Rheinland-Pfälzische Technische Universität Kaiserslautern Landau

Fachbereich Mathematik
AGAG
AG Darstellungstheorie

Trivial source character tables of small finite groups

Vom Fachbereich Mathematik der Rheinland - Pfälzischen Technischen Universität Kaiserslautern - Landau zur Verleihung des akademischen Grades Doktor der Naturwissenschaften (Doctor rerum naturalium, Dr. rer. nat.) genehmigte Dissertation

$G = \mathfrak{S}_5$		Q_v $(1 \leq v \leq 7)$		$\langle 1 \rangle$			C_2		C_2	V_4		V_4	C_4	D_8
$p = 2$		N_v $(1 \leq v \leq 7)$		$\mathfrak{S}_5$			D_{12}		D_8	$\mathfrak{S}_4$		D_8	D_8	D_8
		P	$n_j \in N_v$	1	(123)	(12345)	1	(345)	1	1	(234)	1	1	1
$\chi_1 + \chi_2 + 2\chi_3 + \chi_6 + \chi_7$	1	$\langle 1 \rangle$		24	0	4	0	0	0	0	0	0	0	0
$\chi_4 + \chi_5$	2			8	2	-2	0	0	0	0	0	0	0	0
$\chi_3 + \chi_6 + \chi_7$	3			16	-2	1	0	0	0	0	0	0	0	0
$\chi_1 + \chi_3 + \chi_6$	4	C_2		12	0	2	2	2	0	0	0	0	0	0
χ_4	5			4	1	-1	2	-1	0	0	0	0	0	0
$\chi_1 + \chi_2 + \chi_6 + \chi_7$	6	C_2		12	0	2	0	0	4	0	0	0	0	0
$\chi_1 + \chi_2$	7	V_4		2	2	2	0	0	2	2	2	0	0	0
$\chi_6 + \chi_7$	8			10	-2	0	0	0	2	2	-1	0	0	0
$\chi_1 + \chi_6$	9	V_4		6	0	1	2	2	2	0	0	2	0	0
$\chi_1 + \chi_7$	10	C_4		6	0	1	0	0	2	0	0	0	2	0
χ_1	11	D_8		1	1	1	1	1	1	1	1	1	1	1

vorgelegt von

Bernhard Böhmler

1. Gutachter*in: Jun.-Prof. Dr. Caroline Lassueur
2. Gutachter*in: Prof. Dr. Robert Boltje

Datum der Disputation: 05. September 2023

DE - 386

Abstract

Trivial source modules, also known as p-permutation modules, arise naturally in the representation theory of finite groups. They are the indecomposable direct summands of the permutation modules and have many interesting applications to modular representation theory. In order to do calculations with trivial source modules the ordinary characters of their lifts from positive characteristic p to characteristic zero are of particular interest. The "trivial source character tables" (also called "species tables of the trivial source ring") collect information about the character values of trivial source modules with all possible vertices, as well as those of their Brauer constructions.

One of the focal points of the present thesis is the computational treatment of these tables. We develop an algorithm which computes the trivial source character tables of all finite groups for any given prime number. This algorithm is only limited by the capacity of the memory of the used computer. Implementations of this algorithm in the computer algebra systems GAP and MAGMA are given. In order to implement it in the open source system GAP, it was necessary to write a program that computes the projective indecomposable modules of a group algebra. This code is also included. The computed trivial source character tables are saved into a database.

In the theoretical part of this thesis, we determine the trivial source characters of all block algebras of domestic representation type. Using this result, we work out the trivial source character tables of the infinite family D_{4v} of dihedral groups of order $4v$, where v is an odd integer, in characteristic 2. Moreover, we compute the trivial source character tables of the alternating groups $\mathfrak{A}_4$, $\mathfrak{A}_5$ and the matrix groups $\mathrm{SL}_2(11)$, $\mathrm{PSL}_2(11)$, $\mathrm{SL}_2(13)$, and $\mathrm{PSL}_2(13)$ in characteristic 2 theoretically.

An application of trivial source character tables is given by p-permutation equivalences, since they are induced by chain complexes consisting only of bimodules which are trivial source modules. These equivalences are connected to Broué's abelian defect group conjecture which predicts a categorical equivalence between a block and its Brauer correspondent with isomorphic abelian defect groups. As the number of p-permutation equivalences between two blocks is always finite, it is feasable to calculate all p-permutation equivalences using computer algebra and we briefly present one possible computational approach.

"A good stock of examples, as large as possible, is indispensable for a thorough understanding of any concept, and when I want to learn something new, I make it my first job to build one."

Paul Halmos

Acknowledgements

First and foremost, I would like to express my deep gratitude to my supervisor Caroline Lassueur for her guidance, advice, and kindness during the process of writing this thesis. Her enthusiasm and joy for mathematics are contagious and inspiring. Thank you for sharing your knowledge about representation theory with me and for being such a great mentor!

I would like to thank Robert Boltje for many helpful conversations and for his willingness to referee this thesis. Particularly, I am grateful for his invitation to spend four months as his guest at the University of California, Santa Cruz. The visit was a very valuable experience and I am thankful to everyone who helped make my stay enjoyable, especially Ingrid and Dennis Hostetter, my host parents.

I am indebted to Jürgen Müller, Thomas Breuer, Frank Lübeck, Derek Holt, and Alexander Hulpke for many helpful e-mails, comments, and recommendations on the computational aspects of this thesis.

I would like to thank Jeremy Rickard, Markus Linckelmann, Geoffrey Robinson, Aram Mikaelian, Karin Erdmann, Gunter Malle, Claus Fieker, Olivier Dudas, Max Horn, and the developers of MAGMA for useful discussions. I am also thankful to Markus Hillenbrand for installing GAP and the Shared C MeatAxe on Elwetritsch and for his help with technical issues. In addition, I would like to thank Hans Schönemann, Jens Hubrich, and Jan Sauerwein for their help with various computer related problems.

I would like to thank René Marczinzik for many intriguing telephone calls.

Furthermore, I would like to thank the whole computer algebra working group for their warmth, help and friendliness, especially Ruwen, Niamh, Emil, Patrick, Laura, Birte, Carlo, Alessandro, Raul, Erec, Johannes, Dario, Yvonne, Liam, Fabian, Annika, Marie, Arezoo, Maria, Jeroen, Ali, Suri, Sogo, and Lucas.

A special thanks goes to Raul, Ruwen, Carlo, Laura, and Patrick, for their help during my many moves from Kaiserslautern to Kaiserslautern.

Moreover, I would like to thank the SFB-TRR 195 of the German Research Foundation (DFG) and the German Academic Exchange Service (DAAD) for financial support.

I would also like to thank my close friends Stephan, Thomas, and René, as well as my flatmate Lukas, for their friendship during the last years.

Last but not least, I would like to thank my family for their encouragement and continuous support.

Contents

Chapter 1

Introduction

The representation theory of finite groups plays an important rôle in various areas of current research, including questions coming from physics or chemistry. Given a finite group G (e.g. a group of symmetries of an object), a major goal of representation theory is to find interesting properties and invariants of G. This is achieved by examining certain homomorphisms called representations from G to other objects which are better understood. These objects are often either defined over fields of characteristic zero or over fields of positive characteristic. In the former case, the corresponding theory is called ordinary representation theory and in the latter case it is called modular representation theory.

These two theories are linked with each other and trivial source modules are of central importance in that context. The main objective of the present thesis is to determine trivial source character tables of different groups at various prime numbers both theoretically and using computer algebra. This is one possible approach to the study of trivial source modules, which are also known as p-permutation modules.

We now explain the motivational background for this topic. One way to obtain information about representations is to examine their irreducible composition factors. In ordinary representation theory of finite groups, the ordinary character table collects the values of characters of irreducible group representations evaluated at conjugacy classes. In modular representation theory, the Brauer character table with respect to a prime number p dividing the order of G collects the values of characters of irreducible modular group representations evaluated at p-regular conjugacy classes.

Green rings collect information about direct sums and tensor products of modules. Moreover, these rings give us a unified way to view ordinary and Brauer character tables. Let $\mathbb{F}$ be a field. The Green ring $a(\mathbb{F}G)$ of the group algebra $\mathbb{F}G$ is defined to be the free abelian group on the set of isomorphism classes $[M]$ of indecomposable $\mathbb{F}G$-modules, where addition is given by taking direct sums and multiplication is induced by the tensor product over $\mathbb{F}$. Then $A(\mathbb{F}G) := \mathbb{C} \otimes_{\mathbb{Z}} a(\mathbb{F}G)$ is a commutative and associative $\mathbb{C}$-algebra. When $\mathbb{F} = \mathbb{C}$, $a(\mathbb{C}G)$ is the Grothendieck ring of $\mathbb{C}G$ and every ring homomorphism $a(\mathbb{C}G) \to \mathbb{C}$ is given by a trace map $t_g : [M] \mapsto \mathrm{tr}(g, M)$ with $g \in G$.
After tensoring with $\mathbb{C}$, the sum of these maps over a set of representatives $\mathrm{ccls}(G)$ of the conjugacy classes of G is an isomorphism $\sum_{g \in \mathrm{ccls}(G)} t_g : A(\mathbb{C}G) \xrightarrow{\sim} \bigoplus_{\mathrm{ccls}(G)} \mathbb{C}$, and hence $A(\mathbb{C}G)$ is semisimple. In positive characteristic, on the other hand, if $\mathbb{F}$ is a large enough field of characteristic $p > 0$ then the Grothendieck ring is $\mathscr{R}(\mathbb{F}G) := a(\mathbb{F}G)/a_0(\mathbb{F}G, 1)$, where $a_0(\mathbb{F}G)$ is the ideal of $a(\mathbb{F}G)$ spanned by the difference elements $[M_2] - [M_1] - [M_3]$ where $0 \to M_1 \to M_2 \to M_3 \to 0$ is a short exact sequence of $\mathbb{F}G$-modules. In this case every ring homomorphism $\mathscr{R}(\mathbb{F}G) \to \mathbb{C}$ is given by a map $t_g : \mathscr{R}(\mathbb{F}G) \to \mathbb{C}$ where $g \in G$ is a p'-element and for a given $\mathbb{F}G$-module M, $t_g(M)$ is the sum of the lifts to $\mathbb{C}$ of the eigenvalues of g on the restricted module $\mathrm{Res}^G_{\langle g \rangle}(M)$. Once again, af-

ter tensoring with $\mathbb{C}$, the sum of these maps over a set of representatives $ccls(G)_{p'}$ of the p'-conjugacy classes of G yields an isomorphism $\sum_{g\in \mathrm{ccls}(G)_{p'}} t_g : \mathbb{C}\otimes_{\mathbb{Z}} \mathscr{R}(\mathbb{F}G) \xrightarrow{\sim} \bigoplus_{\mathrm{ccls}(G)_{p'}} \mathbb{C}$.

In [BP84] Benson and Parker generalised such constructions and defined a species of any subalgebra, ideal or quotient A of $A(\mathbb{F}G)$, to be an algebra homomorphism $A \to \mathbb{C}$. Evaluating the species of $A(\mathbb{C}G)$ at the simple $\mathbb{C}G$-modules yields the species table of $A(\mathbb{C}G)$, which is in fact just the ordinary character table of G. Similarly, if $\mathbb{F}$ is large enough and of characteristic $p > 0$ then the species table of $\mathbb{C}\otimes_{\mathbb{Z}} \mathscr{R}(\mathbb{F}G)$, calculated by evaluating the species of $\mathbb{C}\otimes_{\mathbb{Z}} \mathscr{R}(\mathbb{F}G)$ at the simple $\mathbb{F}G$-modules, is just the Brauer character table of G. Benson and Parker [BP84] proved that in the latter, positive characteristic case, many of the properties of Green rings, species, and p-blocks are governed by the trivial source ring $a(\mathbb{F}G, \mathrm{Triv})$, which is defined to be the subring of $a(\mathbb{F}G)$ generated by the (finite) set of all isomorphism classes of indecomposable trivial source $\mathbb{F}G$-modules. We will see later that $A(\mathbb{F}G, \mathrm{Triv}) := \mathbb{C}\otimes_{\mathbb{Z}} a(\mathbb{F}G, \mathrm{Triv})$ is semisimple. Evaluating the species of $A(\mathbb{F}G, \mathrm{Triv})$ at the indecomposable trivial source $\mathbb{F}G$-modules yields a square matrix called the trivial source character table (or species table of the trivial source ring), denoted by $\mathrm{Triv}_p(G)$. This table provides us with information about the character values of all trivial source $\mathbb{F}G$-modules and their Brauer quotients at p'-conjugacy classes. For p-groups, $\mathrm{Triv}_p(G)$ is just the table of marks. For all other groups, however, unlike ordinary and Brauer character tables, only a very small number of trivial source character tables have been published in the literature: [Ben84, Appendix] gives $\mathrm{Triv}_2(\mathfrak{A}_5)$ and [LP10, Example 4.10.12] gives $\mathrm{Triv}_3(M_{11})$.

The article [BFL22] gives $\mathrm{Triv}_p(\mathrm{SL}_2(q))$ in the cases in which q is odd, $p \mid (q \pm 1)$ when p is odd and $q \equiv \pm 3 \pmod 8$ when $p = 2$. It was inspired by the computations of $\mathrm{Triv}_2(\mathrm{PSL}_2(11))$, $\mathrm{Triv}_2(\mathrm{SL}_2(11))$, $\mathrm{Triv}_2(\mathrm{PSL}_2(13))$, and $\mathrm{Triv}_2(\mathrm{SL}_2(13))$ given in Chapter 4.

The class of p-permutation modules was studied by Conlon [Con68] and Scott [Sco73]. Another fruitful approach through the Brauer morphism, invariant bases, and G-algebras is due to Puig; it appears in [Bro85] by Broué. Moreover, Thévenaz [Thé95, §27] also provides a detailed introduction in the language of G-algebras, and in [Lin18a, Lin18b] the most exhaustive collection of known results on the topic with detailed proofs can be found.

The articles [Con68] and [BP84] are usually considered as the starting point for research concerning trivial source character tables. In Chapter 7 of the latter article, these tables are introduced and considered column by column. The authors state the theorem that the trivial source ring $A(G, \mathrm{Triv})$ of a finite group G is semisimple and that the species of $A(G, \mathrm{Triv})$ can be expressed via certain Brauer species. Moreover, induction formulae for species are developed. Additionally, some theorems about the origin of species are stated and proved. These cases were extended and generalised to linear source modules by Boltje in [Bol98].

In most cases, the theoretical results mentioned so far exclusively consider the columns of the trivial source character tables and establish connections between the trivial source character tables of groups and some of their subgroups. Since trivial source modules are liftable from positive characteristic p to characteristic 0 in a unique way, it is also possible to consider the ordinary characters of these lifts and to therefore focus more on the rows of the trivial source character tables. This second approach has the advantage of making the problem approachable via computer algebra.

The necessary theorems and statements from [BP84] are reformulated in [LP10]. In the latter book, the authors apply both computational and theoretical techniques in order to calculate the trivial source character table of the Mathieu group M_{11} in characteristic 3. Since there are only very few examples of fully computed trivial source character tables in the literature, it is advantageous to have an electronic database available which is open source and contains as many examples as possible. This is also useful theoretically, since one can test conjectures for many groups at once, whereas calculations by hand would take much longer.

Trivial source modules are often considered due to the following motivation: assume given group algebras kG and kH over the same field k, as well as their representations. We suppose that $\mathrm{char}(k) = p > 0$ if not stated otherwise. It is possible to subdivide the group algebras further into elementary pieces called p-blocks. The p-blocks under consideration often have infinite representation type, but the number of indecomposable trivial source modules is always finite. Nevertheless these modules have enough structural properties to be interesting and useful. In order to study p-blocks of finite groups one fruitful approach is to define categorical equivalences between these blocks on various levels, which preserve many invariants and structural properties.

Theorems and conjectures concerning these equivalences are currently subject to extensive study. In particular, Broué's abelian defect group conjecture (ADGC) has generated a lot of interest recently. Broué's ADGC predicts categorical equivalence between a block b of the group algebra kG and its Brauer correspondent, if they have abelian defect groups. Broué's ADGC can be stated at various different levels. Originally, Broué stated his famous conjecture in [Bro85] in the 1990's on the level of derived equivalences. One decade later, Rickard strengthened Broué's ADGC in [Ric96] by conjecturing that equivalence can be achieved by splendid Rickard complexes between a block b of kG and a block c of kH. In [BX08] Boltje and Xu proved that each such equivalence determines a p-permutation equivalence between b and c. Moreover, Perepelitsky proved in [Per14] that every p-permutation equivalence between b and c implies an isotypy between b and c and that the number of p-permutation equivalences between two fixed blocks is always finite. When formulated in terms of p-permutation equivalences, Broué's ADGC predicts the following. Assume b is a block of kG with an abelian defect group D. Let H be the normaliser of D in G. Let c be the Brauer correspondent of b in H. Then, there exists a p-permutation equivalence between b and c.

A link between trivial source modules and p-permutation equivalences is given by the fact that p-permutation equivalences are induced by chain complexes consisting only of bimodules which are trivial source modules. Hence, it is desirable to obtain an understanding of these particular modules. Since there are only finitely many of them when G and p are fixed, this is indeed feasable.

The thesis is structured as follows. In Chapter 2, we give some background material for modular representation theory of finite groups. In Chapter 3, trivial source character tables are introduced (see Convention 3.2.2). Moreover, $\mathrm{Triv}_p(G)$ is determined in the cases that G is a p-group (Proposition 3.2.17(a)) or a p'-group (Proposition 3.2.17(b)). Furthermore, a formula that simplifies the algorithmic computation of trivial source modules of abelian groups and some related groups is worked out (Proposition 3.1.15).

Chapter 4 is concerned with the determination of various trivial source characters when the characteristic of the ground field equals 2. Our main theoretical results are the determination of the following objects:

1. $\mathrm{Triv}_2(V_4)$, where V_4 denotes the Klein four-group (Proposition 4.1.2);
2. $\mathrm{Triv}_2(\mathfrak{A}_4)$ (Proposition 4.1.5);
3. $\mathrm{Triv}_2(\mathfrak{A}_5)$ (Proposition 4.1.8);
4. the trivial source characters belonging to B when B is a block algebra of domestic representation type (Section 4.2);
5. $\mathrm{Triv}_2(D_{4v})$ (Theorem 4.3.3);
6. $\mathrm{Triv}_2(\mathrm{PSL}_2(11))$ (Theorem 4.3.9);
7. $\mathrm{Triv}_2(\mathrm{SL}_2(11))$ (Theorem 4.3.10);
8. $\mathrm{Triv}_2(\mathrm{PSL}_2(13))$ (Theorem 4.3.16);
9. $\mathrm{Triv}_2(\mathrm{SL}_2(13))$ (Theorem 4.3.17).

In Chapter 5, we present our theoretical algorithms. The main results are:

1. a theoretical algorithm for GAP and the Shared C MeatAxe that computes projective indecomposable modules over finite fields (Strategy 5.1.32 & Strategy 5.1.34);
2. a theoretical algorithm that computes $\mathrm{Triv}_p(G)$ for any given finite group G and any given prime number p dividing the order of G (Strategy 5.3.1);
3. one possible way to establish a database of trivial source character tables and the solutions of certain possible issues (Section 5.4).

Moreover, the behaviour of modules under ground field extensions, as well as the theory of peakwords are presented.

In Chapter 6, we describe how to theoretically obtain all p-permutation equivalences (Section 6.1) and all splendid Morita equivalences (Section 6.2) between two p-blocks.

In Chapter 7 the GAP- and MAGMA-algorithms are given. In GAP, there was no readily available function that computes projective indecomposable matrix representations of group algebras over splitting fields. Combining and extending existing programs we have implemented an algorithm which computes these modules using GAP and the Shared C MeatAxe. Our main results are:

1. an algorithm computing the projective indecomposable modules of G over finite splitting fields using GAP and the Shared C MeatAxe (Section 7.1);
2. an algorithm computing $\mathrm{Triv}_p(G)$ using GAP and the Shared C MeatAxe (Section 7.2);
3. an algorithm computing $\mathrm{Triv}_p(G)$ using MAGMA (Section 7.3);
4. an algorithm computing p-permutation equivalences between (principal) blocks of finite group algebras in MAGMA (Section 7.4);

5. an algorithm computing splendid Morita equivalences between (principal) blocks of finite group algebras in MAGMA (Section 7.5).

The most important bugs we encountered during our computations are stated in Remark 5.4.14.

We emphasize here that by now it is not possible to avoid intensive computations with matrix representations since there is no known method which obtains the same pieces of information in a purely character-theoretic way.

Our algorithm computing $\mathrm{Triv}_p(G)$ has contributed to the article [BFL22], as it was very useful to be able to calculate small instances of trivial source character tables of infinite families of groups with the computer. Moreover, many computed tables are available at a database of trivial source character tables, see
https://agag-lassueur.math.rptu.de/~lassueur/en/TrivSourceDatabase/.

We plan to improve the GAP-algorithm which computes the projective indecomposable modules in the future by using condensation subgroups. Moreover, we aim at extending our database of trivial source character tables and at contributing to databases of splendid Morita and p-permutation equivalence classes of blocks in the cases of small finite groups. It is desirable to know which Morita equivalences lift to splendid Morita equivalences.

This thesis also contributes to one of the projects of the collaborative research centre SFB-TRR 195. In this context, the development of the mentioned algorithms was an essential goal of this thesis. In particular, this led to the realisation of the algorithms in GAP as a part of the open source computer algebra system OSCAR, as well as the implementation of many functions that were not available in GAP before.

Chapter 2

Background material

In this chapter, we introduce preliminary notions which are needed throughout the text. We assume that the reader is familiar with the modular representation theory of finite groups. The content of this chapter is mostly based on [Ben98], [Lin18a], [Lin18b], [Las21], [Web16], [Thé95], [CR90], [CR87], [CR06], and [NT89] to which we refer for convenient reference. Nevertheless, we restate some of the most important notions in order to fix our notation. Moreover, for an overview of the used symbols, we refer to the index of notation at the end of this thesis.

Conventions

Throughout this thesis we assume that the following conventions hold, if not stated otherwise.

1. All rings are unitary and associative.
2. For the theoretical part all modules are finitely generated left modules. For computer calculations (Chapters 5-7) all modules are finitely generated right modules.

Furthermore, in this chapter, R denotes a unitary commutative ring.

2.1 Foundations of representation theory

In this section, let $\mathfrak{R}$ and $\mathfrak{S}$ be rings, let Λ be an R-algebra, and let G be a finite group.

Definition 2.1.1. (a) A **left $\mathfrak{R}$-module** is a triple $(M, +, \cdot)$ satisfying the following axioms:

(i) $(M, +)$ is an abelian group;

(ii) $(r_1 + r_2) \cdot m = r_1 \cdot m + r_2 \cdot m$ for each $r_1, r_2 \in \mathfrak{R}$ and each $m \in M$;

(iii) $r \cdot (m_1 + m_2) = r \cdot m_1 + r \cdot m_2$ for each $r \in \mathfrak{R}$ and all $m_1, m_2 \in M$;

(iv) $(r_1 r_2) \cdot m = r_1 \cdot (r_2 \cdot m)$ for each $r_1, r_2 \in \mathfrak{R}$ and all $m \in M$.

(v) $1_R \cdot m = m$ for each $m \in M$.

The operation $\cdot$ is called a **scalar multiplication**.

(b) A **right $\mathfrak{R}$-module** is defined analogously using a scalar multiplication $\cdot : M \times \mathfrak{R} \to M$, $(m, r) \mapsto m \cdot r$ on the right-hand side.

(c) An $(\mathfrak{R}, \mathfrak{S})$-**bimodule** is an abelian group $(M, +)$ which is both a left $\mathfrak{R}$-module and a right $\mathfrak{S}$-module, and which satisfies the equation

$$r \cdot (m \cdot s) = (r \cdot m) \cdot s \quad \forall\, r \in \mathfrak{R},\ \forall\, s \in \mathfrak{S},\ \forall\, m \in M.$$

Convention: When no confusion is to be made, we will simply write "$\mathfrak{R}$-module" to mean "left $\mathfrak{R}$-module", denote $\mathfrak{R}$-modules by their underlying sets and write rm instead of $r \cdot m$. Right modules and bimodules are treated analogously.

Definition 2.1.2. A **representation** of $\mathfrak{R}$ is a ring homomorphism $\rho : \mathfrak{R} \to \mathrm{End}_{\mathbb{Z}}(M)$, where M is an abelian group, i.e. we have for all $a, b \in \mathfrak{R}$:

$$(*) \quad \rho(a+b) = \rho(a) + \rho(b), \quad \rho(ab) = \rho(a)\rho(b), \quad \rho(1) = \mathrm{Id}_M.$$

The homomorphism ρ induces a map

$$\mathfrak{R} \times M \to M :$$

$$(a, x) \mapsto ax := \rho(a)(x).$$

Since $\rho(a) \in \mathrm{End}_{\mathbb{Z}}(M)$, the equations in (*) turn into:

$$a(x+y) = ax + ay,$$

$$(a+b)x = ax + bx,$$

$$(ab)x = a(bx),$$

$$1 \cdot x = x.$$

Hence, we obtain the well-known correspondence between $\mathfrak{R}$-modules and representations of $\mathfrak{R}$.

Definition 2.1.3. (a) If V is an R-module, we denote by $\mathrm{GL}(V)$ the group of all invertible R-module homomorphisms $V \to V$.

(b) A (linear) **representation** of G (over R) is a group homomorphism $G \to \mathrm{GL}(V)$. If V is a free R-module of rank n, the group $\mathrm{GL}(V)$ is isomorphic to $\mathrm{GL}_n(R)$, the group of non-singular $n \times n$-matrices over R. In this situation, on choosing a basis for V we obtain for each element $g \in G$ a matrix $\rho(g)$, and these matrices multiply together in the manner of the group G.

(c) The **group ring** RG is defined as $RG := \bigoplus_{g \in G} Rg$, the free R-module with basis G.

Elements of RG are written as linear combinations $\sum_{g \in G} a_g g$, $a_g \in R$.

The addition in RG is defined componentwise, that is,

$$\sum_{g \in G} a_g g + \sum_{g \in G} b_g g := \sum_{g \in G} (a_g + b_g) g,$$

and the multiplication in RG is induced by the multiplication in G, that is,

$$(\sum_{g \in G} a_g g) \cdot (\sum_{g \in G} b_g g) = \sum_{g,h \in G} a_g b_h gh = \sum_{e \in G} (\sum_{gh = e} a_g b_h) e.$$

The group ring RG is a ring with unit element $1_{RG} = 1_R 1_G$. Due to the indentification of G with $\varphi(G)$, where $\varphi : G \hookrightarrow RG, \varphi(g) := 1_R g$, we frequently write $1_{RG} = 1_G$ and $G \subseteq RG$. Obviously, we do even have $G \subseteq (RG)^{\times}$, the group of units in RG.

Each representation $\rho : G \to \mathrm{GL}_n(R)$ can be R-linearly extended:

$$\rho' : RG \to \mathrm{Mat}_{n\times n}(R) := \mathrm{End}_R(R^n)$$

$$\rho'(\sum_{g\in G} a_g g) := \sum_{g\in G} a_g \rho(g).$$

Thus, R^n is turned into an RG-module, with scalar multiplication

$$RG \times R^n \to R^n :$$

$$(\alpha, x) \mapsto \alpha x := (\rho'(\alpha))(x).$$

Example 2.1.4. Let $G := C_2 = \{1, c\},\ c^2 = 1$. A representation ρ of G is then given by:

$$\begin{array}{ccc} G & \xrightarrow{\rho} & \mathrm{GL}_n(R) \\ c & \mapsto & A. \end{array}$$

It follows that $\rho(c)$ is equal to a matrix A whose square is the identity matrix $1_{\mathrm{GL}_n(R)}$.

Next, we collect some well-known results from representation theory.

2.1.1 The Jordan-Hölder theorem

From now on, if not stated otherwise, we let Λ be an R-algebra.

Definition 2.1.5. Let M be a Λ-module. Then:

(a) M is called **irreducible** or **simple**, if it has exactly two submodules, otherwise **reducible**;

(b) M is called **decomposable**, if it has two non-zero proper submodules M_1, M_2 such that $M = M_1 \oplus M_2$;

(c) M is called **indecomposable**, if it is non-zero and not decomposable;

(d) M is called **completely reducible** or **semisimple** if there exists a direct sum decomposition of M into simple Λ-submodules.

Definition 2.1.6. Let M be a Λ-module.

(a) A **series** (or **filtration**) of M is a finite chain of submodules

$$\{0\} = M_0 \subseteq M_1 \subseteq \ldots \subseteq M_n = M \quad (n \in \mathbb{Z}_{\geq 0}).$$

(b) A **composition series** of M is a series

$$\{0\} = M_0 \subseteq M_1 \subseteq \ldots \subseteq M_n = M \quad (n \in \mathbb{Z}_{\geq 0})$$

where M_i/M_{i-1} is simple for each $1 \leq i \leq n$. The quotient modules M_i/M_{i-1} are called the **composition factors** (or the **constituents**) of M.

Definition 2.1.7. (a) A Λ-module M is said to satisfy the **descending chain condition** (D.C.C.) on submodules (or to be **Artinian**) if every descending chain

$$M = M_0 \supseteq M_1 \supseteq \ldots \supseteq M_r \supseteq \ldots \supseteq \{0\}$$

of submodules eventually becomes stationary, i.e. there exists a non-negative integer m_0 such that $M_m = M_{m_0}$ for every $m \geq m_0$.

(b) A Λ-module M is said to satisfy the **ascending chain condition** (A.C.C.) on submodules (or to be **Noetherian**) if every ascending chain

$$0 = M_0 \subseteq M_1 \subseteq \ldots \subseteq M_r \subseteq \ldots \subseteq M$$

of submodules eventually becomes stationary, i.e. there exists a non-negative integer m_0 such that $M_m = M_{m_0}$ for every $m \geq m_0$.

(c) The ring Λ is called **left Artinian** (resp. **left Noetherian**) if the regular module Λ^{reg} is Artinian (resp. Noetherian).

Theorem 2.1.8 (Jordan-Hölder)**.** *Given any two series of submodules*

$$\{0\} = M_0 \leq \cdots \leq M_r = M$$
$$\{0\} = M'_0 \leq \cdots \leq M'_s = M$$

of a Λ-module M, we may refine them to series of equal length

$$\{0\} = L_0 \leq \cdots \leq L_n = M$$
$$\{0\} = L'_0 \leq \cdots \leq L'_n = M$$

so that, up to isomorphism, the factors L_i/L_{i-1} $(1 \leq i \leq n)$ are a permutation of the factors L'_j/L'_{j-1} $(1 \leq j \leq n)$. Thus the following conditions on M are equivalent.
(i) The module M has a composition series.
(ii) Every series of submodules of M can be refined to a composition series.
(iii) The module M satisfies A.C.C. and D.C.C. on submodules.

Proof. See [Ben98, Theorem 1.1.4]. □

Remark 2.1.9. (a) It follows that the length of a composition series, if one exists, is an invariant of the module. It is called the **composition length** of the module.

(b) A Λ-module M is said to be **uniserial**, if it has a unique composition series.

Notation 2.1.10. Let M be a Λ-module and let $\{S_1, \ldots, S_n\}$ be a complete set of representatives of isomorphism classes of simple Λ-modules. We write $[M] = a_1S_1 + \ldots + a_nS_n$, $a_i \in \mathbb{Z}_{\geq 0}$ for $1 \leq i \leq n$, if, for all $i \in \{1, \ldots, n\}$, M has the module S_i as a composition factor with multiplicity a_i.

2.1.2 The Jacobson radical

Definition 2.1.11. (a) The **socle** of a Λ-module M is the sum of all the irreducible submodules of M, and is denoted by $\mathrm{Soc}(M)$.

(b) A submodule N of a Λ-module M is called a **maximal submodule (in M)** if the factor module M/N is simple.

(c) The **radical** of a Λ-module M is the intersection of all the maximal submodules of M, and is denoted by $\operatorname{Rad}(M)$ or $J(M)$. The **radical series** or **Loewy series** of M is defined inductively for $n \in \mathbb{Z}_{\geq 0}$ by $\operatorname{Rad}^0(M) := M$, $\operatorname{Rad}^n(M) := \operatorname{Rad}(\operatorname{Rad}^{n-1}(M))$.

(d) The **head** (or **top**) of M is $\operatorname{Head}(M) := \operatorname{Hd}(M) := M/\operatorname{Rad}(M)$.

Definition 2.1.12. Let M be a Λ-module.

(a) The **annihilator** of an element $m \in M$ is the set of all elements $r \in \Lambda$ with $rm = 0$.

(b) The **annihilator** of M is defined to be the intersection of the annihilators of all elements of M and denoted by $\operatorname{ann}_\Lambda(M)$.

Definition 2.1.13. We define $J(\Lambda)$, the **Jacobson radical** of Λ, to be the intersection of all maximal left ideals of Λ.

Remark 2.1.14. We have $J(\Lambda) = \bigcap_{V \text{ simple } \Lambda\text{-module}} \operatorname{ann}_\Lambda(V)$.

Lemma 2.1.15 (Nakayama's Lemma)**.** *Let V be a Λ-module and let $W \leq V$ be a Λ-submodule of V. Then $V = W + J(\Lambda)V$ implies $V = W$.*

Proof. See [NT89, Theorem 1.3.6]. □

Lemma 2.1.16. *A Λ-homomorphism $f : M \to N$ is surjective if and only if the induced morphism $\widetilde{f} : M/\operatorname{Rad}(M) \to N/\operatorname{Rad}(N)$ is surjective.*

Proof. If the homomorphism $\widetilde{f} : M/\operatorname{Rad}(M) \to N/\operatorname{Rad}(N)$ is surjective, then $\operatorname{Im}(f) + \operatorname{Rad}(N) = N$, hence $\operatorname{Im}(f) = N$. The other direction is trivial. □

2.1.3 The Wedderburn structure theorem

Lemma 2.1.17 (Schur's Lemma)**.** *Let S_1 and S_2 be simple Λ-modules. Then*

$$\operatorname{Hom}_\Lambda(S_1, S_2) = \{0\}$$

unless $S_1 \cong S_2$, in which case the endomorphism ring $\operatorname{End}_\Lambda(S_1)$ is a division ring. If Λ is a finite-dimensional algebra over an algebraically closed field k, then every Λ-module endomorphism of S_1 is multiplication by some scalar. Thus, $\operatorname{End}_\Lambda(S_1) \cong k$ in this case.

Proof. See [Web16, Theorem 2.1.1]. □

Theorem 2.1.18 (The Wedderburn-Artin theorem)**.** *Let Λ be a semisimple Artinian ring. Then we have $\Lambda \cong \bigoplus_{i=1}^{r} \Lambda_i$ for some $r \in \mathbb{Z}_{\geq 1}$, where, for each $1 \leq i \leq r$, $\Lambda_i \cong \operatorname{Mat}_{n_i \times n_i}(\Delta_i)$, the ring Δ_i is a division ring and the rings Λ_i are uniquely determined up to permutation of the index i. Furthermore, the ring Λ has exactly r isomorphism classes of simple modules M_i, $i = 1, \ldots, r$, $\operatorname{End}_\Lambda(M_i) \cong \Delta_i^{op}$, the opposite ring of Δ_i, and $\dim_{\Delta_i^{op}}(M_i) = n_i$.*

Proof. See See [Web16, Theorem 1.3.5] □

2.1.4 The Krull-Schmidt theorem

Definition 2.1.19. A Λ-module M has the **unique decomposition property**, if the following two assertions hold.

(a) The Λ-module M is isomorphic to a finite direct sum of indecomposable Λ-modules.

(b) Whenever $M \cong \bigoplus_{i=1}^{m} M_i \cong \bigoplus_{j=1}^{n} M'_j$ with each M_i and each M'_j non-zero indecomposable, then $m = n$, and after a suitable reordering if necessary, $M_i \cong M'_j$.

A ring S is said to have the unique decomposition property, if every finitely generated S-module does.

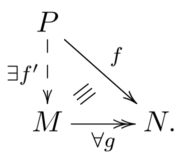
Theorem 2.1.20 (The Krull-Schmidt theorem)**.** *Suppose that Λ is Artinian. Then Λ has the unique decomposition property.*

Proof. See [Ben98, Theorem 1.4.6]. □

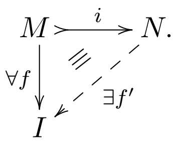
2.1.5 Projective modules, injective modules, and the Cartan matrix

Definition 2.1.21. Let M, N, P, and I be Λ-modules.

(a) The Λ-module P is called **projective** if for every epimorphism $g : M \twoheadrightarrow N$ and for every $f \in \mathrm{Hom}_\Lambda(P, N)$ there exists a morphism $f' \in \mathrm{Hom}_\Lambda(P, M)$ such that $gf' = f$:



(b) The Λ-module I is called **injective** if for every monomorphism $M \xrightarrow{i} N$ and every $f \in \mathrm{Hom}_R(M, I)$ there exists a morphism $f' \in \mathrm{Hom}_R(N, I)$ such that $f'i = f$:



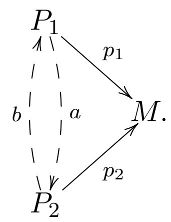
Lemma 2.1.22. *Let P be a Λ-module. The following are equivalent:*

(a) *P is projective;*

(b) *P is a direct summand of a free Λ-module.*

Proof. See [Web16, Proposition 7.1.3]. □

Remark 2.1.23. Since every Λ-module M is a quotient of a free Λ-module, it is certainly a quotient of a projective Λ-module. If the ring Λ is Artinian, and P_1 and P_2 are minimal projective Λ-modules (with respect to direct sum decomposition) mapping onto a finitely generated module M, then we have a diagram



If the composite map $P_1 \xrightarrow{a} P_2 \xrightarrow{b} P_1$ is not an isomorphism of Λ-modules, then by Fitting's lemma P_1 has a summand mapping to zero in M and so P_1 is not minimal. Applying this argument both ways round, we see that $P_1 \cong P_2$.

Definition 2.1.24. Let Λ, M, P_1, and p_1 be as in Remark 2.1.23. Then, the Λ-module P_1 (together with the surjective Λ-module homomorphism p_1) is called the **projective cover** of M. We denote it by $P(M)$.

Remark 2.1.25. Let Λ be as in Remark 2.1.23. Then $\mathrm{Hd}(P) := P/\operatorname{Rad}(P)$ is a simple Λ-module. Moreover, the map $P \mapsto \mathrm{Hd}(P)$ induces a bijection between the set of isomorphism classes of indecomposable projective Λ-modules and the set of isomorphism classes of simple Λ-modules. Projective indecomposable Λ-modules are termed **PIMs** which is an abbreviation for *projective indecomposable modules* and *principal indecomposable modules*, respectively.

Definition 2.1.26. A finite-dimensional algebra Λ over an arbitrary field k is called **symmetric** if there is a linear map $\lambda : \Lambda \to k$ such that $\operatorname{Ker}(\lambda)$ contains no non-zero left or right ideal and for all $a, b \in \Lambda$ we have $\lambda(ab) = \lambda(ba)$.

Example 2.1.27. For every field k the group algebra kG is a symmetric algebra. Moreover, every block of kG is a symmetric algebra (see [Lin18a, Theorem 2.11.11]).

Remark 2.1.28. Let k be a field, let Λ be a symmetric k-algebra, and let P be a projective Λ-module. We have $\operatorname{Soc}(P) \cong P/\operatorname{Rad}(P)$ as Λ-modules.

Definition 2.1.29. Let Λ be a finite-dimensional algebra over an algebraically closed field k. If S and T are simple Λ-modules, then the integer

$$c_{S,T} := \text{ multiplicity of } S \text{ as a composition factor of } P(T)$$

is called the **Cartan invariant** associated to the pair (S, T). The matrix $\mathfrak{C} := (c_{S,T})$ with rows and columns indexed by the isomorphism classes of simple Λ-modules is called the **Cartan matrix** of Λ.

Remark 2.1.30. If Λ is a finite-dimensional algebra over an algebraically closed field k and the algebra Λ is a symmetric algebra, then, after a suitable relabelling of rows and columns, the Cartan matrix of Λ is a symmetric matrix.

2.2 Induction, restriction, and their interactions

Unless stated otherwise, all rings are supposed to have the unique decomposition property from now on.

Let H be a subgroup of G, and let W be an RH-module. Since RG can be considered as an (RG, RH)-bimodule, the tensor product $\operatorname{Ind}_H^G(W) := RG \otimes_{RH} W$ is an RG module, called the **module induced from** W or the **induction of** W **from** H **to** G. We sometimes also write $W{\uparrow}_H^G$ instead of $\operatorname{Ind}_H^G(W)$.

Lemma 2.2.1 ([CR90, §10A], [Web16, Proposition 4.3.1], [LP10, Section 3.2]).
Let $\{g_1, \ldots, g_m\}$ *be a left transversal of the subgroup* H *in* G*, i.e.* $G = \dot{\bigcup} g_i H$*. Furthermore, let* W *be an* RH*-module. Then:*

(a) $RG \cong \bigoplus_{i=1}^{m} g_i RH$ *as* R*-modules and*

$$W\uparrow_H^G = \bigoplus_{i=1}^{m} g_i \otimes W$$

as R*-modules, where* $g_i \otimes W = \{g_i \otimes w \mid w \in W\} \subseteq RG \otimes_{RH} W$*. Each* $g_i \otimes W$ *is isomorphic to* W *as an* R*-module, and if* W *is free as an* R*-module, we have*

$$\mathrm{rank}_R(W\uparrow_H^G) = [G : H] \cdot \mathrm{rank}(W).$$

If $\mathcal{B} = (w_1, \ldots, w_n)$ *is an* R*-basis of* W *then*

$$\mathcal{B}^G := (g_1 \otimes w_1, \ldots, g_1 \otimes w_n, \ldots, g_m \otimes w_1, \ldots, g_m \otimes w_n)$$

is an R*-basis of* W^G*.*

(b) *If* $\rho : H \to \mathrm{GL}(W)$ *is the representation afforded by* W *and*

$$\boldsymbol{\rho} : H \to \mathrm{GL}_n(R), h \mapsto \boldsymbol{\rho}(h) = [\rho(h)]_{\mathcal{B}}$$

is the corresponding matrix representation with respect to the basis $\mathcal{B}$*, then the matrices of the induced representation* $\boldsymbol{\rho}^G : G \to \mathrm{GL}\left(W\uparrow_H^G\right)$ *afforded by the induced module corresponding to the basis* $\mathcal{B}^G$ *are as follows:*

$$\left[\rho^G(g)\right]_{\mathcal{B}^G} = \begin{bmatrix} \dot{\boldsymbol{\rho}}_{11}(g) & \dot{\boldsymbol{\rho}}_{12}(g) & & \dot{\boldsymbol{\rho}}_{1m}(g) \\ \dot{\boldsymbol{\rho}}_{21}(g) & \dot{\boldsymbol{\rho}}_{22}(g) & & \dot{\boldsymbol{\rho}}_{2m}(g) \\ & & \ddots & \\ \dot{\boldsymbol{\rho}}_{m1}(g) & \dot{\boldsymbol{\rho}}_{m2}(g) & & \dot{\boldsymbol{\rho}}_{mm}(g) \end{bmatrix} \quad \text{for} \quad g \in G$$

with

$$\dot{\boldsymbol{\rho}}_{ij}(g) = \begin{cases} \boldsymbol{\rho}\left(g_j^{-1} g g_i\right) \in \mathrm{Mat}_{n\times n}(R) & \text{if } g_j^{-1} g g_i \in H, \\ \mathbf{0}_n \in \mathrm{Mat}_{n\times n}(R) & \text{else.} \end{cases}$$

(c) *Letting* $\lambda : H \to R$ *be the character afforded by* $\boldsymbol{\rho}$*, that is,*

$$\lambda(h) = \mathrm{tr}(\boldsymbol{\rho}(h)), \ h \in H,$$

it is immediate from Part (b) that the character λ^G *afforded by* $\boldsymbol{\rho}^G$ *is given by*

$$\lambda^G(g) = \sum_{i=1}^{m} \dot{\lambda}(g_i^{-1} g g_i), \ \text{for all } g \in G,$$

where $\dot{\lambda}$ *is the extension to* G *of the function* λ*, and is defined by*

$$\dot{\lambda}(g) = \begin{cases} \lambda(g), \ g \in H; \\ 0, \ g \notin H. \end{cases}$$

Definition 2.2.2. Let H be a subgroup of G and let M be an RG-module. Then M may be regarded as an RH-module through a change of the base ring along the inclusion morphism $\iota : RH \hookrightarrow RG$, which we denote by $\mathrm{Res}^G_H(M)$ or simply $M\downarrow^G_H$ and call the **restriction** of M from G to H.

Lemma 2.2.3 ([Web16, Lemma 8.1.2]). *Let H be a subgroup of G.*

(a) *If P is a projective RG-module then $P\downarrow^G_H$ is a projective RH-module.*

(b) *If Q is a projective RH-module then $Q\uparrow^G_H$ is a projective RG-module.*

Lemma 2.2.4 (Frobenius reciprocity / Nakayama relations). *Let $H \leq G$ be a subgroup of G, let V be an RH-module, and let W be an RG-module. We have the R-isomorphisms*

$$\mathrm{Hom}_{RG}(V\uparrow^G_H, W) \cong \mathrm{Hom}_{RH}(V, W\downarrow^G_H)$$

and

$$\mathrm{Hom}_{RG}(W, V\uparrow^G_H) \cong \mathrm{Hom}_{RH}(W\downarrow^G_H, V).$$

Proof. See [Web16, Corollary 4.3.8]. □

Next, we describe how induction and restriction interact.

Definition 2.2.5. Let $H \leq G$ be finite groups, let $g \in G$, and let M be an RH-module.

(a) Let $c_g : G \to G,\ u \mapsto gug^{-1}$ denote the automorphism of G given by conjugation with g. For $h \in G$ and $X \subseteq G$ we denote $c_g(h) := ghg^{-1}$ by ${}^g h$ and $c_g(X)$ by ${}^g X$.

(b) We denote by ${}^g M$ the left $R[{}^g H]$-module with underlying R-module M and ${}^g H$-action given by restricting the H-action along the isomorphism $c_g^{-1} : {}^g H \to H$.

Theorem 2.2.6 (Mackey formula). *Let $H, L \leq G$ and let M be an RL-module. Then, as RH-modules,*

$$M\uparrow^G_L\downarrow^G_H \cong \bigoplus_{g\in[H\backslash G/L]} \left({}^g M\downarrow^{{}^g L}_{H\cap {}^g L}\right)\uparrow^H_{H\cap {}^g L}.$$

Proof. See [CR90, (10.13) Subgroup Theorem]. □

Theorem 2.2.7 (Mackey tensor product theorem). *Let $H_1, H_2 \leq G$, and let L_1 and L_2 be left modules over* RH_1 *and* RH_2*, respectively. Then*

$$L_1\uparrow^G_{H_1} \otimes_R L_2\uparrow^G_{H_2} \cong \bigoplus_{x^{-1}y\in D} (({}^x L_1 \otimes_R {}^y L_2)\downarrow_{{}^x H_1\cap {}^y H_2})\uparrow^G$$

where the sum extends over all (H_1, H_2)-double cosets D in G. There is one summand for each D; namely, we choose a pair (x, y) with $x^{-1}y \in D$, and take the indicated summand.

Proof. See [CR90, (10.18) Tensor Product Theorem]. □

2.3 Vertices, sources, and the Green correspondence

Definition 2.3.1. Let $H \leq G$. An RG-module M is called **relatively H-projective**, or H-projective, if it is isomorphic to a direct summand of an RG-module induced from H.

Proposition 2.3.2 ([Web16, Proposition 11.3.4])**.** *Let $H \leq G$ and let M be an RG-module. The following are equivalent:*

(a) *M is relatively H-projective;*

(b) $M \mid M \downarrow_H^G \uparrow_H^G$.

Any indecomposable RG-module can be seen as a relatively projective module with respect to some subgroup of G:

Proposition 2.3.3 ([Web16, Proposition 11.3.5])**.** *Let $H \leq G$. If the index $[G:H]$ of H in G is invertible in R, then every RG-module is H-projective.*

Remark 2.3.4. In particular, if k is a field of characteristic $p > 0$ and H contains a Sylow p-subgroup of G, then every kG-module is H-projective.

Definition 2.3.5. Let L be an RG-module. If L is free as an R-module then it is called an RG-**lattice**.

When seen as an R-module, an RG-module L may have torsion whereas the $(R/J(R))G$-module $L/J(R)L$ is torsion-free. In order to avoid these cases, we agree on the following.

Convention 2.3.6. If not stated otherwise, all RG-modules are supposed to be RG-lattices from now on.

Moreover, if not stated otherwise, we impose the following restrictions on the ring R from now on.

We assume that R is a field of characteristic p or a complete discrete valuation ring such that the residue field $R/J(R)$ has characteristic p.

We now introduce the concept of vertices and sources which leads to a better understanding of indecomposable modules over group algebras.

Theorem 2.3.7 ([CR90, (19.13) Proposition])**.** *Let M be an indecomposable RG-module.*

(a) *There is a unique conjugacy class of subgroups Q of G which are minimal subject to the property that M is Q-projective.*

(b) *Let Q be a minimal subgroup of G such that M is Q-projective. Then, there exists an indecomposable RQ-module T which is unique, up to conjugacy by elements of $N_G(Q)$, such that M is a direct summand of $T\uparrow_Q^G$.*

Definition 2.3.8. Let M be an indecomposable RG-module.

(a) A **vertex** of M is a minimal subgroup Q of G such that M is relatively Q-projective. The set of all vertices of M is denoted by $\mathrm{vtx}(M)$.

(b) Given a vertex Q of M, an RQ-**source**, or simply a source of M is an RQ-module T such that $M \mid T\uparrow_Q^G$.

Remark 2.3.9. (a) A vertex Q of an indecomposable RG-module M is only defined up to G-conjugacy. Hence, all vertices of M are isomorphic.

(b) For a fixed vertex Q of M, a source of M is defined up to conjugacy by elements of $N_G(Q)$.

As every RG-module is projective relative to a Sylow p-subgroup of G, by minimality, vertices are contained in Sylow p-subgroups. Hence, the vertices of every indecomposable RG-module are p-subgroups of G.

Example 2.3.10. (a) The trivial subgroup $\langle 1 \rangle$ is a vertex of an indecomposable RG-module $U \iff U$ is a PIM of RG.

(b) The vertices of the trivial RG-module are the Sylow p-subgroups of G, i.e. $\mathrm{vtx}(R) = \mathrm{Syl}_p(G)$, and all sources are trivial.

The Green correspondence allows us to reduce questions about indecomposable modules to a situation where a vertex of the given indecomposable module is a normal subgroup.

Theorem 2.3.11 (Green Correspondence)**.** *Let Q be a p-subgroup of G and let L be a subgroup of G containing $N_G(Q)$.*

(a) *If U is an indecomposable RG-module with vertex Q, then*

$$U\downarrow_L^G \cong f(U) \oplus X$$

where $f(U)$ is the unique indecomposable direct summand of $U\downarrow_L^G$ with vertex Q and every direct summand of X is $L \cap {}^xQ$-projective for some $x \in G \backslash L$.

(b) *If V is an indecomposable RL-module with vertex Q, then*

$$V\uparrow_L^G = g(V) \oplus Y$$

where $g(V)$ is the unique indecomposable direct summand of $V\uparrow_L^G$ with vertex Q and every direct summand of Y is $Q \cap {}^xQ$-projective for some $x \in G \backslash L$.

(c) *With the notation of* (a) *and* (b)*, we then have $g(f(U)) \cong U$ and $f(g(V)) \cong V$. In other words, f and g define a bijection*

$$\left\{ \begin{array}{c} \text{isomorphism classes of indecom-} \\ \text{posable } RG\text{-modules with vertex } Q \end{array} \right\} \xrightarrow{\sim} \left\{ \begin{array}{c} \text{isomorphism classes of indecom-} \\ \text{posable } RL\text{-modules with vertex } Q \end{array} \right\}$$
$$U \mapsto f(U)$$
$$g(V) \leftarrow V.$$

Moreover, corresponding modules have a source in common.

Proof. See [CR90, (20.6) Theorem]. □

Definition 2.3.12. In Theorem 2.3.11, the module $f(U)$ is called the RL-**Green correspondent** of U (or simply the Green correspondent of U) and $g(V)$ is called the RG-**Green correspondent** of V (or simply the Green correspondent of V).

Example 2.3.13. Using the notation from Theorem 2.3.11, we see that the Green correspondent of the trivial RG-module is the trivial RL-module, since $R\downarrow_L^G = R$.

We conclude this subsection with the following two lemmata which are needed in the sequel.

Lemma 2.3.14. *Let M be an indecomposable kG-module. Then, there exists a Sylow p-subgroup P of G and an indecomposable kP-module L such that M is isomorphic to a direct summand of $L{\uparrow}_P^G$.*

Proof. Let Q be a vertex of M and let T be a kQ-source of M. Then, by [Web16, Theorem 11.6.1 (2)], M is isomorphic to a direct summand of $T{\uparrow}_Q^G$. As Q is a p-subgroup of G, Q is contained in a Sylow p-subgroup P of G. By Green's indecomposability criterion, the kP-module $L := T{\uparrow}_Q^P$ is indecomposable. Hence, by transitivity of induction, M is isomorphic to a direct summand of $L{\uparrow}_P^G$. □

Lemma 2.3.15 ([Rob89, Theorem 3])**.** *Let G be a finite group. Let $\widetilde{H} \leq G$, and let S and $\tilde{S}$ respectively be a simple kG-module and a simple $k\widetilde{H}$-module. Then, the multiplicity of the projective cover $P(S)$ of S as a direct summand of $\tilde{S}{\uparrow}_{\widetilde{H}}^G$ is equal to the multiplicity of the projective cover $P(\tilde{S})$ of $\tilde{S}$ as a direct summand of $S{\downarrow}_{\widetilde{H}}^G$.*

2.4 Splitting p-modular systems

In order to relate group representations over a field of positive characteristic to character theory in characteristic zero, there is the notion of a p-modular system which is defined as follows.

Definition 2.4.1. Let p be a prime number.

(a) A triple of rings $(K, \mathcal{O}, k)$ is called a ***p*-modular system** if:

   (i) $\mathcal{O}$ is a complete discrete valuation ring of characteristic zero,

   (ii) $K = \mathrm{Frac}(\mathcal{O})$ is the field of fractions of $\mathcal{O}$ (also of characteristic zero), and

   (iii) $k = \mathcal{O}/J(\mathcal{O})$ is the residue field of $\mathcal{O}$ and has characteristic p.

(b) If G is a finite group, then a p-modular system $(K, \mathcal{O}, k)$ is a called a **splitting *p*-modular system** for G, if both K and k are splitting fields for G.

It is often helpful to visualise p-modular systems and the condition on the characteristic of the rings involved through the following commutative diagram of rings and ring homomorphisms.

$$\begin{array}{ccccc} \mathbb{Q} & \hookleftarrow & \mathbb{Z} & \twoheadrightarrow & \mathbb{F}_p \\ \downarrow & & \downarrow & & \downarrow \\ K & \hookleftarrow & \mathcal{O} & \twoheadrightarrow & k \end{array}$$

Here, the hook arrows are the canonical inclusions and the two-head arrows the quotient morphisms. Clearly, these morphisms also extend naturally to ring homomorphisms

$$KG \hookleftarrow \mathcal{O}G \twoheadrightarrow kG$$

between the corresponding group algebras (each mapping an element $g \in G$ to itself).

Remark 2.4.2. We usually work with a splitting p-modular system for all (quotient groups of all) subgroups of G, because it allows us to avoid problems with field extensions. By a theorem of Brauer such a p-modular system can always be obtained by adjoining a primitive m-th root of unity to $\mathbb{Q}_p$, where m is the exponent of G.

Next, we investigate changes of the coefficients given in the setting of a p-modular system for group algebras involved.

Remark 2.4.3. Let $(K, \mathcal{O}, k)$ be a p-modular system and write $\mathfrak{p} := J(\mathcal{O})$. If L is an $\mathcal{O}G$-module, then:

- setting $L^K := K \otimes_{\mathcal{O}} L$ defines a KG-module, and
- reduction modulo $\mathfrak{p}$ of L, that is, $\bar{L} := L/\mathfrak{p}L \cong k \otimes_{\mathcal{O}} L$ defines a kG-module.

