum_{i=0}^{\lceil \log^*(d)\rceil -1} \mathbb{E}[N_i] \\
&= \sum_{i=0}^{\lceil \log^*(d)\rceil -1} \sum_{k=1}^{\infty} kP[\text{success after exactly } k \text{ selections}] \\
&= \sum_{i=0}^{\lceil \log^*(d)\rceil -1} \sum_{k=1}^{\infty} 2kP[\text{success after exactly } 2k \text{ selections}] \\
&= \sum_{i=0}^{\lceil \log^*(d)\rceil -1} \sum_{k=1}^{\infty} 2kP_i(1-P_i)^{k-1} \\
&= \sum_{i=0}^{\lceil \log^*(d)\rceil -1} 2P_i \sum_{k=1}^{\infty} k(1-P_i)^{k-1}
\end{aligned}$$

and

$$\begin{aligned}
&\sum_{i=0}^{\lceil \log^*(d)\rceil-1} 2P_i \sum_{k=1}^{\infty} k(1-P_i)^{k-1} \\
&= \sum_{i=0}^{\lceil \log^*(d)\rceil-1} 2P_i \sum_{k=1}^{\infty} \frac{\mathrm{d}}{\mathrm{d}P_i}\left(-(1-P_i)^k\right) \\
&= \sum_{i=0}^{\lceil \log^*(d)\rceil-1} (-2)P_i \frac{\mathrm{d}}{\mathrm{d}P_i}\left(\sum_{k=1}^{\infty}(1-P_i)^k\right) \\
&= \sum_{i=0}^{\lceil \log^*(d)\rceil-1} (-2)P_i \frac{\mathrm{d}}{\mathrm{d}P_i}\frac{1}{P_i} = \sum_{i=0}^{\lceil \log^*(d)\rceil-1} (-2)P_i(-1)\frac{1}{P_i^2} = \sum_{i=0}^{\lceil \log^*(d)\rceil-1} \frac{2}{P_i}.
\end{aligned}$$

□

Example 10.23

For SL(100, 121) Lemma 10.22 yields an expected value on the selected random elements of 303. ◀

10.1.6 Complexity results

In this section we compute the complexity of the GoingDown algorithm. Recall that the GoingDown algorithm is a one-sided Monte Carlo algorithm and, therefore, a randomised algorithm. Thus, probability results about successfully computing suitable elements by random procedure are required which was accomplished in the previous sections. Thus, these results are used for the complexity analysis of the GoingDown algorithm in this section.

The GoingDown algorithm calls the GoingDown basic step repeatedly until a terminal group is reached and afterwards a final step algorithm to compute a stingray embedded base case group in the terminal group. Thus, a complexity analysis of the GoingDown algorithm depends on the complexity of the GoingDown basic step and the complexity of the GoingDown final step.

We start with a complexity analysis of the final step. Note that for the final step we are dealing with a classical group CL(d, q) for a specific d as given in Table 3.1.

Lemma 10.24

Let CL(d, q) be a terminal group in its natural representation. The complexity of the GoingDown final step is at most

$$\mathcal{O}(\zeta \log^2(q) + \log^4(q) + \mathfrak{X}(q)\log(q))$$

if q is even and at most

$$\mathcal{O}(\zeta + \log(q) + \mathfrak{X}(q) + \mathfrak{X}(q^2))$$

if q is odd where ζ denotes an upper bound on the number of field operations for computing a random element in $\mathrm{CL}(d,q)$ and $\mathfrak{X}$ denotes an upper bound on the required number of field operations for solving the discrete logarithm in $\mathbb{F}_q$.

Proof. We recall well-known results from the literature in Table 10.3. Note that we refer to results for constructive recognition of classical groups for a specific d from which we can derive (standard) generators for a stingray embedded base case group. Note that constructive recognition of terminal groups has worse complexity than the actual GoingDown final step.

Group	$\mathcal{O}$	Reference
$\mathrm{SL}(4,q)$	$\mathcal{O}(\zeta \log(\log(q)) + \log(q))$	[59, Theorem 13.1]
$\mathrm{Sp}(6,q)$ and q odd	$\mathcal{O}(\zeta + \log(q) + \mathfrak{X}(q))$	[59, Theorem 1.1]
$\mathrm{SU}(6,q)$ and q odd	$\mathcal{O}(\zeta + \log(q) + \mathfrak{X}(q))$	[59, Theorem 1.1]
$\mathrm{SU}(10,q)$ and q even	$\mathcal{O}(\zeta \log^2(q) + \log^4(q) + \mathfrak{X}(q)\log(q))$	[32, Theorem 1.2]
$\Omega(8,q)$ and q even	$\mathcal{O}(\zeta \log^2(q) + \log^4(q) + \mathfrak{X}(q)\log(q))$	[32, Theorem 1.2]
$\Omega(8,q)$ and q odd	$\mathcal{O}(\zeta + \log(q) + \mathfrak{X}(q) + \mathfrak{X}(q^2))$	[59, Theorem 1.1]

Table 10.3: Complexity results for constructive recognition of terminal groups and corresponding references.

□

Remark 10.25

In the proof of Lemma 10.24 we refer to results from the literature which form an upper bound for the actual complexity of the GoingDown final step. Note that the GoingDown final step relies on results from the literature, see Table 3.2, and can be replaced by an algorithm designed for the actual aim of the GoingDown final step. Therefore, we do not perform a complexity analysis of the GoingDown final step in this thesis and use the upper bounds of the GoingDown final step given in Lemma 10.24. Moreover, we denote the complexity of the GoingDown final step for all classical groups in the following by $\mathfrak{Z}(q)$. ◀

We continue this section by computing the complexity of the GoingDown basic step. The

GoingDown basic step calls Algorithm FindStingrayElement [Alg. 5] or FindSelfReciprocalStingrayElement [Alg. 7] depending on the classical group which is why we start by analysing these two algorithms.

Lemma 10.26

Let $\mathrm{CL}(d,q)$ be a classical group in its natural representation. The complexity of one iteration of Algorithm FindStingrayElement [Alg. 5] and Algorithm FindSelfReciprocalStingrayElement [Alg. 7] is

$$\mathcal{O}(d^3\log(d)+d^2\log(d)\log(\log(d))\log(q)+\zeta)$$

where ζ denotes an upper bound on the number of field operations for computing a random element in $\mathrm{CL}(d,q)$.

Proof. The computation of a random element is bounded by ζ and the computation of the characteristic polynomial by $\mathcal{O}(d^3)$ see Remark 2.66. The computation of s^B has complexity $\mathcal{O}(d^3\log(d)+d^2\log(d)\log(\log(d))\log(q))$ by [59, Lemma 10.1] which proves the claim. □

Using the complexity analysis of Algorithm FindStingrayElement [Alg. 5] and Algorithm FindSelfReciprocalStingrayElement [Alg. 7] we can now compute the complexity of one GoingDown basic step. Note that the analysis of one GoingDown basic step involves the probability results of the previous sections.

Lemma 10.27

Let $\mathrm{CL}(d,q)$ be a classical group in its natural representation. The complexity of one GoingDown basic step is

$$\mathcal{O}(d^3\log(d)+d^2\log(d)\log(\log(d))\log(q)+\log(d)^4\log(q)^2+\log(d)\zeta'+\zeta)$$

where ζ denotes an upper bound on the number of field operations for computing a random element in $\mathrm{CL}(d,q)$ and ζ' denotes an upper bound on the number of field operations for computing a random element in $\mathrm{CL}(\mathcal{O}(\log(d)),q)$.

Proof. Note that by the results in Section 10.1 the GoingDown basic step succeeds with at most $\mathcal{O}(1)$

random selections. Thus, Algorithm FindStingrayElement [Alg. 5], respectively FindSelfReciprocalStingrayElement [Alg. 7], is called a constant amount of times in the GoingDown basic step. Algorithm FindSelfReciprocalStingrayElement [Alg. 7] and FindSelfReciprocalStingrayElement [Alg. 7] have complexity given in Lemma 10.26. Algorithm ComputeStingrayBody [Alg. 9] and IsStingrayDuo [Alg. 10] have complexity $\mathcal{O}(d^3)$ since these algorithms are applications of the Gaussian elimination algorithm. Lastly, verifying that $\langle s_1, s_2 \rangle$ is isomorphic to a classical group corresponds to two calls of Algorithm InducedActionRepresentation [Alg. 11], which has complexity $\mathcal{O}(d^3)$, and using a naming algorithm on $\mathcal{O}(\log(d)) \times \mathcal{O}(\log(d))$ matrices has complexity $\mathcal{O}(\log(\log(\log(d)))(\zeta' + \log(d)^3 \log(q)^2))$, see [76, Section 6.1], which proves the claim. □

Corollary 10.28

Let $\mathrm{CL}(d,q)$ be a classical group in its natural representation. The complexity of one GoingDown basic step without calling a naming algorithm is

$$\mathcal{O}(d^3 \log(d) + d^2 \log(d) \log(\log(d)) \log(q) + \zeta)$$

where ζ denotes an upper bound on the number of field operations for computing a random element in $\mathrm{CL}(d,q)$.

Using Lemma 10.24 and 10.27 we compute the complexity of the GoingDown algorithm.

Theorem 10.29

Let $\mathrm{CL}(d,q)$ be a classical group in its natural representation. The complexity of the GoingDown algorithm is

$$\mathcal{O}(d^3 \log(d) + d^2 \log(d) \log(\log(d)) \log(q) + \log(d)^4 \log(q)^2 + \log(d)\zeta' + \zeta + \mathfrak{Z}(q))$$

where ζ denotes an upper bound on the number of field operations for computing a random element in $\mathrm{CL}(d,q)$, ζ' denotes an upper bound on the number of field operations for computing a random element in $\mathrm{CL}(\mathcal{O}(\log(d)),q)$ and $\mathfrak{Z}(q)$ denotes an upper bound on the complexity for the final step.

Proof. Using Lemma 10.27 we know the complexity of one GoingDown basic step and the algorithm

has to be called $\mathcal{O}(\log^*(d))$ times. Using the results from Section 5.1.4 the matrices can be represented as $\mathcal{O}(\log(d)) \times \mathcal{O}(\log(d))$ matrices after the first successful GoingDown basic step. Thus, the complexity of the GoingDown algorithm is given by

$$\mathcal{O}\Big(d^3\log(d) + d^2\log(d)\log(\log(d))\log(q) + \log(d)^4\log(q)^2 + \log(d)\zeta' + \zeta + \log^*(d)\big(\log(d)^3\log(\log(d)) + \log(d)^2\log(\log(d))\log(\log(\log(d)))\log(q) + \log(\log(d))^4\log(q)^2 \log(\log(d))\zeta'' + \zeta'\big) + \mathfrak{Z}(q)\Big)$$

which proves the claim. □

Corollary 10.30

Let $\mathrm{CL}(d,q)$ be a classical group in its natural representation. The complexity of the GoingDown algorithm without calling a naming algorithm is

$$\mathcal{O}(d^3\log(d) + d^2\log(d)\log(\log(d))\log(q) + \log^*(d)\zeta' + \zeta + \mathfrak{Z}(q))$$

where ζ denotes an upper bound on the number of field operations for computing a random element in $\mathrm{CL}(d,q)$, ζ' denotes an upper bound on the number of field operations for computing a random element in $\mathrm{CL}(\mathcal{O}(\log(d)),q)$ and $\mathfrak{Z}(q)$ denotes an upper bound on the complexity for the final step.

10.2 Complexity of the BaseCase algorithm

In this section we state a theorem which summarises the complexity of the BaseCase algorithm for all classical groups. The complexity of the BaseCase algorithm depends on whether the given classical group is an orthogonal group. Note that we do not deal with base case groups in this thesis and, therefore, the theorem is proven by results from the literature.

Theorem 10.31

Subject to the availability of a discrete logarithm oracle for $\mathbb{F}_q$, SLPs for standard generators and other elements of $\langle X\rangle = \mathrm{CL}(d,q)$ for $d \leq 4$ and $\mathrm{CL} \neq \Omega$ can be constructed in $\mathcal{O}(\zeta\log(\log(q)) + \log(q))$. Moreover, there is an $\mathcal{O}(\log(q)(\log^2(q) + \mathcal{E}\log(q) + \zeta))$-time Las Vegas algorithm which constructively recognises H, with probability greater than $3/4$, when given $\langle X\rangle = H \cong \Omega^+(6,q)$ and having

available a constructive recognition algorithm for $\mathrm{SL}(2,q)$ and a discrete log oracle. Here, ζ is the complexity to construct a (nearly) uniformly distributed random element of H as an MSLP in X and $\mathscr{E}$ is the complexity of constructively recognising (a homomorphic image of) $\mathrm{SL}(2,q)$.

Proof. For $\mathrm{CL} \neq \Omega$, see [59, Theorem 13.1] and for $\Omega^+(6,q)$, see Theorem 8.10. □

In the following we denote the complexity for constructively recognising a base case group by $\mathfrak{Y}(q)$. Thus, we do not have to distinguish between the complexity results for base case groups of orthogonal groups and the remaining classical groups.

Remark 10.32

In the StandardGenerators algorithm a function $\mathrm{MaximalRandomSelections}_{\mathrm{CL}}(d,q,\epsilon)$ is used to compute for a given $\epsilon > 0$ three positive integers $N_1, N_2, N_3 \in \mathbb{N}$ such that if the GoingDown algorithm selects at most N_1 random elements, the BaseCase algorithm selects at most N_2 random elements and the GoingUp algorithm selects at most N_3 random elements, then the StandardGenerators algorithm succeeds with probability at least $1-\epsilon$. As we are not dealing with base case groups in this thesis, we refer to the literature on how to choose N_2 for the BaseCase algorithm, see Table 3.4. ◀

10.3 Complexity of the GoingUp algorithm

In this section we compute the complexity of the GoingUp algorithm. Recall that both the GoingDown and GoingUp algorithm are randomised. Hence, a complexity analysis requires an analysis of successfully finding a prescribed element by selecting random elements. In contrast to the GoingDown algorithm we do not perform such a probability analysis in this thesis. Even though we are not proving any probability results of the GoingUp algorithm, we state a conjecture and practical tests show that this conjecture is realistic.

Conjecture 10.33

A conjugate of a strong doubling element in a classical group $\mathrm{CL}(d,q)$ in its natural representation can be found in $\mathcal{O}(1)$ selections of random elements.

A probability analysis of successfully finding a required element for a GoingUp step by selecting random elements is already in process but not discussed in this thesis.

Based on Conjecture 10.33 we continue analysing the complexity of the GoingUp algorithm. As for the GoingDown algorithm we start with a complexity analysis of the GoingUp step.

Theorem 10.34

Suppose Conjecture 10.33 is true. Let $G = \mathrm{CL}(d,q)$ be a classical group in its natural representation and let $\langle Y_n \rangle = H \in GL(d,q)$ be a stingray embedded subgroup with $H \cong \mathrm{CL}(n,q)$ of the same type as G. If G is a special linear or symplectic group, then the complexity of one GoingUp step is

$$\mathcal{O}(d^3 + \zeta)$$

where ζ denotes an upper bound on the number of field operations for computing a random element in $\mathrm{CL}(d,q)$. If G is a unitary or orthogonal group, then the complexity of one GoingUp step is

$$\mathcal{O}(d^3 + \zeta + \mathfrak{V}(q))$$

where $\mathfrak{V}(q)$ is an upper bound on the number of field operations for computing a square root.

Proof. Recall the seven phases we are performing during one GoingUp step for each classical group. Note that the phases slightly differ for each classical group but for a complexity analysis we can focus on the phases described in Remark 5.62 for special linear groups. We are discussing the complexity of each phase.

1) In the first phase we are constructing an MSLP for an element $t \in \mathrm{CL}(d,q)$ as a word in Y_n which can be done in $\mathcal{O}(1)$ for all classical groups. Note that t can also be written down as a matrix at a cost of $\mathcal{O}(d^2)$.

2) In the second phase we conjugate t by an element $a \in \mathrm{CL}(d,q)$ which has complexity $\mathcal{O}(d^3)$. Checking the conditions $(C1)$ and $(C2)$ for $\tilde{g} := t^a$ corresponds to applications of the Gaussian elimination algorithm which has complexity $\mathcal{O}(d^3)$. By Conjecture 10.33 we find a conjugate of a strong doubling element in $\mathcal{O}(1)$ selections of random elements. Hence, the second phase

has a complexity of $\mathcal{O}(d^3 + \zeta)$.

3) In the third phase we compute a conjugate $g := L^{-1}\tilde{g}L$ for a specific element $L \in H$ which has complexity $\mathcal{O}(d^3)$. Moreover, writing L as a word in the standard generators Y_n has complexity $\mathcal{O}(n^3)$. For unitary and orthogonal groups the computation of $L \in H$ requires a constant number of computations of square roots for which we use the upper bound $\mathfrak{V}(q)$.

4) In the fourth phase we compute a new base change matrix and conjugate g with the new base change matrix which has complexity $\mathcal{O}(d^3)$ resulting in a strong doubling element c. Writing down the new base change matrix involves an application of the Gaussian algorithm which also has complexity $\mathcal{O}(d^3)$.

5) In the fifth phase we write the standard basis vectors in a given basis which can be done in $\mathcal{O}(n^3)$ using the Gaussian elimination algorithm. Expressing all the coefficients in an $\mathbb{F}_p$-basis for $\mathbb{F}_q$ can be done in $\mathcal{O}(1)$.