As mentioned in Convention 2.3.6, when seen as an $\mathcal{O}$-module, an $\mathcal{O}G$-module L may have torsion, which is lost on passage to K. In order to avoid this issue, we only work with $\mathcal{O}G$-lattices:

Convention 2.4.4. If not stated otherwise, we make the following assumptions from now on.

1. The symbols G and H always denote finite groups.
2. The triple $(K, \mathcal{O}, k)$ denotes a p-modular system, where $\mathcal{O}$ is a complete discrete valuation ring of characteristic zero with unique maximal ideal $\mathfrak{p} := J(\mathcal{O})$, algebraically closed residue field $k = \mathcal{O}/J(\mathcal{O})$ of characteristic p, and field of fractions $K = \mathrm{Frac}(\mathcal{O})$, which we assume to be large enough for G and its subgroups in the sense that K contains a root of unity of order $\exp(G)$.
3. All $\mathcal{O}G$-modules are free as $\mathcal{O}$-modules

In this way, we obtain functors

$$KG\text{-mod} \leftarrow \mathcal{O}G\text{-lat} \rightarrow kG\text{-mod}$$

between the corresponding categories of finitely generated $\mathcal{O}G$-lattices and finitely generated KG-, kG-modules.

Question: which KG-modules, respectively kG-modules, come from $\mathcal{O}G$-lattices?

In the case of KG-modules we have the following answer.

Proposition 2.4.5. *Let $\mathcal{O}$ be a complete discrete valuation ring and let $F := \mathrm{Frac}(\mathcal{O})$ be the fraction field of $\mathcal{O}$. Then, for any finitely generated KG-module V there exists an $\mathcal{O}G$-lattice L which has an $\mathcal{O}$-basis which is also a K-basis. In this situation, we have $V \cong L^K$.*

Proof. See [LP10, Theorem 4.1.4] □

Definition 2.4.6. In the situation of Proposition 2.4.5, we call the $\mathcal{O}G$-lattice L an **$\mathcal{O}$-form** of V.

On the other hand, the question has a negative answer for kG-modules.

Definition 2.4.7. Let $\mathcal{O}$ be a commutative local ring with unique maximal ideal $\mathfrak{p} := J(\mathcal{O})$ and residue field $k := \mathcal{O}/\mathfrak{p}$. A kG-module is called **liftable** if there exists an $\mathcal{O}G$-lattice $\widetilde{M}$ whose reduction modulo $\mathfrak{p}$ is isomorphic to M, that is

$$\widetilde{M}/\mathfrak{p}\widetilde{M} \cong M.$$

Sometimes, it is also said that M is liftable to an $\mathcal{O}G$-lattice, or liftable to $\mathcal{O}$, or liftable to characteristic zero.

Even though every $\mathcal{O}G$-lattice can be reduced modulo $\mathfrak{p}$ to produce a kG-module, not every kG-module is liftable to an $\mathcal{O}G$-lattice. Being liftable is a rather rare property for a kG-module. One interesting class of modules with this property is given by the class of trivial source modules.

Remark 2.4.8. If M is a kG-module, we can also view it as an $\mathcal{O}G$-module via restriction along $\mathcal{O}G \to kG$.

2.5 Brauer characters and decomposition matrices

The prerequisites on our p-modular systems have the implication that K and k both contain a primitive a-th root of unity, where a is the p'-part of the exponent of G. We now examine the relationship between the roots of unity in K and in k. We let

$$\mu_K := \{a\text{-th roots of } 1 \text{ in } K\},$$

$$\mu_k := \{a\text{-th roots of } 1 \text{ in } k\}.$$

Lemma 2.5.1 ([Web16, Lemma 10.1.1])**.** *With the above notation we have:*

(a) *the set μ_K is contained in $\mathcal{O}$, and*

(b) *the quotient homomorphism $\mathcal{O} \to \mathcal{O}/J(\mathcal{O}) = k$ induces an isomorphism $\mu_K \to \mu_k$.*

We write the bijection between a-th roots of unity in K and in k as $\widehat{\xi} \to \xi$, so that if ξ is an a-th root of unity in k then $\widehat{\xi}$ is the root of unity in $\mathcal{O}$ which maps onto it.

Remark 2.5.2. Let $\rho : G \to \mathrm{GL}(V)$ be a k-representation of G. Let $b := |G|_{p'}$ and let $g \in G_{p'}$ be a p-regular element. Then, $\rho(g)^b = 1$. Hence, the minimal polynomial $\mu(x)$ of $\rho(g)$ divides $p(x) := x^b - 1 \in k[x]$. The roots of $p(x)$ are the b distinct b-th roots of unity in k. Hence, $\rho(g)$ is diagonalisable. Moreover, the eigenvalues of $\rho(g)$ are $o(g)$-th roots of unity, since $\rho(g^{o(g)}) = \mathrm{Id}_V$.

This leads to the following definition.

Definition 2.5.3. Let V be a kG-module of dimension $n \in \mathbb{Z}_{\geq 1}$ and let $\rho_V : G \to \mathrm{GL}(V)$ be the associated k-representation. The **Brauer character** of G afforded by V (respectively of ρ_V) is the K-valued function

$$\begin{aligned} \varphi_V : G_{p'} &\to \mathcal{O} \subseteq K \\ g &\mapsto \widehat{\xi}_1 + \cdots + \widehat{\xi}_n, \end{aligned}$$

where $\xi_1, \ldots, \xi_n \in \mu_k$ are the eigenvalues of $\rho_V(g)$. The integer n is also called the **degree** of φ_V. Moreover, φ_V is called **irreducible** if V is simple (resp. if ρ_V is irreducible), and it is called **linear** if $n = 1$. We denote by $\mathrm{IBr}_p(G)$ the set of all irreducible Brauer characters of G and we write $1_{G_{p'}}$ for the Brauer character of the trivial kG-module.

Remark 2.5.4. If the values of Brauer characters are considered as complex numbers, then $\varphi_V(g)$ depends on the choice of the embedding of μ_K into $\mathbb{C}$. However, for a fixed embedding, $\varphi_V(g)$ is uniquely determined up to similarity of $\rho_V(g)$. All of our computations of trivial source character tables employ such a fixed embedding, see Section 5.2.2.

Proposition 2.5.5 ([Web16, Proposition 10.1.3])**.** *Let $(K, \mathcal{O}, k)$ be a p-modular system, let G be a finite group, and let U, V, W be finite dimensional kG-modules. Then the following assertions hold.*

(a) *We have* $\varphi_U(1) = \dim_k U$.

(b) *The Brauer character* φ_U *is a class function on p-regular conjugacy classes.*

(c) *We have* $\varphi_U\left(g^{-1}\right) = \overline{\varphi_U(g)}$.

(d) *The Brauer character of the tensor product of* U *and* V *is given by* $\varphi_{U\otimes V} = \varphi_U \cdot \varphi_V$.

(e) *If*

$$0 \to U \to V \to W \to 0$$

is a short exact sequence of kG-modules then

$$\varphi_V = \varphi_U + \varphi_W.$$

In particular, φ_U *depends only on the isomorphism type of* U*. Furthermore, if the composition factors of* U *are* $S_1, \ldots, S_m$ $(m \in \mathbb{Z}_{\geq 1})$ *with multiplicities* $n_1, \ldots, n_m$ *respectively, then* $\varphi_U = \sum_{i=1}^{m} n_i \varphi_{S_i}$.

(f) *If* U *is liftable to an* $\mathcal{O}G$*-lattice* $\widetilde{U}$ *and the ordinary character of* $\widetilde{U} \otimes_{\mathcal{O}} K$ *is* $\chi_{\widetilde{U}}$*, then* $\varphi_U(g) = \chi_{\widetilde{U}}(g)$ *on p-regular elements* $g \in G$.

Remark 2.5.6. The set $\mathrm{IBr}_p(G)$ of irreducible Brauer characters of G forms a K-basis of the K-vector space $\mathrm{Cl}_K\left(G_{p'}\right)$ of class functions on $G_{p'}$ and we have

$$|\mathrm{IBr}_p(G)| = \dim_K \mathrm{Cl}_K\left(G_{p'}\right) = \text{number of conjugacy classes of } p\text{-regular elements in } G.$$

Next, we investigate the connections between representations of G over K and representations of G over k through the connections between their K-characters and Brauer characters.

Remark 2.5.7. Let V be a KG-module with K-character χ_V. Then:

(a) there exists an $\mathcal{O}G$-lattice L such that $V \cong K \otimes_{\mathcal{O}} L$;

(b) $\chi_V|_{G_{p'}} = \varphi_{\bar{L}}$ and is called the reduction modulo p of χ_V;

(c) if $V \in \mathrm{Irr}_K(G)$, there exist non-negative integers $d_{\chi\varphi}$ such that

$$\chi_V|_{G_{p'}} = \sum_{\varphi \in \mathrm{IBr}_p(G)} d_{\chi\varphi}\varphi.$$

Definition 2.5.8. The matrix $\mathfrak{D}(kG) := \left(d_{\chi,\varphi}\right)_{\substack{\chi\in\mathrm{Irr}_K(G)\\ \varphi\in\mathrm{IBr}_p(G)}}$ is termed **decomposition matrix** of kG and the entries of $\mathfrak{D}(kG)$ are termed **decomposition numbers** of kG.

Remark 2.5.9. (a) The matrix $D^{\mathrm{T}}D = \left(c_{\varphi\mu}\right)_{\varphi,\mu\in\mathrm{IBr}_p(G)}$ is equal to the Cartan matrix $\mathfrak{C}$ of kG.

(b) The decomposition matrix $\mathfrak{D}(kG)$ has full rank, namely $|\mathrm{IBr}_p(G)|$.

(c) The Cartan matrix $\mathfrak{C}$ of kG is a symmetric positive definite matrix with non-negative integer entries.

This is related to the CDE-triangle which we do not define here. For more details we refer to [Web16, Section 9.5]. Since projective kG-modules are liftable, this enables us to associate a K-character of G to each PIM of kG, in fact in a unique way in this case.

Definition 2.5.10. Let $\varphi \in \mathrm{IBr}_p(G)$ be an irreducible Brauer character afforded by a simple kG-module S. Let $P(S)$ be the projective cover of S and let $\widetilde{P(S)}$ denote a lift of $P(S)$ to $\mathcal{O}$. Then, the K-character of $\left(\widetilde{P(S)}\right)^K$ is denoted by Φ_φ and is called the **projective indecomposable character** associated to S or φ.

Remark 2.5.11. Let $\varphi \in \mathrm{IBr}_p(G)$. Then:

(a) $\Phi_\varphi = \sum_{\chi \in \mathrm{Irr}_K(G)} d_{\chi\varphi}\chi$; and

(b) $\Phi_\varphi|_{G_{p'}} = \sum_{\mu \in \mathrm{IBr}_p(G)} c_{\varphi\mu}\mu$.

2.6 Blocks and defect groups

As before, let $(K, \mathcal{O}, k)$ be a splitting p-modular system for G and all of its subgroups. We want to introduce the following notions simultaneously for K, $\mathcal{O}$, and k and, therefore, we let $\mathcal{K} \in \{K, \mathcal{O}, k\}$.

Definition 2.6.1. Let $\mathcal{K}G = B_1 \oplus \cdots \oplus B_n$ be a decomposition of the group algebra $\mathcal{K}G$ into indecomposable $(\mathcal{K}G, \mathcal{K}G)$-subbimodules. The summands $B_1, \ldots, B_n$ are called the **blocks** or **block algebras** of $\mathcal{K}G$. We denote the set of blocks of $\mathcal{K}G$ by $\mathrm{Bl}(\mathcal{K}G)$.

Although we defined blocks here using bimodules, we can still work with one-sided modules as follows.

Convention 2.6.2. If M is a $\mathcal{K}[G \times H]$-module, then we also consider M as a $(\mathcal{K}G, \mathcal{K}H)$-bimodule via

$$gmh := (g, h^{-1})m, \text{ for all } m \in M,\ g \in G,\ h \in H,$$

and vice versa.

Blocks correspond to certain idempotent elements of the group algebra. We refer to [Web16, Chapter 3] for the basics about idempotent elements.

Remark 2.6.3. A block decomposition

$$\mathcal{K}G = B_1 \oplus \cdots \oplus B_n$$

is equivalent to a decomposition

$$1 = e_1 + \cdots + e_n$$

where the e_i are orthogonal primitive central idempotents of $Z(\mathcal{K}G)$ for all $1 \leq i \leq n$. After reordering, we have $e_i = 1_{B_i} \in B_i$ and $B_i = \mathcal{K}Ge_i$ for all $1 \leq i \leq n$. The elements $e_1, \ldots, e_n$ are called **block idempotent elements** of $\mathcal{K}G$.

Definition 2.6.4. An indecomposable $\mathcal{K}G$-module M **belongs to a block** $B_i = \mathcal{K}Ge_i$ if we have $e_iM = M$ and $e_jM = 0$ for all $1 \leq j \leq n$ with $j \neq i$. The block of $\mathcal{K}G$ that contains the trivial module is called the **principal block** $B_0(\mathcal{K}G)$ of $\mathcal{K}G$.

There is a correspondence between the blocks of $\mathcal{O}G$ and kG. In order to understand this relation, we first have to consider the notion of lifting of idempotent elements.

Remark 2.6.5. Every idempotent element of kG can be lifted to an idempotent element of $\mathcal{O}G$, see [Web16, Theorem 7.3.5 & Corollary 7.3.6]: for any idempotent element

$$\bar{e} \in kG = [\mathcal{O}/J(\mathcal{O})]G$$

there exists an idempotent element $e \in \mathcal{O}G$ such that $\bar{e} = e + J(\mathcal{O})$. Moreover, reduction modulo $\mathfrak{p}$ maps an idempotent element $e \in \mathcal{O}G$ to an idempotent element $\bar{e} \in kG$. In fact, this induces a bijection between the primitive idempotent elements of $Z(\mathcal{O}G)$ and the primitive idempotent elements of $Z(kG)$. This bijection maps any decomposition of the identity element of $\mathcal{O}G$ into a sum of primitive central idempotents to a decomposition of the identity element of kG into a sum of primitive central idempotents of kG, i.e.

$$1_{\mathcal{O}G} = e_1 + \cdots + e_r \mapsto 1_{kG} = \bar{e}_1 + \cdots + \bar{e}_r.$$

This induces a bijection between the blocks of $\mathcal{O}G$ and the blocks of kG via

$$\{\text{blocks of } \mathcal{O}G\} \to \{\text{blocks of } kG\}, \quad B_i = \mathcal{O}Ge_i \mapsto \overline{B}_i := kG\bar{e}_i.$$

Definition 2.6.6. Using the correspondence from Remark 2.6.5, we say that the blocks of $\mathcal{O}G$ or kG are the ***p*-blocks** of G.

The name *block* originates from the following property of the Cartan matrix.

Remark 2.6.7. If we group the simple kG-modules together in their respective blocks, the Cartan matrix of kG has a block diagonal form where every block matrix corresponds to a block of kG. This is, up to a permutation of the blocks or the modules in the same block, the unique finest decomposition of the Cartan matrix into block diagonal form, see [Web16, Corollary 12.1.8].

For the rest of this section, we only consider the blocks of kG and keep in mind that they imply analogous results for the blocks of $\mathcal{O}G$. We want to associate a p-subgroup of G to every block of kG. In order to do this, we need the diagonal embedding of G in $G \times G$ and denote it by $\Delta : G \longrightarrow G \times G, g \mapsto (g, g)$.

Theorem 2.6.8 ([Web16, Corollary 12.3.2])**.** *Let B be a block of kG that we consider as an indecomposable $k[G \times G]$-module. Then every vertex of B is of the form $\Delta(D)$ where D is some p-subgroup of G. The subgroup D is unique up to conjugation in G.*

Definition 2.6.9. Let B be a block of kG. We say that a p-group $D \leq G$ is a **defect group** of B if $\Delta(D)$ is a vertex of B as a $k[G \times G]$-module. The **defect** of B is the integer d with $|D| = p^d$. We often write $D(B)$ instead of D in order to indicate that D is a defect group of the block algebra B.

By Theorem 2.6.8, defect groups are unique up to conjugation in G. This also shows that the defect of a block is well-defined. It follows from the definition and [Web16, Corollary 12.4.6] that every indecomposable kG-module belonging to a block $B \in \mathrm{Bl}(kG)$ with defect group D is relatively D-projective.

Next, we define the important notion of Brauer correspondence and state Brauer's First Main Theorem.

Definition 2.6.10. Let H be a subgroup of G and let b be a block of kH. We say that a block $B \in \mathrm{Bl}(kG)$ **corresponds to** b if and only if B is the unique block of kG such that b is a summand of $B \downarrow^{G \times G}_{H \times H}$. In this case, we write $B = b^G$.

Theorem 2.6.11 (Brauer's First Main Theorem)**.** *For any p-group $D \leq G$ there is a bijection*

$$\{\textit{blocks of } kN_G(D) \textit{ with defect group } D\} \to \{\textit{blocks of } kG \textit{ with defect group } D\}$$

*given by $b \mapsto b^G$. Then b and b^G are called **Brauer correspondents**.*

Proof. See [Web16, Theorem 12.6.4]. □

With this, we can state Broué's famous conjecture about blocks with abelian defect groups. It asserts a categorical equivalence between a block and its Brauer correspondent.

Broué's Abelian Defect Group Conjecture ([Bro90]). Let B be a block of kG with abelian defect group D and let $b \in \mathrm{Bl}(N_G(D))$ be its Brauer correspondent. Then the derived categories $\mathcal{D}^b(B\text{-mod})$ and $\mathcal{D}^b(b\text{-mod})$ of bounded complexes of finitely generated modules over B and b are equivalent as triangulated categories.

Remark 2.6.12. A few years later, Rickard strengthened this in [Ric96] by conjecturing that there even exists a splendid derived equivalence between B and b. This is especially interesting for us, since the bimodules involved in the complexes are trivial source bimodules. See Definition 2.10.29 for the definition of a splendid derived equivalence.

2.7 Green rings and Grothendieck rings

Trivial source character tables can also be seen as matrices collecting evaluations of certain algebra homomorphisms at the trivial source modules. In this chapter, we briefly present this approach, as well as some related notions, since it is often useful for calculations to have both this view point and the view point via ordinary character theory at hand.

Notation-Definition 2.7.1. (a) If $\mathscr{C}$ is any additive category, the **Grothendieck group** $\mathscr{R}(\mathscr{C})$ of $\mathscr{C}$ is the group with one generator $[X]$ for all $X \in \mathrm{Ob}(\mathscr{C})$ and relations $[X] = [X'] + [X'']$ whenever there is an exact sequence $0 \to X' \to X \to X'' \to 0$ in $\mathscr{C}$.

(b) Since the Grothendieck group $\mathscr{R}(\mathscr{C})$ of a monoidal additive category $\mathscr{C}$ inherits a ring structure from the tensor product in $\mathscr{C}$, $\mathscr{R}(\mathscr{C})$ becomes a ring in this case, called the **Grothendieck ring**.

(c) If $\mathbb{F}$ is a field and G is a finite group, then we define the **Green ring** $a(\mathbb{F}G)$ of G over $\mathbb{F}$ to be the free abelian group on the set of isomorphism classes $[M]$ of indecomposable $\mathbb{F}G$-modules, with addition given by taking direct sums and multiplication induced by the tensor product over $\mathbb{F}$.

(d) We define $a_0(\mathbb{F}G, 1)$ to be the ideal of $a(\mathbb{F}G)$ spanned by the difference elements $[M_2] - [M_1] - [M_3]$ where $0 \to M_1 \to M_2 \to M_3 \to 0$ is a short exact sequence of $\mathbb{F}G$-modules.

(e) Set $\mathscr{R}(\mathbb{F}G) := \mathscr{R}({}_{\mathbb{F}G}\mathrm{mod})$, $A(\mathbb{F}G) := \mathbb{C} \otimes_{\mathbb{Z}} a(\mathbb{F}G)$, and $A_0(\mathbb{F}G, 1) := \mathbb{C} \otimes_{\mathbb{Z}} a_0(\mathbb{F}G, 1)$.

Proposition 2.7.2 ([Ben98, Proposition 5.2.3 & Theorem 5.3.3]). (a) *If* $\boxed{\mathbb{F} = \mathbb{C}}$*, then* $a(\mathbb{F}G) = \mathscr{R}(\mathbb{F}G)$ *and every ring homomorphism* $a(\mathbb{F}G) \to \mathbb{C}$ *is given by a trace map* $t_g : [M] \mapsto \mathrm{tr}(g, M)$ *with* $g \in G$*.*

Moreover, $\sum_{g \in ccls(G)} t_g : A(\mathbb{F}G) \xrightarrow{\sim} \bigoplus_{ccls(G)} \mathbb{C}$ *is an isomorphism and hence* $A(\mathbb{C}G)$ *is semisimple.*

(b) *If* $\boxed{\mathbb{F} = k}$*, then* $\mathscr{R}(\mathbb{F}G) = a(kG)/a_0(kG, 1)$ *and every ring homomorphism* $\mathscr{R}(\mathbb{F}G) \to \mathbb{C}$ *is given by a map* $t_g : [M] \mapsto t_g(M)$ *where* $g \in G_{p'}$ *and* $t_g(M) := \sum$ *lifts to* $\mathbb{C}$ *of the eigenvalues of* g *on* $\mathrm{Res}^G_{\langle g \rangle}(M)$*.*

Moreover, $\sum_{g \in ccls_{p'}(G)} t_g : A(\mathbb{F}G) \xrightarrow{\sim} \bigoplus_{ccls_{p'}(G)} \mathbb{C}$ *is an isomorphism and hence* $A(kG)$ *is semisimple.*

In [BP84], Benson and Parker further generalised these constructions as follows:

Definition 2.7.3. A **species** of any subalgebra, ideal or quotient A of $A(\mathbb{F}G)$ is an algebra homomorphism $A \to \mathbb{C}$.

Example 2.7.4. (a) Evaluating the species of $A(\mathbb{C}G)$ at the simple $\mathbb{C}G$-modules yields the species table of $A(\mathbb{C}G)$ which is just the ordinary character table of G.

(b) The species table of $A(kG)$ - calculated by evaluating the species of $A(kG)$ at the simple kG-modules - is just the p-Brauer character table of G.

Remark 2.7.5. Via this approach, ordinary character tables and Brauer character tables are just two different instances of species tables.

2.8 Permutation modules

In this section we consider permutation modules and some of their properties.

Definition 2.8.1. Let Ω be a finite, non-empty G-set. We denote by $R\Omega$ the RG-module which is, as an R-module, free having the set Ω as basis, with RG-module structure obtained by extending bilinearly the action of elements of G on elements of Ω. More explicitly, the elements of $R\Omega$ are formal sums $\sum_{m \in \Omega} \mu_m m$ endowed with the action of RG given by

$$\left(\sum_{x \in G} \lambda_x x\right) \cdot \left(\sum_{m \in \Omega} \mu_m m\right) = \sum_{x \in G} \sum_{m \in \Omega} \lambda_x \mu_m x m$$
$$= \sum_{m \in \Omega} \left(\sum_{(x,n) \in G \times \Omega,\, xn = m} \lambda_x \mu_n\right) m.$$

An RG-module V is called a **permutation module** if $V \cong R\Omega$ for some G-set Ω, or equivalently, if V is R-free and has an R-basis B that is G-stable. If G acts transitively on an R-basis of V, then V is called a **transitive permutation module**.

Example 2.8.2. (a) The trivial RG-module R is a permutation module obtained from the trivial action of G on a set with one element.

(b) The regular left RG-module is the permutation module obtained from the regular action of G on itself by left multiplication.

Remark 2.8.3. Transitive permutation modules need not be indecomposable.

Remark 2.8.4. Each element $g \in G$ acts on a permutation module $R\Omega$ by means of a permutation matrix with respect to the basis B. Such a permutation matrix has trace equal to the number of entries 1 on the diagonal, the other diagonal entries being 0. The number 1 on the diagonal is produced precisely each time the corresponding basis element is fixed by g.

We conclude this section with a definition which is related to permutation modules and which is needed in Section 2.10.

Definition 2.8.5. An RG-module M is called a **p-permutation module** if, for some Sylow p-subgroup P of G, the restriction $M{\downarrow}_P^G$ is a permutation RP-module.

2.9 The Burnside ring and the table of marks

Definition 2.9.1. Let Ω be a G-set and let $H \subseteq G$ be a subset of G. We put

$$\mathrm{Fix}_\Omega(g) := \{\omega \in \Omega \mid g \cdot \omega = \omega\} \quad \text{and} \quad \mathrm{Fix}_\Omega(H) := \bigcap_{h \in H} \mathrm{Fix}_\Omega(h).$$

The elements of $\mathrm{Fix}_\Omega(H)$ are called the **fixed points** of H on Ω. They are often also denoted by Ω^H.

It is a consequence of Remark 2.8.4 that for a transitive permutation module $\mathbb{C}\Omega$ the value of its character at g equals the number of fixed points of g on Ω:

$$\chi_{\mathbb{C}\Omega}(g) = |\mathrm{Fix}_\Omega(\langle g \rangle)|.$$

In the following, we derive even more pieces of information from permutation modules. In this section we follow [LP10].

Definition 2.9.2. Let G be a finite group and let p be a prime number. We denote by $\mathscr{S}(G)$ a complete set of representatives of the set of conjugacy classes of subgroups of G. Moreover, we denote by $\mathscr{S}_p(G)$ a complete set of representatives of the set of conjugacy classes of p-subgroups of G. Note that whenever we use $\mathscr{S}(G)$ and $\mathscr{S}_p(G)$ there is a choice involved.

We recall that every transitive G-set is isomorphic to a set of left cosets G/H for some subgroup $H \leq G$ and that G/H is isomorphic as a G-set to G/U if and only if H and U are conjugate in G.

Definition 2.9.3. Let G be a finite group. Choose $\mathscr{S}(G) = \{H_1, \ldots, H_n\}$, where we suppose that that $|H_i| \leq |H_j|$ for every $1 \leq i \leq j \leq n$.

(a) If Ω is a G-set then the function $m_\Omega : \mathscr{S}(G) \to \mathbb{Z}_{\geq 0} : \ H \mapsto |\mathrm{Fix}_\Omega(H)|$ is called the **mark** of Ω. (It is often considered as a row-vector.)

(b) The **table of marks** of G is the square matrix $\mathcal{M}(G) := [m_{G/H_i}(H_j)]_{1 \leq i,j \leq n} \in \mathrm{Mat}_{n \times n}(\mathbb{Q})$.

Remark 2.9.4. It is well-known that conjugate subgroups of G have the same number of fixed points on any G-set. Thus the mark of a G-set is independent of the choice of representatives in $\mathscr{S}(G)$ and likewise $\mathcal{M}(G)$. Of course, $\mathcal{M}(G)$ depends on the ordering of $\mathscr{S}(G)$.

Lemma 2.9.5 ([LP10, Corollary 3.5.4])**.** *Let G be a finite group.*

(a) *The matrix $\mathcal{M}(G)$ is invertible in* $\mathrm{Mat}_{n \times n}(\mathbb{Q})$.

(b) *Two finite G-sets Ω and Ω' are isomorphic if and only if they have the same mark, i.e. if and only if $m_\Omega = m_{\Omega'}$.*

(c) *If Ω is a finite G-set with mark m_Ω and with a_i orbits isomorphic to G/H_i then*

$$[a_1, \ldots, a_n] = m_\Omega \cdot \mathcal{M}(G)^{-1},$$

where $[a_1, \ldots, a_n]$ is the row vector with the entries $a_1, \ldots, a_n$ in that order.

We recall that for two G-sets M, N the disjoint union $M\dot{\cup}N$ and the Cartesian product $M \times N$ are also G-sets in a natural way. Since

$$\mathrm{Fix}_{M\dot{\cup}N}(H) = \mathrm{Fix}_M(H)\dot{\cup}\,\mathrm{Fix}_N(H)$$

and

$$\mathrm{Fix}_{M\times N}(H) = \mathrm{Fix}_M(H) \times \mathrm{Fix}_N(H)$$

for G-sets M, N and any subgroup $H \leq G$, it follows that

$$m_{M\dot{\cup}N} = m_M + m_N, \quad m_{M\times N} = m_M \cdot m_N,$$

with the usual point-wise operations. Thus the $\mathbb{Z}$-linear combinations of marks of G form a ring which we denote by $R_{\mathcal{M}}$.

Definition 2.9.6. The **Burnside ring** $\mathcal{B}(G)$ is the Grothendieck ring of the category of finite G-sets; i.e. as an abelian group $\mathcal{B}(G) = F/F_0$, where F is the free abelian group generated by the isomorphism classes (M) of finite G-sets and

$$F_0 = \langle (M\dot{\cup}N) - (M) - (N) \mid M, N \text{ finite } G\text{-sets} \rangle_{\mathbb{Z}}\,.$$

Defining $(M) \cdot (N) := (M \times N)$ on F, and extending $\mathbb{Z}$-linearly, F becomes a ring and F_0 an ideal, so $\mathcal{B}(G)$ also becomes a ring. Moreover, for any G-set M we denote the image of (M) in $\mathcal{B}(G)$ by $[M]$.

Lemma 2.9.7 ([CR87, (80.4) Lemma & (80.6) Corollary])**.** *We keep our notation from Definition 2.9.3 and Definition 2.9.6.*

(a) *Let M, N be G-sets. Then we have $[M] = [N]$ if and only if M and N are isomorphic as G-sets.*

(b) *The Burnside ring $\mathcal{B}(G)$ is a commutative ring with $\mathbb{Z}$-basis $([G/H_1], \ldots, [G/H_n])$, and with identity element $[G/G]$.*

Lemma 2.9.8. *The rings $\mathcal{B}(G)$ and $R_{\mathcal{M}}$ are isomorphic as rings.*

Proof. We construct a ring isomorphism $\alpha : \mathcal{B}(G) \xrightarrow{\sim} R_{\mathcal{M}}$. Define α on the basis elements of $\mathcal{B}(G)$ given by Lemma 2.9.7 as follows: let

$$\alpha([G/H_i]) := m_{G/H_i} \text{ for all } i \in \{1, \ldots, n\}.$$

Then, for any $1 \leq i, j \leq n$, we have by Lemma 2.9.5(b) that $\alpha([G/H_i] \cdot [G/H_j]) =$

$$\alpha([G/H_i \times G/H_j]) = m_{G/H_i \times G/H_j} = m_{G/H_i} \cdot m_{G/H_j} = \alpha([G/H_i]) \cdot \alpha([G/H_j])$$

and that $\alpha([G/H_i] + [G/H_j]) =$

$$\alpha([G/H_i\dot{\cup}G/H_j]) = m_{G/H_i\dot{\cup}G/H_j} = m_{G/H_i} + m_{G/H_j} = \alpha([G/H_i]) + \alpha([G/H_j]).$$

□

The product of marks m_{G/H_i} of transitive G-sets has a simple group-theoretical interpretation as follows.

Lemma 2.9.9 ([LP10, Lemma 3.5.12])**.** *For $H_i, H_j \in \mathscr{S}(G)$ let $D_{ij} := [H_i \setminus G/H_j]$ be a set of double coset representatives of H_i, H_j in G, i.e.*

$$G = \bigsqcup_{d \in D_{ij}} H_i d H_j.$$

Then

$$m_{G/H_i} \cdot m_{G/H_j} = \sum_{d \in D_{ij}} m_{G/\left(H_i^d \cap H_j\right)}.$$

Remark 2.9.10. Since any transitive action of G is equivalent to an action on the cosets of a subgroup of G, one sees that the table of marks completely characterizes the set of all permutation representations of G. For further general facts about Burnside rings we refer to [LP10], [CR87], [Bou00], and [Pfe97].

2.10 Categorical equivalences and related concepts

This section is concerned with categorical equivalences and isomorphisms between certain Grothendieck groups.

Remark 2.10.1.
(a) Let A and B be finite-dimensional K-algebras. We set $\mathscr{R}(A, B) := \mathscr{R}(A \otimes_K B^{\mathrm{op}})$, where B^{op} denotes the opposite algebra of B.

(b) For an idempotent element $e \in Z(KG)$ we identify $\mathscr{R}(KGe)$ with the virtual character group of KGe, the free $\mathbb{Z}$-span of $\mathrm{Irr}_K(KGe)$. This way, $\mathscr{R}(KGe) \subseteq \mathscr{R}(KG)$. Similarly, if e is an idempotent element in $Z(kG)$ we identify $\mathscr{R}(kGe)$ with the group of virtual Brauer characters belonging to kGe. This way, $\mathscr{R}(kGe) \subseteq \mathscr{R}(kG)$. By scalar extension from $\mathbb{Z}$ to K we view these Grothendieck rings also as embedded into K-vector spaces.

(c) Let $e \in Z(KG)$ and $f \in Z(KH)$ be idempotent elements. By Part (a) we obtain the Grothendieck group $\mathscr{R}(KGe, KHf) := \mathscr{R}(K[G \times H](e \otimes_K f^*))$. Here, $-^* : KH \to KH$ denotes the K-algebra anti-isomorphism given by $h \mapsto h^{-1}$.

2.10.1 Perfect isometries and isotypies

Perfect isometries are certain character bijections 'with signs' between blocks of finite groups which preserve many numerical and structural invariants of the blocks. They were introduced by Broué in the 1990's in order to relate the character theories of p-blocks of finite groups. See [Bro90]. For the definition and basic properties of Brauer pairs we refer to [Nav98].

Definition 2.10.2. If f is a block idempotent element of the group algebra kG, then the block idempotent element of $\mathcal{O}G$ corresponding to f is denoted by $\widetilde{f}$ (wherefore, in particular, $\overline{\widetilde{f}} = f$).

Definition 2.10.3. Let x be a p-element of G and let e be a central idempotent element of $kC_G(x)$. We define the **generalized decomposition map** $d_G^{(x,e)} : K\mathscr{R}(KG) \to K\mathscr{R}\left(KC_G(x)\widetilde{e}\right)$ by

$$d_G^{(x,e)}(\chi)\left(x'\right) := \begin{cases} \chi\left(xx'\widetilde{e}\right) & \text{if } x' \in C_G(x) \text{ is a } p'\text{-element;} \\ 0 & \text{if } x' \in C_G(x) \text{ is not a } p'\text{-element.} \end{cases}$$

If $e = 1$, we set $d_G^{(x,e)} =: d_G^x$. If additionally $x = 1$, then d_G^x is called the **decomposition map** and is denoted by d_G.

Definition 2.10.4. Let σ and τ be central idempotents of $\mathcal{O}G$ and $\mathcal{O}H$, respectively. Following [Bro90], we define a **perfect isometry** between $\mathcal{O}G\sigma$ and $\mathcal{O}H\tau$ as an element $\mu \in \mathscr{R}(KG\sigma, KH\tau)$ satisfying the following conditions.

(a) We have $\mu(g,h) \in |C_G(g)|\,\mathcal{O} \cap |C_H(h)|\,\mathcal{O}$ for every $g \in G$ and $h \in H$.

(b) If $g \in G$ and $h \in H$ such that $\mu(g,h) \neq 0$, then g is a p'-element if and only if h is a p'-element.

(c) The map $I_\mu : \mathscr{R}(KH\tau) \to \mathscr{R}(KG\sigma), \chi \mapsto \mu \otimes_{KH} \chi$, is an isometry.

Perfect isometries between block algebras tend to come in families (called isotypies) that are compatible with the local structure and the decomposition maps of the considered blocks. In the following, the element $\widetilde{f}$ is the block idempotent element of $\mathcal{O}G$ corresponding to $f \in kG$.

Definition 2.10.5. Let G and H be finite groups, let $B = kGe_B$ be a block of kG, and let $B' = kHe_{B'}$ be a block of kH. An **isotypy** between B and B' is a tuple

$$I = \left(D, e, \varphi, E, f, (\mu_W)_{W \leq E}\right)$$

satisfying the following conditions:

(a) The pair (D,e) is a maximal B-Brauer pair, the pair (E,f) is a maximal B'-Brauer pair, and $\varphi : E \to D$ is an isomorphism. Let $\mathcal{B}$ be the fusion system associated with (D,e) and let $\mathcal{B}'$ be the fusion system associated with (E,f). For $Q \leq D$, let e_Q denote the unique block idempotent element of $kC_G(Q)$ such that $(Q,e_Q) \leq (D,e)$, and for $W \leq E$, let f_W denote the unique block idempotent element of $kC_H(W)$ such that $(W, f_W) \leq (E,f)$.

(b) The isomorphism $\varphi : E \to D$ is an isomorphism between $\mathcal{B}'$ and $\mathcal{B}$.

(c) For every $W \leq E$, μ_W is a perfect isometry between $kC_H(W)f_W$ and $kC_G(Q)e_Q$, where $Q := \varphi(W)$. We denote the K-linear map

$$K\mathscr{R}\left(KC_H(W)\widetilde{f_W}\right) \to K\mathscr{R}\left(KC_G(Q)\widetilde{e_Q}\right),\ \chi \mapsto \mu \otimes_{KC_H(W)} \chi$$

by I_W.

(d) Let $W \leq E$ and let $Q = \varphi(W)$. For $g \in G$ and $h \in H$ such that $c_g \in \mathrm{Hom}_{\mathcal{B}}(Q,D)$ and $c_h \in \mathrm{Hom}_{\mathcal{B}'}(W,E)$ such that $c_h = \varphi^{-1} \circ c_g \circ \varphi$ in $\mathrm{Hom}_{\mathcal{B}'}(W,E)$, we have $I_{{}^hW} = {}^{(g,h)}I_W$, where ${}^{(g,h)}I_W$ denotes the K-linear map $c_g \circ I_W \circ c_{h^{-1}}$.

(e) Let $W \leq E$, let $Q = \varphi(W)$, let $y \in C_E(W)$, and let $x = \varphi(y) \in C_D(Q)$. The equality

$$d^{(x, e_{Q\langle x\rangle})}_{C_G(Q)} \circ I_W = I_{W\langle y\rangle} \circ d^{(y, f_{W\langle y\rangle})}_{C_H(W)}$$

holds.

Example 2.10.6 ([Sam20, Example 8.2])**.** Let B be the principal 2-block of $G := \mathfrak{A}_5$. Then $D := D(B) = V_4$ and $N := \mathrm{N}_G(D) = \mathfrak{A}_4$. Let b_D denote the Brauer correspondent block of B in $N_G(D)$. The blocks B and b_D are perfectly isometric via $\chi_1 \mapsto -\psi_1$ and $\chi_i \mapsto \psi_i$ for $i = 2,3,4$ as can be seen from the ordinary character tables of G and N. Moreover, B and b_D are isotypic.

2.10.2 (Splendid) Morita equivalences and related notions

In this subsection, we let R be a field or a complete discrete valuation ring.

Definition 2.10.7. Let A, B be R-algebras. We say that A and B are **Morita equivalent**, if the abelian categories ${}_A\mathrm{Mod}$ and ${}_B\mathrm{Mod}$ are equivalent as R-linear categories. Moreover, a functor $F : {}_A\mathrm{Mod} \to {}_B\mathrm{Mod}$ defining such an equivalence is called a **Morita equivalence between** A **and** B.

Theorem 2.10.8 ([Lin18a, Theorem 2.8.2])**.** *Let A, B be R-algebras. The following assertions are equivalent.*

(a) *The algebras A and B are Morita equivalent.*

(b) *The R-linear categories of finitely generated modules ${}_A\mathrm{mod}$ and ${}_B\mathrm{mod}$ are equivalent.*

(c) *There is a progenerator M of A such that $\mathrm{End}_A(M)^{\mathrm{op}} \cong B$.*

(d) *There exist an (A, B)-bimodule M and a (B, A)-bimodule N such that $M \otimes_B N \cong A$ as (A, A)-bimodules and $N \otimes_A M \cong B$ as (B, B)-bimodules, and such that M, N are finitely generated projective as left and right modules. In that case, M and N are left and right progenerators, and we have bimodule isomorphisms:*

$$N \cong \mathrm{Hom}_A(M, A) \cong \mathrm{Hom}_{B^{\mathrm{op}}}(M, B),$$
$$M \cong \mathrm{Hom}_B(N, B) \cong \mathrm{Hom}_{A^{\mathrm{op}}}(N, A).$$

(e) *There exist an (A, B)-bimodule M and a (B, A)-bimodule N, both finitely generated projective as left and right modules, and there are bimodule isomorphisms $\Phi : M \otimes_B N \to A$ and $\Psi : N \otimes_A M \to B$ such that*

$$\mathrm{Id}_N \otimes \Phi = \Psi \otimes \mathrm{Id}_N, \Phi \otimes \mathrm{Id}_M = \mathrm{Id}_M \otimes \Psi,$$

where we identify $N \otimes_A A = N = B \otimes_B N$ and $M \otimes_B B = M = A \otimes_A M$.

Definition 2.10.9. An (A, B)-bimodule M and a (B, A)-bimodule N are said to **induce a Morita equivalence between** A **and** B if M, N are finitely generated projective as left and right modules, and if there are bimodule isomorphisms $M \otimes_B N \cong A$ and $N \otimes_A M \cong B$.

The notion of relative projectivity generalises to arbitrary subalgebras of an algebra:

Definition 2.10.10. Let A be an R-algebra and let B be a subalgebra of A. A module $U \in {}_A\mathrm{Mod}$ is called **relatively B-projective** if there exists a module $V \in {}_B\mathrm{Mod}$ such that U is isomorphic to a direct summand of $A \otimes_B V$.

Definition 2.10.11. A homomorphism $\varphi : U \to V$ of modules $U, V \in {}_A\mathrm{Mod}$ **factors through a relatively R-projective module**, if there is a relatively R-projective module $P \in {}_A\mathrm{Mod}$ and A-homomorphisms $\alpha : U \to P$ and $\beta : P \to V$ such that $\varphi = \beta \circ \alpha$. We denote by $\mathrm{Hom}_A^{\mathrm{pr}}(U, V)$ the subset of $\mathrm{Hom}_A(U, V)$ consisting of all A-homomorphisms from U to V that factor through a relatively R-projective module.

Remark 2.10.12. If $\varphi, \varphi' \in \mathrm{Hom}_A(U, V)$ factor through relatively R-projective modules P, P', respectively, then for any scalar $\lambda \in R$ the homomorphism $\lambda\varphi$ factors through P and the sum $\varphi + \varphi'$ factors through the direct sum $P \oplus P'$. Thus $\mathrm{Hom}_A^{\mathrm{pr}}(U, V)$ is an R-submodule of $\mathrm{Hom}_A(U, V)$. Given two composable A-homomorphisms φ, ψ, if one of them factors through a relatively R-projective A module, then so does their composition $\psi \circ \varphi$.

Hence, the following is well-defined:

Definition 2.10.13. Let A be an R-algebra. The **R-stable category of** ${}_A\mathrm{Mod}$, denoted ${}_A\underline{\mathrm{Mod}}$, is the category whose objects are all the A-modules and, for any two A-modules U, V, the space of morphisms from U to V in ${}_A\underline{\mathrm{Mod}}$ is the quotient space

$$\underline{\mathrm{Hom}}_A(U,V) = \mathrm{Hom}_A(U,V)/\mathrm{Hom}_A^{\mathrm{pr}}(U,V)$$

with composition of morphisms in ${}_A\underline{\mathrm{Mod}}$ induced by the composition of A-homorphisms in ${}_A\mathrm{Mod}$.

Remark 2.10.14. Thus a morphism from U to V in ${}_A\underline{\mathrm{Mod}}$ is a class of A-homomorphisms, usually denoted $\underline{\varphi}$, of an A-homomorphism $\varphi : U \to V$ modulo A-homomorphisms that factors through a relatively R-projective A-module.

Relatively R-projective modules are exactly the modules that get identified to zero in the R-stable category:

Proposition 2.10.15 ([Lin18a, Proposition 2.13.3])**.** *Let A be an R-algebra and $U \in {}_A\mathrm{Mod}$. The following are equivalent.*

(a) *The module U is isomorphic to the zero object in ${}_A\underline{\mathrm{Mod}}$.*

(b) *The morphism $\underline{\mathrm{Id}}_U$ is the zero endomorphism of U in $\underline{\mathrm{Hom}}_A(U,U)$.*

(c) *The module U is relatively R-projective.*

Lemma 2.10.16 ([Lin18a, Corollary 2.13.5])**.** *Let A be an R-algebra such that A is finitely generated as an R-module. Let $U, V \in {}_A\mathrm{Mod}$. Suppose that V is finitely generated. A homomorphism $\varphi : U \to V$ factors through a relatively R-projective A-module if and only if it factors through a finitely generated relatively R-projective A-module.*

Definition 2.10.17. Let A, B be R-algebras, let $M \in {}_A\mathrm{Mod}_B$, and let $N \in {}_B\mathrm{Mod}_A$. We say that M and N induce a **stable equivalence of Morita type between A and B** if M, N are finitely generated projective as left and right modules with the property that $M \otimes_B N \cong A$ in ${}_{A\otimes_R A^{\mathrm{op}}}\underline{\mathrm{Mod}}$ and $N \otimes_A M \cong B$ in ${}_{B\otimes_R B^{\mathrm{op}}}\underline{\mathrm{Mod}}$.

Proposition 2.10.18 ([Lin18a, Proposition 2.17.5])**.** *Let A, B be R-algebras, let $M \in {}_A\mathrm{Mod}_B$, and let $N \in {}_B\mathrm{Mod}_A$. Suppose further that M and N induce a stable equivalence of Morita type between A and B. Then the functors $M \otimes_B -$ and $N \otimes_A -$ induce inverse equivalences ${}_A\underline{\mathrm{Mod}} \cong {}_B\underline{\mathrm{Mod}}$.*

Corollary 2.10.19. Let A, B be R-algebras and let $\mathscr{M} : {}_A\mathrm{Mod} \to {}_B\mathrm{Mod}$ be a Morita equivalence. Then $\mathscr{M}$ induces a stable equivalence ${}_A\mathrm{mod} \cong {}_B\mathrm{mod}$.

Proof. By Theorem 2.10.8, $\mathscr{M}$ induces a stable equivalence of Morita type $\mathscr{M}_{\mathrm{stM}}$ between A and B. It follows from Proposition 2.10.18 that $\mathscr{M}_{\mathrm{stM}}$ induces an equivalence $\mathcal{F}$: ${}_A\underline{\mathrm{Mod}} \to {}_B\underline{\mathrm{Mod}}$. Lemma 2.10.16 implies that the R-stable category ${}_A\underline{\mathrm{mod}}$ of finitely generated A-modules can be identified with the full subcategory of ${}_A\underline{\mathrm{Mod}}$ of all finitely generated A-modules. As under an equivalence of categories ${}_A\mathrm{Mod} \to {}_B\mathrm{Mod}$ finitely generated modules are mapped into finitely generated modules, the result follows. □

Definition 2.10.20. Let A, B be R-algebras. A Morita equivalence between the algebras A and B is termed **splendid Morita equivalence** if the (A,B)-bimodule M from Theorem 2.10.8(e) can be chosen to be a trivial source (A,B)-bimodule.

2.10.3 p-permutation equivalences

Notation-Definition 2.10.21. Let $p_1 : G\times H \to G$ be the projection homomorphism of $G\times H$ onto G and let $p_2 : G\times H \to H$ be the projection homomorphism of $G\times H$ onto H. For a subgroup U of $G\times H$, we denote by $k_1(U)$ the normal subgroup $\{g \in G \mid (g,1) \in U\}$ of $p_1(U)$ and by $k_2(U)$ the normal subgroup $\{h \in H \mid (1,h) \in U\}$ of $p_2(U)$.

Definition 2.10.22. A subgroup U of $G \times H$ is called **twisted diagonal** if there are isomorphic subgroups Q of G and W of H and an isomorphism $\varphi : W \to Q$ such that $U = \{(\varphi(w), w) \mid w \in W\}$. In this case, we denote the twisted diagonal subgroup U by $\Delta(Q,\varphi,W)$. For $Q \leq G$, we denote the subgroup $\Delta\left(Q, 1_Q, Q\right)$ of $G\times G$ by ΔQ.

Note that a subgroup U of $G \times H$ is twisted diagonal if and only if $k_1(U) = k_2(U) = 1$. If X is a subgroup of $G \times H$, then the subgroup $X^o := \{(h,g) \in H \times G \mid (g,h) \in G \times H\}$ of $H \times G$ is called the **opposite subgroup** of X.

By $\mathrm{T}(kG)$ we denote the Grothendieck ring of the category of p-permutation kG-modules, see Notation-Definition 3.2.1. The isomorphism classes $[M]$ of indecomposable p-permutation kG-modules M form a $\mathbb{Z}$-basis of $\mathrm{T}(kG)$, and for any p-permutation kG-module M we denote by $[M]$ its associated element in $\mathrm{T}(kG)$. For a p-subgroup P of G and an idempotent element σ of $Z(kG)$ we denote by $\mathrm{T}^P(kG\sigma)$ the subgroup of $\mathrm{T}(kG)$ generated by the elements $[M]$ where M is a relatively P-projective p-permutation kG-module satisfying $\sigma M = M$.
Let σ (resp. τ) be an idempotent element of $Z(kG)$ (resp. $Z(kH)$). Then we set $\mathrm{T}(kG\sigma, kH\tau) := \mathrm{T}(k[G \times H](\sigma \otimes \tau^*))$. Here, $-^* : kH \to kH$ denotes the k-algebra anti-isomorphism given by $h \mapsto h^{-1}$. For a p-subgroup S of G denote by $\mathrm{T}^S(kG\sigma)$ the subgroup of $\mathrm{T}(kG)$ generated by the elements $[M]$ where M is a relatively S-projective RG-module satisfying $M\sigma = M$. We set $\mathrm{T}^S(kG\sigma, kH\tau) := \mathrm{T}^S(k[G \times H](\sigma \otimes \tau^*))$ by identifying $k[G\times H]$ with $kG\otimes_k kH$ via $(g,h) \mapsto g\otimes h$. Consequently, we can also identify $\mathrm{T}(k[G \times H])$ and $\mathrm{T}(kG, kH)$.

Note that $-\otimes_{kH} -$ induces a $\mathbb{Z}$-linear map

$$\mathrm{T}(kG, kH) \times \mathrm{T}(kH, kL) \to \mathrm{T}(kG, kL), \quad (\alpha, \beta) \mapsto \alpha \cdot_H \beta,$$

for any group L. By abuse of notation, we sometimes write $\alpha \otimes_{kH} \beta$ for $\alpha \cdot_H \beta$.
For a (kG, kH)-bimodule M, we usually view its k-dual $M^\vee := \mathrm{Hom}_k(M, k)$ as a (kH, kG)-bimodule via $(hfg)(m) := f(gmh)$, for $f \in M^\vee, m \in M, g \in G$, and $h \in H$. On the other hand, if $X \leq G\times H$ and M is a left kX-module, we can also view $M^\vee$ as left kX^o-module via $((h,g)f)(m) := f((g^{-1},h^{-1})m)$ for $f \in M^\vee, m \in M$, and $(g,h) \in X$. Hence, for σ and τ as above, taking the k-dual

$$M^\vee := \mathrm{Hom}_k(M, k)$$

of a p-permutation $(kG\sigma, kH\tau)$-bimodule M results in a p-permutation $(kH\tau, kG\sigma)$-bimodule and induces an isomorphism

$$\mathrm{T}(kG\sigma, kH\tau) \to \mathrm{T}(kH\tau, kG\sigma), \quad \alpha \mapsto \alpha^\vee.$$

We use the same notation for the maps with k replaced by $\mathcal{O}$.

Definition 2.10.23. Let G and H be finite groups, let A be a direct sum of blocks of kG, and let B be a direct sum of blocks of kH. We denote by $T_o^\Delta(A,B)$ the set of all elements γ of $T^\Delta(A,B)$ such that $\gamma \otimes_B \gamma^\vee = [A]$ in $T^\Delta(A,A)$ and $\gamma^\vee \otimes_A \gamma = [B]$ in $T^\Delta(B,B)$. An element of $T_o^\Delta(A,B)$ is called a **p-permutation equivalence**.

Remark 2.10.24. In fact, either one of the two equations in Definition 2.10.23 is equivalent to the other one, see [BP20, 12.3 Theorem].

Remark 2.10.25. Using the notation from above, we mention that p-permutation $(kG\sigma, kH\tau)$-bimodules are projective as one-sided modules, see [Lin18b, Section 9.4].

2.10.4 (Splendid) derived equivalences

We do not give all definitions and results in most generality here, but adapt the results to our setting, i.e. to blocks of finite groups. Let $R \in \{\mathcal{O}, k\}$.

Definition 2.10.26. (a) A **cochain complex** of R-modules is a sequence

$$(C^\bullet, d_C) = \left(\cdots \to C^{n-1} \overset{d_C^{n-1}}{\to} C^n \overset{d_C^n}{\to} C^{n+1} \to \cdots \right),$$

where for each $n \in \mathbb{Z}$, C^n is an R-module and $d_C^n \in \mathrm{Hom}_R(C^n, C^{n+1})$ satisfies $d_C^{n+1} \circ d_C^n = 0$. We often write simply $C^\bullet$ instead of $(C^\bullet, d_C)$.

(b) Let $(C^\bullet, d_C)$ and $(D^\bullet, d_D)$ be two cochain complexes of R-modules. The **tensor product** $C^\bullet \otimes_R D^\bullet$ of the cochain complexes $C^\bullet$ and $D^\bullet$ is the cochain complex $(E^\bullet, d_E)$ defined by

$$E^n := \bigoplus_{i+j=n} C^i \otimes_R D^j$$

where $d_E^n := \bigoplus\limits_{i+j=n} d_C^i \otimes_R 1_{D^j} + (-1)^i \cdot 1_{C^i} \otimes_R d_D^j$.

Definition 2.10.27. Let G, H be two finite groups. Let $B \in \mathrm{Bl}(RG)$, and let $B' \in \mathrm{Bl}(RH)$. Then B and B' are called **derived equivalent** if and only if there exists an equivalence $\mathcal{F} : \mathcal{D}^b(B) \to \mathcal{D}^b(B')$ of triangulated categories.

In the following result from the literature, the left derived tensor product is involved. For a definition see, e.g., [Zim14, Definition 3.7.5].

Theorem 2.10.28 (see [Ric91])**.** *Let G, H be two finite groups. Let $B \in \mathrm{Bl}(RG)$, and let $B' \in \mathrm{Bl}(RH)$. Then B and B' are derived equivalent if and only if there exist two tilting complexes $P^\bullet \in \mathcal{D}^b(B \otimes_R B'^{op})$ and $Q^\bullet \in \mathcal{D}^b(B' \otimes_R B^{op})$ such that $P^\bullet \otimes_{B'}^L Q^\bullet \simeq {}_B B_B$ in $\mathcal{D}^b(B \otimes_R B^{op})$ and $Q^\bullet \otimes_B^L P^\bullet \simeq {}_{B'} B'_{B'}$ in $\mathcal{D}^b(B' \otimes_R B'^{op})$.*

Note that the functor $Q^\bullet \otimes_B^L - : \mathcal{D}^b(B) \to \mathcal{D}^b(B')$ is an equivalence of triangulated categories.

Definition 2.10.29. Let G, H be two finite groups. Let $B \in \mathrm{Bl}(RG)$, and let $B' \in \mathrm{Bl}(RH)$. Suppose that B and B' have isomorphic defect groups D and D', respectively. Then a derived equivalence between B and B' is called a **splendid derived equivalence** if and only if the complex inducing the derived equivalence is splendid, that is, its components are relative $\Delta(D, \varphi, D')$-projective p-permutation $R(G \times H)$-modules, for some group isomorphism $\varphi : D \to D'$.

2.10.5 Implications between equivalences

In this subsection, we collect results from the literature which state the various implications between all the equivalences treated so far. With the aid of the following figure we illustrate which concept implies which. Note however that not all occurring notions are equivalences. A grey arrow from one box to another box means that the former concept implies the latter.