6) The complexity of the sixth phase is at most the same as the one of the fifth phase.

7) In the seventh phase we compute an MSLP for the new permutation matrices of each classical groups using the elements computed in the fifth and sixth phase. This involves no computations which is why the seventh phase has complexity $\mathcal{O}(1)$.

Overall, the GoingUp step for special linear and symplectic groups has a complexity of

$$\mathcal{O}(1 + d^3 + \zeta + d^3 + d^3 + n^3 + n^3 + 1) \subseteq \mathcal{O}(d^3 + \zeta)$$

and for unitary and orthogonal groups has a complexity of

$$\mathcal{O}(1 + d^3 + \zeta + d^3 + \mathfrak{V}(q) + d^3 + n^2 + n^3 + 1) \subseteq \mathcal{O}(d^3 + \zeta + \mathfrak{V}(q))$$

which proves the claim. □

Remark 10.35

A square root in a finite field $\mathbb{F}_q$ can be computed in polynomial time using a discrete logarithm oracle but depending on q there are many cases where the square root can be computed with more efficient algorithms, see [1]. ◀

Using Theorem 10.34 we compute the complexity of the GoingUp algorithm.

Theorem 10.36

Suppose Conjecture 10.33 is true. Let $\mathrm{CL}(d,q)$ be a classical group in its natural representation. If G is a special linear or symplectic group, then the complexity of the GoingUp algorithm is

$$\mathcal{O}(d^3\frac{\log(d)}{\log(\log(d))}+\frac{\log(d)}{\log(\log(d))}\zeta)$$

where ζ denotes an upper bound on the number of field operations for computing a random element in $\mathrm{CL}(d,q)$. If G is a unitary or orthogonal group, then the complexity of the GoingUp algorithm is

$$\mathcal{O}(d^3\frac{\log(d)}{\log(\log(d))}+\frac{\log(d)}{\log(\log(d))}\zeta+\log(d)\mathfrak{V}(q))$$

where $\mathfrak{V}(q)$ is an upper bound on the number of field operations for computing a square root.

Proof. Recall that we are doubling the dimension n from a stingray embedded subgroup H in $\mathrm{CL}(d,q)$ in each GoingUp step, see Section 3.3. Using the results from Section 5.3.7 we are dealing with at most $\mathcal{O}(\log(d))\times\mathcal{O}(\log(d))$ matrices until n is roughly $\log(d)$. Thus, using Theorem 10.34 the complexity of the first GoingUp steps is

$$\mathcal{O}(\log(\log(d))(\log(d)^3+\zeta'))\subseteq\mathcal{O}(d^3+\log(\log(d))\zeta'+\zeta)$$

where ζ' denotes an upper bound on the number of field operations for computing a random element in $\mathrm{CL}(\log(d),q)$. From that point on computations are performed with $d\times d$ matrices. Hence, we have to use $d\times d$ matrices for $\log(d-\log(d))=\frac{\log(d)}{\log(\log(d))}$ GoingUp steps. Using the complexity for one GoingUp step given in Theorem 10.34 this yields the following complexity of the GoingUp

algorithm which is

$$\mathcal{O}(\frac{\log(d)}{\log(\log(d))}(d^3+\zeta)) = \mathcal{O}(d^3\frac{\log(d)}{\log(\log(d))} + \frac{\log(d)}{\log(\log(d))}\zeta).$$

Since $\mathcal{O}(d^3+\log(\log(d))\zeta'+\zeta) \subseteq \mathcal{O}(d^3\frac{\log(d)}{\log(\log(d))} + \frac{\log(d)}{\log(\log(d))}\zeta)$ this proves the claim. □

Corollary 10.37

Suppose Conjecture 10.33 is true. For a given $\epsilon > 0$ in the StandardGenerators algorithm a function MaximalRandomSelections$_{CL}(d,q,\epsilon)$ is used to compute three positive integers $N_1, N_2, N_3 \in \mathbb{N}$ such that if the GoingDown algorithm selects at most N_1 random elements, the BaseCase algorithm selects at most N_2 random elements and the GoingUp algorithm selects at most N_3 random elements, then the StandardGenerators algorithm succeeds with probability at least $1-\epsilon$. As we are not performing a probability analysis on the success of finding a strong doubling element element by selection of random elements in this thesis, we cannot discuss on how to choose N_3 for a given ϵ. In practice, $N_3 := 100$ is sufficient for all tests so far.

10.4 Complexity results of algorithms

In this section we compute the complexity of the StandardGenerators algorithm described in Section 3.4. Recall that the StandardGenerators algorithm is randomised and uses the subalgorithms GoingDown, BaseCase and GoingUp which have been analysed in Section 10.1, Section 10.2 and Section 10.3 respectively. Note that the complexity of the GoingUp algorithm is given in Theorem 10.36 based on Conjecture 10.33. Thus, the complexity of the StandardGenerators algorithm can also only be formulated based on Conjecture 10.33.

Theorem 10.38

Suppose Conjecture 10.33 is true. For a special linear or symplectic group the complexity of Algorithm StandardGenerators [Alg. 3] as stated in Theorem 3.10 is

$$\mathcal{O}(d^3\log(d) + d^2\log(d)\log(\log(d))\log(q) + \frac{\log(d)}{\log(\log(d))}\zeta + \mathfrak{Y}(q) + \mathfrak{Z}(q))$$

where ζ denotes an upper bound on the number of field operations required for computing a random element, $\mathfrak{Y}(q)$ denotes an upper bound on the number of field operations for constructively recognising a base case group and $\mathfrak{Z}(q)$ denotes an upper bound on the required number of field operations for the final step. For a unitary or orthogonal group the complexity of Algorithm StandardGenerators [Alg. 3] as stated in Theorem 3.10 is

$$\mathcal{O}(d^3\log(d)+d^2\log(d)\log(\log(d))\log(q)+\frac{\log(d)}{\log(\log(d))}\zeta+\mathfrak{Y}(q)+\mathfrak{Z}(q)+\log(d)\mathfrak{V}(q))$$

where $\mathfrak{V}(q)$ is an upper bound on the number of field operations for computing a square root.

Proof. Algorithm StandardGenerators [Alg. 3] calls the GoingDown, BaseCase and GoingUp algorithm in this order. In Corollary 10.30 we computed the complexity of the GoingDown algorithm, in Theorem 10.31 the complexity of the BaseCase algorithm and in Conjecture 10.36 the complexity of the GoingUp algorithm. Thus, this yields the following complexity of Algorithm StandardGenerators [Alg. 3]

$$\begin{aligned}&\mathcal{O}\Big(d^3\log(d)+d^2\log(d)\log(\log(d))\log(q)+\log^*(d)\zeta'+\zeta+\mathfrak{Z}(q)+\\&\quad\mathfrak{Y}(q)+d^3\frac{\log(d)}{\log(\log(d))}+\frac{\log(d)}{\log(\log(d))}\zeta\Big)\\&=\mathcal{O}\Big(d^3\log(d)+d^2\log(d)\log(\log(d))\log(q)+\frac{\log(d)}{\log(\log(d))}\zeta+\mathfrak{Y}(q)+\mathfrak{Z}(q)\Big).\end{aligned}$$

□

Chapter 11

Implementation

The algorithms GoingDown, BaseCase, GoingUp and StandardGenerators for all classical groups in this thesis have been implemented in GAP [37] by the author. However, the GoingUp algorithm based on involutions, see Chapter 9, has not been implemented in GAP due to numerous missing functionalities. Nevertheless, an implementation of this algorithm has been tested in Magma [16]. The location of the GAP implementation is described in Section 11.1.

In this chapter we also compare the run-time of the constructive recognition algorithms from this thesis with the DLLO constructive recognition algorithm [59, 32]. The constructive recognition algorithms of this thesis and the DLLO constructive recognition algorithms are implemented in different software as the algorithms of this thesis are in GAP, while the DLLO algorithms are in Magma. Therefore, a direct run-time comparison is challenging, but more on that topic is discussed in Section 11.2.

The GAP code for the algorithms in this thesis spans more than 15,000 lines, which is why the code is not included in an appendix but is available on GitHub [82].

11.1 Recog package

The recog package [70], initiated by Max Neunhöffer and Ákos Seress, is available for GAP [37] and offers algorithms for efficient matrix group computations, with the main goal of implementing

an algorithm for computing the composition tree of matrix groups, see Section 1.1.3. The recog package already includes a variety of useful algorithms, such as basic computations and applications using a composition tree, constructive recognition (see Section 1.1.8) of permutation groups and naming algorithms (see Section 1.1.7) for simple groups. Since the recog package offers many useful algorithms for the constructive recognition algorithms in this thesis and because efficient constructive recognition algorithms for classical groups are essential for computations involving composition trees, the algorithms GoingDown, BaseCase, GoingUp and StandardGenerators from this thesis have been integrated into the recog package by the author. The recog package is still under development, so function names and paths may change. Thus, a snapshot of the recog package at the time of this thesis submission is available on GitHub [82] which is the recog version "recog 1.4.2DEV".

We now present a table listing the locations of the algorithms from this thesis in the recog package [70] version "recog 1.4.2DEV". All algorithms from this thesis are located in the folder with the path recog/gap/projective/constructive_recognition. In this folder, there is one GAP file and five sub-folders. The GAP file named main.gi contains a function, which can be applied to a matrix group G, called ConstructiveRecognitionClassicalGroupsNaturalRepresentation. This function uses naming algorithms, see Section 1.1.7, to determine if G is a classical group in its natural representation. If this is the case, then the function calls the corresponding constructive recognition algorithm from this thesis. The five sub-folders are named SL, Sp, SU, O and utils. Within each of the four folders SL, Sp, SU and O, there are four files named main.gi, GoingDown.gi, BaseCase.gi, GoingUp.gi, which implement the algorithms from this thesis. The file main.gi in each folder contains the StandardGenerators algorithm of Chapter 3, which combines the GoingDown, BaseCase and GoingUp algorithms into a single algorithm for each group. Details about which algorithm is found in which file are also summarised in Table 11.1.

	GoingDown	BaseCase	GoingUp	StandardGenerators
$\mathrm{SL}(d,q)$	SL/GoingDown.gi	SL/BaseCase.gi	SL/GoingUp.gi	SL/main.gi
$\mathrm{Sp}(d,q)$	Sp/GoingDown.gi	Sp/BaseCase.gi	Sp/GoingUp.gi	Sp/main.gi
$\mathrm{SU}(d,q)$	SU/GoingDown.gi	SU/BaseCase.gi	SU/GoingUp.gi	SU/main.gi
$\Omega(d,q)$	O/GoingDown.gi	O/BaseCase.gi	O/GoingUp.gi	O/main.gi

Table 11.1: The algorithms of this thesis are contained in recog package [70] under the path recog/gap/projective/constructive_recognition.

The GoingDown algorithm, utilising stingray elements, has many similarities across all classical groups. Consequently, the algorithm is implemented only once as `constructppdTwoStingray` in `recog/gap/projective/constructive_recognition/utils/utils.gi`. An additional input, a string `type`, is required for distinguishing the type of the classical groups. The `type` parameter can be assigned values `"SL"`, `"Sp"`, `"SU"` and `"O"` which sets variables within the implementation of `constructppdTwoStingray` such that the GoingDown algorithm is executed as described in this thesis. This configuration is automated in the file `GoingDown.gi` for each classical group.

11.2 Run-time results

In this section, we compare the run-time of the constructive recognition algorithm from this thesis implemented in GAP with the constructive recognition algorithm by Dietrich, Leedham-Green, Lübeck and O'Brien [32, 59] implemented in Magma.

Due to the implementation of these algorithms in different computer algebra systems, which have distinct performance profiles, a direct and fair comparison is challenging. Generally, the underpinning support functions for working with matrices and polynomials over finite fields in Magma outperform those in GAP. Despite these influencing factors, we present the run-time results in Table 11.2, detailing the performance of the constructive recognition algorithm from this thesis in GAP compared to the DLLO constructive recognition algorithm in Magma.

	$q=4$		$q=5$		$q=121$	
d	GAP	Magma	GAP	Magma	GAP	Magma
100	<1	7	<1	3	<1	86
200	6	20	1	8	2	436
300	11	39	4	16	7	12595[a]
500	62	104	16	62	31	timeout
700	168	257	40	249	79	—
1000	337	515	81	16047[b]	176	—
2000	2671	4014	532	timeout	2220	—
3000	9626	14780	2028	timeout	11637	—

[a] Four runs completed, six exceeded the time limit

[b] Two runs completed, eight exceeded the time limit

Table 11.2: Time in seconds comparing our GAP implementation against Magma. An entry with a dash means we did not measure this computation. Entries with "timeout" indicate that no run completed within 20 000 seconds.

All computations for both implementations were performed on a Linux server with two AMD EPYC 9554 64-core processors and 1.5 TB RAM, running Gentoo 2.14 with kernel version 6.1.69. For `Magma` version 2.28-8 was used and for `GAP` version 4.13.0 together with its default set of packages loaded plus the `cvec` package version 2.8.1 (for faster characteristic polynomials) with a modified version of `recog` 1.4.2, where the modification is that we inserted our new algorithms. These modifications will be part of a future release of the `recog` package.

For `Magma` we used `ClassicalConstructiveRecognition(SL(d,q),"SL",d,q)` as test command while for GAP `RECOG.FindStdGensSmallerMatrices_SL(SL(d,q))` was used. For each used parameter pair (d,q) we ran the command ten times in each system. Table 11.2 reports the average of these runs. Note that computations that exceeded 20 000 seconds (about 5.5 hours) were aborted. In some cases only a subset of the runs for a given pair (d,q) terminated within that time limit. In this case we treated the aborted computations as if they had completed within 20 000 seconds (which is generous as in some cases we actually waited for 15 hours before aborting).

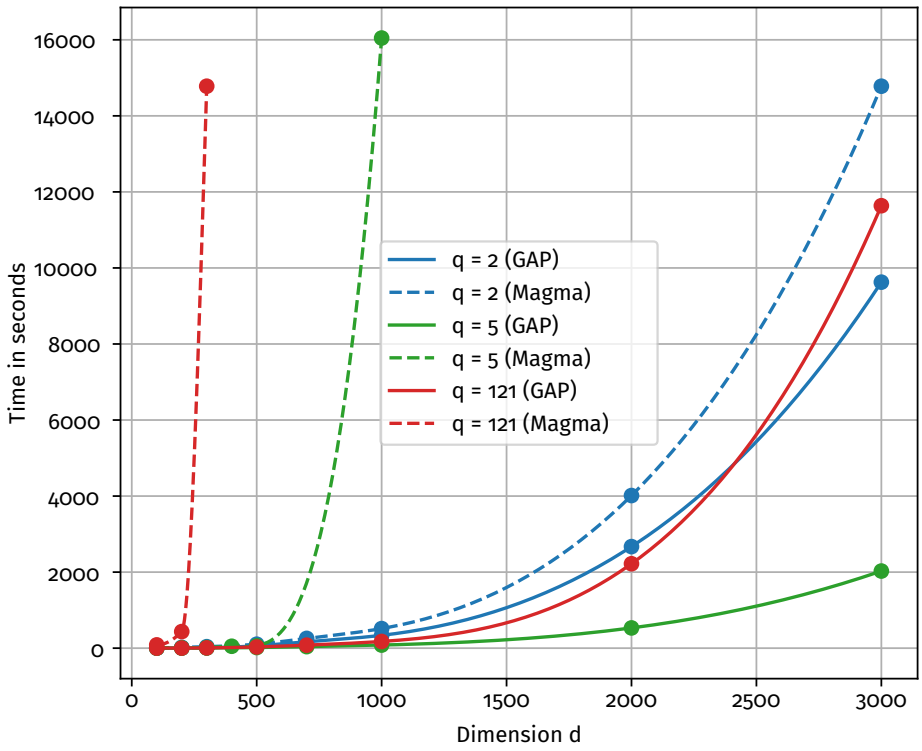


Figure 11.1: The graph visualises the data from Table 11.2 that compares the performance of the `GAP` implementation of this thesis against `Magma`.

Table 11.2 indicates that for special linear groups the constructive recognition algorithm of this thesis outperforms the constructive recognition algorithm by Dietrich, Leedham-Green, Lübeck

and O'Brien [32, 59]. Especially as q grows Table 11.2 highlights that the algorithms in this thesis have better complexity by a factor such as $\mathcal{O}(\log(q))$. This observation is supported by the claimed complexity result for the algorithms from this thesis given in Theorem 10.38 and the current results for the complexity of the constructive recognition algorithm by Dietrich, Leedham-Green, Lübeck and O'Brien [32, 59] presented in Section 1.2.

Chapter 12

Outlook

In Chapter 11, we observed a significant improvement in the run-time of the constructive recognition algorithms for classical groups from this thesis compared to the DLLO algorithm [32, 59]. However, there are several open questions which are not addressed in this thesis. Those questions can be roughly categorised as follows:

1) What is the complexity of the StandardGenerators algorithm from this thesis?

2) Can the algorithm StandardGenerators be adapted for non-natural irreducible matrix representations of classical groups?

3) If modifying the StandardGenerators algorithm for non-natural irreducible representations of classical groups is possible, is it possible to design a version for black-box groups?

4) Can stingray elements be utilised in additional ways within the composition tree?

In the following sections we delve deeper into each of these questions.