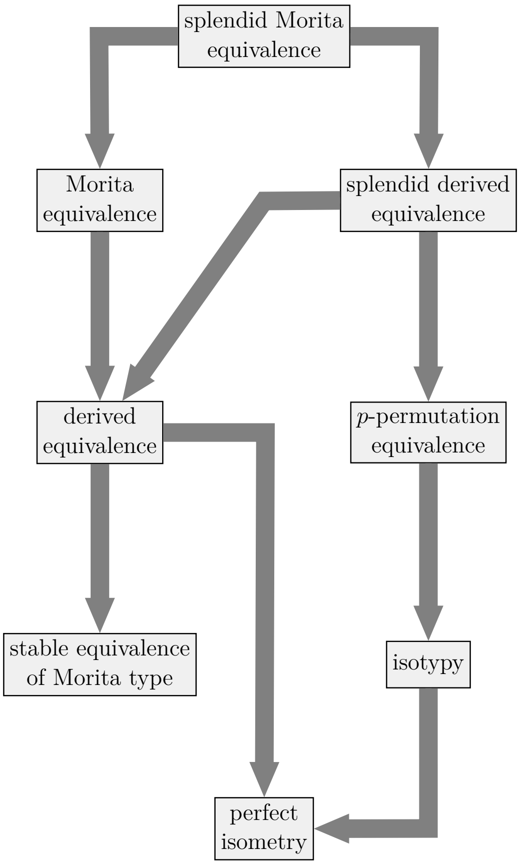


Figure 2.1: implications between categorical equivalences and related concepts

Theorem 2.10.30. *Let G, H be two finite groups, let $B \in \mathrm{Bl}(kG)$, and let $B' \in \mathrm{Bl}(kH)$.*

(a) *Each splendid Morita equivalence between B and B' induces a Morita equivalence between B and B' as well as a splendid derived equivalence between B and B'.*

(b) *Each Morita equivalence between B and B' induces a derived equivalence between B and B'.*

(c) *Each derived equivalence between B and B' induces a stable equivalences of Morita type between B and B'.*

(d) *Each splendid derived equivalence between B and B' induced a derived equivalence between B and B'.*

(e) *Each splendid derived equivalence between B and B' induced a p-permutation equivalence between B and B'.*

(f) *Each p-permutation equivalence between B and B' induces an isotypy between B and B'.*

(g) *Each derived equivalence between B and B' induces a perfect isometry between B and B'.*

(h) *Each isotypy between B and B' induces a perfect isometry between B and B'.*

Proof. The first assertion in part (a) follows from the definition; for the second statement use the chain complex where the only non-trivial module, namely the bimodule inducing the splendid Morita equivalence, is concentrated in degree zero. Part (b) follows analogously. Part (c) is proved in [Ric91]. Part (d) is trivial. The assertion in (e) is proved in [BX08]. Part (f) is proved in [Per14, Theorem 14.5]. Cf. [BP20, 1.6 Theorem (b)]. Part (g) is proved in [Bro90, 3.1 Théorème]. The claim in (h) follows from the definition. □

Remark 2.10.31. As any generalised character μ of $\mathbb{Z}\mathrm{Irr}_K(G \times H)$ induces a group homomorphism $\Phi_\mu : \mathbb{Z}\mathrm{Irr}_K(H) \to \mathbb{Z}\mathrm{Irr}_K(G)$ (see, e.g., [Lin18b, Section 9.2] for more details), we deduce: if we are given a p-permutation equivalence γ between B and B' and if the generalised character $\chi_\gamma \in \mathbb{Z}\mathrm{Irr}_K(G \times H)$ of γ has already been computed, then we can easily calculate the perfect isometry between B and B' induced by γ.

Chapter 3

Trivial source modules and trivial source character tables

In this chapter, we introduce trivial source modules, trivial source character tables, and related concepts. Throughout this chapter, we let $R \in \{\mathcal{O}, k\}$. Cf. Convention 2.4.4.

3.1 Trivial source modules

Definition 3.1.1. Let G be a finite group. An indecomposable RG-module M is called a **trivial source module** if for some vertex Q of M the trivial RQ-module R is a source of M.

Hence, in such a case as in Definition 3.1.1, the kG-module M is isomorphic to a direct summand of the kG-module $k{\uparrow}_Q^G$, a permutation kG-module.

Example 3.1.2. The trivial RG-module R is a particular example of a trivial source module, and it has precisely all of the Sylow p-subgroups of G as vertices. If an indecomposable RG-module M has a trivial source for some vertex, then M has a trivial source for any vertex, because pairs consisting of a vertex and a source are permuted transitively, up to isomorphism, by the conjugation action of G.

Definition 3.1.3. Let M, N be two $\mathcal{O}G$-modules. Then, for any $f \in \mathrm{Hom}_{\mathcal{O}G}(M, N)$, we have $f(J(\mathcal{O})M) \subseteq J(\mathcal{O})N$ wherefore f induces a kG-homomorphism $\overline{f} : \overline{M} \to \overline{N}$ given by $\overline{f}(\overline{m}) = \overline{f(m)}$. Here, $\overline{m}$ is the image of m in $\overline{M} = M/J(\mathcal{O})M$ and $\overline{f(m)}$ is the image of $f(m)$ in $\overline{N} = N/J(\mathcal{O})N$.

Theorem 3.1.4 ([Lin18a, Theorem 5.10.2] and [LP10, Lemma 4.10.2(a)])**.** (i) *An indecomposable RG-module M is a trivial source module if and only if M is a direct summand of a permutation module.*

(ii) *If M, N are direct sums of trivial source $\mathcal{O}G$-modules then the natural map $\mathrm{Hom}_{\mathcal{O}G}(M, N) \to \mathrm{Hom}_{kG}(\overline{M}, \overline{N}), f \mapsto \overline{f}$ is surjective.*

(iii) *If M is a trivial source $\mathcal{O}G$-module with vertex Q then $\overline{M}$ is a trivial source kG-module with vertex Q; in particular, $\overline{M}$ remains indecomposable.*

(iv) *For any trivial source kG-module U there is, up to isomorphism, a unique trivial source $\mathcal{O}G$-module M such that $\overline{M} \cong U$.*

The next result shows that for indecomposable modules the concepts *trivial source module* and *p-permutation module* coincide.

Theorem 3.1.5 ([Lin18a, Theorem 5.11.2 (i)])**.** *An RG-module M is a p-permutation module if and only if M is a direct sum of trivial source RG-modules.*

In general, trivial source kG-modules afford several lifts to $\mathcal{O}G$-modules, but there is a unique one amongst these which is a trivial source $\mathcal{O}G$-module. We denote this trivial source lift by $\widehat{M}$ and we let $\chi_{\widehat{M}}$ be the ordinary character afforded by $K \otimes_{\mathcal{O}} \widehat{M}$. Character values of trivial source modules have the following properties.

Lemma 3.1.6 ([Lan83, Lemma II. 12.6])**.** *Let M be an indecomposable trivial source kG-module and let $x \in G$ be a p-element. Then the following holds:*

(a) *the algebraic integer $\chi_{\widehat{M}}(x)$ is a non-negative integer equal to the multiplicity of the trivial $k\langle x\rangle$-module as a direct summand of $M{\downarrow}^G_{\langle x\rangle}$;*

(b) *$\chi_{\widehat{M}}(x) \neq 0$ if and only if x belongs to a vertex of M.*

Proposition 3.1.7. *Let M, N be trivial source kG-modules. Then*

$$\dim_k(\mathrm{Hom}_{kG}(M,N)) = \langle \chi_{\widehat{M}}, \chi_{\widehat{N}} \rangle,$$

where the usual scalar product of ordinary characters is denoted by $\langle -,- \rangle$.

Proof. Combine [Lan83, Theorem II.12.4 iii)] and [Lan83, Proposition I.14.8]. □

Up to isomorphism, there are only finitely many trivial source kG-modules and we study them vertex by vertex. Thus, we denote by $\mathrm{TS}(G;Q)$ the set of isomorphism classes of indecomposable trivial source kG-modules with vertex Q. Since projective indecomposable kG-modules are trivial source modules with vertex $\langle 1\rangle$, with this notation, $\mathrm{TS}(G;\langle 1\rangle)$ is the set of isomorphism classes of projective indecomposable modules of kG.

In general, if Q is p-subgroup of G, then we set $\overline{N}_G(Q) := N_G(Q)/Q$.
Next, we define the Brauer morphism. We follow [Gil10, 1.1.1.7].

Definition 3.1.8. Let M be a kG-module and let $Q \leq G$ be a non-trivial p-subgroup of G.

(a) For any $H \leq H' \leq G$, let $\mathrm{Tr}_H^{H'} : M^H \to M^{H'}$, $m \mapsto \sum_i mg_i$ be the relative trace map where the $g_i \in H'$ form a transversal for the cosets of H in H'. We define the $\overline{N}_G(Q)$-module

$$M[Q] := M^Q / \sum_{P<Q} \mathrm{Tr}_P^Q(M^P).$$

(b) The natural surjection $\mathrm{Br}_Q^G : M^Q \twoheadrightarrow M[Q]$ is called the **Brauer morphism** and the quotient $M[Q]$ is called the **Brauer construction** with respect to Q.

Note that the construction of $M[Q]$ is only well-defined since both modules in the quotient are $N_G(Q)$-invariant. The Brauer morphism is a homomorphism of $kN_G(Q)$-modules that can be considered for all kG-modules M but is especially interesting for p-permutation modules.

Remark 3.1.9. The Brauer morphism allows us to construct bases for $M[Q]$ where M is a p-permutation kG-module and Q is a p-subgroup of G. If $\mathcal{B}$ is a Q-stable basis of M, then one can see that $\{\mathrm{Br}_Q^G(x) \mid x \in \mathcal{B}^Q\}$ is a k-basis of $M[Q]$. Here, $\mathcal{B}^Q$ consists of the elements of $\mathcal{B}$ that are fixed by all elements of Q. In particular, this implies $\dim_k M[Q] \leq \dim_k M[P]$ if $P \leq Q$.
Assume that M is a permutation kG-module and $\mathcal{B}$ is a G-stable basis for M. Then, one can see that $M^Q = \langle \mathcal{B}^Q \rangle \oplus \sum_{P<Q} \mathrm{Tr}_P^Q(M^P)$ and $M[Q]$ can be identified with $\langle \mathcal{B}^Q \rangle$. For more details we refer to [Bro85, (1.1)(3)].

Definition 3.1.10. Let $H \leq G$. There exists an indecomposable direct summand M of the kG-module $L := k\uparrow_H^G$ such that the trivial kG-module k is a submodule of M. This is due to the fact that $\dim_k(\mathrm{Hom}_{kG}(k, L)) = \dim_k(\mathrm{Hom}_{kH}(k, k)) = 1$ wherefore the trivial kG-module k occurs with multiplicity equal to 1 in $\mathrm{Soc}(L)$. The kG-module M is called the **Scott module** of G associated to H. We denote it by $\mathrm{Sc}(G, H)$.

Proposition 3.1.11. (a) *The p-permutation modules are preserved under direct sums, tensor products, inflation, restriction and induction.*

(b) *If $Q \leq G$ is an p-subgroup of G and M is a p-permutation kG-module, then $M[Q]$ is a p-permutation $k\overline{N}_G(Q)$-module.*

(c) *The vertices of a trivial source kG-module M are the maximal p-subgroups Q of G such that $M[Q] \neq \{0\}$.*

(d) *A trivial source kG-module M has vertex Q if and only if $M[Q]$ is a non-zero projective $k\overline{N}_G(Q)$-module. Moreover, if this is the case, then the $kN_G(Q)$-Green correspondent $f(M)$ of M is $M[Q]$ (viewed as a $kN_G(Q)$-module). Thus, there are bijections:*

$$\begin{array}{ccccc} \mathrm{TS}(G;Q) & \longleftrightarrow & \mathrm{TS}(N_G(Q);Q) & \longleftrightarrow & \mathrm{TS}(\overline{N}_G(Q);\langle 1\rangle) \\ M & \mapsto & f(M) & \mapsto & M[Q]. \end{array}$$

These sets are also in bijection with the set of p'-conjugacy classes of $\overline{N}_G(Q)$.

(e) *Let $H \leq G$. Then the Scott module $\mathrm{Sc}(G, H)$ is a trivial source kG-module lying in $B_0(G)$. If $Q \in Syl_p(H)$, then Q is a vertex of $\mathrm{Sc}(G, H)$ and $\mathrm{Sc}(G, H) \cong \mathrm{Sc}(G, Q)$.*

(f) *The trivial source kG-modules, together with their vertices, are preserved under splendid Morita equivalences.*

(g) *The trivial source kG-modules are preserved under taking duals.*

(h) *Let $H \leq G$. Then a kH-module M is a trivial source module if and only if ${}^g(M)$ is a trivial source $k[{}^gH]$-module.*

Proof. Statements (a) to (e) are proved in [Bro85]. Statement (f) follows from [Lin18b, Theorem 6.4.10 & Theorem 9.7.4]. Part (g) follows from [Alp86, Lemma III.5]. Part (h) follows because of the the following argument. For all $g \in G$ and for all subgroups J of H and for all kJ-modules L we have the $k[{}^gH]$-module isomorphism ${}^g(L\uparrow_J^H) \cong ({}^gL)\uparrow_{{}^gJ}^{{}^gH}$. The claim follows now from spacialising to the case in which L is the trivial kJ-module, as M is a trivial source kH-module and, therefore, a direct summand of $k\uparrow_{\widetilde{J}}^H$ for some subgroup $\widetilde{J}$ of H. □

Lemma 3.1.12. *Let Q be a p-subgroup of G.*

(a) *Let $g \in N_G(Q)$ such that gQ is a p'-element in $N_G(Q)/Q$. Then there exists an element $x \in N_G(Q)_{p'}$ such that $gQ = xQ$.*

(b) *Let $g, h \in N_G(Q)_{p'}$. If gQ is conjugate to hQ in $N_G(Q)/Q$ then g is conjugate to h in $N_G(Q)$.*

Proof. (a) Write $g = g_p \cdot g_{p'}$ where g_p is the p-part of g. Since $Q \trianglelefteq \langle Q, g\rangle$ and $Q \in \mathrm{Syl}_p(\langle Q, g\rangle)$, it follows that Q is the unique Sylow p-subgroup of $\langle Q, g\rangle$, wherefore $g_p \in Q$. Thus, $gQ = g_{p'} \cdot g_pQ = g_{p'}Q$. Define $x := g_{p'}$ and note that $x \in N_G(Q)$ because $g \in N_G(Q)$.

(b) See [Dei97, Lemma 5.3]. □

3.1.1 The trivial source modules of abelian groups and related groups

In this section, which is based on an idea of Robert Boltje, let G be a finite group and let $P \trianglelefteq G$ be a normal Sylow p-subgroup of G such that G/P is abelian. We remark that this setting encompasses all abelian groups.
Consequently, P is the only Sylow p-subgroup of G. Choose $\mathscr{S}_p(G)$. Note that $Q \in \mathscr{S}_p(G)$ implies $Q \leq P$. Set $H := N_G(Q)$. We deduce that $P \cap H = \mathbf{O}_p(H) \in \mathrm{Syl}_p(H)$. By the Schur-Zassenhaus Theorem, we choose a complement C of $P \cap H$ in H. This leads to the following diagram:

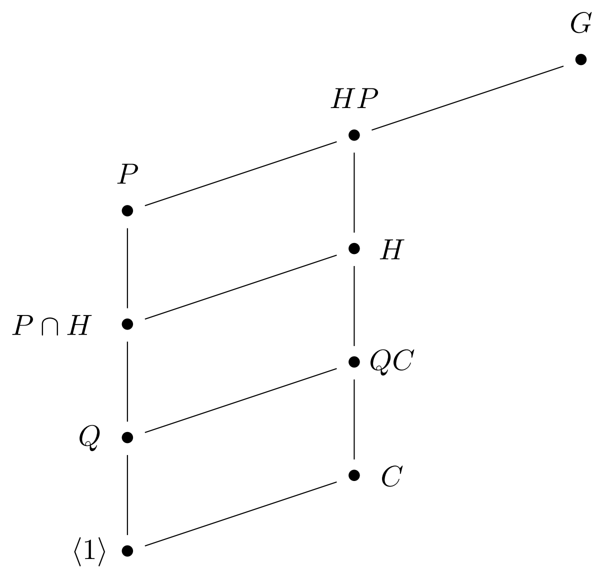


Figure 3.1: parts of the lattice of subgroups in G

Lemma 3.1.13. *For each $Q \in \mathscr{S}_p(G)$, the following sets are in bijection with one another:*

(a) $\mathrm{TS}(G; Q)$*;*

(b) $\mathrm{TS}(H/Q; \langle 1 \rangle)$*;*

(c) $\{$*simple* $k[H/Q]$*-modules*$\}/\cong$*;*

(d) $\{$*simple* kH*-modules annihilated by* $J(kQ) \cdot kH\}/\cong$*;*

(e) $\{$*simple* kH*-modules*$\}/\cong$*;*

(f) $\{$*simple* kH*-modules annihilated by* $J(k[H \cap P]) \cdot kH\}/\cong$*;*

(g) $\{$*simple* kC*-modules*$\}/\cong$*.*

Proof. The bijection between the sets in (a) and (b) follows from the Brauer construction. The map $P \mapsto P/\operatorname{Rad}(P)$ induces a bijection between the sets in (b) and (c). The sets in (c), (d) and (e) are in bijective correspondence with each other, since $k[H/Q] \cong \frac{kH}{kH \cdot J(kQ)}$: as $Q \trianglelefteq H$, we have $kH \cdot J(kQ) \subseteq J(kH)$, and since $J(kH) = \bigcap_{V \text{ simple } kH\text{-module}} \mathrm{ann}_{kH}(V)$, the claim follows. The bijections to the sets in (f) and (g) follow analogously, noting that $C \cong H/(H \cap P)$. □

Remark 3.1.14. Upon identification of the groups C and $H/(H \cap P)$, we choose the bijections in Lemma 3.1.13 as follows: given a simple kH-module S from the set in (e), the underlying vector space of S is mapped to itself in order to obtain a bijection between the sets in (c), (d), (e), (f), and (g). Only the acting group algebras are different, when the respective module structures are considered.

In Proposition 3.1.15, we identify the two isomorphic groups QC/Q and C.

Proposition 3.1.15. *Let S be a simple kC-module, let $L := \mathrm{Ind}_{QC/Q}^{H/Q}(S)$, let $U := \mathrm{Inf}_{H/Q}^{H}(L)$, and let $M := \mathrm{Ind}_H^G(U)$. Then, the following holds.*

(a) *The $k[H/Q]$-module L is the projective indecomposable module corresponding to S via the bijections in Lemma 3.1.13.*

(b) *The kG-module M is an indecomposable p-permutation module with vertex Q.*

Proof. (a) By the Mackey Formula, see Theorem 2.2.6, we have

$$\mathrm{Res}_{(P\cap H)/Q}^{H/Q}(L) = \mathrm{Res}_{(P\cap H)/Q}^{H/Q}\mathrm{Ind}_{QC/Q}^{H/Q}(S) \cong$$

$$\bigoplus_{s\in[(QC/Q)\backslash(H/Q)/((P\cap H)/Q)]} \mathrm{Ind}_{{}^s(QC/Q)\cap((P\cap H)/Q)}^{(P\cap H)/Q}\mathrm{Res}_{{}^s(QC/Q)\cap((P\cap H)/Q)}^{QC/Q}({}^sS).$$

As the group C is a complement of $P \cap H$ in H, this equals

$$\mathrm{Ind}_{Q/Q}^{(P\cap H)/Q}\mathrm{Res}_{Q/Q}^{QC/Q}(S) \cong \mathrm{Ind}_{Q/Q}^{(P\cap H)/Q}(k),$$

since $\dim_k(S) = 1$. Indeed, the group G/P is abelian wherefore the subgroup $HP/P \cong H/(H \cap P) \cong C$ is abelian, as well. Due to the fact that $(P \cap H)/Q$ is a p-group, we obtain that $\mathrm{Res}_{(P\cap H)/Q}^{H/Q}(L) \cong \mathrm{Ind}_{Q/Q}^{(P\cap H)/Q}(k)$ is indecomposable and projective. Hence, also L is indecomposable. Since $(P\cap H)/Q$ is a Sylow p-subgroup of H/Q, it follows from [Lin18a, Corollary 2.6.3] that L is projective. Next, we prove that the $k[H/Q]$-module L corresponds to the simple kC-module S. Let $T := \mathrm{Soc}(L)$. Then,

$$0 < |\mathrm{Hom}_{k[H/Q]}(L, T)| = |\mathrm{Hom}_{k[QC/Q]}(S, \mathrm{Res}_{QC/Q}^{H/Q}(T))|.$$

But this happens if and only if $\mathrm{Res}_{QC/Q}^{H/Q}(T) \cong S$, by Schur's Lemma and Remark 3.1.14.

(b) The kH-module U is indecomposable, as L is indecomposable. Moreover, Q is a vertex of U by Proposition 3.1.11. Next, we prove that $M = \mathrm{Ind}_H^G(U)$ is indecomposable. We have

$$\begin{aligned} M = \mathrm{Ind}_H^G(U) &= \mathrm{Ind}_H^G\mathrm{Inf}_{H/Q}^H(L) \cong \mathrm{Ind}_H^G\mathrm{Inf}_{H/Q}^H\mathrm{Ind}_{QC/Q}^{H/Q}(S) \\ &\cong \mathrm{Ind}_H^G\mathrm{Ind}_{QC}^H\mathrm{Inf}_{QC/Q}^{QC}(S) \cong \mathrm{Ind}_{QC}^G\mathrm{Inf}_{QC/Q}^{QC}(S). \end{aligned}$$

The $k[QC]$-module $S' := \mathrm{Inf}_{QC/Q}^{QC}(S)$ is simple, since Q is normal in QC and S is simple. Moreover, $\dim_k(S') = 1$, as C is abelian. By the Mackey Formula, see Theorem 2.2.6, we have

$$\begin{aligned} \mathrm{Res}_P^G(M) = \mathrm{Res}_P^G\mathrm{Ind}_{QC}^G(S') &\cong \bigoplus_{s\in[P\backslash G/QC]} \mathrm{Ind}_{{}^s(QC)\cap P}^P\mathrm{Res}_{{}^s(QC)\cap P}^{{}^s(QC)}({}^sS') \\ &\cong \bigoplus_{s\in[P\backslash G/QC]} \mathrm{Ind}_{{}^sQ}^P\mathrm{Res}_{{}^sQ}^{{}^s(QC)}({}^sS') \cong \bigoplus_{s\in G/PC} \mathrm{Ind}_{{}^sQ}^P(k), \end{aligned}$$

since $P \trianglelefteq G$ and $\dim_k(S') = 1$. We notice that each vertex of every direct summand of

$$Y := \bigoplus_{s \in G/PC} \mathrm{Ind}_{{}^sQ}^{P}(k)$$

has order $|Q|$. Assume for a contradiction that $\mathrm{Ind}_H^G(U) \cong f(U) \oplus X$ for some non-zero kG-module X. Then, X is a finite direct sum of indecomposable kG-modules $X_1, \ldots, X_n$ for some $n \in \mathbb{Z}_{\geq 1}$. By the Green Correspondence, the vertices of X_i have orders different from $|Q|$ for each $1 \leq i \leq n$. It follows now from [NT89, Chapter 4, Lemma 3.5(ii)] that Y would have to have a direct summand whose vertices have orders which are different from $|Q|$, as every indecomposable direct summand of the kG-module X is P-projective. This is a contradiction. Thus, the kG-module M is indecomposable. By the Green Correspondence, Q is a vertex of M. As U is a trivial source module, M is a trivial source module, as well.

□

Remark 3.1.16. This facilitates the algorithmic computation of the trivial source modules of abelian groups.

3.2 Trivial source character tables

Notation-Definition 3.2.1 (cf. [BT10, §2])**.** 1. Set $\mathcal{P}_{G,p} := \{(Q, E) \mid Q \text{ is a } p\text{-subgroup of } G,\ E \in \mathrm{TS}(\overline{N}_G(Q); 1)\}$. The group G acts by conjugation on $\mathcal{P}_{G,p}$ and we let $[\mathcal{P}_{G,p}]$ denote a set of representatives of the G-orbits on $\mathcal{P}_{G,p}$. Then, given $(Q, E) \in [\mathcal{P}_{G,p}]$, we let $M_{(Q,E)} \in \mathrm{TS}(G; Q)$ denote the unique trivial source kG-module (up to isomorphism) such that $M_{(Q,E)}[Q] \cong E$ given by Proposition 3.1.11(d).

2. The **trivial source ring** $\mathrm{T}(kG)$ of kG is the Grothendieck group of the category of p-permutation kG-modules, with relations corresponding to direct sum decompositions, i.e. $[M] + [N] = [M \oplus N]$, and endowed with the multiplication induced by the tensor product over k. The identity element is the class of the trivial kG-module k. Moreover, $\mathrm{T}(kG)$ is a finitely generated free abelian group with basis $\mathcal{B} := \{[M_{(Q,E)}] \mid (Q, E) \in [\mathcal{P}_{G,p}]\}$.

3. Set $\mathcal{Q}_{G,p} := \{(Q, s) \mid Q \text{ is a } p\text{-subgroup of } G,\ s \in \overline{N}_G(Q)_{p'}\}$. Again, G acts on $\mathcal{Q}_{G,p}$ by conjugation and we let $[\mathcal{Q}_{G,p}]$ be a set of representatives of the G-orbits on $\mathcal{Q}_{G,p}$. We have $|[\mathcal{Q}_{G,p}]| = |[\mathcal{P}_{G,p}]|$.

4. Given $(Q, s) \in \mathcal{Q}_{G,p}$, there is a ring homomorphism

$$\begin{array}{rccc} \tau_{Q,s}^G : & \mathrm{T}(kG) & \longrightarrow & K \\ & [M] & \mapsto & \varphi_{M[Q]}(s) \end{array}$$

mapping the class of a p-permutation kG-module M to the value at s of the Brauer character $\varphi_{M[Q]}$ of the Brauer quotient $M[Q]$. The species $\tau_{Q,s}^G$ only depends on the G-orbit of (Q, s), that is, $\tau_{Q^x,s^x}^G = \tau_{Q,s}^G$ for every $x \in G$, see e.g. [Bol98, 2.3 Proposition]. Moreover, this ring homomorphism extends to a K-algebra homomorphism

$$\hat{\tau}_{Q,s}^G : K \otimes_{\mathbb{Z}} \mathrm{T}(kG) \longrightarrow K\,,$$

and the set $\{\hat{\tau}^G_{Q,s} \mid (Q,s) \in [\mathcal{Q}_{G,p}]\}$ is the set of all distinct species (= K-algebra homomorphisms) from $K \otimes_{\mathbb{Z}} \mathrm{T}(kG)$ to K. These species induce a K-algebra isomorphism

$$\prod_{(Q,s)\in[\mathcal{Q}_{G,p}]} \hat{\tau}^G_{Q,s} : K \otimes_{\mathbb{Z}} \mathrm{T}(kG) \longrightarrow \prod_{(Q,s)\in[\mathcal{Q}_{G,p}]} K\,.$$

See, e.g., [BT10, 2.18. Proposition]. The matrix of this isomorphism with respect to the basis $\mathcal{B}$, denoted by $\mathrm{Triv}_p(G)$, is called the **trivial source character table** (or **species table of the trivial source ring**) of the group G at the prime p and is a square matrix of size $|[\mathcal{Q}_{G,p}]| \times |[\mathcal{Q}_{G,p}]|$.

Convention 3.2.2. For the purpose of computations we see the trivial source character table as follows, following the approach of [LP10, Section 4.10]. First, we fix a set of representatives $Q_1, \dots, Q_r$ for the conjugacy classes of p-subgroups of G where $Q_1 := \langle 1 \rangle$ and $Q_r \in \mathrm{Syl}_p(G)$, and for each $1 \le v \le r$ we set $N_v := N_G(Q_v)$, $\overline{N}_v := N_G(Q_v)/Q_v$. Then, for each $1 \le i, v \le r$, we define a matrix

$$T_{i,v} := \left[\tau^G_{Q_v,s}([M])\right]_{M\in\mathrm{TS}(G;Q_i),s\in[\overline{N}_v]_{p'}}\,.$$

With this notation, the trivial source character table of G at p is the block matrix

$$\mathrm{Triv}_p(G) := [T_{i,v}]_{1\le i,v\le r}\,.$$

Moreover, following [LP10] we label the rows of $\mathrm{Triv}_p(G)$ with the ordinary characters $\chi_{\widehat{M}}$ instead of the isomorphism classes of trivial source modules M themselves.We label the block columns of the the trivial source character table by the groups Q_v and by the groups N_v, $1 \le v \le r$. Furthermore, we label the columns by p'-elements of the groups N_v, $1 \le v \le r$. This is possible due to Lemma 3.1.12. Whenever we give a concrete example of a trivial source character table with entries in $\mathbb{C}$, we follow [LP10] with the choice of the p-modular system, see Section 5.2.2.

Remark 3.2.3. Two non-isomorphic trivial source modules M and N with vertex Q_i may afford the same ordinary character. Therefore two rows of $\mathrm{Triv}_p(G)$ may be labelled with the same ordinary character. However, this labelling brings additional information about the trivial source modules and they are distinguished by the values in $T_{i,v}$ for some $v > 1$, since every trivial source character table is invertible by Remark 3.2.6(b).

Example 3.2.4. We consider the symmetric group $G := \mathfrak{S}_3$ acting on 3 letters. The ordinary character table $\mathrm{X}(\mathfrak{S}_3)$ of $\mathfrak{S}_3$ is as given in Table 3.1.

	1	$(1,2)$	$(1,2,3)$
χ_1	1	1	1
χ_2	1	-1	1
χ_3	2	0	-1

Table 3.1: **ordinary character table of $\mathfrak{S}_3$**

In order to compute $\mathrm{Triv}_2(\mathfrak{S}_3)$, we set

$$Q_1 := \langle 1 \rangle \text{ and } Q_2 := \langle (1,2) \rangle \in \mathrm{Syl}_2(\mathfrak{S}_3).$$

Furthermore, we choose $\mathscr{S}_2(\mathfrak{S}_3) = \{Q_1, Q_2\}$. Then, the lattice of subgroups in $\mathscr{S}_2(\mathfrak{S}_3)$ is given as follows:

Figure 3.2: the lattice of subgroups in $\mathscr{S}_2(\mathfrak{S}_3)$

Moreover,

$$N_G(Q_1) = G \text{ and } \overline{N}_G(Q_1) \cong G;$$
$$N_G(Q_2) = Q_2 \text{ and } \overline{N}_G(Q_2) \cong \langle 1 \rangle.$$

Therefore, $[\mathcal{Q}_{\mathfrak{S}_3,2}] = \{(Q_1, 1), (Q_1, (1,2,3)), (Q_2, 1)\}$. Hence, $\mathrm{Triv}_2(\mathfrak{S}_3)$ is a 3×3-matrix. By Theorem 3.1.4, the trivial $k\mathfrak{S}_3$-module k is an indecomposable p-permutation $k\mathfrak{S}_3$-module. Since it has the Sylow 2-subgroups of $\mathfrak{S}_3$ as vertices, we deduce that $k[Q_2] \cong k$ as kN_2-modules. Next, by Theorem 3.1.4, the projective indecomposable $k\mathfrak{S}_3$-modules are p-permutation $k\mathfrak{S}_3$-modules. As there are exactly two $2'$-conjugacy classes in $\mathfrak{S}_3$, there are exactly two projective indecomposable $k\mathfrak{S}_3$-modules, and we denote them by P_1 and P_2, respectively. They are direct summands of $k\!\uparrow_{\langle 1 \rangle}^{\mathfrak{S}_3}$. As $1_{\langle 1 \rangle}\!\uparrow_{\langle 1 \rangle}^{\mathfrak{S}_3} = \chi_1 + \chi_2 + 2 \cdot \chi_3$, using Lemma 2.2.1, we see that their ordinary characters are given by $\chi_1 + \chi_2$ and χ_3, respectively, since one of these projective modules occurs twice (up to isomorphism) as a summand in $k\!\uparrow_{\langle 1 \rangle}^{\mathfrak{S}_3}$. Moreover, $P_1[Q_2] = 0 = P_2[Q_2]$, as projective modules have only trivial vertices. Hence, the trivial source character table $\mathrm{Triv}_2(\mathfrak{S}_3)$ of $\mathfrak{S}_3$ in characteristic 2 is as given in Table 3.2.

p-subgroups Q_v of G up to conjugacy in G	$Q_1 := \langle 1 \rangle$		$Q_2 := \langle (1,2) \rangle$
Normalisers N_v	$N_1 := N_{\mathfrak{S}_3}(Q_1) = \mathfrak{S}_3$		$N_2 := N_{\mathfrak{S}_3}(Q_2) = Q_2$
Representatives $n_j \in N_v$	1	$(1,2,3)$	1
$\chi_1 + \chi_2$	2	2	0
χ_3	2	-1	0
χ_1	1	1	1

Table 3.2: **trivial source character table of** $\mathfrak{S}_3$ at $p = 2$

Notation-Definition 3.2.5. A subgroup $H \leq G$ of G is called ***p*-hypo-elementary**, if $H/O_p(H)$ is a cyclic p'-group. We denote the set of p-hypo-elementary subgroups of G by $\mathcal{C}_p(G)$.

Remark 3.2.6. (a) It follows from Lemma 3.2.9 that our definition of a trivial source character table is equivalent to the definition given in [LP10]. See also [BT10, 2.16. Remarks]. Moreover, in the context of [LP10], the following is worth mentioning: the set of coprimordial subgroups for the trivial source ring $a(\mathcal{O}G, \mathrm{Triv})$ coincides by [Bol95, III.4.15 Proposition & the paragraph after III.4.1 Proposition] with the set of p-hypo-elementary subgroups of G. (See [Bol95] for the definition of a coprimordial subgroup.) This also explains the choice of our notation in Notation-Definition 3.2.5.

(b) The block $T_{i,i}$ consists of the values of projective indecomposable characters of $\overline{N}_i$ at the p'-conjugacy classes of $\overline{N}_i$. In particular, $T_{1,1}$ consists of the values of projective indecomposable characters of G at the p'-conjugacy classes of G.

(c) The trivial kG-module k is a trivial source module with vertex Q_r and $k = M_{(Q_r,k)}$. For every $1 \leq v \leq r$ and every $s \in [\overline{N}_v]_{p'}$ we have $\tau^G_{Q_v,s}(k) = 1$ since $k[Q_v] = k$.

(d) For p-subgroups Q_i, Q_v of G, it follows immediately from the definition and Proposition 3.1.11(c) that $\tau^G_{Q_v,s}([M_{(Q_i,E)}]) = 0$ unless $Q_v \leq_G Q_i$. In other words $T_{i,v} = \mathbf{0}$ if $Q_v \not\leq_G Q_i$.

(e) If $v = 1$ and $1 \leq i \leq r$, then $M[\langle 1 \rangle] = M$ and so

$$\tau^G_{\langle 1 \rangle,s}([M]) = \chi_{\widehat{M}}(s)$$

for every $M \in \mathrm{TS}(G; Q_i)$ and every $s \in [G]_{p'}$. In particular, for every $M \in \mathrm{TS}(G; Q_i)$ we have

$$\tau^G_{\langle 1 \rangle,1}([M]) = \dim_k M\,.$$

This explains the terminology *trivial source character table* and our labelling of the rows by the characters.

(f) More generally, if $(Q, s) \in \mathcal{Q}_{G,p}$ with $s = 1$ and M is any p-permutation kG-module, then $\tau^G_{Q,1}([M]) = \dim_k M[Q]$.

(g) Species values of G may be computed using species values of smaller groups through the following formula (see [BT10, 2.16. Remarks])

$$\tau^G_{Q,s} = \tau^{\langle Qs \rangle/Q}_{\langle 1 \rangle,s} \circ \mathrm{Br}^{\langle Qs \rangle}_Q \circ \mathrm{Res}^G_{\langle Qs \rangle},$$

where $\langle Qs \rangle$ denotes the inverse image in $N_G(Q)$ of the cyclic group $\langle s \rangle$ of $\overline{N}_G(Q)$ and $\mathrm{Br}^{\langle Qs \rangle}_Q : A(k\langle Qs \rangle, \mathrm{Triv}) \longrightarrow A(kN_{\langle Qs \rangle}(Q)/Q, \mathrm{Triv})$ denotes the ring homomorphism induced by the correspondence $M \mapsto M[Q]$ for trivial source $k\langle Qs \rangle$-modules M. See [BT10, 2.11. Proposition].

The following generalises the formula in Remark 3.2.6(e) and is often helpful for computer calculations.

Lemma 3.2.7 ([Ric96, Lemma 6.2])**.** *Let G be a finite group, let g_p and $g_{p'}$ be commuting elements of G with g_p a p-element and $g_{p'}$ a p'-element, and let M be a p-permutation $\mathcal{O}G$-module. Let M_{g_p} be the p-permutation $\mathcal{O}C_G(g_p)$-module such that $\overline{M_{g_p}}$ is isomorphic to* $\mathrm{Res}^{N_G(\langle g_p \rangle)}_{C_G(g_p)}(\overline{M}[\langle g_p \rangle])$. *Moreover, let $\chi_{\widehat{\overline{M}}}$ and $\chi_{\widehat{\overline{M_{g_p}}}}$ be the K-characters of $\overline{M}$ and $\overline{M_{g_p}}$ respectively. Then*

$$\chi_{\widehat{\overline{M}}}(g_p g_{p'}) = \chi_{\widehat{\overline{M_{g_p}}}}(g_{p'}).$$

Corollary 3.2.8. Let G be a finite group, let g_p and $g_{p'}$ be commuting elements of G with g_p a p-element and $g_{p'}$ a p'-element, and let M be a trivial source $\mathcal{O}G$-module. Then:

$$\chi_{\widehat{\overline{M}}}(g_p g_{p'}) = \tau^G_{\langle g_p \rangle, g_{p'}}([\overline{M}]).$$

Proof. Set $L := \overline{M}[\langle g_p \rangle]$. Since $g_{p'} \in C_G(g_p)$, we have by Lemma 3.2.7:

$$\tau^G_{\langle g_p \rangle, g_{p'}}([\overline{M}]) = \chi_{\widehat{L}}(g_{p'}) = \chi_{\widehat{L}\downarrow^{N_G(\langle g_p \rangle)}_{C_G(g_p)}}(g_{p'}) = \chi_{\widehat{\overline{M_{g_p}}}}(g_{p'}) = \chi_{\widehat{\overline{M}}}(g_p g_{p'}).$$

□

Lemma 3.2.9. (a) *Every p-hypo-elementary subgroup H of G is contained in the normaliser of a p-subgroup of G.*

(b) *Given a p-subgroup P of G, there exists a p-hypo-elementary subgroup H of G such that $P \leq H \leq N_G(P)$.*

(c) *Let* $Q \in \mathscr{S}_p(G)$, *let* $\widetilde{c} = cQ \in \overline{N}_G(Q)_{p'}$ *for some* $c \in N_{p'}$, *and let* M *be an indecomposable trivial source* kG*-module. Then* $\varphi_{M[Q]}(c) = \varphi_{(M\downarrow^G_{N_G(Q)})^{\supseteq Q}}(c)$.

(d) *The definition of species for the trivial source ring* $\mathrm{T}(kG)$ *given in [LP10, Definition 4.10.4] is equivalent to the definition given in Notation-Definition 3.2.1.*

Proof. Fix $H \in \mathcal{C}_p(G)$. We have $\mathbf{O}_p(H) \trianglelefteq H$ and, therefore, $H \leq N_G(O_p(H))$. This proves (a). Now, assume given a p-subgroup P of G and set $N := N_G(P)$. Let $g \in N$ have the property that gP is a p'-element of N/P. By Lemma 3.1.12, there is an element $x \in N_{p'}$ such that $gP = xP$. Define $H := \langle x, P\rangle \leq N$. Then $P \trianglelefteq H$ and $H/P \cong \langle xP\rangle = \langle gP\rangle$, and hence $H \in \mathcal{C}_p(G)$. This proves (b). Part (c) follows from [Lin18a, Proposition 5.8.7]. Part (d) follows from part (a), part (b), part (c), and [LP10, Lemma 4.10.11]. □

3.2.1 The trivial source character tables of p-groups and p'-groups

Lemma 3.2.10. *Let* G *be a* p*-group. Choose* $\mathscr{S}_p(G)$. *Then, for each* $Q \in \mathscr{S}_p(G)$, *we have*

$$\mathrm{TS}(G;Q) = \{k\uparrow_Q^G\}.$$

Proof. If $M \in \mathrm{TS}(G;Q)$ for some $Q \in \mathscr{S}_p(G)$, then $M \,|\, k\uparrow_Q^G$ by the definition of vertices and sources. We know from the Green indecomposability theorem that $k\uparrow_Q^G$ is indecomposable, since G is a p-group and the trivial kQ-module k is indecomposable. Hence, $[M] = [k\uparrow_Q^G]$. On the other hand, $|\mathrm{TS}(G;Q)|$ is equal to the number of p-regular conjugacy classes of $\overline{N}_G(Q)$ which is equal to 1, as G is a p-group. □

Theorem 3.2.11 (cf. [Büy13, Proposition 2.3.16]). (a) *If* G *is a* p*-group, then the trivial source ring* $\mathrm{T}(kG)$ *and the Burnside ring* $\mathcal{B}(G)$ *are isomorphic as rings.*

(b) *If* G *is a* p'*-group, then the trivial source ring* $\mathrm{T}(kG)$ *and the Brauer character ring* $\mathscr{R}(kG)$ *are isomorphic as rings.*

Proof. (a) Let G be a p-group. By Lemma 3.2.10, $\mathscr{B} := \{[k\uparrow_Q^G] \mid Q \leq G\}$ is a $\mathbb{Z}$-basis of $\mathrm{T}(kG)$. Now, let $P, Q \in \mathscr{S}_p(G)$ with $[k\uparrow_P^G] = [k\uparrow_Q^G]$. Then,

$$P \in \mathrm{vtx}(k\uparrow_P^G) = \mathrm{vtx}(k\uparrow_Q^G) \ni Q.$$

Consequently, $P \sim Q$ in G.
If $Q = {}^gP$ for some $g \in G$, then $[k\uparrow_P^G] = [({}^gk)\uparrow_{{}^gP}^G] = [k\uparrow_Q^G]$. Therefore, $[k\uparrow_P^G] = [k\uparrow_Q^G]$ if and only if $Q = {}^gP$. Thus, we have a well-defined bijection

$$\alpha : [k\uparrow_P^G] \mapsto [G/P],$$

which we can extend $\mathbb{Z}$-linearly to a $\mathbb{Z}$-module homomorphism $\mathrm{T}(kG) \to \mathcal{B}(G)$, which we denote by α as well. Let P, Q be subgroups of G.
It follows from the definition of the multiplication in $\mathrm{T}(kG)$ that

$$\alpha([k\uparrow_P^G] \cdot [k\uparrow_Q^G]) = \alpha([k\uparrow_P^G \otimes_k k\uparrow_Q^G]).$$

By Theorem 2.2.7, this is equal to

$$\alpha(\bigoplus_{PgQ \in P\backslash G/Q} [k\uparrow_{{}^gP\cap Q}^G]) = \sum_{PgQ \in P\backslash G/Q} [G/{}^gP \cap Q].$$

Using Lemma 2.9.9 we deduce that this equals $[G/P][G/Q] = \alpha([k\uparrow_P^G]) \cdot \alpha([k\uparrow_Q^G])$.

(b) Since every trivial source module defines a Brauer character, we obtain a ring homomorphism $\mathrm{T}(kG) \to \mathscr{R}(kG)$. Since G is a p'-group, every kG-module is isomorphic to a direct sum of simple kG-modules. Moreover, projective indecomposable kG-modules and simple kG-modules coincide in that case. On the other hand, each projective indecomposable kG-module is isomorphic to a direct summand of the regular kG-module, which in turn is isomorphic to $k{\uparrow}_{\langle 1\rangle}^{G}$. Thus, the trivial source kG-modules are precisely the simple kG-modules, and the claim follows.

□

Remark 3.2.12. We mention here that Kimmerle and Roggenkamp have constructed examples of non-isomorphic groups G and H which do not only have isomorphic Burnside rings, but also isomorphic complex character tables. That is, their character tables are equal, up to permutations of rows and columns. See [KR94].

As a consequence, it is natural to ask the following question: given a finite group G, is G uniquely determined up to isomorphism by the trivial source character tables $\mathrm{Triv}_p(G)$ where p runs through all prime divisors of $|G|$ and the ordinary character table of G (up to table isomorphisms). The answer is no, as is shown by Example 3.2.13. This example is computer calculated.

Example 3.2.13. Let $G :=$ SmallGroup$(1458, 44)$ and let $H :=$ SmallGroup$(1458, 45)$. Then $\mathrm{Triv}_2(G) \cong \mathrm{Triv}_2(H)$, $\mathrm{Triv}_3(G) \cong \mathrm{Triv}_3(H)$, and $\mathrm{X}(G) \cong \mathrm{X}(H)$, where all isomorphisms are table isomorphisms. The structure description of both the group G and the group H is given by $C_2 \times (((C_9 \rtimes C_9) \rtimes C_3) \rtimes C_3)$. These groups only differ in the action of the outermost cyclic group C_3 on the group $(C_9 \rtimes C_9) \rtimes C_3$. We have $G \cong C_2 \times \widetilde{G}$ for $\widetilde{G} \cong$ SmallGroup$(729, 44)$ and $H \cong C_2 \times \widetilde{H}$ for $\widetilde{H} \cong$ SmallGroup$(729, 45)$. Let $\widetilde{G} = \langle [a, b, c, d] \rangle$ such that these generators correspond to the structure description $((C_9 \rtimes C_9) \rtimes C_3) \rtimes C_3$ of $\widetilde{G}$ in the correct order. Let $\widetilde{H} = \langle [a, b, c, \widetilde{d}] \rangle$ such that these generators correspond to the structure description $((C_9 \rtimes C_9) \rtimes C_3) \rtimes C_3$ of $\widetilde{H}$ in the correct order. Then, d acts on $[a, b, c]$ by $[ab^{-1}, b^{-2}c, d^{-1}cd]$ and $\widetilde{d}$ acts on $[a, b, c]$ by $[ab^{-1}c, \widetilde{d}^{-1}b\widetilde{d}, a^{-1}ca]$.

Definition 3.2.14. Let $H \leq G$. We define

$$\begin{array}{rccll} \tau_H^{\mathcal{B}(G)}: & \mathcal{B}(G) & \longrightarrow & \mathbb{Z} \subseteq K & \\ & [S] & \mapsto & |\mathrm{Fix}_S(H)| & \quad (S \text{ a } G\text{-set}) \end{array}$$

and obtain in this way for each subgroup $H \leq G$ a ring homomorphism, the so-called **Burnside homomorphism**. Moreover, we set $\widetilde{\mathcal{B}}(G) := K \otimes_{\mathbb{Z}} \mathcal{B}(G)$. Then $\tau_H^{\mathcal{B}(G)}$ extends to a K-algebra homomorphism $\hat{\tau}_H^{\mathcal{B}(G)} : \widetilde{\mathcal{B}}(G) \to K$.

Theorem 3.2.15 ([Ben98, Theorem 5.4.2])**.** *The set $\{\hat{\tau}_H^{\mathcal{B}(G)} \mid H \in \mathscr{S}(G)\}$ is the set of all distinct species from $\widetilde{\mathcal{B}}(G)$ to K. These species induce a K-algebra isomorphism*

$$\prod_{H \in \mathscr{S}(G)} \hat{\tau}_H^{\mathcal{B}(G)} : \widetilde{\mathcal{B}}(G) \longrightarrow \prod_{H \in \mathscr{S}(G)} K.$$

Lemma 3.2.16. *Let G be a p-group, let $H \leq G$, and let $V := k{\uparrow}_U^G$, the permutation kG-module afforded by the G-set $\Omega := G/U$ for some $U \leq G$. Then $\tau_{H,1}^{G}([V]) = m_\Omega(H)$.*

Proof. This follows from [LP10, Remark 4.10.5]. □

Proposition 3.2.17. (a) *If G is a p-group, then* $\mathrm{Triv}_p(G) = \mathcal{M}(G)$.

(b) *If G is a p'-group, then* $\mathrm{Triv}_p(G) = \mathrm{BR}_p(G)$.

In both assertions, the equality sign means that we have equality up to permutations of rows and columns.

Proof. (a) This follows from Theorem 3.2.11, Theorem 3.2.15, and Lemma 3.2.16.

(b) This follows from Theorem 3.2.11 and Example 2.7.4.

□

Chapter 4

Computations of some trivial source character tables in characteristic 2

Throughout this chapter, if not stated otherwise, let k denote an algebraically closed field of characteristic $p = 2$.

4.1 Concrete examples

4.1.1 The Klein four-group V_4

Let $G := V_4 := \langle a \rangle \times \langle b \rangle \leq \mathfrak{S}_4$, where $a := (1,2)(3,4)$ and $b := (1,3)(2,4)$. The ordinary character table of G is given as follows:

	1	a	b	ab
χ_1	1	1	1	1
χ_2	1	1	-1	-1
χ_3	1	-1	1	-1
χ_4	1	-1	-1	1

Table 4.1: **ordinary character table of** V_4

We set

$$Q_1 := \langle 1 \rangle, Q_2 := \langle a \rangle, Q_3 := \langle b \rangle, Q_4 := \langle ab \rangle, \text{ and } Q_5 := G.$$

Furthermore, $\mathscr{S}_2(V_4) = \{Q_1, Q_2, Q_3, Q_4, Q_5\}$. The lattice of subgroups in $\mathscr{S}_2(V_4)$ is given as follows:

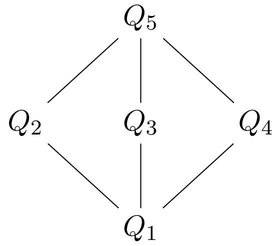


Figure 4.1: lattice of subgroups in $\mathscr{S}_2(V_4)$

Moreover,

$$\begin{aligned} N_G(Q_1) &= G \text{ and } \overline{N}_G(Q_1) \cong G; \\ N_G(Q_2) &= G \text{ and } \overline{N}_G(Q_2) \cong Q_3 \cong C_2; \\ N_G(Q_3) &= G \text{ and } \overline{N}_G(Q_3) \cong Q_2 \cong C_2; \\ N_G(Q_4) &= G \text{ and } \overline{N}_G(Q_4) \cong C_2; \\ N_G(Q_5) &= G \text{ and } \overline{N}_G(Q_5) \cong Q_1. \end{aligned}$$

Remark 4.1.1. We have already seen in Proposition 3.2.17 that $\mathrm{Triv}_2(V_4) = \mathcal{M}(V_4)$. We compute $\mathrm{Triv}_2(V_4)$ here nevertheless, since we also determine the ordinary characters, and this is relevant for other calculations in the sequel.

Proposition 4.1.2. *Labelling the ordinary characters as in Table 4.1, the trivial source character table* $\mathrm{Triv}_2(V_4)$ *is given as follows:*

Q_v $(1 \leq v \leq 5)$	$Q_1 \cong C_1$	$Q_2 \cong C_2$	$Q_3 \cong C_2$	$Q_4 \cong C_2$	$Q_5 \cong V_4$
N_v $(1 \leq v \leq 5)$	$N_1 \cong V_4$	$N_2 \cong V_4$	$N_3 \cong V_4$	$N_4 \cong V_4$	$N_5 \cong V_4$
$n_j \in N_v$	1	1	1	1	1
$\chi_1 + \chi_2 + \chi_3 + \chi_4$	4	0	0	0	0
$\chi_1 + \chi_2$	2	2	0	0	0
$\chi_1 + \chi_3$	2	0	2	0	0
$\chi_1 + \chi_4$	2	0	0	2	0
χ_1	1	1	1	1	1

Table 4.2: **trivial source character table of** V_4 at $p = 2$

Proof. Since V_4 is a 2-group, G has exactly one 2-block. Moreover, the trivial kV_4-module is the unique simple kV_4-module. Labelling the ordinary characters as in Table 4.1, the 2-decomposition matrix $\mathfrak{D}(kV_4)$ is equal to

	$1_{G_{p'}}$
χ_1	1
χ_2	1
χ_3	1
χ_4	1

and, since $a \not\sim b \not\sim ab \not\sim a$ in G, the claim follows from Remark 3.2.6. The K-characters of the trivial source modules follow from *Corollary* 3.2.8. □

4.1.2 The alternating group $\mathfrak{A}_4$

Let $G := \mathfrak{A}_4 := \langle a, c \rangle \leq \mathfrak{S}_4$, where $a := (1,2)(3,4)$ and $c := (1,2,3)$. Moreover, we set $b := (1,3)(2,4) \in G$. Defining $\omega := \exp(\frac{2\pi i}{3}) = \frac{-1+i\sqrt{3}}{2}$, the ordinary character table of $\mathfrak{A}_4$ is given as follows:

	1	a	c	bc^2
χ_1	1	1	1	1
χ_2	1	1	ω	ω^2
χ_3	1	1	ω^2	ω
χ_4	3	-1	0	0

Table 4.3: **ordinary character table of** $\mathfrak{A}_4$

We set

$$Q_1 := \langle 1 \rangle, Q_2 := \langle a \rangle, \text{ and } Q_3 := \langle a, b \rangle \in \mathrm{Syl}_2(G).$$

Furthermore, we fix $\mathscr{S}_2(\mathfrak{A}_4) = \{Q_1, Q_2, Q_3\}$. Then, the lattice of subgroups in $\mathscr{S}_2(\mathfrak{A}_4)$ is given as follows:

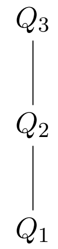


Figure 4.2: lattice of subgroups in $\mathscr{S}_2(\mathfrak{A}_4)$

Moreover,

$$\begin{aligned}
N_G(Q_1) &= G \quad \text{and } \overline{N}_G(Q_1) \cong G;\\
N_G(Q_2) &= Q_3 \text{ and } \overline{N}_G(Q_2) \cong \langle b \rangle \cong C_2;\\
N_G(Q_3) &= G \quad \text{and } \overline{N}_G(Q_3) \cong \langle c \rangle \cong C_3.
\end{aligned}$$

Remark 4.1.3 ([Ben98, Remark after Theorem 4.3.3]). The classification of the indecomposable modules for the alternating group $\mathfrak{A}_4$ in characteristic 2 is well-known. By Lemma 2.3.14, every such module is a summand of a module induced from V_4. In particular, the $k\mathfrak{A}_4$-module $M := k{\uparrow}_{Q_2}^{\mathfrak{A}_4}$ is indecomposable.

Let $\{1, \omega, \overline{\omega}\}$ be the cube roots of unity in k. Denote the three one-dimensional simple $k\mathfrak{A}_4$-modules by k, S_ω, and $S_{\overline{\omega}}$.

Proposition 4.1.4. *The following assertions about the trivial source $k\mathfrak{A}_4$-modules hold.*

(a) *We have* $\mathrm{TS}(\mathfrak{A}_4; Q_1) = \{P(k), P(S_\omega), P(S_{\overline{\omega}})\}$.

(b) *We have* $\mathrm{TS}(\mathfrak{A}_4; Q_2) = \{M\}$.

(c) *We have* $\mathrm{TS}(\mathfrak{A}_4; Q_3) = \{k, S_\omega, S_{\overline{\omega}}\}$.