12.1 Complexity of the StandardGenerators algorithm

A complexity analysis of the StandardGenerators algorithm, as outlined in Chapter 3, involves examining the complexity of the GoingDown, BaseCase and GoingUp algorithm. Complexity results for the StandardGenerators algorithm are provided in Theorem 10.38. While we analyse the complexity of the GoingDown algorithm and refer to results of the BaseCase algorithm from

existing literature in Chapter 10, we do not conduct a complexity analysis of the randomised algorithm GoingUp. Practical tests suggest that selecting $\mathcal{O}(1)$ random elements in one GoingUp step suffices, a presumption utilised in the proof of Theorem 10.38. Nonetheless, an analysis of the complexity of the GoingUp algorithm is absent and would yield theoretical insight into the complexity of the StandardGenerators algorithm.

12.2 Gray-box algorithm

In Section 1.1.6, we introduced black-box groups and black-box algorithms. While black-box algorithms are applicable in every representation of a group, a slightly weaker variant exists known as *gray-box algorithms*. Gray-box algorithms assume that the input group is a matrix group, eliminating the need for oracle calls during multiplication, inversion and equality checks. This setting allows for more efficient operations, which are costly in black-box algorithms, such as computing orders of invertible matrices using algorithms like the pseudo-order algorithm by Celler and Leedham-Green [24].

An important question arising from the algorithms in this thesis is whether they can be adapted for gray-box algorithms. In gray-box algorithms, classical groups are not necessarily given in their natural representation but in any irreducible matrix representation. However, many computations that are efficient in natural representations become less efficient in other representations. For example, computing stingray embedded subgroups becomes more costly as it involves restricting the action of a group element on a subspace, which is less efficient in non-natural irreducible representations. Moreover, the "natural" action of group elements of non-natural irreducible representation cannot be computed efficiently and this limitation affects the efficiency of several phases of the GoingUp step which are based on linear algebra, introduced in the Chapters 5, 6, 7 and 8. Thus, the GoingUp algorithm based on linear algebra is not directly usable as a gray-box algorithm.

In Chapter 9 we presented a second GoingUp algorithm based on involutions. Involutions are already used in gray-box and black-box constructive recognition algorithms for classical groups [33]. Therefore, the GoingUp algorithm based on involutions seems to be promising in gray-box settings and initial tests conducted in `Magma` were successful. However, a thorough proof of correctness and analysis requires additional effort and an extension of the results presented in this thesis.

12.3 Black-box algorithm

In Section 12.2, we explored the utilisation of the Algorithm StandardGenerators in gray-box settings. While gray-box algorithms suffice for practical use so far, as the leaf groups of a composition tree, see Section 1.1.3, are either provided as permutation or matrix groups, adapting the algorithms from this thesis for black-box groups could be interesting in the future.

As highlighted in Section 12.2, the GoingUp algorithm based on linear algebra, presented in the Chapters 5, 6, 7 and 8, heavily relies on the natural action of the input group. However, as black-box algorithms do not accommodate actions, the GoingUp algorithm based on linear algebra is not applicable. Nevertheless, preliminary tests for the GoingUp algorithm based on involutions for gray-box classical groups, conducted in `Magma`, were successful. Therefore, similar tests should be performed for black-box classical groups. As of now, the author has not conducted tests of the GoingUp algorithm based on involutions for black-box classical groups.

Unlike the GoingUp algorithm, the GoingDown algorithm does not depend on the natural action of the input group. The GoingDown basic step involves tasks such as computing random elements and their characteristic polynomials, and verifying if the stingray bodies of a stingray duo intersect trivially. However, in black-box groups computing characteristic polynomials is not possible and, thus, computing stingray elements by random selections requires an alternative approach. A preliminary version of a black-box GoingDown basic step has been developed by Niemeyer and the author but has yet to be published.

12.4 Further improvements of the composition tree

In this thesis we utilise stingray elements within the GoingDown basic step of the GoingDown algorithm, where these elements serve as a powerful tool for computing a descending recognition chain of length at most $\mathcal{O}(\log^*(d))$. Given the pivotal role of stingray elements in this context, it is logical to explore the potential of stingray elements in other applications, such as constructive recognition of other quasi-simple matrix groups or in computing the composition tree, see Section 1.1.3. However, the proportion results for stingray elements published in [39, 42, 75] are currently limited to classical groups. Therefore, further investigations into the proportion of pre-stingray elements

for additional applications would be beneficial.

As of now, the author has not delved into exploring further applications of stingray elements.

List of symbols

Notation	Description	Page List
$\eth$	Embedding of G into $\mathbb{F}G$	43
$^-$	Field automorphism of order 2	27
$(\omega_1,\ldots,\omega_f)$	$\mathbb{F}_p$ basis of $\mathbb{F}_q$	27
$\overline{\mathbb{F}}$	Algebraic closure of finite field $\mathbb{F}$	27
$\amalg$	Set of pre-stingray candidates	83
$A^\perp$	Orthogonal complement	31
$v \perp w$	v and w are orthogonal, i.e. $\Phi(v,w)=0$	31
$\langle \cdot \mid \cdot \rangle$	Standard scalar product on V	38
$\mathfrak{S}$	MSLP	51
Υ	Length of MSLP $\mathfrak{S}$	51
A	Subset	50
a, c	Matrices	27
α	Parameter of Table 4.1	82, 247
B	Exponent for pre-stingray candidate	73
$\mathscr{B}$	Basis of vector space where $\mathscr{B}=(b_1,\ldots,b_d)$	27
$(b_1,\ldots,b_d)$	Basis of vector space where $\mathscr{B}=(b_1,\ldots,b_d)$	27
$\mathfrak{b}$	Memory quota of MSLP $\mathfrak{S}$	50
$\mathscr{L}$	Base change matrix	27
β	Part of the definition of B	73
$\mathrm{char}(\mathbb{F})$	Characteristic of $\mathbb{F}$	27
χ_a	Characteristic polynomial of $a \in \mathrm{GL}(d,q)$	27
$\mathrm{CL}(V)$	One of the classical groups SL, Sp, SU or Ω	33

Notation	Description	Page List
$\mathrm{CL}(d,q)$	One of the classical groups SL, Sp, SU or Ω	27
d	V is a d-dimensional vector space and $G \leq \mathrm{GL}(d,q)$	27
δ	Parameter of Table 4.1	82, 247
$(e_1,\ldots,e_d)$	Standard basis of vector space	27
$\mathscr{E}$	Complexity of constructive recognition of $\mathrm{SL}(2,q)$	170
$E_{i,j}(\lambda)$	Particular transvection of $\mathrm{SL}(d,q)$	27
$\mathrm{End}(V)$	Group of all endomorphisms on V	26
ϵ	Probability parameter for randomised algorithms	67
E_s	Stingray tail of a stingray element s	72
Eval	Evaluation function of MSLPs	51
$\mathbb{F}$ (or $\mathbb{F}_q$)	Finite field of size $q = p^f$ for prime p and $f \in \mathbb{N}$	27
$\mathbb{F}_p$	Sub field of $\mathbb{F}_q$ of size p	27
f	Exponent of Prime power, i.e. $q = p^f$	27
F_{d-n}	$\langle b_{n+1},\ldots,b_d\rangle$ for basis $(b_1,\ldots,b_d)$	28
$\mathbb{F}G$	Group algebra of G over $\mathbb{F}$	43
$\mathrm{Fix}(g)$	$\mathrm{Fix}(g) = \{v \in V \mid vg = v\} \leq V$	28
$\mathrm{Fix}(U)$	$\mathrm{Fix}(U) = \{v \in V \mid vu = v \text{ for all } u \in U\} \leq V$	28
F_X	Free Group on X	51
G,H,U	Groups where U is a subgroup in general	26
$\overline{g}$	Element of $\mathbb{F}G$ for $g \in G$	43
g,h,u	Group elements of G,H,U, respectively	27
$\mathrm{GL}(d,q)$	Group $\{a \in \mathbb{F}_q^{d\times d} \mid a \text{ is invertible}\}$	26
$\mathrm{GL}(V)$	Group of all bijective endomorphisms on V	26
$\wp$	Cost of performing a group operation	49
$\mathscr{I}$	Instruction of MSLP $\mathfrak{S}$	50
i,j,k,r	Elements of natural numbers and indices	27
I_d	Identity matrix of size $d \times d$	27
$I_{i,j}$	Matrix with exactly one non-zero entry	27
$\imath,\jmath,\lambda$	Elements of finite field $\mathbb{F}$	27

Notation	Description	Page List
$\log^*$	Iterated logarithm	46
$\mathscr{M}$	List which is modified by an instruction $\mathscr{I}$	50
$\mathfrak{m}_1,\ldots,\mathfrak{m}_\mathfrak{b}$	Elements of an ordered list $\mathscr{M}$ of an instruction $\mathscr{I}$	50
m	Dimension of irreducible subspace	72
$\mathbb{N}$	Natural numbers	27
n	$U \leq G \leq \mathrm{GL}(d,q)$ with $U \cong \mathrm{SL}(n,q)$ for $n < d$	27
$\mathrm{O}(V)$	Orthogonal group	32
$\Omega(V)$	Omega group	32
$\Omega^\circ(d,q)$	Omega group of circle type	35
$\Omega^+(d,q)$	Omega group of plus type	35
$\Omega^-(d,q)$	Omega group of minus type	35
$\urcorner$	Cost of solving the order oracle	49
$\mathscr{P}$	Set of all primes	27
P	Polynomial and in most cases $P(x) \in \mathbb{F}_q[x]$	27
p	Prime element	27
Φ	Bilinear form	29
Φ_Q	Polar form of a quadratic form Q	29
π	Projection	133
$\Psi = \Psi(m,q)$	Product of all ppd of $q^k - 1$	75
Q	Quadratic form	29
q	Prime power, i.e. $q = p^f$ for prime p and $f \in \mathbb{N}$	27
$\mathrm{rad}(\Phi)$	Radical of Φ	31
$\mathrm{rad}(Q)$	Radical of Q	31
$\rho_{v,w}$	Siegel transformation with respect to v and w	41
S	Set of standard generators of a special linear group	90
$\mathfrak{s}$	Pre-stingray candidate	73
$\tilde{s}$	Stingray candidate	72
s	Stingray element	72
σ	Isomorphism of stingray embedding	57

Notation	Description	Page List
$\mathrm{SL}(d,q)$	Group $\{a \in \mathbb{F}_q^{d\times d} \mid a \text{ is invertible and } \det(a)=1\}$	26
$\mathrm{SL}(V)$	Subgroup of $\mathrm{GL}(V)$ with determinant 1	26
$\mathrm{SO}(d,q)$	Special orthogonal group	36
$\mathrm{SO}(V)$	Special orthogonal group	32
S^{O}	Set of standard generators of an orthogonal group	210
SO°	Special orthogonal group of circle type	35
SO^+	Special orthogonal group of plus type	35
SO^-	Special orthogonal group of minus type	35
$\mathrm{Sp}(d,q)$	Symplectic group	36
$\mathrm{Sp}(V)$	Symplectic group	32
S^{Sp}	Set of standard generators of a symplectic group	162
S^{SU}	Set of standard generators of a unitary group	191
$\mathrm{SU}(d,q)$	Special unitary group	36
$\mathrm{SU}(V)$	Special unitary group	32
$\mathrm{SX}(d,q)$	One of the groups SL, Sp, SU or SO	27
t	Matrix for the GoingUp algorithm	126
a^{Tr}	Transpose of a matrix $a \in \mathrm{GL}(d,q)$	27
T	Transvection	38
T_{w_1,w_2}	Transvection with respect to w_1 and w_2	38
$\mathrm{U}(V)$	Unitary group	32
V	Finite dimensional $\mathbb{F}$-vector space	27
v, w, ν, b, e	Vector	27
V^*	Dual space	27
ϱ	Map which is not a homomorphism	44
ϑ	Element of the dual space V^*	27
φ	Vector space homomorphism	27
V_n	$\langle b_1, b_2, \ldots, b_n\rangle$ for basis $(b_1, \ldots, b_d)$	28
W	Finite dimensional $\mathbb{F}$-vector space. Mostly $W \leq V$	27
W_s	Stingray body of a stingray element s	72

Notation	Description	Page List
$\hat{\mathscr{X}}$	Extension of representation $\mathscr{X}$	44
$\mathscr{X}$	Representation ($\mathscr{Y}$ also denotes a representation)	42
x	Indeterminate/variable	27
X, Y	Finite generating sets of matrix groups	27
ξ	Complexity parameter for matrix multiplication	46
z_1, z_2	Standard generators of SL	90
$z_1^{\mathrm{O}}, z_2^{\mathrm{O}}, z_3^{\mathrm{O}}, z_4^{\mathrm{O}}, z_5^{\mathrm{O}}, z_6^{\mathrm{O}}$	Standard generators of Sp	210
$z_1^{\mathrm{Sp}}, z_2^{\mathrm{Sp}}, z_3^{\mathrm{Sp}}$	Standard generators of Sp	162
$z_1^{\mathrm{SU}}, z_2^{\mathrm{SU}}, z_3^{\mathrm{SU}}, z_4^{\mathrm{SU}}, z_5^{\mathrm{SU}}$	Standard generators of Sp	191
ζ	Complexity to compute a random element	49

Index

References

[1] G. Adj and F. Rodriguez-Henriquez. **Square root computation over even extension fields**. In: IEEE Trans. Comput. 63.11 (2014), pp. 2829–2841. DOI: 10.1109/TC.2013.145.

[2] M. Aschbacher. **On finite groups of component type**. In: Illinois J. Math. 19 (1975), pp. 87–115.

[3] M. Aschbacher and S. D. Smith. **The classification of quasithin groups. I**. Vol. 111. Mathematical Surveys and Monographs. Structure of strongly quasithin K-groups. American Mathematical Society, Providence, RI, 2004, pp. xiv+477. DOI: 10.1090/surv/111.

[4] M. Aschbacher and S. D. Smith. **The classification of quasithin groups. II**. Vol. 112. Mathematical Surveys and Monographs. Main theorems: the classification of simple QTKE-groups. American Mathematical Society, Providence, RI, 2004, i–xii and 479–1221. DOI: 10.1086/428989.

[5] H. Bäärnhielm, D. Holt, C. R. Leedham-Green, and E. A. O'Brien. **A practical model for computation with matrix groups**. In: J. Symbolic Comput. 68.part 1 (2015), pp. 27–60. DOI: 10.1016/j.jsc.2014.08.006.

[6] L. Babai, A. J. Goodman, W. M. Kantor, E. M. Luks, and P. P. Pálfy. **Short presentations for finite groups**. In: J. Algebra 194.1 (1997), pp. 79–112. DOI: 10.1006/jabr.1996.6980.

[7] L. Babai. **Local expansion of vertex-transitive graphs and random generation in finite groups**. In: Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing. STOC '91. New Orleans, Louisiana, USA: Association for Computing Machinery, 1991, pp. 164–174. DOI: 10.1145/103418.103440.

[8] L. Babai. **Randomization in group algorithms: conceptual questions**. In: Groups and computation, II (New Brunswick, NJ, 1995). Vol. 28. DIMACS Ser. Discrete Math. Theoret. Comput. Sci. Amer. Math. Soc., Providence, RI, 1997, pp. 1–17. DOI: 10.1090/dimacs/028/01.

[9] L. Babai and R. Beals. **A polynomial-time theory of black box groups. I**. In: Groups St. Andrews 1997 in Bath, I. Vol. 260. London Math. Soc. Lecture Note Ser. Cambridge Univ. Press, Cambridge, 1999, pp. 30–64.

[10] L. Babai, R. Beals, and Á. Seress. **Polynomial-time theory of matrix groups**. In: STOC'09—Proceedings of the 2009 ACM International Symposium on Theory of Computing. ACM, New York, 2009, pp. 55–64.

[11] L. Babai, W. M. Kantor, P. P. Pálfy, and Á. Seress. **Black-box recognition of finite simple groups of Lie type by statistics of element orders**. In: J. Group Theory 5.4 (2002), pp. 383–401. DOI: 10.1515/jgth.2002.010.

[12] L. Babai and E. Szemerédi. **On the complexity of matrix group problems I**. In: Proc. 25th IEEE Sympos. Foundations Comp. Sci. 1984, pp. 229–240.

[13] J. Bamberg, J. De Beule, and M. Horn. **Forms, Sesquilinear and Quadratic, Version 1.2.11**. https://gap-packages.github.io/forms. GAP package. Apr. 2024.

[14] R. Beals, C. R. Leedham-Green, A. C. Niemeyer, C. E. Praeger, and Á. Seress. **A black-box group algorithm for recognizing finite symmetric and alternating groups. I**. In: Trans. Amer. Math. Soc. 355.5 (2003), pp. 2097–2113. DOI: 10.1090/S0002-9947-03-03040-X.

[15] R. Beals, C. R. Leedham-Green, A. C. Niemeyer, C. E. Praeger, and Á. Seress. **Permutations with restricted cycle structure and an algorithmic application**. In: Combin. Probab. Comput. 11.5 (2002), pp. 447–464. DOI: 10.1017/S0963548302005217.

[16] W. Bosma, J. Cannon, and C. Playoust. **The Magma algebra system. I. The user language**. In: J. Symbolic Comput. 24.3-4 (1997). Computational algebra and number theory (London, 1993), pp. 235–265. DOI: 10.1006/jsco.1996.0125.

[17] J. N. Bray, M. D. E. Conder, C. R. Leedham-Green, and E. A. O'Brien. **Short presentations for alternating and symmetric groups**. In: Trans. Amer. Math. Soc. 363.6 (2011), pp. 3277–3285. DOI: 10.1090/S0002-9947-2011-05231-1.