Proof. By counting the p'-conjugacy classes of $\overline{N}_{\mathfrak{A}_4}(Q_v)$, for $1 \leq v \leq 3$, we deduce that

$$|\mathrm{TS}(\mathfrak{A}_4; Q_1)| = 3 = |\mathrm{TS}(\mathfrak{A}_4; Q_3)| \text{ and } |\mathrm{TS}(\mathfrak{A}_4; Q_2)| = 1.$$

The $k\mathfrak{A}_4$-modules $P(k), P(S_\omega)$, and $P(S_{\overline{\omega}})$ are trivial source modules, since they are projective. This proves (a). Every trivial source $k\mathfrak{A}_4$-module which has Q_3 as a vertex is isomorphic to a direct summand of $k{\uparrow}_{Q_3}^{\mathfrak{A}_4}$. As $|\mathrm{TS}(\mathfrak{A}_4; Q_3)| = 3$ and $[\mathfrak{A}_4 : Q_3] = 3$, it follows that $k{\uparrow}_{Q_3}^{\mathfrak{A}_4}$ decomposes into a direct sum of three non-isomorphic 1-dimensional $k\mathfrak{A}_4$-modules. Therefore, $k{\uparrow}_{Q_3}^{\mathfrak{A}_4} \cong k \oplus S_\omega \oplus S_{\overline{\omega}}$ and all simple $k\mathfrak{A}_4$-modules are trivial source modules with Q_3 as a vertex. This proves (c). By Remark 4.1.3, the $k\mathfrak{A}_4$-module M is indecomposable. By Proposition 3.1.11, it is a trivial source module. The claim in (b) follows, as $|\mathrm{TS}(\mathfrak{A}_4; Q_2)| = 1$. □

Proposition 4.1.5. *Labelling the ordinary characters as in Table 4.3, the trivial source character table* $\mathrm{Triv}_2(\mathfrak{A}_4)$ *is given as follows:*

Q_v	$Q_1 \cong C_1$			$Q_2 \cong C_2$	$Q_3 \cong V_4$		
N_v	$N_1 \cong \mathfrak{A}_4$			$N_2 \cong V_4$	$N_3 \cong \mathfrak{A}_4$		
$n_j \in N_v$	1	c	bc^2	1	1	c	bc^2
$\chi_1 + \chi_4$	4	1	1	0	0	0	0
$\chi_2 + \chi_4$	4	ω	ω^2	0	0	0	0
$\chi_3 + \chi_4$	4	ω^2	ω	0	0	0	0
$\chi_1 + \chi_2 + \chi_3 + \chi_4$	6	0	0	2	0	0	0
χ_1	1	1	1	1	1	1	1
χ_2	1	ω	ω^2	1	1	ω	ω^2
χ_3	1	ω^2	ω	1	1	ω^2	ω

Table 4.4: **trivial source character table of** $\mathfrak{A}_4$ at $p = 2$

Proof. The $\mathfrak{p}$-modular reductions of the ordinary irreducible characters χ_1, χ_2, and χ_3 are one-dimensional. They yield all the irreducible Brauer characters $\varphi_k = 1_{G_{p'}}, \varphi_{S_\omega}$ and $\varphi_{S_{\overline{\omega}}}$, respectively. Moreover, they are obviously linearly independent. As $\chi_4^\circ = \chi_1^\circ + \chi_2^\circ + \chi_3^\circ$, the decomposition matrix $\mathfrak{D}(k\mathfrak{A}_4)$ is given as follows:

	φ_k	φ_{S_ω}	$\varphi_{S_{\overline{\omega}}}$
χ_1	1	0	0
χ_2	0	1	0
χ_3	0	0	1
χ_4	1	1	1

Therefore, the ordinary characters of the projective indecomposable $k\mathfrak{A}_4$-modules are given by

$$\chi_1 + \chi_4, \chi_2 + \chi_4, \text{ and } \chi_3 + \chi_4.$$

It follows from Proposition 4.1.4 that $\chi_{\widehat{M}} = \chi_{\widehat{k\uparrow_{Q_2}^{\mathfrak{A}_4}}}$. Since induction commutes with restriction modulo $\mathfrak{p}$, we deduce:

$$\chi_{\widehat{M}} = 1\uparrow_{Q_2}^{\mathfrak{A}_4} = \chi_1 + \chi_2 + \chi_3 + \chi_4.$$

It follows from Proposition 4.1.4 and the decomposition matrix $\mathfrak{D}(k\mathfrak{A}_4)$ that the ordinary characters of k, S_ω, and $S_{\overline{\omega}}$ are as given in Table 4.4. We have

$$\dim_k(k\downarrow^{\mathfrak{A}_4}_{N_{\mathfrak{A}_4}(Q_2)}) = \dim_k(S_\omega\downarrow^{\mathfrak{A}_4}_{N_{\mathfrak{A}_4}(Q_2)}) = \dim_k(S_{\overline{\omega}}\downarrow^{G}_{N_G(Q_2)}) = 1.$$

Because $N_{\mathfrak{A}_4}(Q_2) \cong V_4$, the only 1-dimensional trivial source $kN_{\mathfrak{A}_4}(Q_2)$-module is the trivial $kN_{\mathfrak{A}_4}(Q_2)$-module $k \in \mathrm{TS}(N_{\mathfrak{A}_4}(Q_2); Q_3)$. As $Q_2 \leq Q_3$, we obtain from Remark 3.2.6(f) that

$$T_{3,2} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

Since $N_{\mathfrak{A}_4}(Q_3) = \mathfrak{A}_4$, we deduce that $T_{3,3} = T_{3,1}$. Hence, the claim follows from Proposition 4.1.4. ☐

4.1.3 The alternating group $\mathfrak{A}_5$

Let $G := \mathfrak{A}_5 = \langle a, d \rangle \leq \mathfrak{S}_5$, where $a := (1,2)(3,4)$ and $d := (1,3,5)$. Moreover, we set $b := (1,3)(2,4) \in G$ and $c := (1,2,3) \in G$. Defining $A := \frac{1-\sqrt{5}}{2}$ and ${}^*A := \frac{1+\sqrt{5}}{2}$, the ordinary character table of $\mathfrak{A}_5$ is given as follows:

	1	a	d	ad	$(ad)^2$
χ_1	1	1	1	1	1
χ_2	3	-1	0	A	${}^*A$
χ_3	3	-1	0	${}^*A$	A
χ_4	4	0	1	-1	-1
χ_5	5	1	-1	0	0

Table 4.5: **ordinary character table of $\mathfrak{A}_5$**

We set

$$Q_1 := \langle 1 \rangle, Q_2 := \langle a \rangle, \text{ and } Q_3 := \langle a, b \rangle \in \mathrm{Syl}_2(\mathfrak{A}_5).$$

Furthermore, we set $\omega := \exp(\frac{2\pi i}{3}) = \frac{-1+i\sqrt{3}}{2}$, $\eta := \exp(\frac{2\pi i}{5}) = \frac{1}{4}(\sqrt{5}-1) + i\sqrt{\frac{5}{8}+\frac{\sqrt{5}}{8}}$, and choose $\mathscr{S}_p(\mathfrak{A}_5) = \{Q_1, Q_2, Q_3\}$. Then, the lattice of subgroups in $\mathscr{S}_2(\mathfrak{A}_5)$ is given as follows:

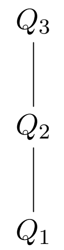


Figure 4.3: lattice of subgroups in $\mathscr{S}_2(\mathfrak{A}_5)$

Moreover,

$$\begin{aligned} N_{\mathfrak{A}_5}(Q_1) &= \mathfrak{A}_5 && \text{and } \overline{N}_{\mathfrak{A}_5}(Q_1) \cong \mathfrak{A}_5; \\ N_{\mathfrak{A}_5}(Q_2) &= Q_3 && \text{and } \overline{N}_G(Q_2) \cong \langle b \rangle \cong C_2; \\ N_{\mathfrak{A}_5}(Q_3) &= \langle a, c \rangle \cong \mathfrak{A}_4 && \text{and } \overline{N}_{\mathfrak{A}_5}(Q_3) \cong \langle c \rangle \cong C_3. \end{aligned}$$

By counting the p'-conjugacy classes of $\mathfrak{A}_5$, we deduce that, up to isomorphism, there exist exactly four simple $k\mathfrak{A}_5$-modules. We denote them by $S_1 := k$, S_2, S_3 and S_4. Moreover, we set $H := \langle (1,2)(3,4), (1,4)(2,5) \rangle \leq \mathfrak{A}_5$ and $\widetilde{M} := k\uparrow_H^{\mathfrak{A}_5}$. Note that $H \cong D_{10}$.

Remark 4.1.6. We identify $N_{\mathfrak{A}_5}(Q_3)$ with the group $\mathfrak{A}_4$ from the previous subsection. Note that these two permutation groups are even identical.

Proposition 4.1.7. *The following assertions about the trivial source $k\mathfrak{A}_5$-modules hold.*

(a) *We have* $\mathrm{TS}(\mathfrak{A}_5; Q_1) = \{P(k), P(S_2), P(S_3), P(S_4)\}$.

(b) *We have* $\mathrm{TS}(\mathfrak{A}_5; Q_2) = \{\widetilde{M}\}$.

(c) *Let the two 5-dimensional indecomposable summands of $k\uparrow_{Q_3}^{\mathfrak{A}_5}$ be denoted by M_7 and M_8, respectively. Then we have* $\mathrm{TS}(\mathfrak{A}_5; Q_3) = \{S_1, M_7, M_8\}$.

Proof. By counting the p'-conjugacy classes of $\overline{N}_{\mathfrak{A}_5}(v)$, for $1 \leq v \leq 3$, we deduce that

$$|\mathrm{TS}(\mathfrak{A}_5; Q_1)| = 4, |\mathrm{TS}(\mathfrak{A}_5; Q_3)| = 3, \text{ and } |\mathrm{TS}(\mathfrak{A}_5; Q_2)| = 1.$$

The $k\mathfrak{A}_5$-modules $P(S_1)$, $P(S_2)$, $P(S_3)$ and $P(S_4)$ are trivial source modules by definition. This proves (a). We have $Q_2 \in \mathrm{Syl}_2(H)$. As there is only one p'-conjugacy class in $\overline{N}_H(Q_2)$, we deduce that $\mathrm{TS}(H; Q_2) = \{k_H\}$. All other trvial source kH-modules are projective and their dimensions are therefore divisible by 2. Since $k\!\uparrow_{Q_2}^{H}$ is 5-dimensional, we obtain

$$k\!\uparrow_{Q_2}^{H} \cong k \oplus X$$

for some projective kH-module X. Therefore,

$$k\!\uparrow_{Q_2}^{\mathfrak{A}_5} = (k\!\uparrow_{Q_2}^{H})\!\uparrow_{H}^{\mathfrak{A}_5} = (k \oplus X)\!\uparrow_{H}^{\mathfrak{A}_5} = k\!\uparrow_{H}^{\mathfrak{A}_5} \oplus X\!\uparrow_{H}^{\mathfrak{A}_5}.$$

Note that the $k\mathfrak{A}_5$-module $\widetilde{M} = k\!\uparrow_{H}^{\mathfrak{A}_5}$ is 6-dimensional. By transitivity of induction, we have

$$k\!\uparrow_{Q_2}^{\mathfrak{A}_5} = (k\!\uparrow_{Q_2}^{N_{\mathfrak{A}_5}(Q_3)})\!\uparrow_{N_{\mathfrak{A}_5}(Q_3)}^{\mathfrak{A}_5}.$$

We know from Proposition 4.1.4 that the $kN_{\mathfrak{A}_5}(Q_3)$-module $k\!\uparrow_{Q_2}^{N_{\mathfrak{A}_5}(Q_3)}$ is indecomposable. As $\dim_k(k\!\uparrow_{Q_2}^{N_{\mathfrak{A}_5}(Q_3)}) = 6$ and $N_{\mathfrak{A}_5}(Q_2) \leq N_{\mathfrak{A}_5}(Q_3)$, it follows from the Green correspondence that $\dim_k(g(k\!\uparrow_{Q_2}^{N_{\mathfrak{A}_5}(Q_3)})) \geq 6$. Hence, $k\!\uparrow_{H}^{\mathfrak{A}_5}$ is indecomposable. This proves (b). Next, consider the $k\mathfrak{A}_5$-module $L := k\!\uparrow_{Q_3}^{\mathfrak{A}_5}$. Its ordinary character $\chi_{\widehat{L}}$ is equal to

$$\chi_{\widehat{L}} = 1\!\uparrow_{Q_3}^{\mathfrak{A}_5} = \chi_1 + \chi_4 + 2 \cdot \chi_5.$$

Since χ_4 does not belong to $\mathrm{Irr}_K(B_0(k\mathfrak{A}_5))$, the set $\mathrm{TS}(\mathfrak{A}_5; Q_3)$ is as asserted. □

Proposition 4.1.8. *Labelling the ordinary characters as in Table 4.5, the trivial source character table* $\mathrm{Triv}_2(\mathfrak{A}_5)$ *is given as follows:*

Q_v	$Q_1 \cong C_1$				$Q_2 \cong C_2$	$Q_3 \cong V_4$		
N_v	$N_1 \cong \mathfrak{A}_5$				$N_2 \cong V_4$	$N_3 \cong \mathfrak{A}_4$		
$n_j \in N_v$	1	d	ad	$(ad)^2$	1	1	c	c^2
$\chi_1+\chi_2+\chi_3+\chi_5$	12	0	2	2	0	0	0	0
$\chi_3+\chi_5$	8	-1	$-\eta^2-\eta^3$	$-\eta-\eta^4$	0	0	0	0
$\chi_2+\chi_5$	8	-1	$-\eta-\eta^4$	$-\eta^2-\eta^3$	0	0	0	0
χ_4	4	1	-1	-1	0	0	0	0
$\chi_1+\chi_5$	6	0	1	1	2	0	0	0
χ_1	1	1	1	1	1	1	1	1
χ_5	5	-1	0	0	1	1	ω	ω^2
χ_5	5	-1	0	0	1	1	ω^2	ω

Table 4.6: **trivial source character table of** $\mathfrak{A}_5$ at $p = 2$

Proof. By [WTP+98, A5mod2.pdf], the decomposition matrix of $\mathfrak{A}_5$ at $p = 2$ is given by

	φ_{S_1}	φ_{S_2}	φ_{S_3}	φ_{S_4}
χ_1	1	0	0	0
χ_2	1	0	1	0
χ_3	1	1	0	0
χ_4	0	0	0	1
χ_5	1	1	1	0

and, therefore, the algebra $k\mathfrak{A}_5$ has exactly two 2-blocks. Moreover, by Proposition 4.1.7, we have

$$\begin{aligned}\Phi_1 &:= \chi_{\widehat{P(k)}} = \chi_1 + \chi_2 + \chi_3 + \chi_5,\\ \Phi_2 &:= \chi_{\widehat{P(S_2)}} = \chi_3 + \chi_5,\\ \Phi_3 &:= \chi_{\widehat{P(S_3)}} = \chi_2 + \chi_5, \text{ and}\\ \Phi_4 &:= \chi_{\widehat{P(S_4)}} = \chi_4.\end{aligned}$$

By the proof of Proposition 4.1.7, we have $\chi_{\widehat{\widehat{M}}} = 1{\uparrow}_H^{\mathfrak{A}_5} = \chi_1+\chi_5$. Since $\chi_{\widehat{L}} = \chi_1+\chi_4+2\cdot\chi_5$ and since $\chi_4 \notin \mathrm{Irr}_K(B_0(k\mathfrak{A}_5))$, the ordinary characters of the trivial source $k\mathfrak{A}_5$-modules are as asserted. Set $M_6 := k$. By Remark 3.2.6 we have

$$\tau^{\mathfrak{A}_5}_{Q_2,1}([M_i]) = \dim_k\left(\mathrm{Br}^{Q_2}_{Q_2} \circ \mathrm{Res}^{\mathfrak{A}_5}_{Q_2}(M_i)\right) = \chi_{\widehat{M_i}}(a) = 1,$$

for $6 \le i \le 8$. All remaining entries of $\mathrm{Triv}_2(\mathfrak{A}_5)$ are determined by Proposition 4.1.7 and Remark 3.2.6. □

We conclude with the following lemma which is used in the sequel. We keep the notation of Section 4.1.3.

Lemma 4.1.9. *The $k\mathfrak{A}_5$-module $M_5 := k{\uparrow}_H^{\mathfrak{A}_5}$ has trivial socle of k-dimension 1.*

Proof. We have $k{\uparrow}_{Q_2}^{\mathfrak{A}_5} \cong M_5 \oplus 2\cdot P(S_4)\oplus P(S_2)\oplus P(S_3)$. Thus, it follows from Lemma 2.3.15 that $S_2{\downarrow}_{Q_2}^{\mathfrak{A}_5} \cong kQ_2{}^{\mathrm{reg}}$ and $S_3{\downarrow}_{Q_2}^{\mathfrak{A}_5} \cong kQ_2{}^{\mathrm{reg}}$. We observe that for all $i \in \{2,3\}$ we have

$$\begin{aligned}\mathrm{Hom}_{kQ_2}(k,k) &\cong \mathrm{Hom}_{kQ_2}(k, \mathrm{Soc}(kQ_2{}^{\mathrm{reg}})) \cong \mathrm{Hom}_{kQ_2}(k, kQ_2{}^{\mathrm{reg}}) \cong \mathrm{Hom}_{kQ_2}(k, S_i{\downarrow}_{Q_2}^{\mathfrak{A}_5})\\ &\cong \mathrm{Hom}_{k\mathfrak{A}_5}(k{\uparrow}_{Q_2}^{\mathfrak{A}_5}, S_i) \cong \mathrm{Hom}_{k\mathfrak{A}_5}(M_5 \oplus 2\cdot P(S_4)\oplus P(S_2)\oplus P(S_3), S_i)\\ &\cong \mathrm{Hom}_{k\mathfrak{A}_5}(\mathrm{Hd}(M_5 \oplus 2\cdot P(S_4)\oplus P(S_2)\oplus P(S_3)), S_i)\\ &\cong \mathrm{Hom}_{k\mathfrak{A}_5}(\mathrm{Hd}(M_5) \oplus 2\cdot S_4\oplus S_2\oplus S_3, S_i),\end{aligned}$$

by Lemma 2.1.16. It follows from $\dim_k(\mathrm{Hom}_{kQ_2}(k,k)) = 1$ that neither S_2 nor S_3 is a direct summand of $\mathrm{Hd}(M_5)$. Using Lemma 2.2.4 and Lemma 2.1.16, we compute for $S_1 = k$:

$$\begin{aligned}\mathrm{Hom}_{kQ_2}(k,k) &\cong \mathrm{Hom}_{kQ_2}(k, S_1{\downarrow}_{Q_2}^{\mathfrak{A}_5})\\ &\cong \mathrm{Hom}_{k\mathfrak{A}_5}(k{\uparrow}_{Q_2}^{\mathfrak{A}_5}, S_1) \cong \mathrm{Hom}_{k\mathfrak{A}_5}(M_5 \oplus 2\cdot P(S_4)\oplus P(S_2)\oplus P(S_3), S_1)\\ &\cong \mathrm{Hom}_{k\mathfrak{A}_5}(\mathrm{Hd}(M_5 \oplus 2\cdot P(S_4)\oplus P(S_2)\oplus P(S_3)), S_1)\\ &\cong \mathrm{Hom}_{k\mathfrak{A}_5}(\mathrm{Hd}(M_5) \oplus 2\cdot S_4\oplus S_2\oplus S_3, S_1).\end{aligned}$$

Since $\dim_k(\mathrm{Hom}_{kQ_2}(k,k)) = 1$, the multiplicity of $S_1 = k$ in $\mathrm{Hd}(M_5)$ is equal to one. Hence, $\mathrm{Hd}(M_5) \cong S_1 = k$, since M_5 belongs to $B_0(k\mathfrak{A}_5)$.

As $\mathrm{TS}(\mathfrak{A}_5; Q_2) = \{M_5\}$, the $k\mathfrak{A}_5$-module M_5 is self-dual. Since $M_5 \cong \mathrm{Sc}(\mathfrak{A}_5, Q_2)$, we conclude that

$$\mathrm{Soc}(M_5) \cong \mathrm{Hd}(M_5)^* \cong k^* \cong k.$$

□

4.2 Domestic representation type

The classification of the trivial source modules in blocks with cyclic defect groups was accomplished by Hiß and Lassueur in [HL21]. Hence, as a next step, it is natural to consider the trivial source modules belonging to an arbitrary block whose defect groups are isomorphic to a Klein four-group. This is done here. We also compute their ordinary characters. We use methods from classical module theory here in order to illustrate with appealing examples how these methods can be applied in order to obtain theoretical results about trivial source modules.

Let the symbol k denote an algebraically closed field of characteristic $p > 0$ for a (not necessarily even) prime number p.

Assume we are given a k-algebra A. Then, there are either finitely many or infinitely many isomorphism classes of indecomposable A-modules. In the former case, we say that A is of *finite representation type*. In the latter case, it is possible to distinguish further between algebras of *tame representation type* - where the isomorphism classes of indecomposable A-modules can still be classified dimension by dimension and grouped into (infinite) families - and algebras of *wild representation type* - where such a classification is not possible.

Theorem 4.2.1 ([Ben98, Theorem 4.4.2]). *Over an algebraically closed field, every finite dimensional algebra is of finite, tame or wild representation type, and these types are mutually exclusive.*

A subclass of all algebras of tame representation type is given by the class of all algebras of *domestic representation type*. Algebras of domestic representation type are tame algebras whose isomorphism classes of indecomposable modules can be classified as those algebras whose indecomposable modules satisfy, up to isomorphism, a condition which is even more restrictive than the condition imposed on the indecomposable modules of tame non-domestic algebras.

Proposition 4.2.2 ([Kra98, The second Corollary]). *Let Λ and Γ be finite-dimensional algebras over an algebraically closed field. Suppose that Λ and Γ are stably equivalent of Morita type. Then the algebra Λ is of domestic representation type if and only if the algebra Γ is of domestic representation type.*

Remark 4.2.3. Hence, domestic representation type is also preserved by Morita equivalences.

For group algebras, there is the following interesting result.

Theorem 4.2.4 ([Ben98, Theorem 4.4.4]). *The following assertions hold.*

(a) *The algebra kG has finite representation type if and only if G has cyclic Sylow p-subgroups.*

(b) *The algebra kG has domestic representation type if and only if $p = 2$ and the Sylow 2-subgroups of G are isomorphic to the Klein four-group.*

(c) *The algebra kG has tame representation type if and only if $p = 2$ and the Sylow 2-subgroups are dihedral, semidihedral or generalised quaternion.*

(d) *In all other cases kG has wild representation type.*

For the rest of this section, we make the following assumptions:

1. H is a finite group;
2. $\mathrm{char}(k) = p = 2$;
3. $B' \in \mathrm{Bl}(kH)$;
4. $D(B')$ is an arbitrary but fixed defect group of B'.

The following results are used in the sequel.

Lemma 4.2.5. *Let G be a finite group and let $B \in \mathrm{Bl}(kG)$ with a cyclic defect group of order two. Suppose that $D(B)$ is an arbitrary but fixed defect group of B. Set $D := D(B)$. Let S be a simple B-module. Then, the following assertions hold.*

(a) *Up to isomorphism, there are exactly two non-isomorphic indecomposable kG-modules that belong to B, namely the module S and its projective cover $P(S)$.*

(b) *The kG-module S has a trivial source.*

Proof. The group algebra $A := kD$ has, up to isomorphism, exactly two non-isomorphic modules, namely k and $P(k)$. The Brauer tree $\sigma(A)$ of A is as given in Fig. 4.4.

$$\circ \text{------} \circ$$

Figure 4.4: the Brauer tree $\sigma(A)$ of A

The Brauer tree $\sigma(B)$ of B has the same shape as $\sigma(A)$. Indeed, the inertial index e has to divide $p - 1 = 2 - 1 = 1$, so $e = 1$. Moreover,

$$|\mathrm{Irr}_K(B)| = e + \frac{|D| - 1}{e} = 1 + \frac{2 - 1}{1} = 2$$

and $|\mathrm{IBr}(B)| = 1$. Since the Brauer tree, together with its planar embedding, encodes the Morita equivalence class of a Brauer tree algebra, it follows that there exists a Morita equivalence $F : {}_B\mathrm{mod} \to {}_A\mathrm{mod}$. Since the algebra A has, up to isomorphism, exactly two non-isomorphic indecomposable modules, the same holds for the algebra B, as Morita equivalences preserve the number of indecomposable modules. Moreover, Morita equivalences preserve simple modules and projective modules. Hence, the assertions in (a) follow. By [Mic75, Theorem 0.2], there is, up to isomorphism, exactly one kG-module with source $k\downarrow^{kG}_{kD}$ belonging to the block algebra B. Since $P(S)$ has source $k\downarrow^{kG}_{k\langle 1\rangle}$, the claim in part (b) follows. ☐

Theorem 4.2.6 ([CEKL11, Theorem 1.1])**.** *Let $B' \in \mathrm{Bl}(kH)$ such that $D(B') \cong V_4$. Then there exists a splendid Morita equivalence between B and either kV_4 or $k\mathfrak{A}_4$ or $B_0(k\mathfrak{A}_5)$.*

Remark 4.2.7. It follows from Theorem 4.2.6, Proposition 4.2.2, and Theorem 4.2.4 that splendid Morita equivalences between blocks of group algebras preserve domestic representation type.

Recall from Proposition 3.1.11 that splendid Morita equivalences preserve trivial source modules and their vertices.

Before we come to the main result of this section, we need the following:

Proposition 4.2.8 ([Bra71, Proposition 7B])**.** *Let $B' \in \mathrm{Bl}(kH)$. Suppose that there exists a splendid Morita equivalence between B' and kV_4. Denote the elements of the defect group $D(B')$ of B' by $1, a, b, ab$. Denote the ordinary irreducible characters in B' by $\chi_\alpha, \chi_\beta, \chi_\gamma, \chi_\delta$. Then, after a suitable relabelling of the characters χ_ι, $\iota \in \{\alpha, \beta, \gamma, \delta\}$, the following holds for the i-th character in $\mathrm{Irr}_K(B')$, for $1 \le i \le 4$.*

(I) *If $a \sim b \sim ab$ in H then there are positive integers n_1, n_2, n_3 and signs $\varepsilon_{i,1}, \varepsilon_{i,2}, \varepsilon_{i,3} \in \{\pm 1\}$ such that*
$$\chi_i(a) = \varepsilon_{i,1} n_1 + \varepsilon_{i,2} n_2 + \varepsilon_{i,3} n_3.$$

(II) *If $a \not\sim b \sim ab$ in H then there are positive integers n_1, n_2, n_3 and signs $\varepsilon_{i,1}, \varepsilon_{i,2}, \varepsilon_{i,3} \in \{\pm 1\}$ such that*
$$\chi_i(a) = \varepsilon_{i,1} n_1; \; \chi_i(b) = \varepsilon_{i,2} n_2 + \varepsilon_{i,3} n_3.$$

(III) *If $a \not\sim b \not\sim ab \not\sim a$ in H then there are positive integers n_1, n_2, n_3 and signs $\varepsilon_{i,1}, \varepsilon_{i,2}, \varepsilon_{i,3} \in \{\pm 1\}$ such that*
$$\chi_i(a) = \varepsilon_{i,1} n_1; \; \chi_i(b) = \varepsilon_{i,2} n_2; \; \chi_i(ab) = \varepsilon_{i,3} n_3.$$

In each of the three formulae (I), (II), and (III) above the integers $\varepsilon_{i,1}, \varepsilon_{i,2}, \varepsilon_{i,3} \in \{\pm 1\}$ in front of n_1, n_2, n_3 are given by
$$\begin{pmatrix} \varepsilon_{1,1} & \varepsilon_{1,2} & \varepsilon_{1,3} \\ \varepsilon_{2,1} & \varepsilon_{2,2} & \varepsilon_{2,3} \\ \varepsilon_{3,1} & \varepsilon_{3,2} & \varepsilon_{3,3} \\ \varepsilon_{4,1} & \varepsilon_{4,2} & \varepsilon_{4,3} \end{pmatrix} = \begin{pmatrix} +1 & +1 & +1 \\ +1 & -1 & -1 \\ -1 & +1 & -1 \\ -1 & -1 & +1 \end{pmatrix}.$$

4.2.1 There exists a splendid Morita equivalence between B' and kV_4

Let $G := V_4$ and let $B := B_0(kG)$. Assume there exists a splendid Morita equivalence between B and B' defined by a functor F. We keep the notation from Section 4.1.1. In particular, we set $\mathrm{Irr}_K(B) := \{\chi_1, \chi_2, \chi_3, \chi_4\}$ and $D(B) := \langle a, b \rangle$. Moreover, we set $\mathrm{Irr}_K(B') := \{\chi_\alpha, \chi_\beta, \chi_\gamma, \chi_\delta\}$ and $D(B') := \langle a', b' \rangle$. Furthermore, we endow the images of the B-modules under F with the symbol $'$.

Remark 4.2.9. Here we assume that only the existence of F is known, but not an explicit bimodule inducing F.

Proposition 4.2.10. *The following assertions about the trivial source B'-modules hold.*

(a) *The trivial source B'-modules are given as follows.*
   (i) *We have $\mathrm{TS}(B'; D(B')) = \{S'\}$, where S' denotes the unique simple B'-module (up to isomorphism).*
   (ii) *We have $\mathrm{TS}(B'; \langle 1 \rangle) = \{P(S')\}$.*
   (iii) *Up to isomorphism, there are exactly three non-isomorphic trivial source B'-modules M_1', M_2' and M_3', respectively, with non-trivial cyclic vertices isomorphic to C_2.*

(b) *After a suitable relabelling of the elements of $\mathrm{Irr}_K(B')$ we have*
$$\chi_{\widehat{S'}} = \chi_\alpha, \chi_{\widehat{P(S')}} = \chi_\alpha + \chi_\beta + \chi_\gamma + \chi_\delta, \chi_{\widehat{M_1'}} = \chi_\alpha + \chi_\beta, \chi_{\widehat{M_2'}} = \chi_\alpha + \chi_\gamma, \chi_{\widehat{M_3'}} = \chi_\alpha + \chi_\delta.$$

(c) *The character values of $\mathrm{Irr}_K(B')$ determine $\chi_{\widehat{S'}}$ uniquely.*

Proof. By Proposition 3.1.11(f) both all trivial source modules and their respective vertices are preserved under splendid Morita equivalences. Moreover, every Morita equivalence preserves projective indecomposable modules and simple modules. This proves (a). Decomposition matrices are preserved by Morita equivalences. Hence, the decomposition matrix of B' looks as follows:

	$\varphi_{S'}$
χ_α	1
χ_β	1
χ_γ	1
χ_δ	1

Therefore, $\chi_{\widehat{P(S')}} = \chi_\alpha + \chi_\beta + \chi_\gamma + \chi_\delta$. Since composition factors are preserved by Morita equivalences, we deduce from the trivial source character table of kV_4 that each M_i' has the composition series

$$\boxed{\begin{matrix} S' \\ S' \end{matrix}}$$

for $1 \leq i \leq 3$. We claim that $\dim_k(\mathrm{Hom}_{kH}(S', M_i')) = 1$ for each $i \in \{1,2,3\}$.
If $f \in \mathrm{Hom}_{kH}(S', M_i')$ then, using the statement and notation of Lemma 2.1.16, we deduce that $\widetilde{f}(\mathrm{Hd}(S')) = \{0\}$, since $S' \ncong M_i'$ for every $i \in \{1,2,3\}$. Thus, every element of $S' \cong \mathrm{Hd}(S')$ is mapped to $\mathrm{Rad}(M_i') \cong S'$. Hence, by Schur's lemma, for each $i \in \{1,2,3\}$,

$$\dim_k(\mathrm{Hom}_{kH}(S', M_i')) = \dim_k(\mathrm{Hom}_{kH}(S', S')) = 1.$$

The multiplicity of χ_α as a constituent in $\chi_{\widehat{M_i'}}$ is therefore equal to 1 by Proposition 3.1.7.

The decomposition matrix of B' implies $\chi_\iota(1) = \dim_k(S')$ for all $\iota \in \{\alpha,\beta,\gamma,\delta\}$. Hence, for each $i \in \{1,2,3\}$, $\chi_{\widehat{M_i'}} = \chi_\alpha + \chi_j$ for some $j \in \{\beta,\gamma,\delta\}$ due to the shape of the composition series of the M_i'.

After a suitable relabelling the assertion in (b) follows now from the fact that

$$\dim_k(\mathrm{Hom}_{kH}(M_i', M_j')) = 1 \text{ for all } 1 \leq i \neq j \leq 3.$$

In order to see this, set $M' := M_i'$ and $N' := M_j'$ for some arbitrary $i,j \in \{1,2,3\}$ with $i \neq j$. Let $f \in \mathrm{Hom}_{kH}(M', N')$. Since $\dim_k(M') = \dim_k(N')$ but $M' \ncong N'$, the shape of the composition series implies the following assertion.

$(*)$ The morphism f maps every element of $\mathrm{Hd}(M')$ to $\mathrm{Rad}(N')$.

Indeed, this follows from $\mathrm{Hd}(M') \cong S' \cong \mathrm{Hd}(N')$ and Schur's lemma. We claim now that in the present case not only $f(\mathrm{Rad}(M')) \leq \mathrm{Rad}(N')$ but also the equation $f(\mathrm{Rad}(M')) = \{0\}$ holds.
In order to see this, let $\widetilde{m} \in \mathrm{Rad}(M') = \mathrm{Rad}(kH) \cdot M'$. Then $\widetilde{m} = j \cdot m$ for some $m \in M'$ and some $j \in \mathrm{Rad}(kH)$. Consequently, $f(\widetilde{m}) = f(j \cdot m) = j \cdot f(m)$. Due to $(*)$ we have $f(m) \in \mathrm{Rad}(N')$. Hence,

$$f(\widetilde{m}) \in \mathrm{Rad}(kH) \cdot \mathrm{Rad}(N') = \mathrm{Rad}(\mathrm{Rad}(N')) = \mathrm{Rad}(S') = \{0\}.$$

Therefore, $\dim_k(\mathrm{Hom}_{kH}(M', N')) = \dim_k(\mathrm{Hom}_{kH}(S', S')) = 1$. The assertions in (b) follow now from Proposition 3.1.7.
The ordinary character $\chi_{\widehat{S'}}$ is the unique ordinary irreducible character χ lying in B' with the property that $\chi(x)$ is as large as possible for each $x \in D(B')$. Indeed, since S' has maximal vertex $D(B')$, this follows from Proposition 4.2.8 and Lemma 3.1.6. □

4.2.2 There exists a splendid Morita equivalence between B' and $k\mathfrak{A}_4$

Let $G := \mathfrak{A}_4$. We begin with the following auxiliary lemma where we keep the notation from Proposition 4.1.4. Moreover we set $S_1 := k, S_2 := S_\omega$, and $S_3 := S_{\overline{\omega}}$.

Lemma 4.2.11. *We have* $\mathrm{Soc}(M) \cong S_1 \oplus S_2 \oplus S_3$.

Proof. Set $\widetilde{H} := \langle(1,2)\rangle$. The kG-module $M := k{\uparrow}_{\widetilde{H}}^{\mathfrak{A}_4}$ is, up to isomorphism, the only trivial source $k\mathfrak{A}_4$-module with a non-trivial cyclic vertex. Hence, $M^* \cong M$, since dual modules have the same vertices. It follows from [Alp86, Lemma III.5] that $\mathrm{Soc}(M) \cong (\mathrm{Hd}(M))^*$. By Lemma 2.2.4, we obtain $\mathrm{Hom}_{kG}(M, S_i) \cong \mathrm{Hom}_{k\widetilde{H}}(k, S_i{\downarrow}_{\widetilde{H}}^{\mathfrak{A}_4})$ for each $1 \leq i \leq 3$. But $S_i{\downarrow}_{\widetilde{H}}^{\mathfrak{A}_4}$ is a one-dimensional trivial source $k\widetilde{H}$-module, hence isomorphic to k for all $i \in \{1,2,3\}$. Consequently, $\dim_k \mathrm{Hom}_{kG}(M, S_i) = \dim_k \mathrm{Hom}_{k\widetilde{H}}(k,k) = 1$ for all $1 \leq i \leq 3$.
Since $\dim_k \mathrm{Hom}_{kG}(M, S_i)$ is equal to the multiplicity of S_i as a direct summand in $\mathrm{Hd}(M)$ for all $i \in \{1,2,3\}$, it follows that $\mathrm{Hd}(M) \cong S_1 \oplus S_2 \oplus S_3$. Hence, as $\mathrm{Soc}(M) \cong (\mathrm{Hd}(M))^*$, we deduce that $\mathrm{Soc}(M) \cong S_1 \oplus S_2 \oplus S_3$. □

Now, let $B := B_0(kG)$ and let $B' \in \mathrm{Bl}(kH)$ be a block of another group algebra kH. Assume there exists a splendid Morita equivalence between B and B' defined by a functor F. We keep the notation from Section 4.1.2. In particular, we set $\mathrm{Irr}_K(B) := \{\chi_1, \chi_2, \chi_3, \chi_4\}$ and $D(B) := \langle a, b\rangle$. Moreover, we set $\mathrm{Irr}_K(B') := \{\chi_\alpha, \chi_\beta, \chi_\gamma, \chi_\delta\}$ and $D(B') := \langle a', b'\rangle$. Furthermore, we denote the image of a B-module Y under F by Y'.

Proposition 4.2.12. *The following assertions about the trivial source B'-modules hold.*

(a) *Up to isomorphism there are exactly 7 trivial source B'-modules. They are given as follows.*
  (i) *We have* $\mathrm{TS}(B'; D(B')) = \{S_1', S_2', S_3'\}$.
  (ii) *We have* $\mathrm{TS}(B'; \langle 1\rangle) = \{P(S_1'), P(S_2'), P(S_3')\}$.
  (iii) *Up to isomorphism, there is exactly one trivial source B'-module M' with non-trivial cyclic vertices isomorphic to C_2.*

(b) *After a suitable relabelling of the elements of* $\mathrm{Irr}_K(B')$ *we have:*

$$\chi_{\widehat{S_1'}} = \chi_\alpha, \chi_{\widehat{S_2'}} = \chi_\beta, \chi_{\widehat{S_3'}} = \chi_\gamma,$$
$$\chi_{\widehat{P(S_1')}} = \chi_\alpha + \chi_\delta, \chi_{\widehat{P(S_2')}} = \chi_\beta + \chi_\delta, \chi_{\widehat{P(S_3')}} = \chi_\gamma + \chi_\delta,$$
$$\chi_{\widehat{M'}} = \chi_\alpha + \chi_\beta + \chi_\gamma + \chi_\delta.$$

(c) *The set* $\mathrm{Irr}_K(B')$ *determines χ_δ uniquely and in a purely character-theoretic way.*

Proof. By Proposition 3.1.11(f) both all trivial source modules and their respective vertices are preserved under splendid Morita equivalences. Moreover, every Morita equivalence preserves projective indecomposable modules and simple modules. This proves (a). Decomposition matrices are preserved by Morita equivalences. Hence, the decomposition matrix of B' looks as follows:

	$\varphi_{S_1'}$	$\varphi_{S_2'}$	$\varphi_{S_3'}$
χ_α	1	0	0
χ_β	0	1	0
χ_γ	0	0	1
χ_δ	1	1	1

As the composition factors of the trivial source $k\mathfrak{A}_4$-module M are given by

$$[M] = 2 \cdot S_1 + 2 \cdot S_2 + 2 \cdot S_3,$$

we deduce that the composition factors of M' are given by

$$[M'] = 2 \cdot S_1' + 2 \cdot S_2' + 2 \cdot S_3'.$$

Since $\mathrm{Soc}(M) \cong S_1 \oplus S_2 \oplus S_3$, we deduce that $\mathrm{Soc}(M') \cong S_1' \oplus S_2' \oplus S_3'$. Investigating the decomposition matrix of B' we see that

$$\chi_\delta(1) = \dim_k(S_1') + \dim_k(S_2') + \dim_k(S_3').$$

Hence, χ_δ can be distinguished from χ_α, χ_β and χ_γ by its degree. Let $f \in \mathrm{Hom}_{kH}(S_1', M')$ be non-trivial. Since $f(S_1') = f(\mathrm{Soc}(S_1')) \leq \mathrm{Soc}(M') \cong S_1' \oplus S_2' \oplus S_3'$, we deduce that $\dim_k(\mathrm{Hom}_{kH}(S_1', M')) = 1$. Analogously it follows that $\dim_k(\mathrm{Hom}_{kH}(S_2', M')) = 1$ and $\dim_k(\mathrm{Hom}_{kH}(S_3', M')) = 1$. Consequently, $\chi_{\widehat{M'}} = \chi_\alpha + \chi_\beta + \chi_\gamma + \chi_\delta$. This proves part (b) and part (c). □

4.2.3 There exists a splendid Morita equivalence between B' and $B_0(k\mathfrak{A}_5)$

Let $G := \mathfrak{A}_5$, let $B := B_0(kG)$, and let $B' \in \mathrm{Bl}(kH)$ be a block of another group algebra kH. Assume there exists a splendid Morita equivalence between B and B' defined by a functor F. We keep the notation from Section 4.1.3. In particular, we set $\mathrm{Irr}_K(B) := \{\chi_1, \chi_2, \chi_3, \chi_5\}$ and $D(B) := \langle a, b\rangle$. Moreover, we set $\mathrm{Irr}_K(B') := \{\chi_\alpha, \chi_\beta, \chi_\gamma, \chi_\delta\}$ and $D(B') := \langle a', b'\rangle$. Furthermore, we endow the images of the B-modules under F with a prime mark.

Proposition 4.2.13. *The following assertions about the trivial source B'-modules hold.*

(a) *Up to isomorphism there are exactly 7 trivial source B'-modules. They are given as follows.*

  (i) *We have* $\mathrm{TS}(B'; D(B')) = \{S_1', M_7', M_8'\}$.

  (ii) *We have* $\mathrm{TS}(B'; \langle 1\rangle) = \{P(S_1'), P(M_7'), P(M_8')\}$.

  (iii) *Up to isomorphism, there is exactly one trivial source B'-module $\widetilde{M}'$ with non-trivial cyclic vertices isomorphic to C_2.*

(b) *After a suitable relabelling of the elements of* $\mathrm{Irr}_K(B')$ *we have:*

$$\chi_{\widehat{S_1'}} = \chi_\alpha, \chi_{\widehat{M_7'}} = \chi_\delta, \chi_{\widehat{M_8'}} = \chi_\delta,$$
$$\chi_{\widehat{P(S_1')}} = \chi_\alpha + \chi_\beta + \chi_\gamma + \chi_\delta, \chi_{\widehat{P(S_2')}} = \chi_\gamma + \chi_\delta, \chi_{\widehat{P(S_3')}} = \chi_\beta + \chi_\delta;$$
$$\chi_{\widehat{\widetilde{M}'}} = \chi_\alpha + \chi_\delta.$$

(c) *The set $\mathrm{Irr}_K(B')$ determines both χ_α and χ_δ uniquely and in a purely character-theoretic way.*

Proof. By Proposition 3.1.11(f) both all trivial source modules and their respective vertices are preserved under splendid Morita equivalences. Moreover, every Morita equivalence preserves projective indecomposable modules and simple modules. This proves (a). Decomposition matrices are preserved by Morita equivalences. Hence, the decomposition matrix of B' looks as follows:

	$\varphi_{S'_1}$	$\varphi_{S'_2}$	$\varphi_{S'_3}$
χ_α	1	0	0
χ_β	1	0	1
χ_γ	1	1	0
χ_δ	1	1	1

As the composition factors of the trivial source $k\mathfrak{A}_5$-module $\widetilde{M}$ are given by

$$[\widetilde{M}] = 2 \cdot S_1 + S_2 + S_3,$$

we deduce that the composition factors of $\widetilde{M}'$ are given by

$$[\widetilde{M}'] = 2 \cdot S'_1 + S'_2 + S'_3,$$

following the order of the decomposition matrix. Moreover, the respective composition factors of the projective indecomposable B'-modules are given as follows:

$$\begin{aligned}[P(S'_1)] &= 4 \cdot S'_1 + 2 \cdot S'_2 + 2 \cdot S'_3,\\ [P(S'_2)] &= 2 \cdot S'_1 + 2 \cdot S'_2 + 1 \cdot S'_3,\\ [P(S'_3)] &= 2 \cdot S'_1 + 1 \cdot S'_2 + 2 \cdot S'_3.\end{aligned}$$

It follows from the decomposition matrix of B' that $\chi_{S'_1} = \chi_\alpha$. By Lemma 4.1.9, we have $\mathrm{Soc}(\widetilde{M}') \cong S'_1$ and, therefore, $\dim_k(\mathrm{Hom}_{kH}(S'_1, \widetilde{M}')) = 1$. Hence, $\langle \chi_\alpha, \chi_{\widehat{\widetilde{M}'}} \rangle = 1$. It follows from the composition factors of $\widetilde{M}'$ and from the decomposition matrix of B' that $\chi_{\widehat{\widetilde{M}'}} = \chi_\alpha + \chi_\delta$. We have $[M'_7] = [M'_8] = S'_1 + S'_2 + S'_3$. The socles of the two trivial source B'-modules M'_7 and M'_8 are S'_2 and S'_3, respectively, since the socles of their preimages under F are S_2 and S_3, respectively. Hence, $\dim_k(\mathrm{Hom}_{kH}(S'_1, M'_7)) = 0 = \dim_k(\mathrm{Hom}_{kH}(S'_1, M'_8))$. Therefore, $\chi_{\widehat{M'_7}} = \chi_\delta = \chi_{\widehat{M'_8}}$. The assertions in (*c*) follow from the decomposition matrix of B'. □

4.3 Families of groups

In this section, we consider certain families of finite group algebras which are of tame domestic representation type. We deduce the following assertions from Section 4.2. Consider the group algebra kG, where k is an algebraically closed field of characteristic 2. If $G = D_{4v}$, the dihedral group of order $4v$, where v is an odd integer, then the group algebra kG is splendidly Morita equivalent to kV_4. If $G = \mathrm{PSL}_2(11)$, then the group algebra kG is splendidly Morita equivalent to $k\mathfrak{A}_4$. If $G = \mathrm{PSL}_2(13)$, then the group algebra kG is splendidly Morita equivalent to $B_0(k\mathfrak{A}_5)$. More generally, it follows from the generic chracter tables of $\mathrm{SL}_2(q)$, where q denotes an odd prime power, that $B_0(kG)$ is splendidly Morita equivalent to $k\mathfrak{A}_4$, if $q \equiv 3 \bmod 8$, and that $B_0(kG)$ is splendidly Morita equivalent to $B_0(k\mathfrak{A}_5)$, if $q \equiv 5 \bmod 8$.

4.3.1 The dihedral groups D_{4v} ($v \in \mathbb{Z}_{\geq 3}$ odd)

Let $G := D_{2n}$, the dihedral group of order $2n$. Assume $2n = 4v$, where $v \geq 3$ is an odd integer. Notice that $G \cong \langle s, r \rangle \leq \mathfrak{S}_n$, where $s := (1, n-1)(2, n-2) \cdots (\frac{n}{2} - 1, \frac{n}{2} + 1)$ and $r := (1, 2, \ldots, n)$ are a reflection and a (smallest) rotation, respectively.

Set $\omega_m := e^{-\frac{2\pi i m}{n}}$.

The $v+3$ conjugacy classes of G are given by

$$\begin{aligned}
[1] &= \{1\},\\
[r] &= \{r, r^{-1}\},\\
[r^2] &= \{r^2, r^{-2}\},\\
&\vdots\\
[r^{v-1}] &= \{r^{v-1}, r^{-(v-1)}\},\\
[r^v] &= \{r^v\},\\
[s] &= \{sr^{2a} \mid 1 \le a \le v\}, \text{ and}\\
[sr] &= \{sr^{2a-1} \mid 1 \le a \le v\}.
\end{aligned}$$

The $\frac{v+1}{2}$ distinct $2'-$conjugacy classes of G are given by $[1], [r^2], [r^4], \ldots, [r^{v-1}]$.

The ordinary character table of G is as given in Table 4.7 (cf. [JL01, §18.3]).

g	1	r	r^2	r^3	$\cdots$	r^{v-1}	r^v	s	sr
χ_1	1	1	1	1	$\cdots$	1	1	1	1
χ_2	1	-1	1	-1	$\cdots$	1	-1	1	-1
χ_3	1	1	1	1	$\cdots$	1	1	-1	-1
χ_4	1	-1	1	-1	$\cdots$	1	-1	-1	1
χ_5	2	$\omega_1 + \overline{\omega}_1$	$\omega_1^2 + \overline{\omega}_1^2$	$\omega_1^3 + \overline{\omega}_1^3$	$\cdots$	$\omega_1^{v-1} + \overline{\omega}_1^{v-1}$	$\omega_1^v + \overline{\omega}_1^v$	0	0
χ_6	2	$\omega_2 + \overline{\omega}_2$	$\omega_2^2 + \overline{\omega}_2^2$	$\omega_2^3 + \overline{\omega}_2^3$	$\cdots$	$\omega_2^{v-1} + \overline{\omega}_2^{v-1}$	$\omega_2^v + \overline{\omega}_2^v$	0	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
χ_{v+3}	2	$\omega_{v-1} + \overline{\omega}_{v-1}$	$\omega_{v-1}^2 + \overline{\omega}_{v-1}^2$	$\omega_{v-1}^3 + \overline{\omega}_{v-1}^3$	$\cdots$	$\omega_{v-1}^{v-1} + \overline{\omega}_{v-1}^{v-1}$	$\omega_{v-1}^v + \overline{\omega}_{v-1}^v$	0	0

Table 4.7: **ordinary character table of D_{4v} for odd v**

We set $Q_1 := \langle 1 \rangle, Q_2 := \langle sr^v \rangle, Q_3 := \langle r^v \rangle, Q_4 := \langle s \rangle$, and $Q_5 := \langle s, r^v \rangle \in \mathrm{Syl}_2(G)$. Furthermore, we choose $\mathscr{S}_2(D_{4v}) = \{Q_1, Q_2, Q_3, Q_4, Q_5\}$. Then, the lattice of subgroups in $\mathscr{S}_2(D_{4v})$ is as given in Fig. 4.5.

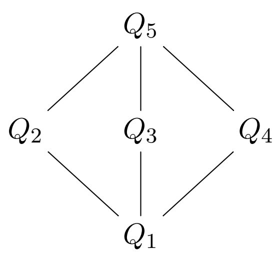


Figure 4.5: the lattice of subgroups in $\mathscr{S}_2(D_{4v})$

Moreover,

$$\begin{aligned}
N_G(Q_1) &= G \quad \text{and } \overline{N}_G(Q_1) \cong G;\\
N_G(Q_2) &= Q_5 \text{ and } \overline{N}_G(Q_2) \cong C_2;\\
N_G(Q_3) &= G \quad \text{and } \overline{N}_G(Q_3) \cong D_{2v};\\
N_G(Q_4) &= Q_5 \text{ and } \overline{N}_G(Q_4) \cong C_2;\\
N_G(Q_5) &= Q_5 \text{ and } \overline{N}_G(Q_5) \cong Q_1.
\end{aligned}$$

We denote the $\frac{v-1}{2}$ simple kG-modules by $S_1 := k, S_2, \ldots, S_{\frac{v-1}{2}}$.

Proposition 4.3.1. (a) *The decomposition matrix* $\mathfrak{D}(B_0(kG))$ *is equal to*

	$1_{G_{p'}}$
χ_1	1
χ_2	1
χ_3	1
χ_4	1

(b) *The decomposition matrix of each non-principal block of* kG *is equal to*

	φ_{S_γ}
χ_α	1
χ_β	1

for suitable $\alpha, \beta \in \{5, \ldots, v+3\}$, $\gamma \in \{2, \ldots, \frac{v+1}{2}\}$.

(c) *The projective indecomposable characters of* kG *are given as follows:*

$$\Phi_1 := \chi_1 + \chi_2 + \chi_3 + \chi_4, \ \Phi_j := \chi_{4+(j-1)} + \chi_{4+(v-(j-1))}, \ \textit{for } 2 \leq j \leq 1 + \frac{v-1}{2}.$$

Proof. Note that $Z(N_G(Q_5)) = Z(Q_5) = Q_5$. Thus, G is p-nilpotent by Burnside's transfer theorem, see [CR90, (13.20) Burnside's Transfer Theorem]. The group G has a normal p-complement if and only if for every simple kG-module S, the composition factors of the projective cover $P(S)$ of S are all isomorphic to S. See [Web16, Theorem 8.4.1]. Hence, the Cartan matrix of kG is diagonal.

As $B_0(kG)$ has the Sylow p-subgroups of G as vertices, we deduce that $D(B_0(kG)) = Q_5 \cong V_4$. Now, by [Lin18b, Theorem 12.1.1], there exists a splendid Morita equivalence between $B_0(kG)$ and either kV_4 or $k\mathfrak{A}_4$ or $B_0(k\mathfrak{A}_5)$. Since the Cartan matrix of kG is diagonal, the only possibility for the decomposition matrix of kG is as stated in part (a).

We see that $\chi_i^\circ(g) = \varphi_1(g)$ for each $i \in \{1,2,3,4\}$ and for every $g \in G_{p'}$. Consequently, we have $\mathrm{Irr}_K(B_0(kG)) = \{\chi_1, \chi_2, \chi_3, \chi_4\}$. In the following we prove that every non-principal block $B_j \in \mathrm{Bl}(kG)$ has defect groups isomorphic to C_2. If $D(B_j) = \langle 1 \rangle$ for some $j \in \{1, \ldots, |\mathrm{Bl}(kG)| - 1\}$, then we would have $|\mathrm{Irr}_K(B_j)| = 1$. Moreover, by [Web16, Theorem 9.6.1], the integer 4 would be a divisor of the degree of the character belonging to $\mathrm{Irr}_K(B_j)$. This cannot be true, since the degree of every ordinary irreducible character of G is less than 4.

If $D(B_i) \cong C_2$ for some $i \in \{1, \dots, |\mathrm{Bl}(kG)| - 1\}$, then the decomposition matrix $\mathfrak{D}(B_i)$ is given as stated in Part (b) due to the following. By [Lin18b, Theorem 11.1.15], each entry of $\mathfrak{D}(B_i)$ is either equal to 0 or equal to 1. Moreover, by [Lin18b, Theorem 11.1.12], we have

$$|\mathrm{Irr}_K(B_i)| = e + \frac{|D(B_i)| - 1}{e}$$

for some positive integer e. Hence, $|\mathrm{Irr}_K(B_i)| = e + \frac{1}{e}$ and, therefore, $e = 1$.

Since

$$\sum_{B \in \mathrm{Bl}(kG)} |\mathrm{IBr}_p(B)| = |\{2' - \text{conjugacy classes of } G\}| = \frac{v+1}{2} \text{ and}$$

$$\sum_{B \in \mathrm{Bl}(kG)} |\mathrm{Irr}_K(B)| = |\{\text{conjugacy classes of } G\}| = v + 3,$$

it follows that all blocks of kG except for the principal block have defect groups isomorphic to C_2. Consequently, if

$$\chi_a^\circ = \chi_b^\circ \text{ for } \chi_a, \chi_b \in \mathrm{Irr}_K(G) \setminus \mathrm{Lin}(G),$$

then χ_a and χ_b belong to the same block. The decomposition matrix $\mathfrak{D}(kG)$ is given as follows:

	φ_{S_1}	φ_{S_2}	φ_{S_3}	$\cdots$
χ_1	1	0	0	$\cdots$
χ_2	1	0	0	$\cdots$
χ_3	1	0	0	$\cdots$
χ_4	1	0	0	$\cdots$
χ_5	0	1	0	$\cdots$
χ_{v+3}	0	1	0	$\cdots$
χ_6	0	0	1	$\cdots$
χ_{v+2}	0	0	1	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

It follows from $\mathrm{X}(G)$ that $1 \leq m \leq v - 1$.
Fix $m_1 \in \{1, \dots, \frac{v-1}{2}\}$. Define $m_2 := v - m_1$. Then, $\omega_{m_1}^j + \overline{\omega}_{m_1}^j = \omega_{m_2}^j + \overline{\omega}_{m_2}^j$ for every even number $j \in \{1, \dots, v\}$, since

$$\begin{aligned}
&e^{\frac{-2m_1\pi ij}{n}} + e^{\frac{2m_1\pi ij}{n}} = e^{\frac{-2(v-m_1)\pi ij}{n}} + e^{\frac{2(v-m_1)\pi ij}{n}} \\
\Leftrightarrow\ & 2 \cdot \cos(\frac{2m_1\pi j}{n}) = 2 \cdot \cos(\frac{2(v-m_1)\pi j}{n}) \\
\Leftrightarrow\ & \cos(\frac{m_1\pi j}{v}) = \cos(\frac{(v-m_1)\pi j}{v}) \\
\Leftrightarrow\ & \cos(\frac{m_1\pi j}{v}) = \cos(\pi j - \frac{m_1\pi j}{v}).
\end{aligned}$$

Hence, $\chi_{4+m_1}^\circ = \chi_{4+m_2}^\circ$ for every choice of m_1.