[18] J. N. Bray. **An improved method for generating the centralizer of an involution**. In: Arch. Math. (Basel) 74.4 (2000), pp. 241–245. DOI: 10.1007/s000130050437.

[19] P. A. Brooksbank. **Constructive recognition of classical groups in their natural representation**. In: J. Symbolic Comput. 35.2 (2003), pp. 195–239. DOI: 10.1016/S0747-7171(02)00132-3.

[20] P. A. Brooksbank. **Fast constructive recognition of black box symplectic groups**. In: J. Algebra 320.2 (2008), pp. 885–909. DOI: 10.1016/j.jalgebra.2008.03.021.

[21] P. A. Brooksbank. **Fast constructive recognition of black-box unitary groups**. In: LMS J. Comput. Math. 6 (2003), pp. 162–197. DOI: 10.1112/S1461157000000437.

[22] P. A. Brooksbank and W. M. Kantor. **Fast constructive recognition of black box orthogonal groups**. In: J. Algebra 300.1 (2006), pp. 256–288. DOI: 10.1016/j.jalgebra.2006.02.024.

[23] F. Celler and C. R. Leedham-Green. **A constructive recognition algorithm for the special linear group**. In: The atlas of finite groups: ten years on (Birmingham, 1995). Vol. 249. London Math. Soc. Lecture Note Ser. Cambridge Univ. Press, Cambridge, 1998, pp. 11–26. DOI: 10.1017/CBO9780511565830.007.

[24] F. Celler and C. R. Leedham-Green. **Calculating the order of an invertible matrix**. In: Groups and computation, II (New Brunswick, NJ, 1995). Vol. 28. DIMACS Ser. Discrete Math. Theoret. Comput. Sci. Amer. Math. Soc., Providence, RI, 1997, pp. 55–60. DOI: 10.1090/dimacs/028/04.

[25] F. Celler, C. R. Leedham-Green, S. H. Murray, A. C. Niemeyer, and E. A. O'Brien. **Generating random elements of a finite group**. In: Comm. Algebra 23.13 (1995), pp. 4931–4948. DOI: 10.1080/00927879508825509.

[26] I. Chatzigeorgiou. **Bounds on the Lambert Function and Their Application to the Outage Analysis of User Cooperation**. In: IEEE Communications Letters 17.8 (Aug. 2013), pp. 1505–1508. DOI: 10.1109/lcomm.2013.070113.130972.

[27] M. D. E. Conder, C. R. Leedham-Green, and E. A. O'Brien. **Constructive recognition of** PSL$(2,q)$. In: Trans. Amer. Math. Soc. 358.3 (2006), pp. 1203–1221. DOI: 10.1090/S0002-9947-05-03756-6.

[28] M. Conder and C. R. Leedham-Green. **Fast recognition of classical groups over large fields**. In: Groups and computation, III (Columbus, OH, 1999). Vol. 8. Ohio State Univ. Math. Res. Inst. Publ. de Gruyter, Berlin, 2001, pp. 113–121.

[29] J. H. Conway, R. T. Curtis, S. P. Norton, R. A. Parker, and R. A. Wilson. ATLAS **of finite groups**. Maximal subgroups and ordinary characters for simple groups, With computational assistance from J. G. Thackray. Oxford University Press, Eynsham, 1985, pp. xxxiv+252.

[30] E. Costi. **Constructive membership testing in classical groups**. In: PhD thesis. Queen Mary, University of London, 2009.

[31] H. Dietrich, M. Lee, and T. Popiel. **The maximal subgroups of the Monster**. 2023. arXiv: 2304.14646 [math.GR].

[32] H. Dietrich, C. R. Leedham-Green, F. Lübeck, and E. A. O'Brien. **Constructive recognition of classical groups in even characteristic**. In: J. Algebra 391 (2013), pp. 227–255. DOI: 10.1016/j.jalgebra.2013.04.031.

[33] H. Dietrich, C. Leedham-Green, and E. O'Brien. **Effective black-box constructive recognition of classical groups**. In: Journal of Algebra 421 (2015). Special issue in memory of Ákos Seress, pp. 460–492. DOI: https://doi.org/10.1016/j.jalgebra.2014.08.039.

[34] J. D. Dixon. **Generating random elements in finite groups**. In: Electron. J. Combin. 15.1 (2008), Research Paper 94, 13. DOI: 10.37236/818.

[35] J. D. Dixon, C. E. Praeger, and Á. Seress. **Strong involutions in finite special linear groups of odd characteristic**. In: J. Algebra 498 (2018), pp. 413–447. DOI: 10.1016/j.jalgebra.2017.11.047.

[36] J.-G. Dumas, C. Pernet, and Z. Wan. **Efficient computation of the characteristic polynomial**. In: ISSAC'05. ACM, New York, 2005, pp. 140–147. DOI: 10.1145/1073884.1073905.

[37] **GAP – Groups, Algorithms, and Programming, Version 4.13.0**. https://www.gap-system.org. The GAP Group. 2024.

[38] S. P. Glasby, A. C. Niemeyer, and C. E. Praeger. **Bipartite q-Kneser graphs and two-generated irreducible linear groups**. 2023. arXiv: 2312.05529 [math.GR].

[39] S. P. Glasby, A. C. Niemeyer, and C. E. Praeger. **The probability of spanning a classical space by two non-degenerate subspaces of complementary dimensions**. In: Finite Fields Appl. 82 (2022), Paper No. 102055, 31. DOI: 10.1016/j.ffa.2022.102055.

[40] S. P. Glasby, A. C. Niemeyer, and C. E. Praeger. **The probability that two elements with large 1-eigenspaces generate a classical group**. In preparation. 2024.

[41] S. P. Glasby, C. E. Praeger, and C. M. Roney-Dougal. **Involution centralisers in finite unitary groups of odd characteristic**. In: J. Algebra 545 (2020), pp. 245–299. DOI: 10.1016/j.jalgebra.2019.09.009.

[42] S. P. Glasby, A. C. Niemeyer, and C. E. Praeger. **Random generation of direct sums of finite non-degenerate subspaces**. In: Linear Algebra Appl. 649 (2022), pp. 408–432. DOI: 10.1016/j.laa.2022.05.016.

[43] S. P. Glasby, A. C. Niemeyer, C. E. Praeger, and A. Zalesski. **Absolutely irreducible quasisimple linear groups containing certain elements of prime order**. In preparation. 2024.

[44] D. Gorenstein, R. Lyons, and R. Solomon. **The classification of the finite simple groups**. Vol. 40.1. Mathematical Surveys and Monographs. American Mathematical Society, Providence, RI, 1994, pp. xiv+165. DOI: 10.1090/surv/040.1.

[45] L. C. Grove. **Classical groups and geometric algebra**. Vol. 39. Graduate Studies in Mathematics. American Mathematical Society, Providence, RI, 2002, pp. x+169. DOI: 10.1090/gsm/039.

[46] P. E. Holmes, S. A. Linton, E. A. O'Brien, A. J. E. Ryba, and R. A. Wilson. **Constructive membership in black-box groups**. In: J. Group Theory 11.6 (2008), pp. 747–763. DOI: 10.1515/JGT.2008.047.

[47] D. Holt, C. R. Leedham-Green, and E. A. O'Brien. **Constructing composition factors for a linear group in polynomial time**. In: J. Algebra 561 (2020), pp. 215–236. DOI: 10.1016/j.jalgebra.2020.02.018.

[48] D. F. Holt, B. Eick, and E. A. O'Brien. **Handbook of computational group theory**. Discrete Mathematics and its Applications (Boca Raton). Chapman & Hall/CRC, Boca Raton, FL, 2005, pp. xvi+514. DOI: 10.1201/9781420035216.

[49] M. Horn, A. Niemeyer, C. Praeger, and D. Rademacher. **Constructive Recognition of Special Linear Groups**. 2024. arXiv: 2404.18860 [math.GR].

[50] A. Hulpke and Á. Seress. **Short presentations for three-dimensional unitary groups**. In: J. Algebra 245.2 (2001), pp. 719–729. DOI: 10.1006/jabr.2001.8943.

[51] S. P. Humphries. **Generation of special linear groups by transvections**. In: J. Algebra 99.2 (1986), pp. 480–495. DOI: 10.1016/0021-8693(86)90041-4.

[52] B. Huppert and W. Willems. **Lineare Algebra**. Vieweg+Teubner Verlag, Wiesbaden, 2010. DOI: https://doi.org/10.1007/978-3-8348-9710-7.

[53] S. Jambor, M. Leuner, A. C. Niemeyer, and W. Plesken. **Fast recognition of alternating groups of unknown degree**. In: J. Algebra 392 (2013), pp. 315–335. DOI: 10.1016/j.jalgebra.2013.06.005.