Therefore, we get a disjoint subdivision of the set $\{1, \dots, v\}$ into pairs and have verified the labelling of the decomposition matrix $\mathfrak{D}(kG)$. This completes the proof. □

Proposition 4.3.2. (a) *The* $\frac{v+1}{2}$ $2'$*-conjugacy classes of* $H := D_{2n}/Q_3 \cong D_{2v}$ *are given by* $[\,\overline{1}\,], [\overline{r^2}], [\overline{r^4}], \ldots, [\overline{r^{v-1}}]$.

(b) *The ordinary characters of the projective indecomposable kH-modules are given as follows:*

(i) *the projective character* $\widetilde{\Phi}_1 := \lambda_1 + \lambda_2$ *where* $\lambda_1, \lambda_2 \in Lin(H)$*;*

(ii) *all* $\frac{v-1}{2}$ *non-linear* $\chi_i \in \mathrm{Irr}_K(H)$*.*

(c) *The ordinary characters of the trivial source kG-modules with vertex* Q_3 *are given as follows:*

(i) *the character* $\chi_1 + \chi_3$*;*

(ii) *the* $\frac{v-1}{2}$ *ordinary irreducible characters* $\chi_{4+1}, \chi_{4+3}, \ldots, \chi_{4+(v-4)}, \chi_{4+(v-2)}$.

(d) *After a suitable relabelling of the rows and columns of* $\mathrm{Triv}_p(G)$ *we have* $T_{3,3} = T_{3,1}$ *and* $T_{3,1} = \frac{1}{2} \cdot T_{1,1}$.

(e) *We have* $\mathrm{TS}(G; Q_2) = \{M_1\}, \mathrm{TS}(G; Q_4) = \{M_2\}$*, and* $\mathrm{TS}(G; Q_5) = \{M_3\}$ *for suitable trivial source kG-modules* M_1, M_2*, and* M_3*. Moreover, the following assertions hold.*

(i) *We have* $\chi_{\widehat{M_1}} = \chi_1 + \chi_4$.

(ii) *We have* $\chi_{\widehat{M_2}} = \chi_1 + \chi_2$.

(iii) *We have* $\chi_{\widehat{M_3}} = \chi_1$.

Proof. In H the following conjugacy classes of G fuse:

- $[1]$ and $[r^v]$,
- $[r]$ and $[r^{v-1}]$,
- $[r^2]$ and $[r^{v-2}]$,

  $\vdots$
- $[r^{\frac{v-1}{2}}]$ and $[r^{\frac{v+1}{2}}]$,
- $[s]$ and $[sr]$.

Hence, H has $2+\frac{v-1}{2}$ conjugacy classes. Since s is a 2-element, this implies part (a). Note that $C_2 \cong D(B_0(kH)) \in \mathrm{Syl}_2(H)$. Hence, $D(B_0(kH))$ is given as follows:

	$1_{H_{p'}}$
λ_1	1
λ_2	1

Moreover, since H is p-solvable, the decomposition matrix of kH contains an identity matrix of maximum possible size. Hence, (b) follows. The trivial source kG-modules with vertices isomorphic to Q_3 are obtained by inflation of the projective indecomposable kH-modules. The same is true for their ordinary characters. We examine which $\chi_t \in \mathrm{Irr}_K(G)$ have Q_3 in their kernel. Since

$$\omega_m^v + \overline{\omega}_m^v = e^{\frac{-2m\pi iv}{n}} + e^{\frac{2m\pi iv}{n}} = 2 \cdot \cos(\frac{2m\pi v}{n}) = 2\cos(m\pi),$$

we deduce that Q_3 is in the kernel of χ_t if and only if $t \in \{1, \dots, v+3\}$ is odd. This proves (c). The first assertion in part (d) follows from part (a) and the fact that inflation of characters from H to G does not change their values at conjugacy classes. By Proposition 4.3.1, for each projective characters of kG the number of constituents with odd indices is equal to the number of constituents with even indices. This proves the second claim of (d). Recall that all the ordinary characters stated in part (e) have to occur as trivial source characters of kG due to Proposition 4.2.10. Note that

$$(\chi_1 + \chi_4)(sr^v) = (\chi_1 + \chi_4)(sr) = 2 > 0 \text{ and}$$

$$(\chi_1 + \chi_2)(s) = 2 > 0.$$

The claim follows now from Lemma 3.1.6, as the number of p'-conjugacy classes of $\overline{N}_G(Q_i)$ is equal to 1 for $i \in \{2, 4, 5\}$. □

Theorem 4.3.3. *Labelling the ordinary characters as in Table 4.7, the trivial source character table of D_{4v} at $p = 2$ is as given in Table 4.8:*

Proof. The assertion follows from Proposition 4.3.1, Proposition 4.3.2 and Remark 3.2.6. □

Q_v	$Q_1 \cong C_1$						$Q_2 \cong C_2$	$Q_3 \cong C_2$						$Q_4 \cong C_2$	$Q_5 \cong V_4$
N_v	$N_1 \cong D_{4v}$						$N_2 \cong V_4$	$N_3 \cong D_{4v}$						$N_4 \cong V_4$	$N_5 \cong V_4$
$n_j \in N_v$	1	r^2	r^4	r^6	$\cdots$	r^{v-1}	1	1	r^2	r^4	$\cdots$	r^{v-1}	1	1
$\chi_1+\chi_2+\chi_3+\chi_4$	4	4	4	4	$\cdots$	4	0	0	0	0	$\cdots$	0	0	0
$\chi_{4+1}+\chi_{4+(v-1)}$	4	$2\left(\omega_1^2+\overline{\omega}_1^2\right)$	$2\left(\omega_1^4+\overline{\omega}_1^4\right)$	$2\left(\omega_1^6+\overline{\omega}_1^6\right)$	$\cdots$	$2\left(\omega_1^{v-1}+\overline{\omega}_1^{v-1}\right)$	0	0	0	0	$\cdots$	0	0	0
$\chi_{4+2}+\chi_{4+(v-2)}$	4	$2\left(\omega_2^2+\overline{\omega}_2^2\right)$	$2\left(\omega_2^4+\overline{\omega}_2^4\right)$	$2\left(\omega_2^6+\overline{\omega}_2^6\right)$	$\cdots$	$2\left(\omega_2^{v-1}+\overline{\omega}_2^{v-1}\right)$	0	0	0	0	$\cdots$	0	0	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\vdots$	$\vdots$
$\chi_{4+\frac{v-1}{2}}+\chi_{4+\frac{v+1}{2}}$	4	$2\left(\omega_{\frac{v-1}{2}}^2+\overline{\omega}_{\frac{v-1}{2}}^2\right)$	$2\left(\omega_{\frac{v-1}{2}}^4+\overline{\omega}_{\frac{v-1}{2}}^4\right)$	$2\left(\omega_{\frac{v-1}{2}}^6+\overline{\omega}_{\frac{v-1}{2}}^6\right)$	$\cdots$	$2\left(\omega_{\frac{v-1}{2}}^{v-1}+\overline{\omega}_{\frac{v-1}{2}}^{v-1}\right)$	0	0	0	0	$\cdots$	0	0	0
$\chi_1+\chi_4$	2	2	2	2	$\cdots$	2	2	0	0	0	$\cdots$	0	0	0
$\chi_1+\chi_3$	2	2	2	2	$\cdots$	2	0	2	2	2	$\cdots$	2	0	0
χ_{4+1}	2	$\omega_1^2+\overline{\omega}_1^2$	$\omega_1^4+\overline{\omega}_1^4$	$\omega_1^6+\overline{\omega}_1^6$	$\cdots$	$\omega_1^{v-1}+\overline{\omega}_1^{v-1}$	0	2	$\omega_1^2+\overline{\omega}_1^2$	$\omega_1^4+\overline{\omega}_1^4$	$\cdots$	$\omega_1^{v-1}+\overline{\omega}_1^{v-1}$	0	0
$\chi_{4+(v-2)}$	2	$\omega_{v-2}^2+\overline{\omega}_{v-2}^2$	$\omega_{v-2}^4+\overline{\omega}_{v-2}^4$	$\omega_{v-2}^6+\overline{\omega}_{v-2}^6$	$\cdots$	$\omega_{v-2}^{v-1}+\overline{\omega}_{v-2}^{v-1}$	0	2	$\omega_{v-2}^2+\overline{\omega}_{v-2}^2$	$\omega_{v-2}^4+\overline{\omega}_{v-2}^4$	$\cdots$	$\omega_{v-2}^{v-1}+\overline{\omega}_{v-2}^{v-1}$	0	0
χ_{4+3}	2	$\omega_3^2+\overline{\omega}_3^2$	$\omega_3^4+\overline{\omega}_3^4$	$\omega_3^6+\overline{\omega}_3^6$	$\cdots$	$\omega_3^{v-1}+\overline{\omega}_3^{v-1}$	0	2	$\omega_3^2+\overline{\omega}_3^2$	$\omega_3^4+\overline{\omega}_3^4$	$\cdots$	$\omega_3^{v-1}+\overline{\omega}_3^{v-1}$	0	0
$\chi_{4+(v-4)}$	2	$\omega_{v-4}^2+\overline{\omega}_{v-4}^2$	$\omega_{v-4}^4+\overline{\omega}_{v-4}^4$	$\omega_{v-4}^6+\overline{\omega}_{v-4}^6$	$\cdots$	$\omega_{v-4}^{v-1}+\overline{\omega}_{v-4}^{v-1}$	0	2	$\omega_{v-4}^2+\overline{\omega}_{v-4}^2$	$\omega_{v-4}^4+\overline{\omega}_{v-4}^4$	$\cdots$	$\omega_{v-4}^{v-1}+\overline{\omega}_{v-4}^{v-1}$	0	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\vdots$	$\vdots$
$\chi_1+\chi_2$	2	2	2	2	$\cdots$	2	2	0	0	0	$\cdots$	0	2	0
χ_1	1	1	1	1	$\cdots$	1	1	1	1	1	$\cdots$	1	1	1

Table 4.8: **trivial source character table of** D_{4v} at $p = 2$ for odd v

4.3.2 The groups $\mathrm{SL}_2(11)$ and $\mathrm{PSL}_2(11)$

In this section, we consider the (projective) special linear groups $\mathrm{SL}_2(11)$ and $\mathrm{PSL}_2(11)$. These groups provide a nice example of how it is possible to use inflation of modules and characters in order to obtain trivial source modules and trivial source characters of new groups. Following [LP10], we choose the ordering $0 < 1 < \ldots < p-1$ in $\mathbb{F}_p$ and let $\mathsf{Z(p)}$ denote the smallest primitive generator of $\mathbb{F}_p^\times$. Moreover, by $\zeta_{p,n} := \mathsf{Z(p^n)}$ we denote a certain root of the n-th Conway polynomial of characteristic p (for $n \in \mathbb{Z}_{\geq 1}$). We refer to Section 5.2.1 for more details.
We consider the special linear group

$$G := \mathrm{SL}_2(11) := \left\{ \begin{pmatrix} a & b \\ c & d \end{pmatrix} \mid ad - bc = 1 \right\} \leq \mathrm{GL}_2(\mathbb{F}_{11})$$

with $|G| = (11-1) \cdot 11 \cdot (11+1) = 1320$. We choose the following representatives of the 15 conjugacy classes of G:

$$\begin{aligned}
\mathrm{I}_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, &\quad -\mathrm{I}_2, \\
\begin{pmatrix} \zeta_{11,1} & 0 \\ 0 & \zeta_{11,9} \end{pmatrix}^i &\quad (1 \leq i \leq 4), \\
\begin{pmatrix} \zeta_{11,8} & \zeta_{11,2} \\ \zeta_{11,1} & \zeta_{11,8} \end{pmatrix}^j &\quad (1 \leq j \leq 5), \\
\mathrm{u}_+ := \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, &\quad -\mathrm{u}_+, \\
\mathrm{u}_- := \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}, &\quad -\mathrm{u}_-.
\end{aligned}$$

The ordinary character table of G is as given in Table 4.9. We set

$$\begin{aligned}
&Q_1 := \langle 1 \rangle, \\
&Q_2 := \langle -\mathrm{I}_2 \rangle = Z(G) =: Z \cong C_2 \\
&Q_3 := \left\langle \begin{pmatrix} 0 & \zeta_{11,3} \\ \zeta_{11,2} & 0 \end{pmatrix} \right\rangle \cong C_4, \text{ and} \\
&Q_4 := \left\langle \begin{pmatrix} 0 & \zeta_{11,3} \\ \zeta_{11,2} & 0 \end{pmatrix}, \begin{pmatrix} 1 & \zeta_{11,6} \\ 1 & \zeta_{11,5} \end{pmatrix} \right\rangle \cong \mathcal{Q}_8.
\end{aligned}$$

Furthermore, we choose $\mathscr{S}_2(\mathrm{SL}_2(11)) = \{Q_1, Q_2, Q_3, Q_4\}$. Then, the lattice of subgroups in $\mathscr{S}_2(\mathrm{SL}_2(11))$ is as given in Fig. 4.6.

g	I_2	$-\mathrm{I}_2$	$\begin{pmatrix}\zeta_{11,1} & 0 \\ 0 & \zeta_{11,9}\end{pmatrix}$	$\begin{pmatrix}\zeta_{11,1} & 0 \\ 0 & \zeta_{11,9}\end{pmatrix}^2$	$\begin{pmatrix}\zeta_{11,1} & 0 \\ 0 & \zeta_{11,9}\end{pmatrix}^3$	$\begin{pmatrix}\zeta_{11,1} & 0 \\ 0 & \zeta_{11,9}\end{pmatrix}^4$	$\begin{pmatrix}\zeta_{11,8} & \zeta_{11,2} \\ \zeta_{11,1} & \zeta_{11,8}\end{pmatrix}$	$\begin{pmatrix}\zeta_{11,8} & \zeta_{11,2} \\ \zeta_{11,1} & \zeta_{11,8}\end{pmatrix}^2$	$\begin{pmatrix}\zeta_{11,8} & \zeta_{11,2} \\ \zeta_{11,1} & \zeta_{11,8}\end{pmatrix}^3$	$\begin{pmatrix}\zeta_{11,8} & \zeta_{11,2} \\ \zeta_{11,1} & \zeta_{11,8}\end{pmatrix}^4$	$\begin{pmatrix}\zeta_{11,8} & \zeta_{11,2} \\ \zeta_{11,1} & \zeta_{11,8}\end{pmatrix}^5$	u_+	$-u_+$	u_-	$-u_-$
χ_1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
χ_2	11	11	1	1	1	1	-1	-1	-1	-1	-1	0	0	0	0
χ_3	12	12	$\frac{-1+\sqrt{5}}{2}$	$\frac{-1-\sqrt{5}}{2}$	$\frac{-1-\sqrt{5}}{2}$	$\frac{-1+\sqrt{5}}{2}$	0	0	0	0	0	1	1	1	1
χ_4	12	12	$\frac{-1-\sqrt{5}}{2}$	$\frac{-1+\sqrt{5}}{2}$	$\frac{-1+\sqrt{5}}{2}$	$\frac{-1-\sqrt{5}}{2}$	0	0	0	0	0	1	1	1	1
χ_5	12	-12	$-\frac{-1+\sqrt{5}}{2}$	$\frac{-1-\sqrt{5}}{2}$	$-\frac{-1-\sqrt{5}}{2}$	$\frac{-1+\sqrt{5}}{2}$	0	0	0	0	0	1	-1	1	-1
χ_6	12	-12	$-\frac{-1-\sqrt{5}}{2}$	$\frac{-1+\sqrt{5}}{2}$	$-\frac{-1+\sqrt{5}}{2}$	$\frac{-1-\sqrt{5}}{2}$	0	0	0	0	0	1	-1	1	-1
χ_7	10	10	0	0	0	0	1	1	-2	1	1	-1	-1	-1	-1
χ_8	10	10	0	0	0	0	-1	1	2	1	-1	-1	-1	-1	-1
χ_9	10	-10	0	0	0	0	0	2	0	-2	0	-1	1	-1	1
χ_{10}	10	-10	0	0	0	0	$\sqrt{3}$	-1	0	1	$-\sqrt{3}$	-1	1	-1	1
χ_{11}	10	-10	0	0	0	0	$-\sqrt{3}$	-1	0	1	$\sqrt{3}$	-1	1	-1	1
χ_{12}	6	-6	-1	1	-1	1	0	0	0	0	0	$-\frac{-1+\sqrt{11}i}{2}$	$\frac{-1+\sqrt{11}i}{2}$	$-\frac{-1-\sqrt{11}i}{2}$	$\frac{-1-\sqrt{11}i}{2}$
χ_{13}	6	-6	-1	1	-1	1	0	0	0	0	0	$-\frac{-1-\sqrt{11}i}{2}$	$\frac{-1-\sqrt{11}i}{2}$	$-\frac{-1+\sqrt{11}i}{2}$	$\frac{-1+\sqrt{11}i}{2}$
χ_{14}	5	5	0	0	0	0	1	-1	1	-1	1	$\frac{-1+\sqrt{11}i}{2}$	$\frac{-1+\sqrt{11}i}{2}$	$\frac{-1-\sqrt{11}i}{2}$	$\frac{-1-\sqrt{11}i}{2}$
χ_{15}	5	5	0	0	0	0	1	-1	1	-1	1	$\frac{-1-\sqrt{11}i}{2}$	$\frac{-1-\sqrt{11}i}{2}$	$\frac{-1+\sqrt{11}i}{2}$	$\frac{-1+\sqrt{11}i}{2}$

Table 4.9: **ordinary character table of** $\mathrm{SL}_2(11)$

Figure 4.6: the lattice of subgroups in $\mathscr{S}_2(\mathrm{SL}_2(11))$

Moreover,

$$\begin{aligned}
&N_G(Q_1) = G && \text{and } \overline{N}_G(Q_1) \cong G;\\
&N_G(Q_2) = G && \text{and } \overline{N}_G(Q_2) = G/Z \cong \mathrm{PSL}_2(11);\\
&N_G(Q_3) = \left\langle \begin{pmatrix} \zeta_{11,8} & \zeta_{11,2} \\ \zeta_{11,1} & \zeta_{11,8} \end{pmatrix}, \begin{pmatrix} 1 & \zeta_{11,6} \\ 1 & \zeta_{11,5} \end{pmatrix} \right\rangle \cong C_3 \rtimes \mathcal{Q}_8 && \text{and } \overline{N}_G(Q_3) \cong \mathfrak{S}_3;\\
&N_G(Q_4) = \left\langle Q_4, \begin{pmatrix} \zeta_{11,9} & \zeta_{11,6} \\ \zeta_{11,4} & \zeta_{11,2} \end{pmatrix} \right\rangle \cong \mathrm{SL}_2(3) && \text{and } \overline{N}_G(Q_4) \cong C_3.
\end{aligned}$$

Next, we consider the group $\overline{G} := G/Z \cong \mathrm{PSL}_2(11)$. We identify $\overline{G}$ with $\mathrm{PSL}_2(11)$. We set

$$\begin{aligned}
\overline{Q_2} &:= \langle 1 \rangle,\\
\overline{Q_3} &:= \left\langle \begin{pmatrix} 0 & \zeta_{11,3} \\ \zeta_{11,2} & 0 \end{pmatrix} Z \right\rangle \cong C_2, \text{ and}\\
\overline{Q_4} &:= \left\langle \begin{pmatrix} 0 & \zeta_{11,3} \\ \zeta_{11,2} & 0 \end{pmatrix} Z, \begin{pmatrix} 1 & \zeta_{11,6} \\ 1 & \zeta_{11,5} \end{pmatrix} Z \right\rangle \cong C_2 \times C_2.
\end{aligned}$$

Moreover, we fix $\mathscr{S}_2(\mathrm{PSL}_2(11)) = \{\overline{Q_2}, \overline{Q_3}, \overline{Q_4}\}$. Then, the lattice of subgroups in $\mathscr{S}_2(\overline{G})$ is as given in Fig. 4.7.

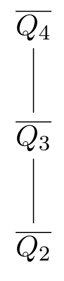


Figure 4.7: the lattice of subgroups in $\mathrm{PSL}_2(11)$

Furthermore,

$$N_{\overline{G}}(\overline{Q_2}) = \overline{G} \qquad \text{and } \overline{N}_{\overline{G}}(\overline{Q_2}) \cong \overline{G};$$
$$N_{\overline{G}}(\overline{Q_3}) = \left\langle \begin{pmatrix} \zeta_{11,8} & \zeta_{11,2} \\ \zeta_{11,1} & \zeta_{11,8} \end{pmatrix} Z, \begin{pmatrix} 1 & \zeta_{11,6} \\ 1 & \zeta_{11,5} \end{pmatrix} Z \right\rangle \cong D_{12} \qquad \text{and } \overline{N}_{\overline{G}}(\overline{Q_3}) \cong D_6;$$
$$N_{\overline{G}}(\overline{Q_4}) = \left\langle \overline{Q_4}, \begin{pmatrix} \zeta_{11,9} & \zeta_{11,6} \\ \zeta_{11,4} & \zeta_{11,2} \end{pmatrix} Z \right\rangle \cong \mathfrak{A}_4 \qquad \text{and } \overline{N}_{\overline{G}}(\overline{Q_4}) \cong C_3.$$

Notation 4.3.4. We choose the following sets of representatives of the $2'$-conjugacy classes of the groups occurring in $\mathscr{S}_2(G)$ and $\mathscr{S}_2(\overline{G})$, respectively, where the bar notation denotes left cosets in the respective quotients $\overline{N}_{\overline{G}}(\overline{Q_i})$ for $2 \leq i \leq 4$:

$$[\overline{N}_G(Q_1)]_{2'} := \{I_2\} \cup \left\{ \begin{pmatrix} \zeta_{11,1} & 0 \\ 0 & \zeta_{11,9} \end{pmatrix}^2, \begin{pmatrix} \zeta_{11,1} & 0 \\ 0 & \zeta_{11,9} \end{pmatrix}^4 \right\} \cup \left\{ \begin{pmatrix} \zeta_{11,4} & \zeta_{11,1} \\ 1 & \zeta_{11,4} \end{pmatrix} \right\} \cup \{u_+, u_-\};$$
$$[\overline{N}_G(Q_2)]_{2'} := \{I_2 Q_2\} \cup \left\{ \begin{pmatrix} \zeta_{11,1} & 0 \\ 0 & \zeta_{11,9} \end{pmatrix}^2 Q_2, \begin{pmatrix} \zeta_{11,1} & 0 \\ 0 & \zeta_{11,9} \end{pmatrix}^4 Q_2 \right\} \cup \left\{ \begin{pmatrix} \zeta_{11,4} & \zeta_{11,1} \\ 1 & \zeta_{11,4} \end{pmatrix} Q_2 \right\} \cup \{u_+ Q_2, u_- Q_2\};$$
$$[\overline{N}_G(Q_3)]_{2'} := \{I_2 Q_3\} \cup \left\{ \begin{pmatrix} \zeta_{11,4} & \zeta_{11,1} \\ 1 & \zeta_{11,4} \end{pmatrix} \right\};$$
$$[\overline{N}_G(Q_4)]_{2'} := \left\{ I_2 Q_4, \begin{pmatrix} \zeta_{11,9} & \zeta_{11,6} \\ \zeta_{11,4} & \zeta_{11,2} \end{pmatrix} Q_4, \begin{pmatrix} \zeta_{11,9} & \zeta_{11,6} \\ \zeta_{11,4} & \zeta_{11,2} \end{pmatrix}^2 Q_4 \right\};$$
$$[\overline{N}_{\overline{G}}(\overline{Q_2})]_{2'} := \left\{ \overline{I_2 Z} \right\} \cup \left\{ \overline{\begin{pmatrix} \zeta_{11,1} & 0 \\ 0 & \zeta_{11,9} \end{pmatrix}^2 Z}, \overline{\begin{pmatrix} \zeta_{11,1} & 0 \\ 0 & \zeta_{11,9} \end{pmatrix}^4 Z} \right\} \cup \left\{ \overline{\begin{pmatrix} \zeta_{11,4} & \zeta_{11,1} \\ 1 & \zeta_{11,4} \end{pmatrix} Z} \right\} \cup \left\{ \overline{u_+ Z}, \overline{u_- Z} \right\};$$
$$[\overline{N}_{\overline{G}}(\overline{Q_3})]_{2'} := \left\{ \overline{I_2 Z} \right\} \cup \left\{ \overline{\begin{pmatrix} \zeta_{11,4} & \zeta_{11,1} \\ 1 & \zeta_{11,4} \end{pmatrix} Z} \right\};$$
$$[\overline{N}_{\overline{G}}(\overline{Q_4})]_{2'} := \left\{ \overline{I_2 Z}, \overline{\begin{pmatrix} \zeta_{11,9} & \zeta_{11,6} \\ \zeta_{11,4} & \zeta_{11,2} \end{pmatrix} Z}, \overline{\begin{pmatrix} \zeta_{11,9} & \zeta_{11,6} \\ \zeta_{11,4} & \zeta_{11,2} \end{pmatrix}^2 Z} \right\}.$$

We identify the ordinary characters of $\overline{G}$ with the ordinary characters of G with the centre Z in their kernel. We label the ordinary characters and the 2-blocks of $\mathrm{PSL}_2(11)$ using the corresponding labelling in $\mathrm{SL}_2(11)$. Consequently, the ordinary character table of $\overline{G}$ is given in Table 4.10.

g	I_2Z	$\begin{pmatrix} \zeta_{11,1} & 0 \\ 0 & \zeta_{11,9} \end{pmatrix}^2 Z$	$\begin{pmatrix} \zeta_{11,1} & 0 \\ 0 & \zeta_{11,9} \end{pmatrix}^4 Z$	$\begin{pmatrix} \zeta_{11,8} & \zeta_{11,2} \\ \zeta_{11,1} & \zeta_{11,8} \end{pmatrix} Z$	$\begin{pmatrix} \zeta_{11,8} & \zeta_{11,2} \\ \zeta_{11,1} & \zeta_{11,8} \end{pmatrix}^3 Z$	$\begin{pmatrix} \zeta_{11,8} & \zeta_{11,2} \\ \zeta_{11,1} & \zeta_{11,8} \end{pmatrix}^4 Z$	u_+Z	u_-Z
χ_1	1	1	1	1	1	1	1	1
χ_2	11	1	1	-1	-1	-1	0	0
χ_3	12	$\frac{-1-\sqrt{5}}{2}$	$\frac{-1+\sqrt{5}}{2}$	0	0	0	1	1
χ_4	12	$\frac{-1+\sqrt{5}}{2}$	$\frac{-1-\sqrt{5}}{2}$	0	0	0	1	1
χ_7	10	0	0	1	-2	1	-1	-1
χ_8	10	0	0	-1	2	1	-1	-1
χ_{14}	5	0	0	1	1	-1	$\frac{-1+\sqrt{11}i}{2}$	$\frac{-1-\sqrt{11}i}{2}$
χ_{15}	5	0	0	1	1	-1	$\frac{-1-\sqrt{11}i}{2}$	$\frac{-1+\sqrt{11}i}{2}$

Table 4.10: **ordinary character table of** $\mathrm{PSL}_2(11)$

Notation 4.3.5. (a) We denote the simple kG-modules by S_i $(1 \leq i \leq 6)$. Moreover, we set $\mathrm{IBr}_2(G) = \{\varphi_1, \varphi_2, \varphi_3, \varphi_4, \varphi_5, \varphi_6\}$ where $\varphi_i := \varphi_{S_i}$ for $1 \leq i \leq 6$.

(b) We denote the simple $k\overline{G}$-modules by T_i $(1 \leq i \leq 6)$. Moreover, we set $\mathrm{IBr}_2(\overline{G}) = \{\phi_1, \phi_2, \phi_3, \phi_4, \phi_5, \phi_6\}$ where $\phi_i := \phi_{T_i}$ for $1 \leq i \leq 6$.

Proposition 4.3.6. (a) *The decomposition matrix* $\mathfrak{D}(kG)$ *is equal to*

	φ_1	φ_2	φ_3	φ_4	φ_5	φ_6
χ_1	1	0	0	0	0	0
χ_2	1	1	1	0	0	0
χ_3	0	0	0	0	0	1
χ_4	0	0	0	0	1	0
χ_5	0	0	0	0	0	1
χ_6	0	0	0	0	1	0
χ_7	0	0	0	1	0	0
χ_8	0	0	0	1	0	0
χ_9	0	1	1	0	0	0
χ_{10}	0	0	0	1	0	0
χ_{11}	0	0	0	1	0	0
χ_{12}	1	0	1	0	0	0
χ_{13}	1	1	0	0	0	0
χ_{14}	0	1	0	0	0	0
χ_{15}	0	0	1	0	0	0

.

(b) *The ordinary irreducible characters of* G *split into the following four blocks of* kG*:*

$$\mathrm{Irr}_K(B_0(kG)) = \{\chi_1, \chi_2, \chi_9, \chi_{12}, \chi_{13}, \chi_{14}, \chi_{15}\},$$

$$\mathrm{Irr}_K(B_1(kG)) = \{\chi_7, \chi_8, \chi_{10}, \chi_{11}\},$$

$$\mathrm{Irr}_K(B_2(kG)) = \{\chi_4, \chi_6\}, \text{ and}$$

$$\mathrm{Irr}_K(B_3(kG)) = \{\chi_3, \chi_5\}.$$

(c) *We have* $d(B_0(kG)) = 3$, $d(B_1(kG)) = 2$, $d(B_2(kG)) = 1$, *and* $d(B_3(kG)) = 1$.

(d) *We have*

$$D(B_0(kG)) \cong \mathcal{Q}_8,\ D(B_1(kG)) \cong C_4,\ D(B_2(kG)) \cong C_2,\ \text{and } D(B_3(kG)) \cong C_2.$$

Proof. The assertions in (a), (b), and (c) follow from [WTP+98, L2(11)mod2.pdf]. It remains to prove part (d). The principal block of kG has the Sylow 2-subgroups as vertices. As the group $\mathcal{Q}_8$ does not have any subgroup isomorphic to $C_2 \times C_2$, part (d) follows immediately from part (c). □

Next, we use Proposition 4.3.6 in order to derive the analogous pieces of information for the group $\overline{G}$.

Proposition 4.3.7. (a) *The decomposition matrix* $\mathfrak{D}(k\overline{G})$ *is equal to*

	ϕ_1	ϕ_2	ϕ_3	ϕ_4	ϕ_5	ϕ_6
χ_1	1	0	0	0	0	0
χ_2	1	1	1	0	0	0
χ_3	0	0	0	0	0	1
χ_4	0	0	0	0	1	0
χ_7	0	0	0	1	0	0
χ_8	0	0	0	1	0	0
χ_{14}	0	1	0	0	0	0
χ_{15}	0	0	1	0	0	0

.

(b) *The ordinary irreducible characters of $\overline{G}$ belong to the following four blocks of $k\overline{G}$:*

$$\mathrm{Irr}_K(B_0(k\overline{G})) = \{\chi_1, \chi_2, \chi_{14}, \chi_{15}\},$$

$$\mathrm{Irr}_K(B_1(k\overline{G})) = \{\chi_7, \chi_8\},$$

$$\mathrm{Irr}_K(B_2(k\overline{G})) = \{\chi_4\}, \text{ and}$$

$$\mathrm{Irr}_K(B_3(k\overline{G})) = \{\chi_3\}.$$

(c) *We have $d(B_0(k\overline{G})) = 2$, $d(B_1(k\overline{G})) = 1$, $d(B_2(k\overline{G})) = 0$, and $d(B_3(k\overline{G})) = 0$.*

(d) *We have*

$$D(B_0(k\overline{G})) \cong C_2 \times C_2,\ D(B_1(k\overline{G})) \cong C_2,\ D(B_2(k\overline{G})) \cong \langle 1\rangle, \text{ and } D(B_3(k\overline{G})) \cong \langle 1\rangle.$$

Proof. Analogous to the proof of Proposition 4.3.6 □

Proposition 4.3.8. *The trivial source $k\overline{G}$-modules and their ordinary characters are as given in Table 4.11.*

Character $\chi_{\widehat{M}}$	Module M	Vertices
$\chi_1 + \chi_2$	$P(T_1)$	$\langle 1\rangle$
$\chi_2 + \chi_{14}$	$P(T_2)$	$\langle 1\rangle$
$\chi_2 + \chi_{15}$	$P(T_3)$	$\langle 1\rangle$
$\chi_7 + \chi_8$	$P(T_4)$	$\langle 1\rangle$
χ_4	$P(T_5)$	$\langle 1\rangle$
χ_3	$P(T_6)$	$\langle 1\rangle$
$\chi_1 + \chi_2 + \chi_{14} + \chi_{15}$	$\mathrm{Sc}(\overline{G}, \overline{Q_3})$	C_2
χ_8	T_4	C_2
$1_{\overline{G}}$	$k_{\overline{G}}$	$C_2 \times C_2$
χ_{14}	T_2	$C_2 \times C_2$
χ_{15}	T_3	$C_2 \times C_2$

Table 4.11: **trivial source** $k\,\mathrm{PSL}_2(11)$**-modules**

Proof. By counting the $2'$-conjugacy classes of $N_{\overline{G}}(\overline{Q_i})$, $2 \leq i \leq 4$, we deduce that

$$|\mathrm{TS}(\overline{G}, \overline{Q_2})| = 6,\ |\mathrm{TS}(\overline{G}, \overline{Q_3})| = 2, \text{ and } |\mathrm{TS}(\overline{G}, \overline{Q_4})| = 3.$$

The ordinary characters of the projective indecomposable $k\overline{G}$-modules follow from the decomposition matrix of $k\overline{G}$. As $D(B_0(k\overline{G})) \cong C_2 \times C_2$, we deduce that there exists a splendid Morita equivalence between $B_0(k\overline{G})$ and $k\mathfrak{A}_4$ since the character degrees exclude all other possibilities by Proposition 4.2.12. The same proposition implies all remaining assertions except for those concerning the trivial source modules with vertex C_2. Since the trivial source module $\mathrm{Sc}(\overline{G}, \overline{Q_3})$ belongs to the principal block of $k\overline{G}$, it only remains to prove the assertions about the module T_4 with vertex C_2. The remaining trivial source $k\overline{G}$-module does not belong to the principal block. Moreover, it is not projective. Consequently, it belongs to a block of kG with a defect group which is isomorphic to C_2. Hence, it belongs to $B_1(k\overline{G})$. We have $\mathrm{Irr}_K(B_1(k\overline{G})) = \{\chi_7, \chi_8\}$. This block is Morita equivalent to kC_2. By Lemma 4.2.5, there exist, up to isomorphism, exactly two $B_1(k\overline{G})$-modules, and they are both trivial source modules. We have already taken the projective indecomposable $B_1(k\overline{G})$-module $P(T_4)$ into account. Consequently, $T_4 \cong \mathrm{Soc}(P(T_4))$ is the remaining trivial source $B_1(k\overline{G})$-module. The ordinary character of T_4 is either equal to χ_7 or equal to χ_8. By Proposition 3.1.7 we deduce that $\chi_{\widehat{T_4}} = \chi_8$. □

Theorem 4.3.9. *Assume* $\overline{G} = \mathrm{PSL}_2(11)$. *Then, the trivial source character table* $\mathrm{Triv}_2(\overline{G})$ *of* $\overline{G}$ *is given as follows.*

(a) *We have* $T_{1,2} = T_{1,3} = T_{2,3} = \mathbf{0}$.

(b) *The matrices* $T_{i,1}$ *with* $1 \le i \le 3$ *are as given in Table 4.12.*

(c) *The matrices* $T_{2,2}$ *and* $T_{3,2}$ *are as given in Table 4.13.*

(d) *The matrix* $T_{3,3}$ *is as given in Table 4.14 where* x *denotes a third root of unity.*

	$\mathrm{I}_2 Z$	$\begin{pmatrix}\zeta_{11,1} & 0\\ 0 & \zeta_{11,9}\end{pmatrix}^2 Z$	$\begin{pmatrix}\zeta_{11,1} & 0\\ 0 & \zeta_{11,9}\end{pmatrix}^4 Z$	$\begin{pmatrix}\zeta_{11,8} & \zeta_{11,2}\\ \zeta_{11,1} & \zeta_{11,8}\end{pmatrix}^4 Z$	$u_+ Z$	$u_- Z$
$\chi_1 + \chi_2$	12	2	2	0	1	1
$\chi_2 + \chi_{14}$	16	1	1	-2	$\frac{-1+\sqrt{11}i}{2}$	$\frac{-1-\sqrt{11}i}{2}$
$\chi_2 + \chi_{15}$	16	1	1	-2	$\frac{-1-\sqrt{11}i}{2}$	$\frac{-1+\sqrt{11}i}{2}$
$\chi_7 + \chi_8$	20	0	0	2	-2	-2
χ_4	12	$\frac{-1+\sqrt{5}}{2}$	$\frac{-1-\sqrt{5}}{2}$	0	1	1
χ_3	12	$\frac{-1-\sqrt{5}}{2}$	$\frac{-1+\sqrt{5}}{2}$	0	1	1
$\chi_1 + \chi_2 + \chi_{14} + \chi_{15}$	22	2	2	-2	0	0
χ_8	10	0	0	1	-1	-1
χ_1	1	1	1	1	1	1
χ_{14}	5	0	0	-1	$\frac{-1+\sqrt{11}i}{2}$	$\frac{-1-\sqrt{11}i}{2}$
χ_{15}	5	0	0	-1	$\frac{-1-\sqrt{11}i}{2}$	$\frac{-1+\sqrt{11}i}{2}$

Table 4.12: $T_{i,1}$ **with** $1 \le i \le 3$ **for** $\overline{G} = \mathrm{PSL}_2(11)$ **and** $p = 2$

	$\mathrm{I}_2 Z$	$\begin{pmatrix}\zeta_{11,4} & \zeta_{11,1}\\ 1 & \zeta_{11,4}\end{pmatrix} Z$
$\chi_1 + \chi_2 + \chi_{14} + \chi_{15}$	2	2
χ_8	2	-1
χ_1	1	1
χ_{14}	1	1
χ_{15}	1	1

Table 4.13: $T_{2,2}$ **and** $T_{3,2}$ **for** $\overline{G} = \mathrm{PSL}_2(11)$ **and** $p = 2$

	$\mathrm{I}_2 Z$	$\begin{pmatrix}\zeta_{11,9} & \zeta_{11,6}\\ \zeta_{11,4} & \zeta_{11,2}\end{pmatrix} Z$	$\begin{pmatrix}\zeta_{11,9} & \zeta_{11,6}\\ \zeta_{11,4} & \zeta_{11,2}\end{pmatrix}^2 Z$
χ_1	1	1	1
χ_{14}	1	x	x^2
χ_{15}	1	x^2	x

Table 4.14: $T_{3,3}$ **for** $\overline{G} = \mathrm{PSL}_2(11)$ **and** $p = 2$

Proof. The fact that $T_{1,2} = T_{1,3} = T_{2,3} = \mathbf{0}$ is immediate from Remark 3.2.6(d). Hence, we may assume that $1 \leq v \leq i \leq 3$.

- **The matrix** $T_{1,1}$. By Remark 3.2.6(b), the matrix $T_{1,1}$ consists of the values of the ordinary characters of the projective indecomposable $k\overline{G}$-modules evaluated at the $2'$-conjugacy classes of $\overline{G}$. Hence, the claim follows from Proposition 4.3.8.

- **The matrix** $T_{2,1}$. By Remark 3.2.6(e), the matrix $T_{2,1}$ consists of the values of the ordinary characters of the trivial source $k\overline{G}$-modules with vertex $\overline{Q_3} \cong C_2$ evaluated at the $2'$-conjugacy classes of $\overline{G}$. Hence, the claim follows from Proposition 4.3.8.

- **The matrix** $T_{3,1}$. By Remark 3.2.6(e), the matrix $T_{2,1}$ consists of the values of the ordinary characters of the trivial source $k\overline{G}$-modules with vertex $\overline{Q_4} \cong C_2 \times C_2$ evaluated at the $2'$-conjugacy classes of $\overline{G}$. Hence, the claim follows from Proposition 4.3.8.

- **The matrix** $T_{2,2}$. By Convention 3.2.2, the matrix $T_{2,2}$ consists of the values of the species $\tau^{\overline{G}}_{\overline{Q_3},s}$, with s running through $[N_{\overline{G}}(\overline{Q_3})]_{2'}$, evaluated at the trivial source modules $[M] \in \mathrm{TS}(\overline{G};\overline{Q_3})$. By Remark 3.2.6(g), $s = 1$ yields

$$\tau^{\overline{G}}_{\overline{Q_3},1}([M]) = \tau^{\overline{Q_3}/\overline{Q_3}}_{\langle 1\rangle,1} \circ \mathrm{Br}^{\overline{Q_3}}_{\overline{Q_3}} \circ \mathrm{Res}^{\overline{G}}_{\overline{Q_3}}([M]).$$

Now, by Remark 3.2.6(e), $\tau^{\overline{Q_3}/\overline{Q_3}}_{\langle 1\rangle,1}$ returns the k-dimension of $\mathrm{Br}^{\overline{Q_3}}_{\overline{Q_3}} \circ \mathrm{Res}^{\overline{G}}_{\overline{Q_3}}(M)$, which is easily computed as follows. Because $\overline{Q_3} \cong C_2$, the indecomposable direct summands of $\mathrm{Res}^{\overline{G}}_{\overline{Q_3}}(M)$ are either trivial or projective and it follows that $\mathrm{Br}^{\overline{Q_3}}_{\overline{Q_3}}$ returns only the trivial summands of the latter module. By Lemma 3.1.6, the multiplicity of the trivial module as a direct summand of $\mathrm{Res}^{\overline{G}}_{\overline{Q_3}}(M)$ is given by $\chi_{\widehat{M}}(z)$ where z is the generator of $\overline{Q_3}$. Therefore, since z is an element of order 2 and the modules $[M] \in \mathrm{TS}(\overline{G};\overline{Q_3})$ afford the characters $\chi_1 + \chi_2 + \chi_{14} + \chi_{15}$ and χ_8, we read from Table 4.10 that

$$\chi_{\widehat{M}}(z) = 2$$

in all cases. Next, we prove that if $M = \mathrm{Sc}(\overline{G}, C_2) = \mathrm{Sc}(\overline{G}, \overline{Q_3})$, then

$$\tau^{\overline{G}}_{\overline{Q_3},s}([M]) = 2 \text{ for each } 1 \neq s \in [N_{\overline{G}}(\overline{Q_3})]_{2'}.$$

By definition $\tau^{\overline{G}}_{\overline{Q_3},s}([M])$ is given by the Brauer character $\varphi_{M[\overline{Q_3}]}$ of $M[\overline{Q_3}]$ evaluated at s. Moreover, by Remark 3.2.6(e), $M[\overline{Q_3}]$ seen as a $kN_{\overline{G}}(\overline{Q_3})$-module is the $kN_{\overline{G}}(\overline{Q_3})$-Green correspondent of M which is again the Scott module with vertex $\overline{Q_3}$, that is,

$$M[\overline{Q_3}] = \mathrm{Sc}(N_{\overline{G}}(\overline{Q_3}), \overline{Q_3})$$

(see [Bro85, §2]). Thus it suffices to prove that the ordinary character $\chi_{\widehat{M[\overline{Q_3}]}}$ takes value 2 at all the $2'$-conjugacy classes of $N_{\overline{G}}(\overline{Q_3})$. Now, the normaliser $N_{\overline{G}}(\overline{Q_3}) =: \mathcal{D}_{12} \cong D_{12}$ is a dihedral group of order $4w$ with w odd. Clearly, Scott modules belong to the principal block because they have a trivial composition factor by definition, and $B_0(\mathcal{D}_{4w})$ is splendidly Morita equivalent to $k[C_2 \times C_2]$ by the main result of [CEKL11], as $\mathcal{D}_{4w}$ is 2-solvable. For $R_2 \leq C_2 \times C_2$ of order 2 it is straightforward to compute that

$$U := \mathrm{Sc}(C_2 \times C_2, R_2) = k \uparrow_{R_2}^{C_2 \times C_2}$$

over $k\,[C_2 \times C_2]$ and that U affords the ordinary character

$$\chi_{\widehat{U}} = 1_{C_2 \times C_2} + 1_b,$$

where $1_b \in \mathrm{Irr}\,(C_2 \times C_2) \setminus \{1_{C_2 \times C_2}\}$. It follows then directly from the character table of $\mathcal{D}_{12}$, see Table 4.7, and the above splendid Morita equivalence that $\mathrm{Irr}_K(B_0(\mathcal{D}_{12})) = \mathrm{Lin}(B_0(\mathcal{D}_{12}))$ and $\chi_{\widehat{M[\overline{Q_3}]}}$ is the sum of two linear characters. The claim now follows from the fact that all the linear characters of $\mathcal{D}_{4w}$ take value 1 at all $2'$-conjugacy classes. The remaining entries of $T_{2,2}$ follow from Remark 3.2.6(b).

- **The matrix** $T_{3,2}$. By Convention 3.2.2, the matrix $T_{3,2}$ consists of the values of the species $\tau^{\overline{G}}_{\overline{Q_3},s}$, with s running through $[\overline{N}_{\overline{G}}(\overline{Q_3})]_{2'}$, evaluated at the trivial source modules $[M] \in \mathrm{TS}(\overline{G}; \overline{Q_4})$. As in the previous case, if $s = 1$, then

$$\tau^{\overline{G}}_{\overline{Q_3},1}([M]) = \dim_k(\mathrm{Br}^{\overline{Q_3}}_{\overline{Q_3}} \circ \mathrm{Res}^{\overline{G}}_{\overline{Q_3}}(M)) = \chi_{\widehat{M}}(z)$$

where z is the generator of $\overline{Q_3}$. Since the three modules $[M] \in \mathrm{TS}(\overline{G}; \overline{Q_4})$ afford the characters $\chi_1, \chi_{14}, \chi_{15}$, we read from Table 4.10 that

$$\chi_{\widehat{M}}(z) = 1$$

in all cases, as required.

Next, we claim that

$$\tau^{\overline{G}}_{\overline{Q_3},s}([M]) = 1 \quad \forall\ [M] \in \mathrm{TS}(\overline{G}; \overline{Q_4}), \forall s \in [\overline{N}_{\overline{G}}(\overline{Q_3})]_{2'}.$$

First, notice that by Remark 3.2.6(f) the above argument yields

$$\dim_k(M[\overline{Q_3}]) = \tau^{\overline{G}}_{\overline{Q_3},1}([M]) = 1 \quad \forall\ [M] \in \mathrm{TS}(\overline{G}; \overline{Q_4}).$$

Now, $\overline{N}_{\overline{G}}(\overline{Q_3})$ has a unique trivial source module with vertex C_2, namely the trivial module. Indeed, this follows from Proposition 3.1.11(d) as $\overline{N}_{\overline{G}}(\overline{Q_3}) \cong D_6$, so that subgroups of order 2 are conjugate and self-normalising. Therefore, we conclude that $M[\overline{Q_3}] = k$ for every $[M] \in \mathrm{TS}(\overline{G}; \overline{Q_4})$. In all cases, by definition $\tau^{\overline{G}}_{\overline{Q_3},s}([M])$ is equal to the Brauer character of the trivial $k\overline{N}_{\overline{G}}(\overline{Q_3})$-module evaluated at s, hence equal to 1, proving the claim.

- **The matrix** $T_{3,3}$. Because $\overline{N}_{\overline{G}}(\overline{Q_4}) \cong C_3$, by Remark 3.2.6(b) the matrix $T_{3,3}$ of $\mathrm{Triv}_2(\overline{G})$ is just the ordinary character table of the cyclic group C_3. □

The trivial source character table $\mathrm{Triv}_2(G) = [T_{i,v}]_{1\leq i,v\leq 4}$ of $G = \mathrm{SL}_2(11)$ is now up to a large extent obtained via inflation from $\overline{G}$. For this reason, we write $T_{i,v}(G)$ for the matrix $T_{i,v}$ of $\mathrm{Triv}_2(G)$ and $T_{i,v}(\overline{G})$ for the matrix $T_{i,v}$ of $\mathrm{Triv}_2(\overline{G})$.

Theorem 4.3.10. *Assume* $G = \mathrm{SL}_2(11)$*. Then the trivial source character table* $\mathrm{Triv}_2(G) = [T_{i,v}]_{1\leq i,v\leq 4}$ *is given as follows.*

(a) *The following holds:*

  (i) $T_{i,v} = \mathbf{0}$ *for every* $1 \leq i < v \leq 4$*;*

  (ii) $T_{i,1}(G) = T_{i,2}(G) = T_{i-1,1}(\overline{G})$ *for every* $2 \leq i \leq 4$*;*

  (iii) $T_{3,3}(G) = T_{2,2}(\overline{G}),\ T_{4,3}(G) = T_{3,2}(\overline{G}),\ T_{4,4}(G) = T_{3,3}(\overline{G})$.

(b) *The matrix* $T_{1,1}$ *is as given in Table 4.15.*

	I_2	$\begin{pmatrix}\zeta_{11,1} & 0\\ 0 & \zeta_{11,9}\end{pmatrix}^2$	$\begin{pmatrix}\zeta_{11,1} & 0\\ 0 & \zeta_{11,9}\end{pmatrix}^4$	$\begin{pmatrix}\zeta_{11,8} & \zeta_{11,2}\\ \zeta_{11,1} & \zeta_{11,8}\end{pmatrix}^4$	u_+	u_-
$\chi_1+\chi_2+\chi_{12}+\chi_{13}$	24	4	4	0	2	2
$\chi_2+\chi_9+\chi_{13}+\chi_{14}$	32	2	2	-4	$-1+\sqrt{11}i$	$-1-\sqrt{11}i$
$\chi_2+\chi_9+\chi_{12}+\chi_{15}$	32	2	2	-4	$-1-\sqrt{11}i$	$-1+\sqrt{11}i$
$\chi_7+\chi_8+\chi_{10}+\chi_{11}$	40	0	0	4	-4	-4
$\chi_4+\chi_6$	24	$-1+\sqrt{5}$	$-1-\sqrt{5}$	0	2	2
$\chi_3+\chi_5$	24	$-1-\sqrt{5}$	$-1+\sqrt{5}$	0	2	2

Table 4.15: $T_{1,1}$ **for** $G = \mathrm{SL}_2(11)$ **and** $p = 2$

Proof. (a) Again, the fact that $T_{i,v} = \mathbf{0}$ for every $1 \leq i < v \leq 4$ is immediate from Remark 3.2.6(d). This proves (i).
Next, we notice that the subgroups Q_2, Q_3, Q_4 all contain the centre $Z = Q_2$. The following assertions follow immediately.

1. For every $2 \leq i \leq 4$ and every $M \in \mathrm{TS}(G, Q_i)$ we have $M[Q_2] = M$ (as kN_2-modules). Therefore, as our choices of $[\overline{N}_1]_{2'}$ and $[\overline{N}_2]_{2'}$ agree modulo Z, by definition of the species we have $T_{i,1}(G) = T_{i,2}(G)$ for every $2 \leq i \leq 4$.
2. For every $2 \leq i \leq 4$, we have $Q_i/Z = \overline{Q_i}$, and so any trivial source module in $\mathrm{TS}(G, Q_i)$ is the inflation from $\overline{G} = G/Z$ to G of a trivial source $k\overline{G}$-module with vertex $\overline{Q_i}$, i.e.

$$\mathrm{TS}(G, Q_i) = \left\{\mathrm{Inf}_{\overline{G}}^{G}(M) \mid M \in \mathrm{TS}(\overline{G}, \overline{Q_i})\right\}$$

and the corresponding characters are

$$\chi_{\widehat{\mathrm{Inf}_{\overline{G}}^{G}(M)}} = \mathrm{Inf}_{\overline{G}}^{G}(\chi_{\widehat{M}}).$$

It follows that $T_{i,2}(G) = T_{i-1,1}(\overline{G})$ for every $2 \leq i \leq 4$ and

$$T_{3,3}(G) = T_{2,2}(\overline{G}),\ T_{4,3}(G) = T_{3,2}(\overline{G}),\ T_{4,4}(G) = T_{3,3}(\overline{G})$$

because our choices of the representatives of the $2'$-conjugacy classes agree modulo Z, proving (ii) and (iii).

(b) The ordinary characters of the PIMs of $B_0(G)$ can be read from the 2-decomposition matrix in Proposition 4.3.6. □

4.3.3 The groups $\mathrm{SL}_2(13)$ and $\mathrm{PSL}_2(13)$

We consider the special linear group

$$G := \mathrm{SL}_2(13) := \left\{ \begin{pmatrix} a & b \\ c & d \end{pmatrix} \mid ad - bc = 1 \right\} \leq \mathrm{GL}_2(\mathbb{F}_{13})$$

with $|G| = (13-1)\cdot 13 \cdot (13+1) = 2184$. We choose the following representatives of the 17 conjugacy classes of G:

$$\begin{aligned} \mathrm{I}_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, &\quad -\mathrm{I}_2, \\ \begin{pmatrix} \zeta_{13,1} & 0 \\ 0 & \zeta_{13,11} \end{pmatrix}^i &\quad (1 \leq i \leq 5), \\ \begin{pmatrix} \zeta_{13,8} & \zeta_{13,1} \\ 1 & \zeta_{13,8} \end{pmatrix}^j &\quad (1 \leq j \leq 6), \\ u_+ := \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, &\quad -u_+, \\ u_- := \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}, &\quad -u_-. \end{aligned}$$

The ordinary character table of G is as given in Table 4.16. We set

$$\begin{aligned} &Q_1 := \langle 1 \rangle, \\ &Q_2 := \langle -\mathrm{I}_2 \rangle = Z(G) =: Z \cong C_2 \\ &Q_3 := \left\langle \begin{pmatrix} \zeta_{13,3} & 0 \\ 0 & \zeta_{13,9} \end{pmatrix} \right\rangle \cong C_4, \text{ and} \\ &Q_4 := \left\langle \begin{pmatrix} \zeta_{13,3} & 0 \\ 0 & \zeta_{13,9} \end{pmatrix}, \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \right\rangle \cong \mathcal{Q}_8. \end{aligned}$$

Furthermore, we choose $\mathscr{S}_2(\mathrm{SL}_2(13)) = \{Q_1, Q_2, Q_3, Q_4\}$. Then, the lattice of subgroups in $\mathscr{S}_2(\mathrm{SL}_2(13))$ is as given in Fig. 4.8.