[54] C. Jansen, K. Lux, R. Parker, and R. Wilson. **An atlas of Brauer characters**. Vol. 11. London Mathematical Society Monographs. New Series. Appendix 2 by T. Breuer and S. Norton, Oxford Science Publications. The Clarendon Press, Oxford University Press, New York, 1995, pp. xviii+327.

[55] W. M. Kantor and Á. Seress. **Black box classical groups**. In: Mem. Amer. Math. Soc. 149.708 (2001), pp. viii+168. DOI: 10.1090/memo/0708.

[56] W. M. Kantor and Á. Seress. **Computing with matrix groups**. In: Groups, combinatorics & geometry (Durham, 2001). World Sci. Publ., River Edge, NJ, 2003, pp. 123–137. DOI: 10.1142/9789812564481_0007.

[57] P. Kleidman and M. Liebeck. **The subgroup structure of the finite classical groups**. Vol. 129. London Mathematical Society Lecture Note Series. Cambridge University Press, Cambridge, 1990, pp. x+303. DOI: 10.1017/CBO9780511629235.

[58] C. R. Leedham-Green. **On a variant of the product replacement algorithm**. In: Glasg. Math. J. 66.1 (2024), pp. 221–228. DOI: 10.1017/s0017089523000435.

[59] C. R. Leedham-Green and E. A. O'Brien. **Constructive recognition of classical groups in odd characteristic**. In: J. Algebra 322.3 (2009), pp. 833–881. DOI: 10.1016/j.jalgebra.2009.04.028.

[60] C. R. Leedham-Green and E. A. O'Brien. **Presentations on standard generators for classical groups**. In: J. Algebra 545 (2020), pp. 324–389. DOI: 10.1016/j.jalgebra.2019.07.024.

[61] C. R. Leedham-Green. **The computational matrix group project**. In: Groups and computation, III (Columbus, OH, 1999). Vol. 8. Ohio State Univ. Math. Res. Inst. Publ. de Gruyter, Berlin, 2001, pp. 229–247.

[62] W. C. W. Li. **Number theory with applications**. Vol. 7. Series on University Mathematics. World Scientific Publishing Co., Inc., River Edge, NJ, 1996, pp. xii+229. DOI: 10.1142/2716.

[63] M. W. Liebeck and E. A. O'Brien. **Finding the characteristic of a group of Lie type**. In: J. Lond. Math. Soc. (2) 75.3 (2007), pp. 741–754. DOI: 10.1112/jlms/jdm028.

[64] M. W. Liebeck and E. A. O'Brien. **Recognition of finite exceptional groups of Lie type**. In: Trans. Amer. Math. Soc. 368.9 (2016), pp. 6189–6226. DOI: 10.1090/tran/6534.

[65] F. Lübeck, K. Magaard, and E. A. O'Brien. **Constructive recognition of** $\mathrm{SL}_3(q)$. In: J. Algebra 316.2 (2007), pp. 619–633. DOI: 10.1016/j.jalgebra.2007.01.020.

[66] F. Lübeck, A. C. Niemeyer, and C. E. Praeger. **Finding involutions in finite Lie type groups of odd characteristic**. In: J. Algebra 321.11 (2009), pp. 3397–3417. DOI: 10.1016/j.jalgebra.2008.05.009.

[67] A. Lubotzky and I. Pak. **The product replacement algorithm and Kazhdan's property (T)**. In: J. Amer. Math. Soc. 14.2 (2001), pp. 347–363. DOI: 10.1090/S0894-0347-00-00356-8.

[68] K. Lux and H. Pahlings. **Representations of groups**. Vol. 124. Cambridge Studies in Advanced Mathematics. A computational approach. Cambridge University Press, Cambridge, 2010, pp. x+460. DOI: 10.1017/CBO9780511750915.

[69] P. M. Neumann and C. E. Praeger. **A recognition algorithm for special linear groups**. In: Proc. London Math. Soc. (3) 65.3 (1992), pp. 555–603. DOI: 10.1112/plms/s3-65.3.555.

[70] M. Neunhöffer, Á. Seress, and M. Horn. **recog, A package for constructive recognition of permutation and matrix groups, Version 1.4.2**. https://gap-packages.github.io/recog. GAP package. Sept. 2022.

[71] M. Neunhöffer and C. E. Praeger. **Computing minimal polynomials of matrices**. In: LMS J. Comput. Math. 11 (2008), pp. 252–279. DOI: 10.1112/S1461157000000590.

[72] M. Neunhöffer and Á. Seress. **A data structure for a uniform approach to computations with finite groups**. In: ISSAC 2006. ACM, New York, 2006, pp. 254–261. DOI: 10.1145/1145768.1145811.

[73] A. C. Niemeyer, T. Popiel, C. E. Praeger, and D. Rademacher. **Showcasing straight-line programs with memory via matrix Bruhat decomposition**. 2024. arXiv: 1305.5617 [cs.DS].

[74] A. C. Niemeyer and C. E. Praeger. **A recognition algorithm for classical groups over finite fields**. In: Proc. London Math. Soc. (3) 77.1 (1998), pp. 117–169. DOI: 10.1112/S0024611598000422.

[75] A. C. Niemeyer and C. E. Praeger. **Elements in finite classical groups whose powers have large 1-eigenspaces**. In: Discrete Math. Theor. Comput. Sci. 16.1 (2014), pp. 303–312.

[76] E. A. O'Brien. **Algorithms for matrix groups**. In: Groups St Andrews 2009 in Bath. Volume 2. Vol. 388. London Math. Soc. Lecture Note Ser. Cambridge Univ. Press, Cambridge, 2011, pp. 297–323.

[77] I. Pak. **What do we know about the product replacement algorithm?** In: Groups and computation, III (Columbus, OH, 1999). Vol. 8. Ohio State Univ. Math. Res. Inst. Publ. de Gruyter, Berlin, 2001, pp. 301–347.

[78] C. W. Parker and R. A. Wilson. **Recognising simplicity of black-box groups by constructing involutions and their centralisers**. In: J. Algebra 324.5 (2010), pp. 885–915. DOI: 10.1016/j.jalgebra.2010.05.013.

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

[80] C. E. Praeger. **Primitive prime divisor elements in finite classical groups**. In: Groups St. Andrews 1997 in Bath, II. Vol. 261. London Math. Soc. Lecture Note Ser. Cambridge Univ. Press, Cambridge, 1999, pp. 605–623. DOI: 10.1017/CBO9780511666148.024.

[81] C. E. Praeger, Á. Seress, and Ş. Yalçınkaya. **Generation of finite classical groups by pairs of elements with large fixed point spaces**. In: J. Algebra 421 (2015), pp. 56–101. DOI: 10.1016/j.jalgebra.2014.08.020.

[82] D. Rademacher. **A new algorithm for constructive recognition of classical groups**. https://github.com/danielrademacher/ConstructiveRecognitionOfClassicalGroups. 2024.

[83] D. Rademacher. **Bruhat Decomposition in Orthogonal Groups over Finite Fields**. Master's thesis. RWTH Aachen University, Aachen, Germany. 2020.

[84] D. Rademacher. **Bruhat Decomposition in Unitary and Symplectic Groups over Finite Fields**. Bachelor's thesis. RWTH Aachen University, Aachen, Germany, 2019.

[85] Á. Seress. **Permutation group algorithms**. Vol. 152. Cambridge Tracts in Mathematics. Cambridge University Press, Cambridge, 2003, pp. x+264. DOI: 10.1017/CBO9780511546549.

[86] C. C. Sims. **Computation with Finitely Presented Groups**. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1994.

[87] C. C. Sims. **Computational methods in the study of permutation groups**. In: Computational Problems in Abstract Algebra (Proc. Conf., Oxford, 1967). Pergamon, Oxford-New York-Toronto, Ont., 1970, pp. 169–183.

[88] R. Steinberg. **Generators for simple groups**. In: Canadian J. Math. 14 (1962), pp. 277–283. DOI: 10.4153/CJM-1962-018-0.

[89] V. Strassen. **Gaussian elimination is not optimal**. In: Numer. Math 13 (1969), pp. 354–356.

[90] M. Suzuki. **On a class of doubly transitive groups**. In: Ann. of Math. (2) 75 (1962), pp. 105–145. DOI: 10.2307/1970423.

[91] D. E. Taylor. **The geometry of the classical groups**. Vol. 9. Sigma Series in Pure Mathematics. Heldermann Verlag, Berlin, 1992, pp. xii+229.

[92] R. A. Wilson. **Computing in the Monster**. In: Groups, combinatorics & geometry (Durham, 2001). World Sci. Publ., River Edge, NJ, 2003, pp. 327–335. DOI: 10.1142/9789812564481_0020.

[93] R. A. Wilson. **Standard generators for sporadic simple groups**. In: J. Algebra 184.2 (1996), pp. 505–515. DOI: 10.1006/jabr.1996.0271.