Figure 4.8: the lattice of subgroups in $\mathscr{S}_2(\mathrm{SL}_2(13))$

g	I_2	$-\mathrm{I}_2$	$\begin{pmatrix}\zeta_{13,1} & 0\\ 0 & \zeta_{13,11}\end{pmatrix}$	$\begin{pmatrix}\zeta_{13,1} & 0\\ 0 & \zeta_{13,11}\end{pmatrix}^2$	$\begin{pmatrix}\zeta_{13,1} & 0\\ 0 & \zeta_{13,11}\end{pmatrix}^3$	$\begin{pmatrix}\zeta_{13,1} & 0\\ 0 & \zeta_{13,11}\end{pmatrix}^4$	$\begin{pmatrix}\zeta_{13,1} & 0\\ 0 & \zeta_{13,11}\end{pmatrix}^5$	$\begin{pmatrix}\zeta_{13,8} & \zeta_{13,1}\\ 1 & \zeta_{13,8}\end{pmatrix}$	$\begin{pmatrix}\zeta_{13,8} & \zeta_{13,1}\\ 1 & \zeta_{13,8}\end{pmatrix}^2$	$\begin{pmatrix}\zeta_{13,8} & \zeta_{13,1}\\ 1 & \zeta_{13,8}\end{pmatrix}^3$	$\begin{pmatrix}\zeta_{13,8} & \zeta_{13,1}\\ 1 & \zeta_{13,8}\end{pmatrix}^4$	$\begin{pmatrix}\zeta_{13,8} & \zeta_{13,1}\\ 1 & \zeta_{13,8}\end{pmatrix}^5$	$\begin{pmatrix}\zeta_{13,8} & \zeta_{13,1}\\ 1 & \zeta_{13,8}\end{pmatrix}^6$	u_+	$-u_+$	u_-	$-u_-$
χ_1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
χ_2	13	13	1	1	1	1	1	-1	-1	-1	-1	-1	-1	0	0	0	0
χ_3	14	14	-1	-1	2	-1	-1	0	0	0	0	0	0	1	1	1	1
χ_4	14	14	1	-1	-2	-1	1	0	0	0	0	0	0	1	1	1	1
χ_5	14	-14	0	-2	0	2	0	0	0	0	0	0	0	1	-1	1	-1
χ_6	14	-14	$-\sqrt{3}$	1	0	-1	$\sqrt{3}$	0	0	0	0	0	0	1	-1	1	-1
χ_7	14	-14	$\sqrt{3}$	1	0	-1	$-\sqrt{3}$	0	0	0	0	0	0	1	-1	1	-1
χ_8	12	12	0	0	0	0	0	$-\exp\left(\frac{2\pi i}{7}\right)-\exp\left(\frac{12\pi i}{7}\right)$	$-\exp\left(\frac{4\pi i}{7}\right)-\exp\left(\frac{10\pi i}{7}\right)$	$-\exp\left(\frac{6\pi i}{7}\right)-\exp\left(\frac{8\pi i}{7}\right)$	$-\exp\left(\frac{6\pi i}{7}\right)-\exp\left(\frac{8\pi i}{7}\right)$	$-\exp\left(\frac{4\pi i}{7}\right)-\exp\left(\frac{10\pi i}{7}\right)$	$-\exp\left(\frac{2\pi i}{7}\right)-\exp\left(\frac{12\pi i}{7}\right)$	-1	-1	-1	-1
χ_9	12	12	0	0	0	0	0	$-\exp\left(\frac{6\pi i}{7}\right)-\exp\left(\frac{8\pi i}{7}\right)$	$-\exp\left(\frac{2\pi i}{7}\right)-\exp\left(\frac{12\pi i}{7}\right)$	$-\exp\left(\frac{4\pi i}{7}\right)-\exp\left(\frac{10\pi i}{7}\right)$	$-\exp\left(\frac{4\pi i}{7}\right)-\exp\left(\frac{10\pi i}{7}\right)$	$-\exp\left(\frac{2\pi i}{7}\right)-\exp\left(\frac{12\pi i}{7}\right)$	$-\exp\left(\frac{6\pi i}{7}\right)-\exp\left(\frac{8\pi i}{7}\right)$	-1	-1	-1	-1
χ_{10}	12	12	0	0	0	0	0	$-\exp\left(\frac{4\pi i}{7}\right)-\exp\left(\frac{10\pi i}{7}\right)$	$-\exp\left(\frac{6\pi i}{7}\right)-\exp\left(\frac{8\pi i}{7}\right)$	$-\exp\left(\frac{2\pi i}{7}\right)-\exp\left(\frac{12\pi i}{7}\right)$	$-\exp\left(\frac{2\pi i}{7}\right)-\exp\left(\frac{12\pi i}{7}\right)$	$-\exp\left(\frac{6\pi i}{7}\right)-\exp\left(\frac{8\pi i}{7}\right)$	$-\exp\left(\frac{4\pi i}{7}\right)-\exp\left(\frac{10\pi i}{7}\right)$	-1	-1	-1	-1
χ_{11}	12	-12	0	0	0	0	0	$\exp\left(\frac{2\pi i}{7}\right)+\exp\left(\frac{12\pi i}{7}\right)$	$-\exp\left(\frac{4\pi i}{7}\right)-\exp\left(\frac{10\pi i}{7}\right)$	$\exp\left(\frac{6\pi i}{7}\right)+\exp\left(\frac{8\pi i}{7}\right)$	$-\exp\left(\frac{6\pi i}{7}\right)-\exp\left(\frac{8\pi i}{7}\right)$	$\exp\left(\frac{4\pi i}{7}\right)+\exp\left(\frac{10\pi i}{7}\right)$	$-\exp\left(\frac{2\pi i}{7}\right)-\exp\left(\frac{12\pi i}{7}\right)$	-1	1	-1	1
χ_{12}	12	-12	0	0	0	0	0	$\exp\left(\frac{6\pi i}{7}\right)+\exp\left(\frac{8\pi i}{7}\right)$	$-\exp\left(\frac{2\pi i}{7}\right)-\exp\left(\frac{12\pi i}{7}\right)$	$\exp\left(\frac{4\pi i}{7}\right)+\exp\left(\frac{10\pi i}{7}\right)$	$-\exp\left(\frac{4\pi i}{7}\right)-\exp\left(\frac{10\pi i}{7}\right)$	$\exp\left(\frac{2\pi i}{7}\right)+\exp\left(\frac{12\pi i}{7}\right)$	$-\exp\left(\frac{6\pi i}{7}\right)-\exp\left(\frac{8\pi i}{7}\right)$	-1	1	-1	1
χ_{13}	12	-12	0	0	0	0	0	$\exp\left(\frac{4\pi i}{7}\right)+\exp\left(\frac{10\pi i}{7}\right)$	$-\exp\left(\frac{6\pi i}{7}\right)-\exp\left(\frac{8\pi i}{7}\right)$	$\exp\left(\frac{2\pi i}{7}\right)+\exp\left(\frac{12\pi i}{7}\right)$	$-\exp\left(\frac{2\pi i}{7}\right)-\exp\left(\frac{12\pi i}{7}\right)$	$\exp\left(\frac{6\pi i}{7}\right)+\exp\left(\frac{8\pi i}{7}\right)$	$-\exp\left(\frac{4\pi i}{7}\right)-\exp\left(\frac{10\pi i}{7}\right)$	-1	1	-1	1
χ_{14}	7	7	-1	1	-1	1	-1	0	0	0	0	0	0	$\frac{1+\sqrt{13}}{2}$	$\frac{1+\sqrt{13}}{2}$	$\frac{1-\sqrt{13}}{2}$	$\frac{1-\sqrt{13}}{2}$
χ_{15}	7	7	-1	1	-1	1	-1	0	0	0	0	0	0	$\frac{1-\sqrt{13}}{2}$	$\frac{1-\sqrt{13}}{2}$	$\frac{1+\sqrt{13}}{2}$	$\frac{1+\sqrt{13}}{2}$
χ_{16}	6	-6	0	0	0	0	0	1	-1	1	-1	1	-1	$\frac{\sqrt{13}-1}{2}$	$\frac{1-\sqrt{13}}{2}$	$\frac{-1-\sqrt{13}}{2}$	$\frac{1+\sqrt{13}}{2}$
χ_{17}	6	-6	0	0	0	0	0	1	-1	1	-1	1	-1	$\frac{-1-\sqrt{13}}{2}$	$\frac{1+\sqrt{13}}{2}$	$\frac{\sqrt{13}-1}{2}$	$\frac{1-\sqrt{13}}{2}$

Table 4.16: **ordinary character table of** $\mathrm{SL}_2(13)$

Moreover,

$$\begin{aligned}
&N_G(Q_1) = G && \text{and } \overline{N}_G(Q_1) \cong G;\\
&N_G(Q_2) = G && \text{and } \overline{N}_G(Q_2) = G/Z \cong \mathrm{PSL}_2(13);\\
&N_G(Q_3) = \left\langle \begin{pmatrix} \zeta_{13,1} & 0 \\ 0 & \zeta_{13,11} \end{pmatrix}, \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \right\rangle \cong C_3 \rtimes \mathcal{Q}_8 && \text{and } \overline{N}_G(Q_3) \cong \mathfrak{S}_3;\\
&N_G(Q_4) = \left\langle Q_4, \begin{pmatrix} \zeta_{13,1} & \zeta_{13,1} \\ \zeta_{13,4} & \zeta_{13,10} \end{pmatrix} \right\rangle \cong \mathrm{SL}_2(3) && \text{and } \overline{N}_G(Q_4) \cong C_3.
\end{aligned}$$

Next, we consider the group $\overline{G} := G/Z \cong \mathrm{PSL}_2(13)$. We identify $\overline{G}$ with $\mathrm{PSL}_2(13)$. We set

$$\begin{aligned}
\overline{Q_2} &:= \langle 1 \rangle,\\
\overline{Q_3} &:= \left\langle \begin{pmatrix} \zeta_{13,3} & 0 \\ 0 & \zeta_{13,9} \end{pmatrix} Z \right\rangle \cong C_2, \text{ and}\\
\overline{Q_4} &:= \left\langle \begin{pmatrix} \zeta_{13,3} & 0 \\ 0 & \zeta_{13,9} \end{pmatrix} Z, \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} Z \right\rangle \cong V_4.
\end{aligned}$$

Moreover, we choose $\mathscr{S}_2(\mathrm{PSL}_2(13)) = \{\overline{Q_2}, \overline{Q_3}, \overline{Q_4}\}$. Then, the lattice of subgroups in $\mathscr{S}_2(\overline{G})$ is as given in Fig. 4.9.

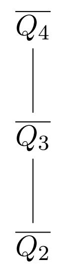


Figure 4.9: the lattice of subgroups in $\mathrm{PSL}_2(13)$

Furthermore,

$$\begin{aligned}
&N_{\overline{G}}(\overline{Q_2}) = \overline{G} && \text{and } \overline{N}_{\overline{G}}(\overline{Q_2}) \cong \overline{G};\\
&N_{\overline{G}}(\overline{Q_3}) = \left\langle \begin{pmatrix} \zeta_{13,1} & 0 \\ 0 & \zeta_{13,11} \end{pmatrix} Z, \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} Z \right\rangle \cong D_{12} && \text{and } \overline{N}_{\overline{G}}(\overline{Q_3}) \cong \mathfrak{S}_3;\\
&N_{\overline{G}}(\overline{Q_4}) = \left\langle \overline{Q_4}, \begin{pmatrix} \zeta_{13,1} & \zeta_{13,1} \\ \zeta_{13,4} & \zeta_{13,10} \end{pmatrix} Z \right\rangle \cong \mathfrak{A}_4 && \text{and } \overline{N}_{\overline{G}}(\overline{Q_4}) \cong C_3.
\end{aligned}$$

Notation 4.3.11. We choose the following sets of representatives of the $2'$-conjugacy classes of the groups occurring in $\mathscr{S}_2(G)$ and $\mathscr{S}_2(\overline{G})$, respectively, where the bar notation denotes left cosets in the respective quotients $\overline{N}_{\overline{G}}(\overline{Q_i})$ for $2 \leq i \leq 4$:

$$\begin{aligned}
[\overline{N}_G(Q_1)]_{2'} :=& \{I_2\} \cup \left\{ \begin{pmatrix} \zeta_{13,4} & 0 \\ 0 & \zeta_{13,8} \end{pmatrix} \right\} \cup \left\{ \begin{pmatrix} \zeta_{13,8} & \zeta_{13,1} \\ 1 & \zeta_{13,8} \end{pmatrix}^2, \begin{pmatrix} \zeta_{13,8} & \zeta_{13,1} \\ 1 & \zeta_{13,8} \end{pmatrix}^4, \begin{pmatrix} \zeta_{13,8} & \zeta_{13,1} \\ 1 & \zeta_{13,8} \end{pmatrix}^6 \right\} \\
& \cup \{u_+, u_-\}; \\
[\overline{N}_G(Q_2)]_{2'} :=& \{I_2 Q_2\} \cup \left\{ \begin{pmatrix} \zeta_{13,4} & 0 \\ 0 & \zeta_{13,8} \end{pmatrix} Q_2 \right\} \\
& \cup \left\{ \begin{pmatrix} \zeta_{13,8} & \zeta_{13,1} \\ 1 & \zeta_{13,8} \end{pmatrix}^2 Q_2, \begin{pmatrix} \zeta_{13,8} & \zeta_{13,1} \\ 1 & \zeta_{13,8} \end{pmatrix}^4 Q_2, \begin{pmatrix} \zeta_{13,8} & \zeta_{13,1} \\ 1 & \zeta_{13,8} \end{pmatrix}^6 Q_2 \right\} \\
& \cup \{u_+ Q_2, u_- Q_2\}; \\
[\overline{N}_G(Q_3)]_{2'} :=& \{I_2 Q_3\} \cup \left\{ \begin{pmatrix} \zeta_{13,4} & 0 \\ 0 & \zeta_{13,8} \end{pmatrix} Q_3 \right\}; \\
[\overline{N}_G(Q_4)]_{2'} :=& \{I_2 Q_4\} \cup \left\{ \begin{pmatrix} \zeta_{13,1} & \zeta_{13,1} \\ \zeta_{13,4} & \zeta_{13,10} \end{pmatrix} Q_4, \begin{pmatrix} \zeta_{13,1} & \zeta_{13,1} \\ \zeta_{13,4} & \zeta_{13,10} \end{pmatrix}^2 Q_4 \right\}; \\
[\overline{N}_{\overline{G}}(\overline{Q_2})]_{2'} :=& \left\{\overline{I_2 Z}\right\} \cup \left\{ \overline{\begin{pmatrix} \zeta_{13,4} & 0 \\ 0 & \zeta_{13,8} \end{pmatrix} Z} \right\} \\
& \cup \left\{ \overline{\begin{pmatrix} \zeta_{13,8} & \zeta_{13,1} \\ 1 & \zeta_{13,8} \end{pmatrix}^2 Z}, \overline{\begin{pmatrix} \zeta_{13,8} & \zeta_{13,1} \\ 1 & \zeta_{13,8} \end{pmatrix}^4 Z}, \overline{\begin{pmatrix} \zeta_{13,8} & \zeta_{13,1} \\ 1 & \zeta_{13,8} \end{pmatrix}^6 Z} \right\} \cup \left\{\overline{u_+ Z}, \overline{u_- Z}\right\}; \\
[\overline{N}_{\overline{G}}(\overline{Q_3})]_{2'} :=& \left\{ \overline{I_2 Z}, \overline{\begin{pmatrix} \zeta_{13,4} & 0 \\ 0 & \zeta_{13,8} \end{pmatrix} Z} \right\}; \\
[\overline{N}_{\overline{G}}(\overline{Q_4})]_{2'} :=& \left\{ \overline{I_2 Z}, \overline{\begin{pmatrix} \zeta_{13,1} & \zeta_{13,1} \\ \zeta_{13,4} & \zeta_{13,10} \end{pmatrix} Z}, \overline{\begin{pmatrix} \zeta_{13,1} & \zeta_{13,1} \\ \zeta_{13,4} & \zeta_{13,10} \end{pmatrix}^2 Z} \right\}.
\end{aligned}$$

We identify the ordinary characters of $\overline{G}$ with the ordinary characters of G with the centre Z in their kernel. We label the ordinary characters and the 2-blocks of $\mathrm{PSL}_2(13)$ using the corresponding labelling in $\mathrm{SL}_2(13)$. Consequently, the ordinary character table of $\overline{G}$ is given in Table 4.17.

g	$\mathrm{I}_2 Z$	$\begin{pmatrix} \zeta_{13,1} & 0 \\ 0 & \zeta_{13,11} \end{pmatrix} Z$	$\begin{pmatrix} \zeta_{13,1} & 0 \\ 0 & \zeta_{13,11} \end{pmatrix}^3 Z$	$\begin{pmatrix} \zeta_{13,1} & 0 \\ 0 & \zeta_{13,11} \end{pmatrix}^4 Z$	$\begin{pmatrix} \zeta_{13,8} & \zeta_{13,1} \\ 1 & \zeta_{13,8} \end{pmatrix}^2 Z$	$\begin{pmatrix} \zeta_{13,8} & \zeta_{13,1} \\ 1 & \zeta_{13,8} \end{pmatrix}^4 Z$	$\begin{pmatrix} \zeta_{13,8} & \zeta_{13,1} \\ 1 & \zeta_{13,8} \end{pmatrix}^6 Z$	$u_+ Z$	$u_- Z$
χ_1	1	1	1	1	1	1	1	1	1
χ_2	13	1	1	1	-1	-1	-1	0	0
χ_3	14	-1	2	-1	0	0	0	1	1
χ_4	14	1	-2	-1	0	0	0	1	1
χ_8	12	0	0	0	$-\exp\left(\frac{4\pi i}{7}\right) - \exp\left(\frac{10\pi i}{7}\right)$	$-\exp\left(\frac{6\pi i}{7}\right) - \exp\left(\frac{8\pi i}{7}\right)$	$-\exp\left(\frac{2\pi i}{7}\right) - \exp\left(\frac{12\pi i}{7}\right)$	-1	-1
χ_9	12	0	0	0	$-\exp\left(\frac{2\pi i}{7}\right) - \exp\left(\frac{12\pi i}{7}\right)$	$-\exp\left(\frac{4\pi i}{7}\right) - \exp\left(\frac{10\pi i}{7}\right)$	$-\exp\left(\frac{6\pi i}{7}\right) - \exp\left(\frac{8\pi i}{7}\right)$	-1	-1
χ_{10}	12	0	0	0	$-\exp\left(\frac{6\pi i}{7}\right) - \exp\left(\frac{8\pi i}{7}\right)$	$-\exp\left(\frac{2\pi i}{7}\right) - \exp\left(\frac{12\pi i}{7}\right)$	$-\exp\left(\frac{4\pi i}{7}\right) - \exp\left(\frac{10\pi i}{7}\right)$	-1	-1
χ_{14}	7	-1	-1	1	0	0	0	$\frac{1+\sqrt{13}}{2}$	$\frac{1-\sqrt{13}}{2}$
χ_{15}	7	-1	-1	1	0	0	0	$\frac{1-\sqrt{13}}{2}$	$\frac{1+\sqrt{13}}{2}$

Table 4.17: **ordinary character table of** $\mathrm{PSL}_2(13)$

Notation 4.3.12. (a) We denote the simple kG-modules by S_i $(1 \leq i \leq 7)$. Moreover, we set $\mathrm{IBr}_2(G) = \{\varphi_1, \varphi_2, \varphi_3, \varphi_4, \varphi_5, \varphi_6, \varphi_7\}$ where $\varphi_i := \varphi_{S_i}$ for $1 \leq i \leq 7$.

(b) We denote the simple $k\overline{G}$-modules by T_i $(1 \leq i \leq 7)$. Moreover, we set $\mathrm{IBr}_2(\overline{G}) = \{\phi_1, \phi_2, \phi_3, \phi_4, \phi_5, \phi_6, \phi_7\}$ where $\phi_i := \phi_{T_i}$ for $1 \leq i \leq 7$.

Proposition 4.3.13. (a) *The decomposition matrix* $\mathfrak{D}(kG)$ *is equal to*

	φ_1	φ_2	φ_3	φ_4	φ_5	φ_6	φ_7
χ_1	1	0	0	0	0	0	0
χ_2	1	1	1	0	0	0	0
χ_3	0	0	0	0	0	0	1
χ_4	0	0	0	0	0	0	1
χ_5	2	1	1	0	0	0	0
χ_6	0	0	0	0	0	0	1
χ_7	0	0	0	0	0	0	1
χ_8	0	0	0	1	0	0	0
χ_9	0	0	0	0	1	0	0
χ_{10}	0	0	0	0	0	1	0
χ_{11}	0	0	0	1	0	0	0
χ_{12}	0	0	0	0	1	0	0
χ_{13}	0	0	0	0	0	1	0
χ_{14}	1	1	0	0	0	0	0
χ_{15}	1	0	1	0	0	0	0
χ_{16}	0	1	0	0	0	0	0
χ_{17}	0	0	1	0	0	0	0

.

(b) *The ordinary irreducible characters of* G *split into the following five blocks of* kG*:*

$$\mathrm{Irr}_K(B_0(kG)) = \{\chi_1, \chi_2, \chi_5, \chi_{14}, \chi_{15}, \chi_{16}, \chi_{17}\},$$

$$\mathrm{Irr}_K(B_1(kG)) = \{\chi_3, \chi_4, \chi_6, \chi_7\},$$

$$\mathrm{Irr}_K(B_2(kG)) = \{\chi_8, \chi_{11}\},$$

$$\mathrm{Irr}_K(B_3(kG)) = \{\chi_9, \chi_{12}\}, \text{ and}$$

$$\mathrm{Irr}_K(B_4(kG)) = \{\chi_{10}, \chi_{13}\}.$$

(c) *We have* $d(B_0(kG)) = 3$, $d(B_1(kG)) = 2$, $d(B_2(kG)) = 1$, $d(B_3(kG)) = 1$, *and* $d(B_4(kG)) = 1$.

(d) *We have* $D(B_0(kG)) \cong \mathcal{Q}_8$, $D(B_1(kG)) \cong C_4$, $D(B_2(kG)) \cong C_2, D(B_3(kG)) \cong C_2$, *and* $D(B_4(kG)) \cong C_2$.

Proof. The assertions in (a), (b), and (c) follow from [WTP$^+$98, L2(13)mod2.pdf]. It remains to prove part (d). The principal block of kG has the Sylow 2-subgroups as vertices. As the group $\mathcal{Q}_8$ does not have any subgroup isomorphic to $C_2 \times C_2$, part (d) follows immediately from part (c). □

Next, we use Proposition 4.3.13 in order to derive the analogous pieces of information for the group $\overline{G}$.

Proposition 4.3.14. (a) *The decomposition matrix* $\mathfrak{D}(k\overline{G})$ *is equal to*

	ϕ_1	ϕ_2	ϕ_3	ϕ_4	ϕ_5	ϕ_6	ϕ_7
χ_1	1	0	0	0	0	0	0
χ_2	1	1	1	0	0	0	0
χ_3	0	0	0	0	0	0	1
χ_4	0	0	0	0	0	0	1
χ_8	0	0	0	1	0	0	0
χ_9	0	0	0	0	1	0	0
χ_{10}	0	0	0	0	0	1	0
χ_{14}	1	1	0	0	0	0	0
χ_{15}	1	0	1	0	0	0	0

.

(b) *The ordinary irreducible characters of $\overline{G}$ split into the following five blocks of $k\overline{G}$:*

$$\mathrm{Irr}_K(B_0(k\overline{G})) = \{\chi_1, \chi_2, \chi_{14}, \chi_{15}\},$$

$$\mathrm{Irr}_K(B_1(k\overline{G})) = \{\chi_3, \chi_4\},$$

$$\mathrm{Irr}_K(B_2(k\overline{G})) = \{\chi_8\},$$

$$\mathrm{Irr}_K(B_3(k\overline{G})) = \{\chi_9\}, \ \textit{and}$$

$$\mathrm{Irr}_K(B_4(k\overline{G})) = \{\chi_{10}\}.$$

(c) *We have $d(B_0(k\overline{G})) = 2$, $d(B_1(k\overline{G})) = 1$, $d(B_2(k\overline{G})) = 0$, $d(B_3(k\overline{G})) = 0$, and $d(B_4(k\overline{G})) = 0$.*

(d) *We have $D(B_0(k\overline{G})) \cong C_2 \times C_2$, $D(B_1(k\overline{G})) \cong C_2$, $D(B_2(k\overline{G})) \cong \langle 1 \rangle \cong D(B_3(k\overline{G}))$, and $D(B_4(k\overline{G})) \cong \langle 1 \rangle$.*

Proof. Analogous to the proof of Proposition 4.3.13 □

Proposition 4.3.15. *The trivial source $k\overline{G}$-modules and their ordinary characters are as given in Table 4.18.*

Character $\chi_{\widehat{M}}$	Module M	Vertices
$\chi_1 + \chi_2 + \chi_{14} + \chi_{15}$	$P(T_1)$	$\langle 1 \rangle$
$\chi_2 + \chi_{14}$	$P(T_2)$	$\langle 1 \rangle$
$\chi_2 + \chi_{15}$	$P(T_3)$	$\langle 1 \rangle$
χ_8	$P(T_4)$	$\langle 1 \rangle$
χ_9	$P(T_5)$	$\langle 1 \rangle$
χ_{10}	$P(T_6)$	$\langle 1 \rangle$
$\chi_3 + \chi_4$	$P(T_7)$	$\langle 1 \rangle$
$\chi_1 + \chi_2$	$\mathrm{Sc}(\overline{G}, \overline{Q_3})$	C_2
χ_3	T_7	C_2
$1_{\overline{G}}$	$k_{\overline{G}}$	$C_2 \times C_2$
χ_2	$\begin{array}{c} T_2 \\ k \\ T_3 \end{array}$	$C_2 \times C_2$
χ_2	$\begin{array}{c} T_3 \\ k \\ T_2 \end{array}$	$C_2 \times C_2$

Table 4.18: **trivial source** $k\,\mathrm{PSL}_2(13)$**-modules**

Proof. By counting the $2'$-conjugacy classes of $N_{\overline{G}}(\overline{Q_i})$, $2 \leq i \leq 4$, we deduce that

$$|\mathrm{TS}(\overline{G}, \overline{Q_2})| = 7, \ |\mathrm{TS}(\overline{G}, \overline{Q_3})| = 2, \text{ and } |\mathrm{TS}(\overline{G}, \overline{Q_4})| = 3.$$

The ordinary characters of the projective indecomposable $k\overline{G}$-modules follow from the decomposition matrix of $k\overline{G}$. As $D(B_0(k\overline{G})) \cong C_2 \times C_2$, we deduce that there exists a splendid Morita equivalence between $B_0(k\overline{G})$ and $B_0(k\mathfrak{A}_5)$ since the character degrees exclude all other possibilities by Proposition 4.2.13. The same proposition implies all remaining assertions except for those concerning the trivial source modules with vertex C_2. Since the trivial source module $\mathrm{Sc}(\overline{G}, \overline{Q_3})$ belongs to the principal block of $k\overline{G}$, it only remains to prove the assertions about the module T_7 with vertex C_2. The remaining trivial source $k\overline{G}$-module does not belong to the principal block. Moreover, it is not projective. Consequently, it belongs to a block of kG with a defect group which is isomorphic to C_2. Hence, it belongs to $B_1(k\overline{G})$. We have $\mathrm{Irr}_K(B_1(k\overline{G})) = \{\chi_3, \chi_4\}$. This block is Morita equivalent to kC_2. By Lemma 4.2.5, there exist, up to isomorphism, exactly two $B_1(k\overline{G})$-modules, and they are both trivial source modules. We have already taken the projective indecomposable $B_1(k\overline{G})$-module $P(T_7)$ into account. Consequently, $T_7 \cong \mathrm{Soc}(P(T_7))$ is the remaining trivial source $B_1(k\overline{G})$-module. The ordinary character of T_7 is either equal to χ_3 or equal to χ_4. By Proposition 3.1.7 we deduce that $\chi_{\widehat{T_7}} = \chi_3$. □

Theorem 4.3.16. *Assume* $\overline{G} = \mathrm{PSL}_2(13)$. *Then, the trivial source character table* $\mathrm{Triv}_2(\overline{G})$ *of* $\overline{G}$ *is given as follows.*

(a) *We have* $T_{1,2} = T_{1,3} = T_{2,3} = \mathbf{0}$.

(b) *The matrices* $T_{i,1}$ *with* $1 \leq i \leq 3$ *are as given in Table 4.19.*

(c) *The matrices* $T_{2,2}$ *and* $T_{3,2}$ *are as given in Table 4.20.*

(d) *The matrix $T_{3,3}$ is as given in Table 4.21 where x denotes a third root of unity.*

	$\mathrm{I}_2 Z$	$\begin{pmatrix}\zeta_{13,4} & 0\\ 0 & \zeta_{13,8}\end{pmatrix} Z$	$\begin{pmatrix}\zeta_{13,8} & \zeta_{13,1}\\ 1 & \zeta_{13,8}\end{pmatrix}^2 Z$	$\begin{pmatrix}\zeta_{13,8} & \zeta_{13,1}\\ 1 & \zeta_{13,8}\end{pmatrix}^4 Z$	$\begin{pmatrix}\zeta_{13,8} & \zeta_{13,1}\\ 1 & \zeta_{13,8}\end{pmatrix}^6 Z$	$u_+ Z$	$u_- Z$
$\chi_1+\chi_2+\chi_{14}+\chi_{15}$	28	4	0	0	0	2	2
$\chi_2+\chi_{14}$	20	2	-1	-1	-1	$\frac{1+\sqrt{13}}{2}$	$\frac{1-\sqrt{13}}{2}$
$\chi_2+\chi_{15}$	20	2	-1	-1	-1	$\frac{1-\sqrt{13}}{2}$	$\frac{1+\sqrt{13}}{2}$
χ_8	12	0	$-\exp\left(\frac{4\pi i}{7}\right)-\exp\left(\frac{10\pi i}{7}\right)$	$-\exp\left(\frac{6\pi i}{7}\right)-\exp\left(\frac{8\pi i}{7}\right)$	$-\exp\left(\frac{2\pi i}{7}\right)-\exp\left(\frac{12\pi i}{7}\right)$	-1	-1
χ_9	12	0	$-\exp\left(\frac{2\pi i}{7}\right)-\exp\left(\frac{12\pi i}{7}\right)$	$-\exp\left(\frac{4\pi i}{7}\right)-\exp\left(\frac{10\pi i}{7}\right)$	$-\exp\left(\frac{6\pi i}{7}\right)-\exp\left(\frac{8\pi i}{7}\right)$	-1	-1
χ_{10}	12	0	$-\exp\left(\frac{6\pi i}{7}\right)-\exp\left(\frac{8\pi i}{7}\right)$	$-\exp\left(\frac{2\pi i}{7}\right)-\exp\left(\frac{12\pi i}{7}\right)$	$-\exp\left(\frac{4\pi i}{7}\right)-\exp\left(\frac{10\pi i}{7}\right)$	-1	-1
$\chi_3+\chi_4$	28	-2	0	0	0	2	2
$\chi_1+\chi_2$	14	2	0	0	0	1	1
χ_3	14	-1	0	0	0	1	1
χ_1	1	1	1	1	1	1	1
χ_2	13	1	-1	-1	-1	0	0
χ_2	13	1	-1	-1	-1	0	0

Table 4.19: $T_{i,1}$ **with** $1 \le i \le 3$ **for** $\overline{G} = \mathrm{PSL}_2(13)$ **and** $p = 2$

	$\mathrm{I}_2 Z$	$\begin{pmatrix}\zeta_{13,4} & 0\\ 0 & \zeta_{13,8}\end{pmatrix} Z$
$\chi_1+\chi_2$	2	2
χ_3	2	-1
χ_1	1	1
χ_{14}	1	1
χ_{15}	1	1

Table 4.20: $T_{2,2}$ **and** $T_{3,2}$ **for** $\overline{G} = \mathrm{PSL}_2(13)$ **and** $p = 2$

	$\mathrm{I}_2 Z$	$\begin{pmatrix}\zeta_{13,1} & \zeta_{13,1}\\ \zeta_{13,4} & \zeta_{13,10}\end{pmatrix} Z$	$\begin{pmatrix}\zeta_{13,1} & \zeta_{13,1}\\ \zeta_{13,4} & \zeta_{13,10}\end{pmatrix}^2 Z$
χ_1	1	1	1
χ_2	1	x	x^2
χ_2	1	x^2	x

Table 4.21: $T_{3,3}$ **for** $\overline{G} = \mathrm{PSL}_2(13)$ **and** $p = 2$

Proof. The fact that $T_{1,2} = T_{1,3} = T_{2,3} = \mathbf{0}$ is immediate from Remark 3.2.6(d). Hence, we may assume that $1 \le v \le i \le 3$.

- **The matrix** $T_{1,1}$. By Remark 3.2.6(b), the matrix $T_{1,1}$ consists of the values of the ordinary characters of the projective indecomposable $k\overline{G}$-modules evaluated at the $2'$-conjugacy classes of $\overline{G}$. Hence, the claim follows from Proposition 4.3.15.
- **The matrix** $T_{2,1}$. By Remark 3.2.6(e), the matrix $T_{2,1}$ consists of the values of the ordinary characters of the trivial source $k\overline{G}$-modules with vertex $\overline{Q_3} \cong C_2$ evaluated at the $2'$-conjugacy classes of $\overline{G}$. Hence, the claim follows from Proposition 4.3.15.
- **The matrix** $T_{3,1}$. By Remark 3.2.6(e), the matrix $T_{2,1}$ consists of the values of the ordinary characters of the trivial source $k\overline{G}$-modules with vertex $\overline{Q_4} \cong C_2 \times C_2$ evaluated at the $2'$-conjugacy classes of $\overline{G}$. Hence, the claim follows from Proposition 4.3.15.
- **The matrix** $T_{2,2}$. By Convention 3.2.2, the matrix $T_{2,2}$ consists of the values of the species $\tau^{\overline{G}}_{\overline{Q_3},s}$, with s running through $[N_{\overline{G}}(\overline{Q_3})]_{2'}$, evaluated at the trivial source

modules $[M] \in \mathrm{TS}(\overline{G}; \overline{Q_3})$. By Remark 3.2.6(g), $s = 1$ yields

$$\tau^{\overline{G}}_{\overline{Q_3},1}([M]) = \tau^{\overline{Q_3}/\overline{Q_3}}_{\langle 1\rangle,1} \circ \mathrm{Br}^{\overline{Q_3}}_{\overline{Q_3}} \circ \mathrm{Res}^{\overline{G}}_{\overline{Q_3}}([M]).$$

Now, by Remark 3.2.6(e), $\tau^{\overline{Q_3}/\overline{Q_3}}_{\langle 1\rangle,1}$ returns the k-dimension of $\mathrm{Br}^{\overline{Q_3}}_{\overline{Q_3}} \circ \mathrm{Res}^{\overline{G}}_{\overline{Q_3}}(M)$, which is easily computed as follows. Because $\overline{Q_3} \cong C_2$, the indecomposable direct summands of $\mathrm{Res}^{\overline{G}}_{\overline{Q_3}}(M)$ are either trivial or projective and it follows that $\mathrm{Br}^{\overline{Q_3}}_{\overline{Q_3}}$ returns only the trivial summands of the latter module. By Lemma 3.1.6, the multiplicity of the trivial module as a direct summand of $\mathrm{Res}^{\overline{G}}_{\overline{Q_3}}(M)$ is given by $\chi_{\widehat{M}}(z)$ where z is the generator of $\overline{Q_3}$. Therefore, since z is an element of order 2 and the modules $[M] \in \mathrm{TS}(\overline{G}; \overline{Q_3})$ afford the characters $\chi_1 + \chi_2$ and χ_8, we read from Table 4.17 that

$$\chi_{\widehat{M}}(z) = 2$$

in all cases. Next, we prove that if $M = \mathrm{Sc}(\overline{G}, C_2) = \mathrm{Sc}(\overline{G}, \overline{Q_3})$, then

$$\tau^{\overline{G}}_{\overline{Q_3},s}([M]) = 2 \text{ for each } 1 \neq s \in [N_{\overline{G}}(\overline{Q_3})]_{2'}.$$

By definition $\tau^{\overline{G}}_{\overline{Q_3},s}([M])$ is given by the Brauer character $\varphi_{M[\overline{Q_3}]}$ of $M[\overline{Q_3}]$ evaluated at s. Moreover, by Remark 3.2.6(e), $M[\overline{Q_3}]$ seen as a $kN_{\overline{G}}(\overline{Q_3})$-module is the $kN_{\overline{G}}(\overline{Q_3})$-Green correspondent of M which is again the Scott module with vertex $\overline{Q_3}$, that is,

$$M[\overline{Q_3}] = \mathrm{Sc}(N_{\overline{G}}(\overline{Q_3}), \overline{Q_3})$$

(see [Bro85, §2]). Thus it suffices to prove that the ordinary character $\chi_{\widehat{M[\overline{Q_3}]}}$ takes value 2 at all the $2'$-conjugacy classes of $N_{\overline{G}}(\overline{Q_3})$. Now, the normaliser $N_{\overline{G}}(\overline{Q_3}) =: \mathcal{D}_{12} \cong D_{12}$ is a dihedral group of order $4w$ with w odd. Clearly, Scott modules belong to the principal block because they have a trivial composition factor by definition, and $B_0(\mathcal{D}_{4w})$ is splendidly Morita equivalent to $k[C_2 \times C_2]$ by the main result of [CEKL11], as $\mathcal{D}_{4w}$ is 2-solvable. For $R_2 \leq C_2 \times C_2$ of order 2 it is straightforward to compute that

$$U := \mathrm{Sc}(C_2 \times C_2, R_2) = k\uparrow^{C_2 \times C_2}_{R_2}$$

over $k[C_2 \times C_2]$ and that U affords the ordinary character

$$\chi_{\widehat{U}} = 1_{C_2 \times C_2} + 1_b,$$

where $1_b \in \mathrm{Irr}(C_2 \times C_2) \setminus \{1_{C_2 \times C_2}\}$. It follows then directly from the character table of $\mathcal{D}_{12}$, see Table 4.7, and the above splendid Morita equivalence that $\mathrm{Irr}_K(B_0(\mathcal{D}_{12})) = \mathrm{Lin}(B_0(\mathcal{D}_{12}))$ and $\chi_{\widehat{M[\overline{Q_3}]}}$ is the sum of two linear characters. The claim now follows from the fact that all the linear characters of $\mathcal{D}_{4w}$ take value 1 at all $2'$-conjugacy classes. The remaining entries of $T_{2,2}$ follow from Remark 3.2.6(b).

- **The matrix** $T_{3,2}$. By Convention 3.2.2, the matrix $T_{3,2}$ consists of the values of the species $\tau^{\overline{G}}_{\overline{Q_3},s}$, with s running through $[N_{\overline{G}}(\overline{Q_3})]_{2'}$, evaluated at the trivial source modules $[M] \in \mathrm{TS}(\overline{G}; \overline{Q_4})$. As in the previous case, if $s = 1$, then

$$\tau^{\overline{G}}_{\overline{Q_3},1}([M]) = \dim_k(\mathrm{Br}^{\overline{Q_3}}_{\overline{Q_3}} \circ \mathrm{Res}^{\overline{G}}_{\overline{Q_3}}(M)) = \chi_{\widehat{M}}(z)$$

where z is the generator of $\overline{Q_3}$. Since the three modules $[M] \in \mathrm{TS}(\overline{G};\overline{Q_4})$ afford the characters χ_1, χ_2, χ_2, we read from Table 4.10 that

$$\chi_{\widehat{M}}(z) = 1$$

in all cases, as required.

Next, we claim that

$$\tau^{\overline{G}}_{\overline{Q_3},s}([M]) = 1 \quad \forall\ [M] \in \mathrm{TS}(\overline{G};\overline{Q_4}), \forall s \in [\overline{N}_{\overline{G}}(\overline{Q_3})]_{2'}.$$

First, notice that by Remark 3.2.6(f) the above argument yields

$$\dim_k(M[\overline{Q_3}]) = \tau^{\overline{G}}_{\overline{Q_3},1}([M]) = 1 \quad \forall\ [M] \in \mathrm{TS}(\overline{G};\overline{Q_4}).$$

Now, $\overline{N}_{\overline{G}}(\overline{Q_3})$ has a unique trivial source module with vertex C_2, namely the trivial module. Indeed, this follows from Proposition 3.1.11(d) as $\overline{N}_{\overline{G}}(\overline{Q_3}) \cong D_6$, so that subgroups of order 2 are conjugate and self-normalising. Therefore, we conclude that $M[\overline{Q_3}] = k$ for every $[M] \in \mathrm{TS}(\overline{G};\overline{Q_4})$. In all cases, by definition $\tau^{\overline{G}}_{\overline{Q_3},s}([M])$ is equal to the Brauer character of the trivial $k\overline{N}_{\overline{G}}(\overline{Q_3})$-module evaluated at s, hence equal to 1, proving the claim.

- **The matrix** $T_{3,3}$. Because $\overline{N}_{\overline{G}}(\overline{Q_4}) \cong C_3$, by Remark 3.2.6(b) the matrix $T_{3,3}$ of $\mathrm{Triv}_2(\overline{G})$ is just the ordinary character table of the cyclic group C_3. □

The trivial source character table $\mathrm{Triv}_2(G) = [T_{i,v}]_{1\leq i,v\leq 4}$ of $G = \mathrm{SL}_2(13)$ is now up to a large extent obtained via inflation from $\overline{G}$. For this reason, we write $T_{i,v}(G)$ for the matrix $T_{i,v}$ of $\mathrm{Triv}_2(G)$ and $T_{i,v}(\overline{G})$ for the matrix $T_{i,v}$ of $\mathrm{Triv}_2(\overline{G})$.

Theorem 4.3.17. *Assume* $G = \mathrm{SL}_2(13)$. *Then the trivial source character table* $\mathrm{Triv}_2(G) = [T_{i,v}]_{1\leq i,v\leq 4}$ *is given as follows.*

(a) *The following holds:*
  (i) $T_{i,v} = \mathbf{0}$ *for every* $1 \leq i < v \leq 4$*;*
  (ii) $T_{i,1}(G) = T_{i,2}(G) = T_{i-1,1}(\overline{G})$ *for every* $2 \leq i \leq 4$*;*
  (iii) $T_{3,3}(G) = T_{2,2}(\overline{G}),\ T_{4,3}(G) = T_{3,2}(\overline{G}),\ T_{4,4}(G) = T_{3,3}(\overline{G})$.

(b) *Set* $B := -\exp\left(\frac{2\pi i}{7}\right) - \exp\left(\frac{12\pi i}{7}\right)$, $C := -\exp\left(\frac{6\pi i}{7}\right) - \exp\left(\frac{8\pi i}{7}\right)$, *and* $D := -\exp\left(\frac{4\pi i}{7}\right) - \exp\left(\frac{10\pi i}{7}\right)$. *The matrix* $T_{1,1}$ *is as given in Table 4.15.*

	I_2	$\begin{pmatrix}\zeta_{13,4} & 0\\ 0 & \zeta_{13,8}\end{pmatrix}$	$\begin{pmatrix}\zeta_{13,8} & \zeta_{13,1}\\ 1 & \zeta_{13,8}\end{pmatrix}^2$	$\begin{pmatrix}\zeta_{13,8} & \zeta_{13,1}\\ 1 & \zeta_{13,8}\end{pmatrix}^4$	$\begin{pmatrix}\zeta_{13,8} & \zeta_{13,1}\\ 1 & \zeta_{13,8}\end{pmatrix}^6$	u_+	u_-
$\chi_1+\chi_2+2\chi_5+\chi_{14}+\chi_{15}$	56	8	0	0	0	4	4
$\chi_2+\chi_5+\chi_{14}+\chi_{16}$	40	4	-2	-2	-2	$1+\sqrt{13}$	$1-\sqrt{13}$
$\chi_2+\chi_5+\chi_{15}+\chi_{17}$	40	4	-2	-2	-2	$1-\sqrt{13}$	$1+\sqrt{13}$
$\chi_8+\chi_{11}$	24	0	$2D$	$2C$	$2B$	-2	-2
$\chi_9+\chi_{12}$	24	0	$2B$	$2D$	$2C$	-2	-2
$\chi_{10}+\chi_{13}$	24	0	$2C$	$2B$	$2D$	-2	-2
$\chi_3+\chi_4+\chi_6+\chi_7$	56	-4	0	0	0	4	4

Table 4.22: $T_{1,1}$ **for** $G = \mathrm{SL}_2(13)$ **and** $p = 2$

Proof. (a) Again, the fact that $T_{i,v} = \mathbf{0}$ for every $1 \leq i < v \leq 4$ is immediate from Remark 3.2.6(d). This proves (i).

Next, we notice that the subgroups Q_2, Q_3, Q_4 all contain the centre $Z = Q_2$. The following assertions follow immediately.

1. For every $2 \leq i \leq 4$ and every $M \in \mathrm{TS}(G, Q_i)$ we have $M[Q_2] = M$ (as kN_2-modules). Therefore, as our choices of $[\overline{N}_1]_{2'}$ and $[\overline{N}_2]_{2'}$ agree modulo Z, by definition of the species we have $T_{i,1}(G) = T_{i,2}(G)$ for every $2 \leq i \leq 4$.
2. For every $2 \leq i \leq 4$, we have $Q_i/Z = \overline{Q_i}$, and so any trivial source module in $\mathrm{TS}(G, Q_i)$ is the inflation from $\overline{G} = G/Z$ to G of a trivial source $k\overline{G}$-module with vertex $\overline{Q_i}$, i.e.

$$\mathrm{TS}(G, Q_i) = \left\{ \mathrm{Inf}_{\overline{G}}^{G}(M) \mid M \in \mathrm{TS}(\overline{G}, \overline{Q_i}) \right\}$$

and the corresponding characters are

$$\chi_{\widehat{\mathrm{Inf}_{\overline{G}}^{G}(M)}} = \mathrm{Inf}_{\overline{G}}^{G}(\chi_{\widehat{M}}).$$

It follows that $T_{i,2}(G) = T_{i-1,1}(\overline{G})$ for every $2 \leq i \leq 4$ and

$$T_{3,3}(G) = T_{2,2}(\overline{G}),\ T_{4,3}(G) = T_{3,2}(\overline{G}),\ T_{4,4}(G) = T_{3,3}(\overline{G})$$

because our choices of the representatives of the $2'$-conjugacy classes agree modulo Z, proving (ii) and (iii).

(b) The ordinary characters of the PIMs of $B_0(G)$ can be read from the 2-decomposition matrix in Proposition 4.3.13. □

Remark 4.3.18. The general case, where, among other cases, the families of groups $\mathrm{SL}_2(q)$ and $\mathrm{PSL}_2(q)$ for $q \equiv \pm 3 \bmod 8$ are considered in characteristic 2, can be found in [BFL22].

Chapter 5

Algorithms

In this chapter, we describe strategies and algorithms that compute trivial source modules and trivial source character tables. We present the necessary theoretical background, describe how the computer algebra systems GAP and MAGMA deal with finite fields and Brauer characters, and explain how we compute projective indecomposable modules with GAP. Moreover, we present an algorithm allowing us to compute trivial source character tables recursively.

We assume that at least version 4.12.2 of GAP (see [GAP]), at least version 2.26 of MAGMA (see [BCP97]), and at least version 1.0.1 of Shared C MeatAxe (see https://users.fmi.uni-jena.de/~king/SharedMeatAxe/index.html) is installed on the used computer.

5.1 Theoretical background

This section is concerned with some theoretical concepts which simplify later computer calculations. We illustrate how trivial source modules, as well as p-blocks of group algebras, behave under scalar extensions. Furthermore, we introduce peakwords and explain why they faciliate the determination of projective indecomposable modules with GAP. Throughout this section, let F be a field of characteristic $p > 0$.

Convention 5.1.1. A field extension $\widetilde{\mathrm{F}}/\mathrm{F}$ of a field $\widetilde{\mathrm{F}}$ containing F is called a **Galois field extension** if the extension is finite, normal, and separable. Its **Galois group** is denoted by $\mathrm{Gal}(\widetilde{\mathrm{F}}/\mathrm{F})$.

5.1.1 Behaviour of modules under ground field extensions

In this subsection, we examine how modules behave under ground field extensions. These considerations are relevant for our computer implementations. In particular, we apply the theory developed here only to the special case of finite fields. We begin with the notions of absolutely irreducible and absolutely indecomposable modules.

Definition 5.1.2. Let A be an F-algebra, let S be a simple A-module, and let ρ_S be its underlying representation.

(a) The A-module S is called **absolutely simple** over F if $S \otimes_{\mathrm{F}} \widetilde{\mathrm{F}}$ is a simple $A \otimes_{\mathrm{F}} \widetilde{\mathrm{F}}$-module for every field $\widetilde{\mathrm{F}}$ containing F.

(b) The representation ρ_S is called **absolutely irreducible** over F, if $\rho_{S\otimes_{\mathrm{F}}\widetilde{\mathrm{F}}}$ is an irreducible representation for every field $\widetilde{\mathrm{F}}$ containing F.

Proposition 5.1.3 ([Web16, Proposition 9.2.5])**.** *Let S be a simple module over a finite-dimensional F-algebra A. The following are equivalent.*

(a) *The A-module S is absolutely simple.*

(b) *We have* $\mathrm{End}_A(S) \cong \mathrm{F}$.

Definition 5.1.4. Let A be an F-algebra and let S be a simple A-module. A field $\widetilde{\mathrm{F}}$ containing F is called a **splitting field for** S if $S \otimes_{\mathrm{F}} \widetilde{\mathrm{F}}$ is isomorphic to a direct sum of absolutely simple $A \otimes_{\mathrm{F}} \widetilde{F}$-modules. Moreover, $\widetilde{\mathrm{F}}$ is called a **splitting field for** A, if all simple $A \otimes_{\mathrm{F}} \widetilde{F}$-modules are absolutely simple.

Let C be an algebra over a commutative ring R and let U be a C-module. If $R \to R'$ is a homomorphism to another commutative ring R', we may form the R'-algebra $C \otimes_R R'$. Then, the module $U \otimes_R R'$ can be seen as a $C \otimes_R R'$-module.

Definition 5.1.5. Let R be a subring of R'. If U is a C-module, we say that the module $V = U \otimes_R R'$ is obtained from U by **extending the scalars from** R **to** R'. If a $C \otimes_R R'$-module V has the form $U \otimes_R R'$, we say that it **can be written in** R.

In this situation, when U is free as an R-module, we may identify U with the subset $U \otimes_R 1_{R'}$ of $U \otimes_R R'$, and an R-basis of U becomes an R'-basis of $U \otimes_R R'$ under this identification.

Proposition 5.1.6 ([Web16, Proposition 9.2.1])**.** *Let* $\widetilde{\mathrm{F}}/\mathrm{F}$ *be an algebraic field extension and let* V *be a finite-dimensional* $A \otimes_{\mathrm{F}} \widetilde{\mathrm{F}}$*-module. Then there exists an intermediate field* $\mathrm{F} \subseteq \mathrm{F}' \subseteq \widetilde{\mathrm{F}}$ *with* $[\mathrm{F}' : \mathrm{F}] < \infty$ *such that* V *can be written in* F'*.*

Definition 5.1.7. Let A be an F-algebra. The A-module M is called **absolutely indecomposable** if $M \otimes_{\mathrm{F}} \widetilde{F}$ is an indecomposable $A \otimes_{\mathrm{F}} \widetilde{F}$-module for every field $\widetilde{\mathrm{F}}$ containing F.

Theorem 5.1.8 ([CR90, (30.29) Theorem])**.** *Let* F *be a perfect field. Let* A *be an* F*-algebra, and let* M *be an* A*-module. The following statements are equivalent.*

(a) *The* A*-module* M *is absolutely indecomposable.*

(b) *We have* $\mathrm{End}_A(M)/J(\mathrm{End}_A(M)) \cong \mathrm{F}$.

Theorem 5.1.9 ([Kar92, Theorem 13.4.5 & Corollary 13.4.7])**.** *Let* A *be an* F*-algebra, let* $\widetilde{\mathrm{F}}/\mathrm{F}$ *be a Galois field extension and let* M *be an indecomposable* A*-module. Then the following properties hold.*

(a) *If* W *is an indecomposable direct summand of* $M \otimes_{\mathrm{F}} \widetilde{\mathrm{F}}$*, then there exists a positive integer* e *such that*
$$M \otimes_{\mathrm{F}} \widetilde{\mathrm{F}} \cong e \cdot \left(\bigoplus_{i=1}^{r} W^{\sigma_i} \right)$$
where $\{\sigma_1, \ldots, \sigma_r\}$ *is a left transversal for the inertia group* H *of* W *in* $\mathrm{Gal}(\widetilde{\mathrm{F}}/\mathrm{F})$*. Moreover,* $\{W^{\sigma_i} \mid 1 \leq i \leq r\}$ *is a set of non-isomorphic Galois conjugates of* W*.*

(b) *If* $\mathrm{End}_A(M)/J(\mathrm{End}_A(M))$ *is isomorphic to a field, then* $e = 1$*. In this case, there is a decomposition* $M \otimes_{\mathrm{F}} \widetilde{\mathrm{F}} = U_1 \oplus \ldots \oplus U_r$ *of* $M \otimes_{\mathrm{F}} \widetilde{\mathrm{F}}$ *into indecomposable submodules* U_i $(1 \leq i \leq r)$ *such that the* U_i $(1 \leq i \leq r)$ *are permuted transitively by* $\mathrm{Gal}(\widetilde{\mathrm{F}}/\mathrm{F})$*.*

Theorem 5.1.10 ([Kar92, Theorem 14.2.5])**.** *Let* F *be a perfect field, let* A *be an* F*-algebra, let* V *be a simple* A*-module, and let* $\widetilde{\mathrm{F}}/\mathrm{F}$ *be a Galois field extension. Then there exists a positive integer* e *and a simple* $A \otimes_{\mathrm{F}} \widetilde{\mathrm{F}}$*-module* W *such that the following properties hold.*

(a) *The $A \otimes_{\mathrm{F}} \widetilde{\mathrm{F}}$-module $V \otimes_{\mathrm{F}} \widetilde{\mathrm{F}}$ is semisimple.*

(b) *We have $V \otimes_{\mathrm{F}} \widetilde{\mathrm{F}} \cong e \cdot \left(\bigoplus_{i=1}^{r} W^{\sigma_i} \right)$ where $\sigma_i \in \mathrm{Gal}(\widetilde{\mathrm{F}}/\mathrm{F})$ and $\{W^{\sigma_i} \mid 1 \leq i \leq r\}$ is a set of non-isomorphic Galois conjugates of W.*

(c) *Each simple $A \otimes_{\mathrm{F}} \widetilde{\mathrm{F}}$-module U occurs as a direct summand in the decomposition of $M \otimes_{\mathrm{F}} \widetilde{\mathrm{F}}$ into indecomposable submodules for some unique, up to isomorphism, simple A-module M.*

(d) *If A can be written in a finite subfield of F, then $e = 1$.*

Lemma 5.1.11. *Let V be an indecomposable $\mathbb{F}_q G$-module and let $\mathbb{F}_{q'}$ be a finite extension of $\mathbb{F}_q$. Assume U is a vertex of V. Then U is a vertex of every component of $V \otimes \mathbb{F}_{q'}$.*

Proof. Analogous to the proof of [Fei82, Lemma 4.14]. □

Corollary 5.1.12. If M is a p-permutation $\mathbb{F}_p G$-module, then $M \otimes_{\mathbb{F}_p} \mathbb{F}_q$ is a p-permutation $\mathbb{F}_q G$-module.

Proof. This follows from the fact that extension of scalars commutes with Ind_Q^G. □

Lemma 5.1.13. *Let A be an $\mathbb{F}_p$-algebra and let $\mathbb{F}_q$ be a finite extension of $\mathbb{F}_p$. Then the extension of scalars functor $- \otimes_{\mathbb{F}_p} \mathbb{F}_q : \mathrm{mod}_A \to \mathrm{mod}_{A \otimes_{\mathbb{F}_p} \mathbb{F}_q}$ is an exact functor.*

Proof. As $- \otimes_{\mathbb{F}_p} \mathbb{F}_q$ is a tensor functor, it is right exact. Since $\mathbb{F}_q$ is free and hence flat as an $\mathbb{F}_p$-module, $- \otimes_{\mathbb{F}_p} \mathbb{F}_q$ is also left exact. □

Lemma 5.1.14. *Let S be a simple A-module and let P be the projective cover of S. Set $\widetilde{A} := A \otimes_{\mathbb{F}_p} \mathbb{F}_q$, $\widetilde{S} := S \otimes_{\mathbb{F}_p} \mathbb{F}_q$, and $\widetilde{P} := P(S) \otimes_{\mathbb{F}_p} \mathbb{F}_q$. Then $\widetilde{P}$ is a projective $\widetilde{A}$-module such that $\mathrm{Hd}(\widetilde{P}) \cong \widetilde{S}$.*

Proof. By Lemma 5.1.11 all occurring summands have trivial vertices, hence $\widetilde{P}$ is a projective $\widetilde{A}$-module. Note that we have

$$J(\widetilde{P}) = J(P \otimes \mathbb{F}_q) = P \otimes 0 + J(P) \otimes \mathbb{F}_q = J(P) \otimes \mathbb{F}_q.$$

By Lemma 5.1.13 it follows that the sequence

$$0 \to J(\widetilde{P}) \to \widetilde{P} \to \widetilde{S} \to 0$$

is exact. Hence $\widetilde{P}$ maps surjectively onto $\widetilde{S}$ and $\mathrm{Hd}(\widetilde{P}) \cong \widetilde{S}$. □

Lemma 5.1.15 ([Wis91, § 22.2])**.** *Let A be an F-algebra. Let P be a projective A-module. Then*

$$\mathrm{End}_A(P)/J(\mathrm{End}_A(P)) \cong \mathrm{End}_A(P/J(P)).$$

The following proposition is used frequently during our computer calculations.

Proposition 5.1.16. *Let A be an algebra over $\mathbb{F}_p$ and let S be a simple A-module. Suppose that $\mathbb{F}_{q'}$ is a finite extension of $\mathbb{F}_p$ such that $\widetilde{S} := S \otimes \mathbb{F}_{q'}$ decomposes into a direct sum of m absolutely indecomposable modules, i.e. $\widetilde{S} \cong \widetilde{S}_1 \oplus \ldots \oplus \widetilde{S}_m$ as $A \otimes \mathbb{F}_{q'}$-modules for some $m \in \mathbb{Z}_{\geq 1}$. Then also $\widetilde{P} := P(S) \otimes \mathbb{F}_{q'}$ decomposes into a direct sum of m absolutely indecomposable modules.*

Proof. Let $\widetilde{A} := A \otimes \mathbb{F}_{q'}$. By Lemma 5.1.14 and Lemma 5.1.15 we have

$$\mathrm{End}_{\widetilde{A}}(\widetilde{P})/J(\mathrm{End}_{\widetilde{A}}(\widetilde{P})) \cong \mathrm{End}_{\widetilde{A}}(\widetilde{P}/J(\widetilde{P})) \cong \mathrm{End}_{\widetilde{A}}(\widetilde{S}) \cong \underbrace{\mathbb{F}_{q'} \times \ldots \times \mathbb{F}_{q'}}_{m \text{ times}}.$$

□

Remark 5.1.17. Hence, this gives us one possibility to compute matrix representations of PIMs over smallest possible fields.

5.1.2 Projective modules via peakwords

This subsection is oriented on [LMR94], [Kal09], and [Hof04].

Let $\mathbb{F}$ be a large enough finite field of positive characteristic $p > 0$ and let A be an algebra over $\mathbb{F}$. In this section, we describe a method to compute projective indecomposable A-modules using GAP and the Shared C MeatAxe. We begin with an explanation of the general idea.

Let V be an A-module and $e \in A$ be an idempotent element. The functorial relationship between the module categories of A and eAe is well-known. Instead of aiming at a Morita-equivalence, it has turned out that it is more practical to choose e to be a primitive idempotent element and to consider each irreducible A-module seperately. It would be very difficult to compute primitive idempotent elements explicitly. However, under suitable conditions, it is sufficient to know their action on V. In the following, we define the Fitting projection and prove that the idempotent element e corresponding to the Fitting projection with respect to an element $a \in A$ is a primitive idempotent element if a is a so-called peakword. First, we need a lemma.

Lemma 5.1.18 ([LMR94, Lemma 2.1])**.** *Let $e \in A$ be an idempotent element and let V be an A-module.*

(a) *If W is an A-submodule of V, then We is an eAe-submodule of Ve and we have*

$$(V/W)e \cong Ve/We$$

as eAe-modules. Conversely, if $\widetilde{W}$ is an eAe-submodule of Ve, then $W := \widetilde{W} \cdot A$ is an A-submodule of V such that $We = \widetilde{W}$.

(b) *If S is a simple A-module, then either $Se = \{0\}$ or Se is a simple eAe-module.*

Definition 5.1.19. For an A-module V let $\rho_V : A \to \mathrm{End}_{\mathbb{F}}(V), a \mapsto a_V$ be the underlying representation. By the Fitting decomposition theorem, for every $a \in A$ there exists an $N \in \mathbb{Z}_{\geq 1}$ such that $\mathrm{Ker}(a_V^N) = \mathrm{Ker}(a_V^{N+1})$, $\mathrm{Im}(a_V^N) = \mathrm{Im}(a_V^{N+1})$, and $V = \mathrm{Ker}(a_V^N) \oplus \mathrm{Im}(a_V^N)$ as vector spaces. The **Fitting projection** from V onto $\mathrm{Ker}(a_V^N)$ with respect to the complement $\mathrm{Im}(a_V^N)$ is then defined as the $\mathbb{F}_p$-morphism

$$FIT: \quad V = \mathrm{Ker}(a_V^N) \oplus \mathrm{Im}(a_V^N) \to \mathrm{Ker}(a_V^N), \quad (x, y) \mapsto x.$$

Definition 5.1.20. Let S be a simple A-module.

(a) An A-module V is called S-**local** if $V/J(V) \cong S$ as A-modules.

(b) A primitive idempotent element $e \in A$ is called S-**primitive** if the module eA is S-local.

Example 5.1.21. The projective cover $P(S)$ of a simple A-module S is S-local.

Theorem 5.1.22 ([LMR94, Theorem 2.2(a)])**.** *Let $e \in A$ be an idempotent element such that $(eA)/J(eA)$ has S as its only composition factor. Then* $\dim_{\mathbb{F}}(\mathrm{End}_A(S))$ *divides* $\dim_{\mathbb{F}}(Se)$*. Equality holds if and only if e is S-primitive.*

Definition 5.1.23. Let V be a faithful A-module and let S be a composition factor of V. An element $a \in A$ is called an S-**peakword** with respect to V if and only if the following conditions are fulfilled:

(a) $\mathrm{Ker}(a_T) = 0$ for each composition factor T of V which is not isomorphic to S;

(b) $\dim_{\mathbb{F}}(\mathrm{Ker}(a_S^2)) = \dim_{\mathbb{F}}(\mathrm{End}_A(S))$.

Proposition 5.1.24 ([Mül03, (2.5) Proposition])**.** *Let S be a simple A-module. Then there exists an S-peakword with respect to A^{reg} in A.*

Theorem 5.1.25 ([LMR94, Theorem 3.1 & Theorem 3.4])**.** *Let V be a faithful A-module and let $a \in A$. Then there exists an element $e \in A$ which induces the Fitting projection on V with respect to a, i.e.* $\mathrm{Ker}(a_V^N) = Ve$ *and* $\mathrm{Im}(a_V^N) = V(1-e)$*. Furthermore, e is uniquely determined and an idempotent element. If a is an S-peakword with respect to V, then e is S-primitive.*

Definition 5.1.26. If a is an S-peakword with respect to V and e the associated idempotent element as in Theorem 5.1.25, then $\mathrm{Ker}(a_V^N) = Ve$ is called the **stable peakword kernel** of the S-peakword a.

Definition 5.1.27. Let S be a simple A-module and let $e \in A$ be an S-primitive idempotent element. We define

$$\mathcal{M}_S(V) := \{A\text{-submodules } W \text{ of } V \text{ such that } W/J(W) \cong \bigoplus_{i=1}^{m} S \text{ for some } m \in \mathbb{Z}_{\geq 1}\}/\cong$$

and

$$\mathcal{M}(Ve) := \{eAe\text{-submodules of } Ve\}/\cong.$$

Theorem 5.1.28 ([LMR94, Theorem 2.1])**.** *We keep the assumptions of Definition 5.1.27. The map*

$$\kappa : \mathcal{M}_S(V) \to \mathcal{M}(Ve), \quad W \mapsto We$$

is an isomorphism of submodule lattices. Its inverse is given by

$$\kappa^{-1} : \mathcal{M}(Ve) \to \mathcal{M}_S(V), \quad \widetilde{W} \mapsto \widetilde{W} \cdot A.$$

Next, we consider the so-called spinning algorithm (cf. [LP10]). Suppose that k is a field, that A is a finite-dimensional k-algebra with algebra generators $(a_1, \ldots, a_r)$, and that V is an A-module (given by the action of these generators on V). Then, given a non-zero vector $v \in V$, Algorithm 1 computes a basis B of the A-module $\langle v\rangle_{A-\mathrm{span}}$.

Algorithm 1 Spinning algorithm

```
Input: 0 ≠ v ∈ V and algebra generators (a_1, ..., a_r) of A
Output: a k-basis B for ⟨v⟩_{A−span}
Initialise: B ← (v)
for b in B do
    for i in {1, ..., r} do
```

$w := b \cdot a_i$
if w is not contained in $\langle B \rangle_{k-\text{span}}$ **then**
Append w to the list B
return B

Remark 5.1.29. Algorithm 1 terminates since V was supposed to be finite-dimensional. By construction, B is linearly independent over k and $B \subseteq v \cdot A$. Furthermore, $\langle B \rangle_{k-\text{span}}$ is invariant under the action of the generators of A and hence under A. Thus $v \cdot A = \langle B \rangle_{k-\text{span}}$.

Definition 5.1.30. The process of calculating $v \cdot A$ using Algorithm 1 is usually referred to as **spinning up the vector** v.

Proposition 5.1.31. *Let $A \in \mathrm{Bl}(\mathbb{F}_p G)$, let V denote the regular A-module, and S be a composition factor of V. Let e_S be an S-primitive idempotent element of A. Then, every $\mathbb{F}_p$-basis of Ve_S contains a vector that spins up to the S-local submodule $P(S)$ of V.*

Proof. Let U denote the sum of all submodules $Z_i \leq V$ of V which do not contain any submodule isomorphic to $P(S)$. Since A is a symmetric algebra and $P(S)$ is a projective indecomposable A-module, we can replace the word submodule by the word summand in the definition of U. We claim that U is a submodule of V, maximal with respect to the property that it does not contain any submodule isomorphic to $P(S)$: assume for a contradiction that $P(S)$ does occur in a direct sum decomposition of U. Then we can write

$$U = P(S) \oplus L = \sum_{Z_i \leq V} Z_i$$

for some A-module L and the A-modules Z_i as above. Intersecting both sides of this equation with $P(S)$, we deduce that

$$P(S) \cap \sum_{Z_i \leq V} Z_i \not\subseteq J(P(S)).$$

Therefore, there is an index j such that there exists an element $z \in Z_j$ which is an element of $P(S) \setminus J(P(S))$. Therefore,

$$J(P(S)) \subsetneq J(P(S)) + \langle z \rangle = P(S),$$

since $P(S)/J(P(S)) \cong S$. Hence, it follows from Lemma 2.1.15 that $\langle z \rangle = P(S)$. Consequently, $P(S)$ is a summand of Z_j which contradicts the definition of U. It is, moreover, clear from the definition of U that U is a maximal submodule of V, as every module not containing a summand isomorphic to $P(S)$ is contained in U.
It follows from Lemma 5.1.18 and Theorem 5.1.22 that

$$\begin{aligned} \dim_{\mathbb{F}}(Ve_S) - \dim_{\mathbb{F}}(Ue_S) &= \dim_{\mathbb{F}}(Ve_S/Ue_S) = \dim_{\mathbb{F}}((V/U)e_S) \\ &= |\{\text{summands of } V \text{ isomorphic to } P(S)\}/\cong| \cdot \dim_{\mathbb{F}}(Se_S) \\ &\geq 1. \end{aligned}$$

But, as e_S is S-primitive, each vector in $Ve_S \setminus Ue_S$ spins up to an S-local submodule W of V which does contain a submodule M isomorphic to $P(S)$. But M is then a direct summand of W, since A is a symmetric algebra. Hence, each vector in $Ve_S \setminus Ue_S$ spins up to a module which is isomorphic to $P(S)$. Moreover, every $\mathbb{F}_p$-basis of Ve_S obviously contains a vector lying in $Ve_S \setminus Ue_S$. ☐

In the following, we describe the strategy we used in our GAP algorithm (see Section 7.1) in order to compute all projective indecomposable $\mathbb{F}_pG$-modules.

Strategy 5.1.32. 1. Compute the block idempotent elements of $\mathbb{F}_pG$. This is described in Section 5.1.3.

2. Regard each block B of $\mathbb{F}_pG$ as an algebra over $\mathbb{F}_p$ and set $V := B^{\text{reg}}$. Notice that the B-module V is faithful.
3. Compute a composition series of V.
4. Fix a simple composition factor S of V and compute an S-peakword $a \in B$. It is described in the article [LMR94] how to achieve this. Recall that such an S-peakword exists by Proposition 5.1.24.
5. Compute the stable peakword kernel Ve of a, where e is the S-primitive idempotent element corresponding to a.
6. Compute a basis $\mathcal{B}$ of Ve.
7. Compute the $\mathbb{F}_p$-dimension of $P(S)$ from the knowledge of S and the decomposition matrix of $\mathbb{F}_qG$.
8. Spin up elements v of $\mathcal{B}$ until the dimension of $\langle v \rangle$ equals the dimension of $P(S)$.

Output: matrix representations for a set of representatives of the PIMs of $\mathbb{F}_pG$

Remark 5.1.33. By Proposition 5.1.31 and the fact that the $\mathbb{F}_pG$-module $P(S)$ is S-local, we deduce that this algorithm terminates.

We conclude this section with a method to obtain all PIMs of G over a field $\mathbb{F}_q$ which is a splitting field for $\mathbb{F}_pG$.

Strategy 5.1.34. Let P be a projective indecomposable $\mathbb{F}_pG$-module.

1. Compute an $\mathbb{F}_q$-basis of $C_1 := \text{Hom}_{A\otimes_{\mathbb{F}_p}\mathbb{F}_q}(P \otimes_{\mathbb{F}_p} \mathbb{F}_q, P \otimes_{\mathbb{F}_p} \mathbb{F}_q)$.
2. Cosinder C_1 as subalgebra of $C := \text{Mat}_{n\times n}(\mathbb{F}_q)$ where $n := \dim_{\mathbb{F}_q}(P \otimes_{\mathbb{F}_p} \mathbb{F}_q)$.
3. Compute a complete set $\widetilde{\mathcal{S}}$ of primitive orthogonal idempotents of $C_2 := C_1/\text{Rad}(C_1)$ and lift it to a set $\mathcal{S}$ of primitive idempotents for C_1.
4. Note that the elements of $\mathcal{S}$ are matrices $M_1, ..., M_t \in \text{End}_{A\otimes_{\mathbb{F}_p}\mathbb{F}_q}(P \otimes_{\mathbb{F}_p} \mathbb{F}_q)$.
5. For each $1 \leq j \leq t$, compute with GAP the submodule Z_j of $P \otimes_{\mathbb{F}_p} \mathbb{F}_q$ generated by $\text{Im}(M_j)$.

Output: matrix representations for a set of representatives of those PIMs of $\mathbb{F}_qG$ which occur in a direct sum decomposition of $P \otimes_{\mathbb{F}_p} \mathbb{F}_q$

Issue 5.1.35. Since the computation of C_1 involves the solution of a lot of equations over the field $\mathbb{F}_q$, this can easily become very time consuming.

Solution. The computation of C_1 is often avoided or not necessary: for example, if the group G is solvable, then we can use Proposition 5.4.12 instead of computing C_1. Moreover, it happens quite often that a GAP computation reveals that $\mathbb{F}_p$ is already a splitting field for many simple $\mathbb{F}_pG$-modules.

In the following, we describe how we compute the block idempotent elements of $\mathbb{F}_pG$ in our algorithms.

5.1.3 Block idempotent elements

In this subsection, let G be an arbitrary finite group of exponent m and let p be a prime number dividing $|G|$. Let $\xi_m := e^{\frac{2\pi i}{m}}$. In order to obtain the subdivision of $\mathbb{F}_pG$ into p-blocks, we compute the modular block idempotents of $\mathbb{F}_pG$. In the sequel, we describe a way to accomplish this. First, we need some auxiliary means.

Definition 5.1.36. For every $\chi \in \mathrm{Irr}(G)$ we define $e_\chi := \frac{\chi(1)}{|G|}\sum_{g\in G}\chi(g^{-1})\cdot g$.

Remark 5.1.37. (a) Let $\rho : G \to \mathrm{GL}_n(\mathbb{C})$ be a representation affording the ordinary irreducible character χ. It is well-known that e_χ is the idempotent element in $\mathbb{C}G$ which corresponds to ρ.

(b) We emphasise that $e_\chi \in Z(\mathbb{Q}(\xi_m)[G])$.

Definition 5.1.38. We set $\widetilde{R} := \mathbb{Z}[\xi_m]_{\mathfrak{P}}$, where $\mathfrak{P}$ is a prime ideal of $\mathbb{Z}[\xi_m]$ with $p \in \mathfrak{P}$.

Note that $\widetilde{R}$ is a ring of local integers for p.

Definition 5.1.39. Let $B \in \mathrm{Bl}(kG)$. Let $\mathrm{Irr}_K(B)$ be the set of those ordinary irreducible characters of G which lie in B. We define $e_B := \sum_{\chi\in\mathrm{Irr}_K(B)} e_\chi$.

Note that in general $e_\chi \notin \widetilde{R}[G]$, but we have the following result:

Theorem 5.1.40 ([Isa06, (15.26) Theorem])**.** *Let $B \in \mathrm{Bl}(kG)$. Write $e_B = \sum_{g\in G} a_g g$.*

(a) *We have $a_g = \frac{1}{|G|}\sum_{\chi\in\mathrm{Irr}_K(B)}\chi(1)\chi(g^{-1})$.*

(b) *We have $a_g \in \widetilde{R}$ for all $g \in G$.*

(c) *We have $a_g = 0$ if $p \mid o(g)$.*

Remark 5.1.41. Recall that $\widetilde{R} = \mathbb{Z}[\xi_m]_{\mathfrak{P}} = \{\frac{a}{b} \mid a, b \in \mathbb{Z}[\xi_m] \text{ and } b \notin \mathfrak{P}\}$. Hence, with the aid of the usual norm function $\mathfrak{N}_{\mathbb{Q}(\xi_m)/\mathbb{Q}}$, we can write $\frac{a}{b} \in \mathbb{Z}[\xi_m]_{\mathfrak{P}}$ as

$$\frac{a}{b} = \frac{a\cdot(\mathfrak{N}_{\mathbb{Q}(\xi_m)/\mathbb{Q}}(b)/b)}{\mathfrak{N}_{\mathbb{Q}(\xi_m)/\mathbb{Q}}(b)} = \frac{a\cdot\left(\prod_{\sigma\in\mathrm{Gal}(\mathbb{Q}(\xi_m)/\mathbb{Q}),\ \sigma\neq 1}\sigma(b)\right)}{\mathfrak{N}_{\mathbb{Q}(\xi_m)/\mathbb{Q}}(b)} = \frac{a'}{b'}$$

for some $a' \in \mathbb{Z}[\xi_m]$ and some $b' \in \mathbb{Z}$ with $p \nmid b'$. Hence, although $e_\chi \notin \widetilde{R}[G]$ in general, it is nevertheless always possible to compute the $\mathfrak{p}$-modular reduction of the idempotent element e_B.

This is crucial for an algorithmic computation of the block idempotent elements of $\mathbb{F}_qG$. Next, assume given the block idempotent elements of $\mathbb{F}_qG$. Then, it is possible to compute the block idempotent elements of $\mathbb{F}_pG$ as follows.

Since the Galois group $\mathrm{Gal}(\mathbb{F}_q/\mathbb{F}_p)$ acts via $\mathbb{F}_p$-algebra automorphisms on the group algebra $\mathbb{F}_qG$ and also on $Z(\mathbb{F}_qG)$, it permutes the block idempotent elements of $\mathbb{F}_qG$.

Theorem 5.1.42 ([BKY20, Proposition 4.1])**.** (a) *Let b be a block idempotent element of $\mathbb{F}_qG$. Then*

$$\widetilde{b} := \mathrm{tr}_{\mathrm{Gal}(\mathbb{F}_q/\mathbb{F}_p)}(b) := \sum_{\sigma\in\mathrm{Gal}(\mathbb{F}_q/\mathbb{F}_p)/\mathrm{Stab}_{\mathrm{Gal}(\mathbb{F}_q/\mathbb{F}_p)}(b)} {}^{\sigma}b$$

is a block idempotent element of $\mathbb{F}_pG$.

(b) *The assignment* $b \mapsto \widetilde{b}$ *induces a bijection between the set of* $\mathrm{Gal}(\mathbb{F}_q/\mathbb{F}_p)$*-orbits of block idempotent elements of* $\mathbb{F}_qG$ *and the set of block idempotent elements of* $\mathbb{F}_pG$*.*

(c) *If* b *is a block idempotent element of* $\mathbb{F}_qG$ *and* $\widetilde{b} := \mathrm{tr}_{\mathrm{Gal}(\mathbb{F}_q/\mathbb{F}_p)}(b)$ *is the block idempotent element of* $\mathbb{F}_pG$ *associated to it as in part* (a)*, then* b *and* $\widetilde{b}$ *have a common defect group.*

This leads to the following method.

Strategy 5.1.43. 1. For each $\chi \in \mathrm{Irr}_K(G)$ compute e_χ.

2. For each $B \in \mathrm{Bl}(kG)$ compute e_B.

3. For each e_B, compute f_B, the reduction modulo $\mathfrak{p}$ of e_B, by reading the coefficients of e_B in $\mathbb{F}_q$.

4. Fix f_B. Repeatedly apply the Frobenius automorphism to each coefficient of f_B until the element f_B is obtained again. The sum of all calculated distinct elements is equal to a block idempotent element e of $\mathbb{F}_pG$. By Theorem 5.1.42, every block idempotent element of $\mathbb{F}_pG$ can be computed in that way.

Output: a complete set of block idempotent elements of $\mathbb{F}_pG$

We finish this section by explaining a method to obtain the regular module B^{reg} of a block $B \in \mathrm{Bl}(\mathbb{F}_pG)$. Equipped with the block idempotent elements $e_1, \dots, e_r$ of $\mathbb{F}_pG$ we compute the regular representations of the blocks of $\mathbb{F}_pG$ as follows. We extend

$$\rho^{\mathrm{reg}} : G \to \mathrm{GL}_n(\mathbb{F}_p), \quad g \mapsto \rho^{\mathrm{reg}}(g)$$

$\mathbb{F}_p$-linearly to

$$\widetilde{\rho} : \mathbb{F}_pG \to \mathrm{Mat}_{n\times n}(\mathbb{F}_p), \quad \sum a_g g \mapsto \sum a_g \rho^{\mathrm{reg}}(g).$$

Note that $E_i := \widetilde{\rho}(e_i)$ is an idempotent matrix, commuting with $\widetilde{\rho}(g)$ for all $1 \le i \le r$. We have

$$\mathbb{F}_p^n = E_1\mathbb{F}_p^n \oplus \cdots \oplus E_r\mathbb{F}_p^n$$

as $\mathbb{F}_p$-vector spaces. This leads to the following method:

Strategy 5.1.44. We obtain a matrix representation of B^{reg} for every p-block $B \in \mathrm{Bl}(\mathbb{F}_pG)$ as follows: for every $j \in \{1, \dots, r\}$, construct the submodule of $\mathbb{F}_pG^{\mathrm{reg}}$ spanned by all vectors lying in $E_j\mathbb{F}_p^n$ and compute the induced submodule action on $\mathrm{Mat}_{m\times m}(\mathbb{F}_p)$ where $m := \dim_{\mathbb{F}_p}(E_j\mathbb{F}_p^n)$.

Output: a matrix representation of B^{reg} for every p-block $B \in \mathrm{Bl}(\mathbb{F}_pG)$

5.2 Setting for computations in GAP and MAGMA

In this section, which is based on [Maa11] and [LP10], we introduce a "standard setting" which is underlying our computations with GAP and MAGMA. We remark that there are also other and more modern ways to treat finite fields and related concepts computationally. See, for example, [Lü23]. We do not use this here, as we would like to compare the trivial source character tables computed by GAP with those computed by MAGMA.

5.2.1 Finite fields

An explicit field arithmetic is crucial for working with concrete modular matrix representations. The following definition of a finite field with p^n elements is used consistently by MAGMA, GAP, and the Shared C MeatAxe programs.

Let p be a prime number. We fix the representatives $0, 1, \ldots, p-1$ of the cosets of integers modulo p to denote the elements of $\mathrm{GF}(p)$, and we agree on the ordering $0 < 1 < \ldots < p-1$. The smallest primitive element of $\mathrm{GF}(p)$ is then denoted by ζ_p. Now suppose that we have $n > 1$. An ordering on the polynomials of degree n over $\mathrm{GF}(p)$ is defined as follows: if $f = \sum_{i=0}^{n} f_i x^i$ and $g = \sum_{i=0}^{n} g_i x^i$, then

$$f < g \text{ if and only if } (-1)^{n-k} f_k < (-1)^{n-k} g_k,$$

where k is the maximal index such that $f_k \neq g_k$. With respect to this ordering, the Conway polynomial for $\mathrm{GF}(p^n)$ is defined recursively as the smallest monic primitive polynomial $f_{p,n}$ of degree n such that for every proper divisor m of n the $\frac{p^n-1}{p^m-1}$-th power of a root of $f_{p,n}$ is a root of $f_{p,m}$. Such a polynomial always exists, see [LP10]. Thus we can define

$$\mathrm{GF}(p^n) := \mathrm{GF}(p)[x]/(f_{p,n})$$

with designated primitive element $\zeta_{p,n} := x + (f_{p,n})$. As $f_{p,n}$ is primitive, every root of $f_{p,n}$ is a generator of $\mathrm{GF}(p^n)^\times$. Consequently, if $m \mid n$ and ω is a root of $f_{p,n}$, and $d := \frac{p^n-1}{p^m-1}$, then

$$\mathrm{GF}(p^m)^\times = \left\langle \omega^d \right\rangle \leqslant \mathrm{GF}(p^n)^\times$$

ensures the embedding of $\mathrm{GF}(p^m)$ as a subfield of $\mathrm{GF}(p^n)$. We identify $\zeta_{p,n}^d = \zeta_{p,m}$.

5.2.2 p-modular systems and computer algebra

Let G be a finite group of exponent m and let p be a prime number. Write $m = p^a b$, where a and b are suitable integers with $(p, b) = 1$. Let $\xi_m := \exp(\frac{2\pi i}{m}) \in \mathbb{C}$. Since, by Bézout's Lemma, there are suitable integers r, s such that $rp^a + sb = 1$, we may define the p-part of ξ_m by $\xi_{p^a} := \xi_m^{sb}$ and the p'-part of ξ_m by $\xi_b := \xi_m^{rp^a}$. Then,

$$o(\xi_{p^a}) = p^a, \ o(\xi_b) = b, \text{ and } \xi_m = \xi_{p^a} \cdot \xi_b = \xi_b \cdot \xi_{p^a}.$$

Note that this is well-defined, as by the classification of linear Diophantine equation systems, the choice of the solution in Bézout's Lemma does not change the values modulo m of $r \cdot p^a$ and $s \cdot b$. Next, we choose $n \in \mathbb{Z}_{\geq 1}$ minimally subject to the condition $p^n \equiv 1 \bmod b$ and set $\xi_{n,p} := \exp(\frac{2\pi i}{p^n-1}) \in \mathbb{C}$.
Let $\mathsf{f} := \mathrm{GF}(p^n)$ with primitive element $\zeta_{n,p}$. We remark that in GAP, we have $\xi_{n,p} = \mathsf{E}(\mathsf{p}^\mathsf{n} - 1)$ and $\zeta_{p,n} = \mathsf{Z}(\mathsf{p}^\mathsf{n})$. Declaring $\hat{\zeta}_{n,p} := \xi_{n,p}$, each element $\zeta_{n,p}^k \in \mathsf{f}^\times$ may be lifted to $\hat{\zeta}_{n,p}^k = \xi_{n,p}^k \in \mathbb{Z}[\xi_{n,p}]$. Conversely, $\bar{\xi}_{n,p} := \zeta_{n,p}$ induces a ring epimorphism $^- : \mathbb{Z}[\xi_{n,p}] \to \mathsf{f}$. Note that

$$\xi_{n,p}^{\frac{p^n-1}{b}} \in \mathbb{Q}(\xi_m) \text{ and } \bar{\xi}_b \in \mathsf{f}$$

are primitive b-th roots of unity. In the following, we assume that $(\mathcal{F}, \mathcal{O}, \mathsf{f}, \eta)$ is a standard p-modular system for G as in [LP10, Definition 4.2.10], that is, $\mathcal{F}$ is the field of fractions of the completion $\mathcal{O}$ of a valuation ring in $\mathbb{Q}(\xi_m)$ and $\eta : \mathcal{O} \to \mathsf{f}$ is a ring epimorphism with $\eta|_{\mathbb{Z}[\xi_b]} = {}^-|_{\mathbb{Z}[\xi_b]}$. In particular, $\xi_m \in \mathcal{F}$ and $\bar{\xi}_b \in \mathsf{f}$ ensure that both $\mathcal{F}$ and f are splitting fields for G and all of its subgroups.

5.2.3 Brauer characters

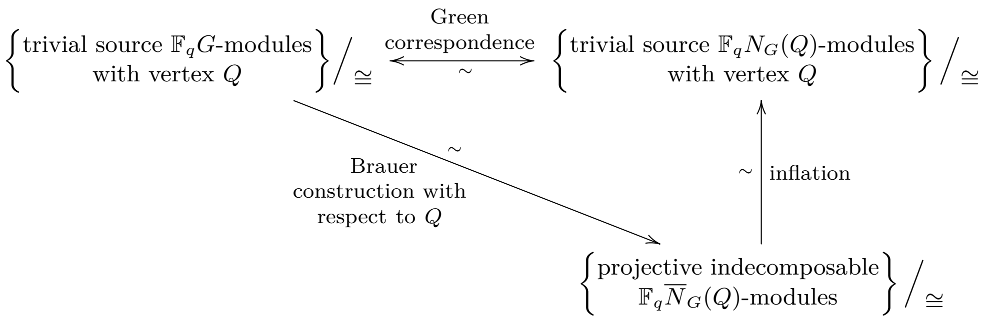
In GAP and MAGMA Brauer characters are defined in the following way.
If W is an $\mathsf{f}G$-module with underlying representation $\rho : G \to G_d(\mathsf{f})$, we define its Brauer character as follows. Choose representatives $g_1, \ldots, g_l$ of the G-conjugacy classes contained in $G_{p'}$. For each $i \in \{1, \ldots, l\}$, we lift the eigenvalues $\epsilon_1, \ldots, \epsilon_d \in \langle \zeta_{n,p} \rangle$ of $\rho(g_i)$ to elements $\hat{\epsilon}_1, \ldots, \hat{\epsilon}_d \in \mathbb{Z}[\xi_{n,p}]$ and set

$$\varphi(g_i) := \sum_{j=1}^{d} \hat{\epsilon}_j.$$

Since conjugate matrices have the same multiset of eigenvalues, this defines a class function $\varphi : \mathrm{G}_{p'} \to \mathbb{C}$ which is the p-modular character afforded by W. As our p-modular system is fixed, we shall often call φ simply a Brauer character of G.

5.3 An algorithm to calculate trivial source character tables

Let G be a finite group and let p be a prime divisor of $|G|$. In this chapter, we develop an algorithm for the computation of all trivial source $\mathbb{F}_q G$-modules, where $\mathbb{F}_q$ is a large enough finite field of characteristic p. We keep the notation from Chapter 3, although the ground field is $\mathbb{F}_q$ instead of k in this chapter. This is no restriction since all occurring modules can be written in $\mathbb{F}_q$. See also Theorem 5.3.4.
Recall that there exist the following bijections by Proposition 3.1.11:



In the following, we make substantial use of the fact that it is possible to compute all modules occurring in $\mathrm{TS}(\overline{N}_G(Q); \langle 1 \rangle)$ for all $Q \in \mathscr{S}_p(G)$. Our strategy to determine all trivial source $\mathbb{F}_q G$-modules is as follows:

Strategy 5.3.1. 1. Choose and fix a set $\mathscr{S}_p(G)$ of representatives of the p-subgroups of G up to conjugacy in G. These groups are the vertices of our trivial source $\mathbb{F}_q G$-modules

2. Sort all groups $Q_i \in \mathscr{S}_p(G)$ by size in ascending order.

3. For all $1 \leq i \leq |\mathscr{S}_p(G)|$: compute $N_G(Q_i)$ and $\overline{N}_G(Q_i)$.

4. For all $1 \leq i \leq |\mathscr{S}_p(G)|$: compute $\mathrm{TS}(\overline{N}_G(Q_i); \langle 1 \rangle)$.

5. Compute the ordinary characters of all projective indecomposable modules occurring in Part 4. and save them in a list as follows:

$$[[\mathrm{PIM}_1, \chi_{\widehat{\mathrm{PIM}_1}}], [\mathrm{PIM}_2, \chi_{\widehat{\mathrm{PIM}_2}}], ..., [\mathrm{PIM}_r, \chi_{\widehat{\mathrm{PIM}_r}}]].$$

6. By Part 2., the algorithm calculates all the PIMs of $\mathbb{F}_q G$ (i.e. $\mathrm{PIM}_1, ..., \mathrm{PIM}_t$) during the first loop.

7. Initialize a new list $\text{list}_{\text{total}}$ which eventually contains all the trivial source $\mathbb{F}_q G$-modules as entries (in list format, so it is a list of lists)

8. For $1 \leq t \leq |\text{TS}(G; \langle 1 \rangle)|$: add $[\langle () \rangle, \text{PIM}_1, \chi_{\widehat{\text{PIM}_1}}], ..., [\langle () \rangle, \text{PIM}_t, \chi_{\widehat{\text{PIM}_t}}]$ to $\text{list}_{\text{total}}$, such that all entries of $\text{list}_{\text{total}}$ have the form [vertex of M, t.s. kG – module $M, \chi_{\widehat{M}}$].

9. Now, we take a closer look at the second step. We have $Q_2 \cong C_p$ (by Cauchy's Theorem) and we are in the following situation:

group	$\overline{N}_G(Q)$	$N_G(Q)$	G
module	PIM	$L := \text{Inf}_{\overline{N}}^{N}(\text{PIM})$	$M := \text{Ind}_N^G(L)$
ordinary character	$\chi_{\widehat{\text{PIM}}}$	$\psi := \text{Inf}_{\overline{N}}^{N}(\chi_{\widehat{\text{PIM}}})$	$\text{Ind}_N^G(\psi)$

We would like to decompose the module M. By the Green correspondence it decomposes as $M \cong g(L) \oplus X$, where $X = \bigoplus_{i=1}^{t} M_i$ is not necessarily zero. Applying the Green correspondence again, we deduce that all direct summands occurring in X have subgroups of G as vertices whose orders are strictly smaller than $|C_p| = p$. Therefore, all summands except for M are isomorphic to a module of our list $\text{list}_{\text{total}}$. We determine these summands with the computer as follows: every time an indecomposable summand is isomorphic to a module in our register $\text{list}_{\text{total}}$, we subtract its ordinary character from $\text{Ind}_N^G(\psi)$. In that way we determine the ordinary character of the Green correspondent $g(L)$ of L.

10. Continuing this process iteratively yields all trivial source $\mathbb{F}_q G$-modules. In order to compute the off-diagonal entries of $\text{Triv}_p(G)$ which do not lie in the first block column, it is by Lemma 3.2.9(c) possible to use restriction, direct sum decomposition, computing module isomorphisms, or discarding their existence.

Output: $\text{Triv}_p(G)$

Remark 5.3.2. Strategy 5.3.1 is used by our algorithms, see Section 7.2 and Section 7.3.

Remark 5.3.3. We mention here that Strategy 5.3.1 does indeed only compute all trivial source $\mathbb{F}_q G$-modules (up to isomorphism). This is due to the nature of computer algebra systems. However, in the theoretical part, we have assumed that our ground field k is, besides other things, algebraically closed. Although this seems like an obstacle, it can be easily resolved as follows in the case of trivial source modules:

Theorem 5.3.4 ([BG07, Theorem 1.9]). *Let k' denote a field of characteristic p that contains a root of unity of order $\exp(G)_{p'}$. Assume that $\widetilde{k}$ is an algebraically closed field of characteristic p that contains k'. Then the functor $\widetilde{k} \otimes_{k'} -$ induces a bijection between the isomorphism classes of indecomposable p-permutation $k'G$-modules and the isomorphism classes of indecomposable p-permutation $\widetilde{k}G$-modules. This bijection preserves defect pairs and is compatible with the Green correspondence. In particular, the induced ring homomorphism*

$$a(k'G, \text{Triv}) \to a(\widetilde{k}G, \text{Triv})$$

is an isomorphism.

Remark 5.3.5. In the sequel, GAP- and MAGMA-implementations of an adapted and improved version of this strategy are given. In particular, we have written a GAP-implementation of the algorithm whose pesuedocode is given in [BL08] which computes maximal common direct summands of finite modules over finite algebras quickly (see Section 7.2). Thereby, computations of complicated endomorphism rings are often avoided. Moreover, as p-permutation modules are closed under induction, restriction, taking direct summands and taking direct sums, this approach is designated for an implementation of Strategy 5.3.1.

5.4 A database of trivial source character tables

For convenient reference and in order to save run time during computer calculations, it is useful to establish a database of trivial source chracter tables. Moreover, it is advantageous to compute trivial source character tables recursively. These aspects are discussed in the present section.

Issue 5.4.1. Assume given groups $H \leq G$. Suppose that we are interested in $\mathrm{Triv}_p(H)$. Moreover, let $\widetilde{H}$ be a group in the database which is isomorphic to H and whose trivial source character table has already been computed. We would like to achieve that the Brauer character values occurring in $\mathrm{Triv}_p(H)$ are consistent with our chosen p-modular system. Let $z := \exp\left(\frac{2\pi i}{3}\right)$. Suppose that one block matrix of $\mathrm{Triv}_p(\widetilde{H})$ is given as follows:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & z & \overline{z} \\ 1 & \overline{z} & z \end{bmatrix}.$$

It is a priori not obvious how we can consistently assign the ordinary characters of H and the conjugacy classes of H to the rows and columns, respectively.

Solution. Choose and fix a group isomorphism $\psi : H \to \widetilde{H}$. Then, transport all the data belonging to $\widetilde{H}$, in particular the ordinary character table, to H via ψ^{-1}.

Issue 5.4.2. The generators of $H := \overline{N}_G(Q)$ do not coincide with the generators of $\widetilde{H}$ from the database, but the underlying matrix representations of our trivial source modules are by default only declared on the generators.

Solution. Use the fact that (matrix) representations are group homomorphisms. More precisely: given a set of generators of a group H, there are efficient algorithms that compute an expression of any group element in terms of these generators. After such an expression is found, multiply the matrices accordingly.

Issue 5.4.3. It is a priori not obvious how we can avoid the computation of endomorphism rings in the direct sum decompisition of trivial source modules over $\mathbb{F}_p$.

Solution. By Strategy 5.3.1, the only remaining issue is to strip off direct summands of an induced module that are not the Green correspondent of this module. This is achived by using our implementation of the algorithm described in [BL08] which computes maximal common (i.e., isomorphic) direct summands of two given $\mathbb{F}_pG$-modules. Although this sometimes also uses the computation of certain sets of module homomorphisms, the dimensions of the involved modules are often significantly smaller. Of course, we first test if the possibility that a trivial source module of our list is a summand can be discarded right away using the computed characters.

Issue 5.4.4. It is a priori not obvious how we can decompose non-projective trivial source modules over $\mathbb{F}_q$.

Solution. We are in the following situation: assume given a finite group G, a p-subgroup Q of G, and a projective indecomposable $\mathbb{F}_p\overline{N}_G(Q)$-module P. Suppose that $\mathbb{F}_q \otimes_{\mathbb{F}_p} P \cong P_1 \oplus P_2$ as $\mathbb{F}_q\overline{N}_G(Q)$-modules. As we work with a computer algebra system, we need to consider matrix representations. Consider the following diagram where we write modules instead of representations and suppress the tensor products which indicate the extensions of scalars for the sake of clarity:

$$\begin{array}{c|ccc}
\text{level} & \mathbb{F}_p & & \mathbb{F}_q \\ \hline
\overline{N}_G(Q) & P & \xrightarrow{\text{conjugate with } \alpha} & P_1 \oplus P_2 \\
N_G(Q) & M & \xrightarrow{\text{conjugate with } \alpha} & M_1 \oplus M_2 \\
G & M\uparrow_{N_G(Q)}^G & \xrightarrow{\text{conjugate with } \begin{pmatrix} \alpha & & \\ & \ddots & \\ & & \alpha \end{pmatrix}} & (M_1 \oplus M_2)\uparrow_{N_G(Q)}^G
\end{array}.$$

The isomorphism $(M_1 \oplus M_2) \uparrow_{N_G(Q)}^G \cong M_1 \uparrow_{N_G(Q)}^G \oplus M_2 \uparrow_{N_G(Q)}^G$ is now obtained by an appropriate base change.
In order to achieve a direct sum decomposition of $\mathbb{F}_q \otimes_{\mathbb{F}_p} g(M)$, we compute maximal direct summands of $\mathbb{F}_q \otimes_{\mathbb{F}_p} g(M)$ and $M_1 \uparrow_{N_G(Q)}^G$, as well as of $\mathbb{F}_q \otimes_{\mathbb{F}_p} g(M)$ and $M_2 \uparrow_{N_G(Q)}^G$. These maximal direct summands are again computed via our implementation of the algorithm in [BL08], this time over the field $\mathbb{F}_q$.

Issue 5.4.5. It is a priori not obvious how we can compute the off-diagonal entries in $\mathrm{Triv}_p(G)$.

Solution. Use Lemma 3.2.9(c) and the solution of Issue 5.4.4.

Issue 5.4.6. It is a priori not obvious how we can intersect two trivial source $\mathbb{F}_qG$-modules.

Solution. In all occurring cases, we are in the comfortable situation that we can reformulate this problem to an equivalent one where only the computation of maximal common direct summands is involved.

Since we aim at avoiding computations over $\mathbb{F}_q$ whenever possible, we do only save the matrix representations of trivial source $\mathbb{F}_pG$-modules, as well as the base change matrices to the .txt-files on our hard drive. Note that this is enough information: if

$$\rho : G \to \mathrm{Mat}_{n\times n}(\mathbb{F}_q),\ g \mapsto \rho(g)$$

is a matrix representation of G and $\{g_1, g_2, g_3\}$ is a complete set of generators of G, then

$$\alpha\rho(g_1g_2g_3)\alpha^{-1} = \alpha\rho(g_1)\alpha^{-1}\alpha\rho(g_2)\alpha^{-1}\alpha\rho(g_3)\alpha^{-1}$$

for each base change matrix $\alpha \in \mathrm{Mat}_{n\times n}(\mathbb{F}_q)$. Nevertheless, this leads to the following issue.

Issue 5.4.7. By our solution to Issue 5.4.4, we do not compute the ordinary characters of the trivial source $\mathbb{F}_qG$-modules on the way. It is a priori not obvious how we can derive these pieces of information without storing all trivial source character tables of all normalisers in cache during the whole computation of $\mathrm{Triv}_p(G)$.

Solution. Compute the Brauer character values of the (restrictions of the) trivial source $\mathbb{F}_qG$-modules column by column. After that, apply Corollary 3.2.8.

Remark 5.4.8. In most cases, we did not compute the Brauer character values, but used the ordinary character values of trivial source modules of smaller groups. The values could be looked up from our database, as they had been computed previously using the same algorithm. Nevertheless, the computation of Brauer character values over $\mathbb{F}_q$ is sometimes necessary. For example, we need to know the Brauer characters values of simple $\mathbb{F}_qG$-modules, in order to be able to compute projective indecomposable $\mathbb{F}_qG$-modules. In general it is faster to compute a composition series of the module in question first and then sum the Brauer characters of the composition factors than to compute the Brauer character of a (much larger) matrix representation.

The following Proposition is therefore very useful.

Proposition 5.4.9 ([CR06, (82.5) Corollary]). *Let S and T be absolutely simple $\mathbb{F}_qG$-modules, with underlying representations ρ_S and ρ_T, respectively. Then $S \cong T$ as $\mathbb{F}_qG$-modules if and only if* $\mathrm{tr}(\rho_S(x)) = \mathrm{tr}(\rho_T(x))$ *for all p-regular elements $x \in G$.*

Remark 5.4.10. We use Proposition 5.4.9 in our algorithms as follows: we compute the Frobenius character values of each irreducible Brauer character and store these values in a dictionary. This way, the Brauer character values of composition factors can be recovered without computing Brauer character values, but via taking traces.

Remark 5.4.11. Proposition 5.4.9 is used in Section 7.1.

If G is a p-solvable group, it is also possible to compute the projective indecomposable $\mathbb{F}_qG$-modules in a different way. This is due to the following result.

Proposition 5.4.12 ([Fon62, Corollary (2E)]). *Let G be a p-solvable group and let C be a Sylow p-complement of G. Then, the following assertion holds. For each projective indecomposable kG-module U_i ($1 \leq i \leq |\mathrm{TS}(G; \langle 1 \rangle)|$), there exists a simple kC-module S_i such that $U_i \cong \mathrm{Ind}_C^G(S_i)$ as kG-modules.*

Remark 5.4.13. We have not used Proposition 5.4.12 in our computer implementations yet, but the idea is as follows: compute the induction of all simple kC-modules from a Sylow p-complement C of G to G. Discard all induced characters which do not vanish on all p-singular elements of G. Each remaining induced character ψ for which additionally the equation

$$\langle \varphi, \psi \rangle_{G_{p'}} \geq 0$$

holds for all $\varphi \in \mathrm{IBr}_p(G)$ is a p-projective character. See [LP10, Corollary 4.3.4]. Hence, in the case of p-solvable groups, matrix representations of projective indecomposable $\mathbb{F}_qG$-modules are much easier to compute than in the general case.

5.4.1 A list containing several bugs in GAP and MAGMA

Remark 5.4.14. Our computations led to the discovery of several bugs in GAP and MAGMA. The most serious bugs are enumerated in the following list.

1. A bug in the GAP file lib/meatauto.gi
2. A GAP problem with vector objects, see github.com/gap-system/gap/issues/5030
3. A GAP problem with AbsolutelyIrreducibleModules, see github.com/gap-system/gap/issues/5061
4. A GAP problem concerning the multiplication of matrices and vectors of different types, see github.com/gap-system/gap/issues/5062

5. A GAP problem concerning 8 Bit matrix representations,
   see github.com/gap-system/gap/issues/5066
6. A GAP problem concerning problem with compressed vectors,
   see github.com/gap-system/gap/issues/5123
7. A bug in the GAP package qpa (Size(HomOverAlgebra(M,M)) yielded another result than Dimension(EndOfModuleAsQuiverAlgebra(M)))
8. A bug within the MAGMA function DecompositionMatrix
9. A bug within the MAGMA function BrauerCharacter
10. A bug concerning an internal memory problem of MAGMA

A special thanks goes to the following people because by the year 2024 these bugs have been resolved due to their committed work: Thomas Breuer, Max Horn, Frank Lübeck, Alexander Hulpke, Øyvind Solberg, Olexandr Konovalov, Derek Holt, Nicole Sutherland, and Allan Steel.

Chapter 6

Using trivial source character tables in modular representation theory

This chapter is concerned with two possible applications of trivial source modules and trivial source character tables to modular representation theory of finite groups.

Throughout, we let G and H be two finite groups, we let $B \in \mathrm{Bl}(kG)$ and $B' \in \mathrm{Bl}(kH)$ be two blocks, and we assume B and B' have isomorphic defect groups. We cosinder the following two problems.

Problem 6.0.1. Compute all p-permutation equivalences between B and B'.

Problem 6.0.2. Compute all splendid Morita equivalences between B and B' or discard the existence of such an equivalence.

6.1 p-permutation equivalences via trivial source character tables

In this section, we consider *Problem* 6.0.1. Recall from Section 2.10.3 that an element $\gamma \in T^{\Delta}(B, B')$ is called a *p-permutation equivalence* between B and B' if

$$\gamma \otimes_{kH} \gamma^{\vee} = [B] \in T^{\Delta}(B, B).$$

Remark 6.1.1. For a given left $k[G \times H]$-module M_{left}, we can define an equivalent right $k[G \times H]$-module M_{right} in MAGMA, where the elements of M_{left} and M_{right} are the same, but, for $m \in M_{\text{left}}$, we have $(g, h)m \in M_{\text{left}}$ is equal to $m(g^{-1}, h^{-1}) \in M_{\text{right}}$.

Below, we explain how to identify B and B' in $\mathrm{Triv}_p(G \times G)$ and $\mathrm{Triv}_p(H \times H)$, respectively, and how it is possible to use $\mathrm{Triv}_p(G \times H)$ and $\mathrm{Triv}_p(H \times G)$ in order to find all p-permutation equivalences with the help of a computer algebra system. In the sequel, we work with GAP. Recall that by Remark 5.3.3 we may replace k with a finite field $\mathbb{F}_q$ which is large enough.

Issue 6.1.2.

1. GAP can neither deal with bimodules nor with tensor products of bimodules.
2. Assume we are given $\mathrm{Irr}_K(B)$. Provided we have already computed $\mathrm{Triv}_p(G \times G)$ with GAP, it is not a priori clear how we can identify the trivial source $\mathbb{F}_q[G \times G]$-module corresponding to the bimodule B.

In order to determine all p-permutation equivalences between B and B' we apply the following method:

Strategy 6.1.3.

1. Compute $\mathrm{Triv}_p(G \times H)$, $\mathrm{Triv}_p(H \times G)$, and $\mathrm{Triv}_p(G \times G)$.

2. Determine which of the trivial source $\mathbb{F}_q[G \times H]$-modules have twisted diagonal vertices. Denote them by $M_1, \dots, M_r$.

3. Regard $M_1, \dots, M_r$ as $(\mathbb{F}_qG, \mathbb{F}_qH)$-bimodules and denote them by $B_1, \dots, B_r$.

4. Compute the dual $(\mathbb{F}_qH, \mathbb{F}_qG)$-bimodules $B_1^\vee, \dots, B_r^\vee$.

5. Regard now $B_1^\vee, \dots, B_r^\vee$ as $\mathbb{F}_q[H \times G]$-modules and denote them by $M_1', \dots, M_r'$.

6. Consider the equation

$$(x_1[M_1] \oplus \cdots \oplus x_r[M_r]) \otimes_{\mathbb{F}_qH} (x_1[M_1'] \oplus \cdots \oplus x_r[M_r']) = [B],$$

where $x_i \in \mathbb{Z}$ for each $1 \leq i \leq r$. It is known that the resulting Diophantine equation system has only finitely many solutions, see [Per14, Theorem 13.2].

7. By Theorem 2.10.30, each p-permutation equivalence between B and B' induces a perfect isometry between B and B'. So, compute all perfect isometries between B and B' (which can be easily done with GAP or MAGMA).

8. For each perfect isometry μ between B and B' compute which tuples $[x_1, \dots, x_r]$ imply the present perfect isometry μ, (see Theorem 2.10.30); this yields a linear equation system in the variables $x_1, \dots, x_r$. Find all r-tuples that solve such an equation.

9. Plug every such r-tuple into the Diophantine equation system and check if the thereby obtained equations hold.

Output: a list containing all r-tuples $(x_1, \dots, x_r)$ which solve our Diophantine equation system

Remark 6.1.4. Using our algorithm in Section 7.4, it is possible to obtain the Diophantine equation system for all cases in which B and B' are principal blocks. Hence, a contribution to a solution of *Problem* 6.0.1 has been obtained.

Remark 6.1.5. Next, we describe how to obtain the dual modules M_i' for $1 \leq i \leq r$ from Strategy 6.1.3 Part 5. with GAP.

(a) If V is an $\mathbb{F}_qG$-module with underlying matrix representation $\rho : G \to \mathrm{GL}_n(\mathbb{F}_q)$, then the dual of V is an $\mathbb{F}_qG$-module with underlying matrix representation

$$\rho' : G \to \mathrm{GL}_n(\mathbb{F}_q), g \mapsto \rho(g^{-1})^{\mathrm{T}}.$$

(b) In order to obtain M_1' from M_1 without using bimodules, we do the following. Recall that M_1 is an $\mathbb{F}_q[G \times H]$-module. Compute the dual of M_1 as an $\mathbb{F}_q[G \times H]$-module. Let $\rho : G \times H \to \mathrm{GL}_n(\mathbb{F}_q)$ be an underlying matrix representation. Then M_1' affords the matrix representation

$$\rho' : H \times G \to \mathrm{GL}_n(\mathbb{F}_q), (h, g) \mapsto \rho((g, h)).$$

It remains to explain how to identify the row of $\mathrm{Triv}_p(G \times G)$ corresponding to B with GAP. This is done in Strategy 6.1.7.

Next, we describe how to identify the full group algebra kG seen as a trivial source (kG, kG)-bimodule in $\mathrm{Triv}_p(G \times G)$. We set $\Omega := (G \times G)/\Delta(G)$.

Lemma 6.1.6. *Consider the regular bimodule ${}_{kG}kG_{kG}$ and regard it as a right $k[G\times G]$-module. Let P be a p-subgroup of $G\times G$ and choose $u\in N_{G\times G}(P)_{p'}$ such that $s:=uP\in \overline{N}_{G\times G}(P)_{p'}$. Then*

$$\tau^{G\times G}_{P,s}([kG]) = |\{\omega\in\Omega \;:\; \omega\cdot h=\omega \text{ for all } h\in\langle P,u\rangle\}|.$$

Moreover, if the group algebra kG has only one block, then the row of $\mathrm{Triv}_p(G\times G)$ *corresponding to kG is uniquely determined by the species values above.*

Proof. As $k[G\times G]$-modules,

$$kG\cong k{\uparrow}^{G\times G}_{\Delta(G)},$$

the $k[G\times G]$-permutation module on Ω. See, e.g., [Lin18a, Proposition 2.8.19]. Now, the asserted equation follows from the equations in (4.42) of [LP10, Remark 4.10.5]. The second claim is clear, as $\mathrm{Triv}_p(G\times G)$ is invertible. ☐

Strategy 6.1.7. The row of $\mathrm{Triv}_p(G\times G)$ corresponding to B can be identified with GAP provided $\mathrm{Irr}_K(B)$ is given:

1. Compute the species values of the $k[G\times G]$-module kG using Lemma 6.1.6 and write them as a linear combination of rows of $\mathrm{Triv}_p(G\times G)$.
2. Recall from Strategy 5.3.1 that for each row of $\mathrm{Triv}_p(G\times G)$ we also know a matrix representation of the corresponding $k[G\times G]$-module, as well as its vertex and its ordinary character. Thus, we obtain a direct sum decomposition of kG:
$$kG\cong\bigoplus_{i=1}^{r}\widetilde{M}_i.$$
3. Recall that for $1\le i\le r$ the $k[G\times G]$-modules $\widetilde{M}_i$ are projective as one-sided modules. So, via the identification $k[G\times G]\cong kG\otimes kG$, compute the ordinary characters of these projective modules (restrict the matrix representations and then use the fact that a Brauer character of a PIM determines the ordinary character of that PIM) and compare them to $\mathrm{Irr}_K(B)$. In that way, we obtain the information which direct summand corresponds to the block B.

Output: the row of $\mathrm{Triv}_p(G\times G)$ corresponding to B

The next lemma allows us to circumvent the problem of computing tensor products of bimodules for the Diophantine equation system.

Lemma 6.1.8 ([BY22, 2.6 Corollary])**.** *Let L be a p-permutation (kG,kH)-bimodule and let M be a p-permutation (kH,kJ)-bimodule. Suppose that all of the indecomposable summands of L and M have twisted diagonal vertices. Moreover, let $\Delta(U,\gamma,W)$ be a twisted diagonal subgroup of $G\times J$. Let $\Gamma_H(U,\gamma,W)$ denote the set of triples (α,V,β) where V is a subgroup of H, and $\alpha: V\to U$ and $\beta: W\to V$ are group isomorphisms with the property that $\gamma=\alpha\circ\beta$. Then, for any diagonal pair $(\Delta(U,\gamma,W),(s,t))$ of $G\times J$ where $(s,t)\in N_{G\times J}(\Delta(U,\gamma,W))_{p'}$ is such that $(s,t)\Delta(U,\gamma,W)\in\overline{N}_{G\times J}(\Delta(U,\gamma,W))_{p'}$, we have*

$$\tau^{G\times J}_{\Delta(U,\gamma,W),(s,t)}(L\otimes_{kH}M)=\frac{1}{|H|}\sum_{\substack{(\alpha,V,\beta)\in\Gamma_H(U,\gamma,W),h\in H_{p'}\\(s,h)\in N_{G\times H}(\Delta(U,\alpha,V))\\(h,t)\in N_{H\times J}(\Delta(V,\beta,W))}}\tau^{G\times H}_{\Delta(U,\alpha,V),(s,h)}(L)\tau^{H\times J}_{\Delta(V,\beta,W),(h,t)}(M).$$

In order to compute *all* p-permutation equivalences between B and B', the following lemma is crucial.

Lemma 6.1.9. *We have*

$$T_o^{\Delta}(B,B') = \{\gamma_{\text{particular}} \otimes_{kH} \gamma' \mid \gamma' \in T_o^{\Delta}(B',B')\} = \{\gamma'' \otimes_{kG} \gamma_{\text{particular}} \mid \gamma'' \in T_o^{\Delta}(B,B)\},$$

where $\gamma_{\text{particular}} \in T_o^{\Delta}(B,B')$ *denotes an arbitrary but fixed p-permutation equivalence between B and B'.*

Proof. Let $\gamma, \widetilde{\gamma} \in T_o^{\Delta}(B,B')$. Then, on the one hand we have

$$\widetilde{\gamma} \cong (\gamma \otimes_{kH} \gamma^{\vee}) \otimes_{kG} \widetilde{\gamma} \cong \gamma \otimes_{kH} (\gamma^{\vee} \otimes_{kG} \widetilde{\gamma}),$$

and on the other hand we have

$$\widetilde{\gamma} \cong \widetilde{\gamma} \otimes_{kH} (\gamma^{\vee} \otimes_{kG} \gamma) \cong (\widetilde{\gamma} \otimes_{kH} \gamma^{\vee}) \otimes_{kG} \gamma.$$

Moreover, $\gamma^{\vee} \otimes_{kG} \widetilde{\gamma} \in T_o^{\Delta}(B',B')$, since

$$(\gamma^{\vee} \otimes_{kG} \widetilde{\gamma}) \otimes_{kH} (\gamma^{\vee} \otimes_{kG} \widetilde{\gamma})^{\vee} \overset{(*)}{\cong} (\gamma^{\vee} \otimes_{kG} \widetilde{\gamma}) \otimes_{kH} (\widetilde{\gamma}^{\vee} \otimes_{kG} \gamma) \cong \gamma^{\vee} \otimes_{kG} \gamma = [B'],$$

and, analogously, $\widetilde{\gamma} \otimes_{kH} \gamma^{\vee} \in T_o^{\Delta}(B,B)$. The asserted isomorphism in $(*)$ holds by [Lin18a, Corollary 2.12.5]. ☐

This leads to the following method to solve the Diophantine equation system.

Strategy 6.1.10.
1. If the relevant blocks B and B' do not coincide, find one particular solution of the Diophantine equation system and (pre)compose it with the p-permutation auto-equivalences of B or B' in order to get all solutions.
2. If B and B' coincide, then compute all p-permutation auto-equivalences of B.
3. It follows from [BP20, 7.4 Corollary (b)] that, in the situation of Part 2, we can additionally apply the following method to compute all solutions of our given Diophantine equation system. We can solve it iteratively as follows: we begin with those trivial source bimodules which have maximal vertices and solve it modulo coefficients of bimodules whose vertices have smaller orders. Only then do we consider the non-maximal bimodules. We can picture our solutions like a tree. On the upper level, we write all solutions only involving the coefficients of the maximal bimodules. For each fixed solution we computed, on the level below we write down all solutions of the now easier Diophantine equation system where we only take all of those bimodules whose vertices have maximal orders (among the non-maximal bimodules) into account. Continuing this process iteratively, we obtain all solutions.
4. In the CAS Mathematica, there exists the command "FindInstance". This command can be used for Part 1 in order to obtain a particular solution of the Diophantine equation system.

Output: a method to determine all solutions of our Diophantine equation system in a faster way

6.2 Splendid Morita equivalences via trivial source bimodules

In this section, we consider *Problem* 6.0.2. Since splendid Morita equivalences are special cases of p-permutation equivalences, they can be computed analogously to Strategy 6.1.3. Since MAGMA is able to do computations with bimodules (including tensor products of bimodules), we consider an example here, where the computations are done with MAGMA.

Remark 6.2.1. Assume given two group algebras $\mathbb{F}_q G$ and $\mathbb{F}_q H$, where one of them has only one block. Our MAGMA algorithm in Section 7.5 computes all splendid Morita equivalences between the principal blocks of $\mathbb{F}_q G$ and $\mathbb{F}_q H$. Moreover, it returns an empty list, if there exists no splendid Morita equivalence between $B_0(\mathbb{F}_q G)$ and $B_0(\mathbb{F}_q H)$.

Example 6.2.2. Let $p := 3$. Define $G := C_3 \times \mathfrak{S}_3$ and $H := C_3 \times C_3$. Then, there is no splendid Morita equivalence between $B_0(kG)$ and $B_0(kH)$.

Algorithm 2 MAGMA pseudocode for Example 6.2.2

The following MAGMA pseudocode verifies that there is no splendid Morita equivalence between $B_0(kG)$ and $B_0(kH)$.

1: G := SmallGroup(18,3);
2: H := SmallGroup(9,2);
3: Define the direct product $G \underset{p_G}{\overset{i_G}{\rightleftarrows}} G \times H \underset{i_H}{\overset{p_H}{\rightleftarrows}} H$.
4: D_G := SylowSubgroup(G,3);
5: D_H := SylowSubgroup(H,3);
6: bool, phi := IsIsomorphic(D_G,D_H); // This returns "true" and an isomorphism.
7: ΔD := $\{i_G(x) \cdot i_H(\varphi(x)) \mid x \in D_G\} \leq G \times H$;
8: $k := \mathbb{F}_3$ // This field contains all $\exp(G \times H)$-th roots of unity.
9: TM := TrivialModule(ΔD,k);
10: Ind := TM$\uparrow_{\Delta D}^{G\times H}$
11: DIR := DirectSumDecomposition(Ind); // DIR is indecomposable.
12: B1 := Bimodule(G,H,DIR[1]);
13: C1 := B1$^\vee$; // C1 is the dual of B1 in the sense of Section 2.10.3.
14: T11 := TensorProduct(B1,C1); // T11 has k-dimension 36.

Caveat 6.2.3. In line 6 of Algorithm 2, the computer chooses an isomorphism between the defect group D_G of $B_0(kG)$ and the defect group D_H of $B_0(kH)$. Hence, if the output is an empty list, this does not necessarily imply that there is no splendid Morita equivalence between $B_0(kG)$ and $B_0(kH)$. We have to test it for all isomorphisms between the groups D_G and D_H.

Remark 6.2.4. In the particular case of Example 6.2.2, the two blocks $B_0(kG)$ and $B_0(kH)$ are not Morita equivalent. Hence, they are not splendidly Morita equivalent.

The MAGMA code for Example 6.2.2:

```
GG := SmallGroup(18,3);
HH := SmallGroup(9,2);
phiG, G := MinimalDegreePermutationRepresentation(GG);
phiH, H := MinimalDegreePermutationRepresentation(HH);
GxH, i_GxH, p_GxH := DirectProduct(G,H);
i_G_GxH := i_GxH[1];
```

```
i_H_GxH := i_GxH[2];
D_G := SylowSubgroup(G,3);
D_H := SylowSubgroup(H,3);
bool, phi := IsIsomorphic(D_G,D_H);
ElsD_G := [y: y in D_G];
temp := [];
for x in ElsD_G do Append(~temp, i_G_GxH(x)*i_H_GxH(phi(x))); end for;
DeltaD := sub<GxH | temp>;
p := 3;
m := Exponent(GxH);
K := GF(p);
Kx<x> := PolynomialRing(K);
f := x^m -1;
L := SplittingField(f);
u := #L;
k := GF(u);
TM := TrivialModule(DeltaD,k);
IND := Induction(TM,GxH);
DIR := DirectSumDecomposition(IND);
B1 := Bimodule(G,H,DIR[1]);
LOM1 := LeftOppositeModule(B1);
RM1 := RightModule(B1);
C1 := Bimodule(Dual(RM1), Dual(LOM1));
T11 := TensorProduct(B1,C1);
```

Proposition 6.2.5. (a) *Let G be a finite group and let B be a block of $\mathcal{O}G$ with normal defect group D and inertial quotient $I(B)$. Then B is Morita equivalent to a twisted group algebra*

$$\mathcal{O}_\gamma[D \rtimes I(B)]$$

where $\gamma \in O_{p'}(\mathrm{H}^2(I(B);\mathcal{O}^\times)) \cong O_{p'}(\mathrm{H}^2(I(B);\mathbb{C}^\times))$.

(b) *Let G be a finite cyclic group. Consider $\mathbb{C}^\times$ with the trivial action on G. Then the group $H^2(G;\mathbb{C}^\times)$ is trivial.*

(c) *Let G be a finite group and let B be a block of $\mathcal{O}G$ with abelian defect group D and inertial quotient $I(B)$. If B is the principal block of $\mathcal{O}G$ then $I(B) = N_G(D)/C_G(D)$.*

Proof. For part (a) see [Sam14, Theorem 1.19] and for part (b) see [Lin18a, Proposition 1.2.10]. Part (c) follows from the definitions. □

Remark 6.2.6. As our computer cannot handle computations over $\mathcal{O}$, it may be slightly surprising that we consider blocks of $\mathcal{O}[G \times H]$ in a computational section, but this can be explained as follows. For our examples, we only consider blocks of groups that fulfill the assumptions (and, in particular, the Morita equivalence) from Proposition 6.2.5.
In order to test blocks for splendid Morita equivalence, it is enough to do the computations over k (or, more precisely, over $\mathbb{F}_q$), for the following reason: on the one hand, each trivial source $k[G \times H]$-module lifts in a unique way to a trivial source $\mathcal{O}[G \times H]$-module, and on the other hand, reduction modulo $\mathfrak{p}$ yields the inverse mapping. Hence, such a splendid Morita equivalence exists over $\mathcal{O}$ if and only if it exists over k.

Example 6.2.7. Let $p := 3$ and let $(K, \mathcal{O}, k)$ be a splitting p-modular system. Define $G := C_3 \times \mathfrak{S}_3$ and $H := C_3 \times Q$ where $Q := C_3 \rtimes C_4$ is isomorphic to the dicyclic group of order 12. Then, the principal block $B_0(\mathcal{O}G)$ of $\mathcal{O}G$ and the principal block $B_0(\mathcal{O}H)$ of $\mathcal{O}H$ are Morita equivalent.

Proof. We choose the following concrete representation of H as a permutation group:

$$H := \langle (7,8,9,10)(15,16), (11,12,13), (7,9)(8,10), (14,15,16) \rangle.$$

Moreover, we choose $D_H := \langle(11,12,13),(14,15,16)\rangle$. Note that D_H is abelian and a normal subgroup of H. We compute $C_H(D_H) = \langle(11,12,13),(14,15,16),(7,9)(8,10)\rangle$. Set $H \ni v := (7,8,9,10)(14,15) \notin C_H(D_H)$. It follows from Proposition 6.2.5 that

$$I(B_0(\mathcal{O}H)) = N_H(D_H)/C_H(D_H),$$

and the latter expression is isomorphic to $\langle vC_H(D_H)\rangle$ as $|N_H(D_H)/C_H(D_H)| = 2$. By Proposition 6.2.5, $B_0(\mathcal{O}H)$ is Morita equivalent to $\mathcal{O}[D_H \rtimes I(B_0(\mathcal{O}H))]$. We denote the outer semidirect product $D_H \rtimes I(B_0(\mathcal{O}H))$ by G. We see that D_H is an abelian normal p-Sylow subgroup of G. Note that $\mathcal{O}G = B_0(\mathcal{O}G)$ and, therefore, D_H is a defect group of the principal block of $\mathcal{O}G$. □

Question: Are the two blocks $B_0(\mathcal{O}G)$ and $B_0(\mathcal{O}H)$ in Example 6.2.7 splendidly Morita equivalent?

We choose the following concrete representation of G as a permutation group:

$$G := \langle(1,4)(2,5)(3,6),(1,2,3)(4,5,6),(1,3,2)(4,5,6)\rangle.$$

Moreover, we choose $D_G := \langle(1,2,3),(4,5,6)\rangle$ as Sylow 3-subgroup.

Define the group isomorphism $\phi : D_G \to D_H$ by

$$\phi((1,2,3)) := (11,12,13) \text{ and } \phi((4,5,6)) := (14,15,16).$$

Then, by counting p'-classes of $N_{G\times H}(\Delta D)/\Delta D$, we see that there are exactly two trivial source $\mathcal{O}[G \times H]$-modules with vertices isomorphic to ΔD. A MAGMA computation proves that they are given as the two indecomposable summands of $\mathrm{Ind}_{\Delta D}^{G\times H}(\mathcal{O})$ and that their respective $\mathcal{O}$-ranks are equal to 36. Another MAGMA computation reveals that

$$\mathrm{rk}_{\mathcal{O}}(B_1 \otimes_{\mathcal{O}H} B_1{}^{\vee}) = \mathrm{rk}_{\mathcal{O}}(B_2 \otimes_{\mathcal{O}H} B_2{}^{\vee}) = 72$$

and

$$\mathrm{rk}_{\mathcal{O}}(B_1 \otimes_{\mathcal{O}H} B_2{}^{\vee}) = \mathrm{rk}_{\mathcal{O}}(B_2 \otimes_{\mathcal{O}H} B_1{}^{\vee}) = 0,$$

where B_1 and B_2 are the aforementioned $\mathcal{O}[G \times H]$-modules, regarded as bimodules. Since $B_0(\mathcal{O}G) = \mathcal{O}G$, we have $\mathrm{rk}_{\mathcal{O}}({}_{\mathcal{O}G}\mathcal{O}G_{\mathcal{O}G}) = 18 \notin \{0;72\}$.

However, due to Caveat 6.2.3, this does not necessarily imply that the two blocks $B_0(kG)$ and $B_0(kH)$ (or, equivalently, the two blocks $B_0(\mathcal{O}G)$ and $B_0(\mathcal{O}H)$) are not splendidly Morita equivalent. Indeed, if we define the group isomorphism $\phi : D_G \to D_H$ by

$$\phi((1,2,3)) := (11,13,12)(14,16,15) \text{ and } \phi((4,5,6)) := (11,13,12)(14,15,16),$$

then, a MAGMA computation reveals that an indecomposable direct summand of the $\mathcal{O}[G \times H]$-module $\mathrm{Ind}_{\Delta D}^{G\times H}(\mathcal{O})$, seen as an $(\mathcal{O}G, \mathcal{O}H)$-bimodule, induces a splendid Morita equivalence between $B_0(\mathcal{O}G)$ and $B_0(\mathcal{O}H)$. Note that this is in accordance with [Lin18b, Theorem 6.14.1].

Chapter 7

APPENDIX: computer implementations

In this chapter, we present our implemented algorithms.

7.1 An algorithm for the computation of principal indecomposable modules using the Shared C MeatAxe and GAP

```
# The following programs are written in order to be able to test
# efficiently with GAP, whether a given indecomposable trivial
# source module is (isomorphic to) a direct summand of a given
# p-permutation module.
# Extensive use of the ideas and pseudo-code mentioned in the
# article "Testing isomorphism of modules"
# by Brooksbank and Luks is made.
# We remark that there does exist an implementation of the
# aforementioned pseudo-code in MAGMA. The name of the function
# is SummandIsomorphism. Alas, the MAGMA code is not open source.
# Hence, we have translated those parts of the pseudo-code that
# we need for our purposes into GAP code.
#
# The notation is inspired by the above article .
# Unless stated otherwise, we assume that all entered modules
# and vector spaces are over the field GF(p).

# Furthermore, we make the following remarks.
# 1) All entered groups are supposed to be permutation groups.
# 2) During the computations, files get deleted automatically.
# Hence, we recommend to do the computations in a folder not containing important files .
# 3) We suppose that a standalone version of
# the Shared C MeatAxe (see, e.g., https://users.fmi.uni-jena.de/~king/SharedMeatAxe/index.html)
# is installed .
# 4) We assume that the GAP system is installed and the version is at least 4.12.2.

# The following two functions are concerned with Lemma 2.1 of
# the article by Brooksbank and Luks.

FromWToWX:= function(W,mats)
# We assume that mats is a list of matrices.

    local BVS, i, j, W_now, temp;

    BVS:=BasisVectors(Basis(W));

    temp:=[];
    for i in mats do
        for j in BVS do
            Add(temp, j*i);
        od;
    od;
    W_now := Subspace(W,temp);

    return W_now;
end;



IsEnvNilpotent:= function(W,mats)

```

```
    local U, flag;

    U:=ShallowCopy(W);

    while Dimension(U) <> Dimension(FromWToWX(U,mats)) do
        U:=FromWToWX(U,mats);
    od;

    flag:=false;

    if IsZero(Dimension(U)) then
        flag := true;
        return flag;
    fi;

    return flag;
end;



# The following function is concerned with Algorithm 2 of
# the article. See also Algorithm 1 in the article. We
# always assume that the function delta mentioned therein
# is trivial.

FromSplitterToFactorization := function(V,LISTE_X)
# We remark that the entered variable V must be a full
# row vector space here as we use this function only once
# where this has to be the case.

    local i, LISTE_now, W, W_full, m, Y, z, d, r, y,
    Y_new, List_LeftMultiplications_Of_z_in_reversed_order,
    j, List_LeftMultiplications_Of_z;

    i:=1;
    LISTE_now:=[LISTE_X[1]];
    W:=ShallowCopy(V);
    W_full:=ShallowCopy(V);

    while IsEnvNilpotent(W,LISTE_now) do
        i:=i+1;
        Add(LISTE_now,LISTE_X[i]);
    od;

    m:= i; # This is indeed correct. There is a typo in the
    # article. It is written correctly in Algorithm 1 of the article.

    Y:=[];

    for j in [1..m-1] do
        Add(Y,LISTE_X[j]);
    od;

    z:=LISTE_X[m];

    d:=Size(LISTE_X[1]);

    List_LeftMultiplications_Of_z_in_reversed_order:=[];

    while IsZero(z^d) do # This means: while z is nilpotent do the following:
        while Dimension(FromWToWX(W,Y)+FromWToWX(W,[z])) < Dimension(W) do
            W:=FromWToWX(W,Y)+FromWToWX(W,[z]);
            Print("W ist im Moment gerade gleich: ");Print(W);Display(W);
        od;

        # Now, we find an element y such that W(y*z) is not a subspace of XY:
        r:=1;
        y:=Y[1];
        while IsSubspace(FromWToWX(W,Y),FromWToWX(W,[y*z])) do
            r:=r+1;
            y:=Y[r];
        od;

        Y_new:=ShallowCopy(Y);
        Add(Y_new,y*z);
        if IsEnvNilpotent(W,Y_new) then
            Y:=ShallowCopy(Y_new);
```

```
        else
            z:=y*z;
            Add(List__LeftMultiplications__Of__z__in__reversed__order,y);
        fi ;
    od;

    List__LeftMultiplications__Of__z :=
    Reversed(List__LeftMultiplications__Of__z__in__reversed__order);

return [z,List__LeftMultiplications__Of__z];

end;



# The following is an adapted version of a function taken from R. Zimmermann's PhD thesis (Jena).

ReadRepFrom:=function(file,nrgens,field)
    # Reading the MeatAxe matrices file .1, .., file .nrgens, which are binary files , into GAP
    local rep, i , LocationOfZPRAsString, path, rm, stdin, stdout, MyDir, options, pro;
    LoadPackage("io");
    ChangeDirectoryCurrent("/home/bernhard");

    stdin := InputTextUser();;
    stdout := OutputTextUser();;
    MyDir:=Directory("/home/bernhard");
    LocationOfZPRAsString := "/home/bernhard/Schreibtisch/shared__meataxe-1.0/bin/zpr";

    rep :=[];
    for i in [1.. nrgens] do

        options:=[Concatenation(file,".",String(i)), Concatenation(file,".text") ];

        pro := Process(MyDir, LocationOfZPRAsString, stdin, stdout, options);

        if not IsZero(pro) then
            Print("The last process did not return zero!");
            return(fail);
        fi ;
        rep[i]:=ScanMeatAxeFile(Concatenation(file,".text"),Size( field ));
    od;

    path := DirectoriesSystemPrograms();
    rm := Filename(path,"rm");

    options:=[Concatenation(file,".text") ];

    pro := Process(MyDir, rm, stdin, stdout, options);
    if not IsZero(pro) then
        Print("The last process did not return zero !!! ");
        return(fail);
    fi ;

    return rep;
end;



# The following function computes the GF(p)G - linear homomorphisms from M to N.
# It uses the Shared C MeatAxe, i.e. programs written by M. Szöke et al. frequently .

Hom__FpG__MNViaSzoeke := function(M,N)
# This function returns a list of matrices which form a GF(p)-basis for the hom space in question.

    local F,FF, gensOfM, gensOfN, p, repM, repN, i, U, DIMensionM, DIMensionN, ZEROMat, j;

    stdin := InputTextUser();;
    stdout := OutputTextUser();;
    MyDir:=Directory("/home/bernhard");
    LocationOfCHOPAsString := "/home/bernhard/Schreibtisch/shared__meataxe-1.0/bin/chop";
    LocationOfPWKONDAsString := "/home/bernhard/Schreibtisch/shared__meataxe-1.0/bin/pwkond";
    LocationOfMKHOMAsString := "/home/bernhard/Schreibtisch/shared__meataxe-1.0/bin/mkhom";
    LocationOfZMUAsString := "/home/bernhard/Schreibtisch/shared__meataxe-1.0/bin/zmu";
    LocationOfZIVAsString := "/home/bernhard/Schreibtisch/shared__meataxe-1.0/bin/ziv";

    path := DirectoriesSystemPrograms();
    rm := Filename(path,"rm");
```

```
if IsVectorSpace(M) then # Our other programs are written in such a manner that this
# happens exactly when M is the zero module.
    return(M);
fi;

if IsVectorSpace(N) then # Our other programs are written in such a manner that this
# happens exactly when N is the zero module.
    return(N);
fi;

F:=M.field;
FF:=N.field;

if not IsZero(Size(F) - Size(FF)) then
    Print("The acting fields so not coincide.");
    return(fail);
fi;


if not IsPrime(Size(F)) then
    Print("The acting fields have to be of the form GF(p).");
    return(fail);
fi;

gensOfM:=M.generators;
gensOfN:=N.generators;

DIMensionM := M.dimension;
DIMensionN := N.dimension;

p:=Characteristic(F);

LoadPackage("io");
ChangeDirectoryCurrent("/home/bernhard");


files := Filtered(DirectoryContents(MyDir), f -> Length(f)>3 and f[1] = 'r' and f[2] = 'e' and f[3] = 'p' and f[4] = 'M');
for f in files do
    # Skip all file names not beginning with MAT. or being of the form MAT<zahl>
    if f[5] <> '.' and not ForAll(f{[5..Length(f)]}, IsDigitChar) then
        continue;
    fi;
    f := Filename(MyDir, f);
    RemoveFile(f);
od;

files := Filtered(DirectoryContents(MyDir), f -> Length(f)>3 and f[1] = 'r' and f[2] = 'e' and f[3] = 'p' and f[4] = 'N');
for f in files do
    if f[5] <> '.' and not ForAll(f{[5..Length(f)]}, IsDigitChar) then
        continue;
    fi;
    f := Filename(MyDir, f);
    RemoveFile(f);
od;


files := Filtered(DirectoryContents(MyDir), f -> Length(f)>3 and f[1] = 'h' and f[2] = 'o' and f[3] = 'm' and f[4] = 'M'
  and f[5] = 'N');
for f in files do
    if f[6] <> '.' and not ForAll(f{[6..Length(f)]}, IsDigitChar) then
        continue;
    fi;
    f := Filename(MyDir, f);
    RemoveFile(f);
od;

files := Filtered(DirectoryContents(MyDir), f -> Length(f)>3 and f[1] = 'h' and f[2] = 'o' and f[3] = 'm' and f[4] = 'N'
  and f[5] = 'M');
for f in files do
    if f[6] <> '.' and not ForAll(f{[6..Length(f)]}, IsDigitChar) then
        continue;
    fi;
    f := Filename(MyDir, f);
    RemoveFile(f);
od;
```

```
files := Filtered(DirectoryContents(MyDir), f -> Length(f)>3 and f[1] = 'H' and f[2] = 'o' and f[3] = 'm' and f[4] = 'M' and f[5] = 'N');
for f in files do
    if f[6] <> '.' and not ForAll(f{[6..Length(f)]}, IsDigitChar) then
        continue;
    fi;
    f := Filename(MyDir, f);
    RemoveFile(f);
od;

files := Filtered(DirectoryContents(MyDir), f -> Length(f)>3 and f[1] = 'H' and f[2] = 'o' and f[3] = 'm' and f[4] = 'N' and f[5] = 'M');
for f in files do
    if f[6] <> '.' and not ForAll(f{[6..Length(f)]}, IsDigitChar) then
        continue;
    fi;
    f := Filename(MyDir, f);
    RemoveFile(f);
od;


if IsExistingFile("bc") then
    options:=["bc"];

    pro := Process(MyDir, rm, stdin, stdout, options);
    if not IsZero(pro) then
        Print("The last process did not return zero!");
        return(fail);
    fi;
fi;


repM:=Filename(DirectoryCurrent(), "repM");
repN:=Filename(DirectoryCurrent(), "repN");

for i in [1.. Size(gensOfM)] do
    CMtxBinaryFFMatOrPerm(gensOfM[i],p,Concatenation(repM,".",String(i)));
od;
for i in [1.. Size(gensOfN)] do
    CMtxBinaryFFMatOrPerm(gensOfN[i],p,Concatenation(repN,".",String(i)));
od;




options:=["-g",String(Size(gensOfM)), repM];

pro := Process(MyDir, LocationOfCHOPAsString, stdin, stdout, options);
if not IsZero(pro) then
    Print("The last process did not return zero!");
    return(fail);
fi;


options:=[repM];

pro := Process(MyDir, LocationOfPWKONDAsString, stdin, stdout, options);
if not IsZero(pro) then
    Print("The last process did not return zero!");
    return(fail);
fi;


options:=["repM", "repN", "homMN"];

pro := Process(MyDir, LocationOfMKHOMAsString, stdin, stdout, options);
if not IsZero(pro) then
    Print("The last process did not return zero!");
    return(fail);
fi;


if IsExistingFile(Concatenation("homMN.",String(1))) then # This file exists
# if and only if the hom sets are nontrivial.
```

```
        # The suffix spb is an abbreviation for spinning basis . Cf. line 1093 of the
        # file mkhom.c and the explanations at the end of that file . We compute the
        # inverse matrix of repM.spb using the function ziv .

        options:=[Concatenation(repM,".spb"), "bc"];

        pro := Process(MyDir, LocationOfZIVAsString, stdin, stdout, options);
        if not IsZero(pro) then
            Print("The last process did not return zero!");
            return(fail);
        fi;

        i:=0;
        while IsExistingFile(Concatenation("homMN.",String(i+1))) do
            i:=i+1;
            options:=
            ["bc", Concatenation("homMN.",String(i)), Concatenation("HomMN.",String(i))];
            pro := Process(MyDir, LocationOfZMUAsString, stdin, stdout, options);
            # All basis elements of homMN are multiplied by the inverse of repM.spb
            # in order to transform the output to the standard bases (of M and N) again.
            if not IsZero(pro) then
                Print("The last process did not return zero!");
                return(fail);
            fi;
        od;

        U:=ReadRepFrom("HomMN",i,F);
        return U;
    fi;

    ZEROMat:=List([1..DIMensionM], x -> []);
    for i in [1.. DIMensionM] do
        for j in  [1.. DIMensionN] do
            Add(ZEROMat[i],0);
        od;
    od;

    return [ZEROMat];
end;



StripOffOneCopyOfNFromMIfPossible := function(IND,L)

    local M,N, IRR_CT, DIFFERENZ, HomMN, HomNM, B, C, f, fC, i, DIM_M, FIELD_M, WW,
    flag2 , s, d, PHI, KER, bas, RestOfM, p, MATRIXs, MATRIXphi, List_s, basKerPHI,
    basImPHI, basComplete;

    LoadPackage("io");
    ChangeDirectoryCurrent("/home/bernhard");
    stdin := InputTextUser();;
    stdout := OutputTextUser();;
    MyDir:=Directory("/home/bernhard");

    LocationOfZPOAsString := "/home/bernhard/Schreibtisch/shared_meataxe-1.0/bin/zpo";

    path := DirectoriesSystemPrograms();
    rm := Filename(path,"rm");

    MATRIXs:=Filename(DirectoryCurrent(), "MATRIXs");
    MATRIXphi:=Filename(DirectoryCurrent(), "MATRIXphi");

    M:=ShallowCopy(IND);
    N:=ShallowCopy(L);

    if IsVectorSpace(M) then # this happens exactly when the dimension of M equals 0
        return([0,M]);
    fi;

    if IsVectorSpace(N) then # this happens exactly when the dimension of N equals 0
        return([0,N]);
    fi;


    HomMN := Hom_FpG_MNViaSzoeke(M,N); # this returns a list whose elements
    # are rectangular matrices which together form a basis of Hom_FpG(M,N)

```

```
if HomMN = fail then
    HomMN := MTX.BasisModuleHomomorphisms(M,N);
fi;


HomNM := Hom_FpG_MNViaSzoeke(N,M);

if HomNM = fail then
    HomNM := MTX.BasisModuleHomomorphisms(N,M);
fi;


# Next, we want to compute a non−trivial direct summand of M

B:=ShallowCopy(HomMN);
C:= ShallowCopy(HomNM);
for f in B do # recall that GAP acts from the right
    fC := [];
    for i in [1.. Size(C)] do
        Add(fC, f*(C[i]));
    od;

    DIM_M:=M.dimension;
    FIELD_M:=M.field;

    p := Characteristic(FIELD_M);

    WW := FullRowSpace(FIELD_M,DIM_M);

    flag2:=false;
    if not IsEnvNilpotent(WW,fC) then # hence, f is a splitter in this case ...
        # ...now we compute the variable s from Lemma 3.4 in the article of
        # Brooksbank and Luks
        flag2:=true; # i.e., there exists a splitter in the basis B
        s := FromSplitterToFactorization(WW,fC)[1];
        # We remark that we are in a special situation here: our module N
        # is indecomposable. Now, we use the definition of f−decomposition (see
        # Section 3.1 in the article of Brooksbank and Luks);
        # furthermore, we use the definition of splitter ... in
        # particular: (N_1)f is a direct summand of M2, Ker(f) is a submodule
        # of K_1 and fg is not nilpotent for some g;
        # this implies that the f−image of N_1 is isomorphic to our module N
        # due to the following reasons:
        # 1) the restriction of f to N_1 is injective
        # 2) our module N is indecomposable
        # 3) (N_1)f is simultaneously a submodule and a direct summand of N

        # Next, we compute the kernel of s^d ....... this makes sense due to
        # the Proof of Lemma 3.4 in the article of Brooksbank and Luks

        d:=Size(s);

        List_s:=[s];
        for i in [1.. Size(List_s)] do
CMtxBinaryFFMatOrPerm(List_s[i],p,Concatenation(MATRIXs,".",String(i)));
        od;

        options:=["MATRIXs.1", String(d), "MATRIXphi.1"];

        pro := Process(MyDir, LocationOfZPOAsString, stdin, stdout, options);
        # PO in ZPO is an abbreviation for power...
        # ...we compute s^d here using the Shared C MeatAxe.
        if not IsZero(pro) then
            Print("The last process did not return zero!");
            return(fail);
        fi;

        PHI:=ReadRepFrom(MATRIXphi,1,FIELD_M)[1];
        # we define the matrix PHI := s^d

        options:=["MATRIXs.1"];

        pro := Process(MyDir, rm, stdin, stdout, options);
        if not IsZero(pro) then
            Print("The last process did not return zero !!! ");
            return(fail);
        fi;
```

```
            options:=["MATRIXphi.1"];

            pro := Process(MyDir, rm, stdin, stdout, options);
            if not IsZero(pro) then
                Print("The last process did not return zero !!! ");
                return(fail);
            fi;

            KER := NullspaceMat(PHI); # we compute the kernel of s^d here

            PHI_basis:=BasisVectors(Basis(VectorSpace(GF(p),PHI)));

            ConjugationMatr := [];
            Append(ConjugationMatr,PHI_basis); Append(ConjugationMatr, KER);

            BlockDiagonalGens:=[];
            for i in [1.. Size(M.generators)] do
Add(BlockDiagonalGens, ConjugationMatr * M.generators[i] * ConjugationMatr^-1);
            od;


            # we consider the case KER=0 seperately

            if Size(KER) > 0 then
                # RestOfM := MTX.InducedActionSubmodule(M,basKerPHI);
                Vaux:=VectorSpace(GF(p),KER);
                GensForRestOfM:= List(BlockDiagonalGens, x -> ExtractSubMatrix(x,
[Size(x) - Dimension(Vaux)+1..Size(x)], [Size(x) - Dimension(Vaux)+1..Size(x)] ) );
                RestOfM := GModuleByMats(GensForRestOfM,GF(p));
            else
                RestOfM:=FullRowModule(GF(Characteristic(FIELD_M)),0);
                # Also kann man später bei übergeordneten Programmen das so schreiben,
                # dass man in dem Moment abbricht, wenn die Dimension von RestOfM
                # gleich 0 ist !!!
            fi;

            basComplete := ShallowCopy(ConjugationMatr);

            return([1,RestOfM,basComplete]);
            # if N is isomorphic to a summand of M then return [1,RestOfM,basComplete];
            # here, 1 means true, RestOfM is the rest of M (after stripping off an
            # isomorphic copy of N from M), and basComplete is a conjugation matrix C
            # such that C * rho_M * C^-1 has block diagonal form, where rho_M is
            # the underlying matrix representation of M from above
        fi;
    od;

    return([0,M]);

end;



MaxCommonDirectSummandFq := function(GreenCorresp_Fq,IND_Fq) # we always put the module
    # that we wish to decompose as the first argument; in the program
    # TSModulesAndLiftsOverFq, this is the module GRE where GRE is given by extending
    # scalars of the Green correspondent over Fp from Fp to Fq;
    # we remark further that both modules have to be defined over the
    # same field Fq here; moreover, we remark that in the Green correspondent situation
    # mentioned two lines above, the module which is given as the second argument is
    # not necessarily indecomposable; in this case, the algorithm works nevertheless,
    # as we enter only certain modules (namely: (summands of) GRE as the first argument
    # and sums SU of indecomposable modules such that each summand of GRE occurs in SU
    # with multiplicity at most one (or an indecomposable module) as the second argument)

    local M, N, Hom_FqG_MN, Hom_FqG_NM, f, fC, i, DIM_M, FIELD_M, p, B, C, WW, flag2,
    s, RestOfM;

    LoadPackage("io");
    ChangeDirectoryCurrent("/home/bernhard");

    M := ShallowCopy(GreenCorresp_Fq);
    N := ShallowCopy(IND_Fq);

    if IsVectorSpace(M) then # das passiert nur, wenn die Dimension von M gleich 0 ist...
    # ansonsten habe ich es so programmiert, dass immer MeaTAXe-Moduln anstatt
```

```
    # VectorSpaces herauskommen.
        return([0,M]);
    fi;

    if IsVectorSpace(N) then # das passiert nur, wenn die Dimension von N gleich 0 ist...
        # ...ansonsten habe ich es so programmiert, dass immer MeaTAXe−Moduln anstatt
        # VectorSpaces herauskommen.
        return([0,N]);
    fi;

    Hom_FqG_MN := MTX.BasisModuleHomomorphisms(M,N);
    # this uses GAP's MeaTAxe since the involved field Fq my be
    # too large for the Shared C MeatAxe

    Hom_FqG_NM := MTX.BasisModuleHomomorphisms(N,M);

    B:=ShallowCopy(Hom_FqG_MN);
    C:= ShallowCopy(Hom_FqG_NM);
    for f in B do # GAP acts from the right
        fC := [];
        for i in [1.. Size(C)] do
            Add(fC, f*(C[i]));
        od;

        DIM_M:=M.dimension;
        FIELD_M:=M.field;

        p := Characteristic(FIELD_M);

        WW := FullRowSpace(FIELD_M,DIM_M);

        flag2:=false;
        if not IsEnvNilpotent(WW,fC) then # hence, f is a splitter now
            flag2:=true; # also gibt es einen Splitter in B !!!
            s := FromSplitterToFactorization(WW,fC)[1];

            d:=Maximum(M.dimension,N.dimension);

            PHI := s^d;

            PHInew1 := ShallowCopy(PHI)*One(FIELD_M);


            basImPHI := MTX.SpinnedBasis(PHInew1,M.generators,M.field);

            PHIasModule := MTX.InducedActionSubmodule(M,basImPHI);

            PHInew2 := ShallowCopy(PHI)*One(FIELD_M);

            KER := TriangulizedNullspaceMat(PHInew2);

            KER := ShallowCopy(KER)*One(FIELD_M);

            basKerPHI := MTX.SpinnedBasis(KER,M.generators,M.field);

            ConjugationMatr := [];
            Append(ConjugationMatr,basImPHI); Append(ConjugationMatr, basKerPHI);

            BlockDiagonalGens:=[];
            for i in [1.. Size(M.generators)] do
Add(BlockDiagonalGens, ConjugationMatr * M.generators[i] * ConjugationMatr^−1);
            od;

            if Size(basKerPHI) > 0 then
                Vaux:=VectorSpace(M.field,basKerPHI);

                GensForRestOfM:= List(BlockDiagonalGens, x −> ExtractSubMatrix(x,
[Size(x) − Dimension(Vaux)+1..Size(x)], [Size(x) − Dimension(Vaux)+1..Size(x)] ) );

                RestOfM := GModuleByMats(GensForRestOfM,M.field);
            else
                RestOfM := FullRowModule(M.field,0);
            fi;

            return([PHIasModule,RestOfM,ConjugationMatr]);
                # if there are several splitters, they yield isomorphic copies
                # of the searched submodule...
```

```
                #... therefore, we stop the present calculation as soon as one splitter
                # is found; we mention here that we have not fully implemented
                # a GAP/Shared C MeatAxe version of the pseudo code to find a maximal
                # common direct summand of two non-isomorphic modules from the article
                # of Brooksbank and Luks; instead, we have only implemented a
                # simplified version of it which only works for our purposes.
        fi;

    od;
    return([0,M]);
end;
```

```
# The program ReadRepFrom is an adapted version of a program taken from René Zimmermann's
# PhD thesis, see R. Zimmermann, Vertizes einfacher Moduln Symmetrischer Gruppen,
# PhD thesis (German), University of Jena, Jena, 2004.

ReadRepFrom:=function(file,nrgens,field)
    # Reading the MeatAxe matrices file .1, .., file .nrgens, which are binary files , into GAP
    local rep, i, LocationOfZPRAsString, path, rm, stdin, stdout, MyDir, options, pro;
    LoadPackage("io");
    ChangeDirectoryCurrent("/home/bernhard");

    stdin := InputTextUser();;
    stdout := OutputTextUser();;
    MyDir:=Directory("/home/bernhard");
    LocationOfZPRAsString := "/home/bernhard/Schreibtisch/shared_meataxe-1.0/bin/zpr";

    rep:=[];
    for i in [1.. nrgens] do
        options:=[Concatenation(file,".",String(i)), Concatenation(file,".text")];
        pro := Process(MyDir, LocationOfZPRAsString, stdin, stdout, options);
        if not IsZero(pro) then
            Print("The last process did not return zero!");
            return(fail);
        fi;
        rep[i]:=ScanMeatAxeFile(Concatenation(file,".text"),Size( field ));
    od;

    path := DirectoriesSystemPrograms();
    rm := Filename(path,"rm");

    options:=[Concatenation(file,".text")];

    pro := Process(MyDir, rm, stdin, stdout, options);
    if not IsZero(pro) then
        Print("The last process did not return zero!");
        return(fail);
    fi;
    return rep;
end;

# The program CoefficientsOfOsimaIdempotent is written by Thomas Breuer.
# See http://www.math.rwth-aachen.de/~Thomas.Breuer/ctblocks/doc/chap3.html#X79C6AA8A7AD53766

#####################################################################
##
#F  CoefficientsOfOsimaIdempotent( <tbl>, <p>, <b> )
##
CoefficientsOfOsimaIdempotent := function( tbl, p, b )

    local blocks, coeffs, irr, i, chi;

    blocks:= PrimeBlocks( tbl, p );
    coeffs:= 0 * [ 1 .. NrConjugacyClasses( tbl ) ];
    irr:= List( Irr( tbl ), ValuesOfClassFunction );
    for i in [ 1 .. Length( blocks.block ) ] do
      if blocks.block[i] = b then
        chi:= irr[i];
        coeffs:= coeffs + chi[1] * chi;
      fi;
    od;

    return List( coeffs, ComplexConjugate ) / Size( tbl );
    end;
#####################################################################

# The program EquivalentLibraryCharacterTableWithGroup is written by Thomas Breuer:

#####################################################################
##
#F  EquivalentLibraryCharacterTableWithGroup( <G> )
##
EquivalentLibraryCharacterTableWithGroup:= function( G )
    local init, Gcopy, name, attr, Gtbl, tbl, trans, compat, ccl, new, i;

    # If the group stores already an ordinary character table
    # then we cannot set the attributes consistently .
    if HasOrdinaryCharacterTable( G ) then
      Error( "<G> has already a character table" );
```

```
    fi;

    # Test cheap attributes first, and exclude duplicates.
    init:= AllCharacterTableNames( Size, Size( G ),
                  NrConjugacyClasses, NrConjugacyClasses( G ),
                  IsDuplicateTable, false );
    if Length( init ) = 0 then
      # No expensive tests are needed.
      # In particular, do not compute a character table.
      return fail;
    fi;

    # Create a copy of the group, in order to compute its character table
    # without storing it.
    # (Note that calling 'AttributeValueNotSet' for 'OrdinaryCharacterTable'
    # does not help, since 'Irr' etc. would appear silently.)
    # Store the known attributes of 'G' in the copy,
    # in particular 'Gcopy' and 'G' have the same ordering of conj. classes.
    Gcopy:= GroupWithGenerators( GeneratorsOfGroup( G ) );
    for name in KnownAttributesOfObject( G ) do
      attr:= ValueGlobal( name );
      Setter( attr )( Gcopy, attr( G ) );
    od;

    # Compute the character table of the copy.
    Gtbl:= OrdinaryCharacterTable( Gcopy );
    for name in init do
      tbl:= CharacterTable( name );
      trans:= TransformingPermutationsCharacterTables( tbl, Gtbl );
      if trans <> fail then
        # Take this library table:
        # - Permute the classes stored in the group.
        compat:= ListPerm( trans.columns, NrConjugacyClasses( tbl ) );
        ccl:= ConjugacyClasses( G ){ compat };

        # - Copy the contents of the library table.
        new:= ConvertToLibraryCharacterTableNC(
                 rec( UnderlyingCharacteristic := 0 ) );

        # - Set the supported attribute values except 'Irr'.
        for i in [ 3, 6 .. Length( SupportedCharacterTableInfo ) ] do
          if Tester( SupportedCharacterTableInfo[ i-2 ] )( tbl )
             and SupportedCharacterTableInfo[ i-1 ] <> "Irr" then
            Setter( SupportedCharacterTableInfo[ i-2 ] )( new,
                SupportedCharacterTableInfo[ i-2 ]( tbl ) );
          fi;
        od;

        # - Set the irreducibles.
        SetIrr( new, List( Irr( tbl ),
            chi -> Character( new, ValuesOfClassFunction( chi ) ) ) );

        # - Set the group in the table.
        SetUnderlyingGroup( new, G );
        SetConjugacyClasses( new, ccl );
        SetIdentificationOfConjugacyClasses( new, compat );

        # - Set the table in the group.
        SetOrdinaryCharacterTable( G, new );

        return new;
      fi;
    od;

    # No library table fits.
    # However, we set the computed character table, since we know it.
    SetOrdinaryCharacterTable( G, Gtbl );
    return fail;
    end;

# The following program FromStringToMatrix is an auxiliary program.
#
# Let G be a group with generators gensG, e.g. constructed by the GAP command
# GroupWithGenerators(gensG). Here, gensG is a list of nrgens generators of G.
# Let x be an element of G.
# Moreover, suppose that the matrices M.1,..., M.nrgens are already constructed in
# binary format. In our applications, these matrices are the images of the group generators
```

```
# gensG under a linear group representation .
#
# The string returned by the command Factorization(G,x) is transformed into the corresponding
# product of matrices. The latter product is the output of our auxiliary program.
# Example: if x=(g1*g2)^3, then the matrix (M.1*M.2)^3 is returned.

FromStringToMatrix := function(G,gensG,p,fAsString)

    local DirOfChop, M, i, RES, ERGEBNIS, MAT, StringNow, z, PositionsOpenParentheses,
        PositionsCloseParentheses, KLAMMER, r, SSS, STR, SPLITnow, KlammerAuf, KlammerZu, u,
        StringNow1, StringNow2, StringNow3, StringYNow, StringToChange, SPLIT, INP,
        ergebnis_to_return, stdin, stdout, MyDir, LocationOfZPRAsString,
        LocationOfZPOAsString, LocationOfZMUAsString, path, rm, options, pro, dir, files, f;

    LoadPackage("io");
    # DirOfChop:=Directory("/home/bernhard/Schreibtisch/shared_meataxe-1.0/src/");
    # ChangeDirectoryCurrent("/home/bernhard");

    MyDir:=Directory("/home/bernhard");
    stdin := InputTextUser();;
    stdout := OutputTextUser();;
    LocationOfZPRAsString := "/home/bernhard/Schreibtisch/shared_meataxe-1.0/bin/zpr";
    LocationOfZPOAsString := "/home/bernhard/Schreibtisch/shared_meataxe-1.0/bin/zpo";
    LocationOfZMUAsString := "/home/bernhard/Schreibtisch/shared_meataxe-1.0/bin/zmu";
    path := DirectoriesSystemPrograms();
    rm := Filename(path,"rm");

    RES:=Filename(MyDir, "RES");

    ERGEBNIS:=Filename(MyDir, "ERGEBNIS");
    MAT:=Filename(DirectoryCurrent(), "MAT");
    KLAMMER:=Filename(DirectoryCurrent(), "KLAMMER");

    # My directory:
    dir := Directory("/home/bernhard");
    # All files in the directory which start with RES and are not called M
    files := Filtered(DirectoryContents(MyDir),
    f -> Length(f)>3 and f[1] = 'R' and f[2] = 'E' and f[3] = 'S');
    for f in files do
        # Skip all files with names not starting with RES. or having the form RES<zahl>
        if f[4] <> '.' and not ForAll(f{[4..Length(f)]}, IsDigitChar) then
            continue;
        fi;
        f := Filename(MyDir, f);
        RemoveFile(f);
    od;

    files := Filtered(DirectoryContents(MyDir),
    f -> Length(f)>8 and f[1] = 'E' and f[2] = 'R' and f[3] = 'G' and f[4] = 'E' and f[5] =
    'B' and f[6] = 'N' and f[7] = 'I' and f[8] = 'S');
    for f in files do
# Skip all files with names not starting with ERGEBNIS. or having the form ERGEBNIS<zahl>
        if f[9] <> '.' and not ForAll(f{[9..Length(f)]}, IsDigitChar) then
            continue;
        fi;
        f := Filename(MyDir, f);
        RemoveFile(f);
    od;

    files := Filtered(DirectoryContents(MyDir), f -> Length(f)>3 and f[1] =
    'M' and f[2] = 'A' and f[3] = 'T');
    for f in files do
        # Skip all files with names not starting with MAT. or having the form MAT<zahl>
        if f[4] <> '.' and not ForAll(f{[4..Length(f)]}, IsDigitChar) then
            continue;
        fi;
        f := Filename(MyDir, f);
        RemoveFile(f);
    od;

    files := Filtered(DirectoryContents(MyDir), f -> Length(f)>7 and f[1] = 'K' and f[2] =
    'L' and f[3] = 'A' and f[4] = 'M' and f[5] = 'M' and f[6] = 'E' and f[7] = 'R');
    for f in files do
        # Skip all files with names not starting with KALMMER. or having the form KLAMMER<zahl>
        if f[8] <> '.' and not ForAll(f{[8..Length(f)]}, IsDigitChar) then
            continue;
        fi;
```

```
        f := Filename(MyDir, f);
        RemoveFile(f);
    od;

    StringNow:=ShallowCopy(fAsString);
    z:=0;

    # The first step is to eliminate all occurring parentheses in the string fAsString.
    # We search for the innermost pair of parentheses and evaluate the expression enclosed by
    # them. Iteratively, after finitely many steps we are done.

    while '(' in StringNow do
        PositionsOpenParentheses := Positions(StringNow,'(');
        PositionsCloseParentheses := Positions(StringNow,')');
        z:=z+1;
        KlammerZu:=PositionsCloseParentheses[1]; # find first closing parenthesis
        u:=PositionsCloseParentheses[1];
        while (u in PositionsOpenParentheses)=false do
            u:=u-1;
        od;
        KlammerAuf:=u;
        StringToChange := StringNow{ [KlammerAuf+1..KlammerZu-1] } ;
        SPLIT:=SplitString(StringToChange, "*");

        for i in [1.. Size(SPLIT)] do
            STR:=SPLIT[i];
            SPLITnow:=SplitString(STR,"^");
            if Size(SPLITnow)=2 then # In this case the expression contains an exponent.
                SSS:=ReplacedString(SPLITnow[1],"x","M");
                SSS:=ReplacedString(SSS,"y","KLAMMER.");

                options:=[SSS, String(SPLITnow[2]), Concatenation("MAT.",String(i))];

                pro := Process(MyDir, LocationOfZPOAsString, stdin, stdout, options);
                if not IsZero(pro) then
                    Print("The last process did not return zero!");
                    return(fail);
                fi;
            else
                SSS:=ReplacedString(SPLITnow[1],"x","M");
                SSS:=ReplacedString(SSS,"y","KLAMMER.");

                options:=[SSS, String(1), Concatenation("MAT.",String(i))];

                pro := Process(MyDir, LocationOfZPOAsString, stdin, stdout, options);
                if not IsZero(pro) then
                    Print("The last process did not return zero!");
                    return(fail);
                fi;
            fi;
        od;

        options:=["MAT.1","MAT.2","RES.2"];

        pro := Process(MyDir, LocationOfZMUAsString, stdin, stdout, options);
        if not IsZero(pro) then
            Print("The last process did not return zero!");
            return(fail);
        fi;

        for i in [2.. Size(SPLIT)-1] do

            options:=[Concatenation("RES.",String(i)),
            Concatenation("MAT.",String(i+1)),Concatenation("RES.",String(i+1))];

            pro := Process(MyDir, LocationOfZMUAsString, stdin, stdout, options);
            if not IsZero(pro) then
                Print("The last process did not return zero!");
                return(fail);
            fi;
        od;

        r:=Maximum(Size(SPLIT),2);

        options:=[Concatenation("RES.",String(r)), "1", Concatenation("KLAMMER.",String(z))];

        pro := Process(MyDir, LocationOfZPOAsString, stdin, stdout, options);
```

```
        if not IsZero(pro) then
            Print("The last process did not return zero!");
            return(fail);
        fi;

        if KlammerAuf > 1 then
            StringNow1 := StringNow{ [1..KlammerAuf-1] } ;
        else
            StringNow1 := "";
        fi;

        StringNow2 := Concatenation("KLAMMER.",String(z));

        if KlammerZu < Size(StringNow) then
            StringNow3 := StringNow{ [KlammerZu+1..Size(StringNow)] } ;
        else
            StringNow3 := "";
        fi;
        StringYNow := Concatenation("y",String(z));
        StringNow:=Concatenation(StringNow1,StringYNow,StringNow3);
    od;

    # We have finally eliminated all parentheses.

    StringToChange := StringNow; # This string does not contain any parentheses anymore,
    # but it can still contain exponents or multiplication symbols.

    SPLIT:=SplitString(StringToChange, "*");

    for i in [1.. Size(SPLIT)] do
        STR:=SPLIT[i];
        SPLITnow:=SplitString(STR,"^");
        if Size(SPLITnow)=2 then # this means that ^ occurs in the present string
            SSS:=ReplacedString(SPLITnow[1],"x","M");
            SSS:=ReplacedString(SSS,"y","KLAMMER.");

            options:=[SSS, String(SPLITnow[2]), Concatenation("MAT.",String(i))];

            pro := Process(MyDir, LocationOfZPOAsString, stdin, stdout, options);
            if not IsZero(pro) then
                Print("The last process did not return zero!");
                return(fail);
            fi;
        else
            SSS:=ReplacedString(SPLITnow[1],"x","M");
            SSS:=ReplacedString(SSS,"y","KLAMMER.");

            options:=[SSS, String(1), Concatenation("MAT.",String(i))];

            pro := Process(MyDir, LocationOfZPOAsString, stdin, stdout, options);
            if not IsZero(pro) then
                Print("The last process did not return zero!");
                return(fail);
            fi;
        fi;
    od;

    if Size(SPLIT) = 1 then

        options:=[Concatenation(MAT,".",String(1)), Concatenation(ERGEBNIS,".text")];

        pro := Process(MyDir, LocationOfZPRAsString, stdin, stdout, options);
        if not IsZero(pro) then
            Print("The last process did not return zero!");
            return(fail);
        fi;
        INP := InputTextString
( Concatenation("ergebnis"," := ScanMeatAxeFile(","\"",Concatenation(ERGEBNIS,".text"),"\"",");" ));;
        Read(INP);
    else

        options:=["MAT.1","MAT.2","RES.2"];

        pro := Process(MyDir, LocationOfZMUAsString, stdin, stdout, options);
        if not IsZero(pro) then
            Print("The last process did not return zero!");
            return(fail);
```

```
        fi;

        for i in [2.. Size(SPLIT)-1] do

            options:=[Concatenation("RES.",String(i)), Concatenation("MAT.",String(i+1)),
            Concatenation("RES.",String(i+1))];

            pro := Process(MyDir, LocationOfZMUAsString, stdin, stdout, options);
            if not IsZero(pro) then
                Print("The last process did not return zero!");
                return(fail);
            fi;
        od;

        r:=Maximum(Size(SPLIT),2);

        options:=[Concatenation("RES.",String(r)), "1", Concatenation("ERGEBNIS.",String(1))];

        pro := Process(MyDir, LocationOfZPOAsString, stdin, stdout, options);
        if not IsZero(pro) then
            Print("The last process did not return zero!");
            return(fail);
        fi;

        options:=[Concatenation(ERGEBNIS, ".", String(1)), Concatenation(ERGEBNIS, ".text")];

        pro := Process(MyDir, LocationOfZPRAsString, stdin, stdout, options);
        if not IsZero(pro) then
            Print("The last process did not return zero!");
            return(fail);
        fi;

        INP := InputTextString
( Concatenation("ergebnis"," := ScanMeatAxeFile(","\"",Concatenation(ERGEBNIS,".text"),"\"",");" ));;
        Read(INP);
    fi;

    ergebnis_to_return := ShallowCopy(ergebnis);

    return ergebnis_to_return;
end;

#################################################################

# The following program is the main program:

# In order to compute the projective indecomposable modules over a splitting field k we apply
# the following strategy:
#
# 1) Computation of the regular GF(p)G-module as a matrix representation
# 2) Computation of the centrally primitive idempotents of the group algebra GF(p)G
# 3) Evaluation of the regular representation at these idempotent group algebra elements yields
# idempotent matrices which blockwise decompose the regular representation as a direct sum of
# subrepresentations
# 4) Apply the programs of Magdolna Szöke, Klaus Lux, Jürgen Müller and Michael Ringe to those
# new representations which are still projective (and hence faithful when considered as modules
# over the block algebra) in order to get the PIMs over GF(p)G
# 5) Compute Hom_{kG}(k\oplus_{GF(p)}P,k\oplus_{GF(p)}P) for each projective indecomposable
# GF(p)G-module as a subalgebra of a full matrix algebra where k is a minimal splitting field
# and hence possibly varying for each P
# 6) Compute a complete set of orthogonal primitive idempotents of this endomorphism ring
# 7) Compute the submodules corresponding to the latter primitive idempotents and save a basis
# with respect to which the generator matrices of the modules in 5) have block diagonal form
# since this is needed later
# 8) Compute the Brauer characters and the ordinary characters of the projective indecomposable
# kG-modules

PIMsFqG:=function(G,p)

    local DirOfChop, gensG, F, RegularModuleOverF, REG, MatricesRegularRep, i, mat,
    NumberOfSimplesOverF, MatricesSimplesOverF, ctG, UUU, ctGmodp, dec, cclsG, p_prime_cclsG,
    BrauerCharsSimplesOverFasClassFunctions, a1, hom, BrVals_hom, phi,
    BrauerCharsOverSplittingField, pModularReductionsOfOrdinaryChars, j, temp,
    BrauerCharsPIMsOverSplittingField, ScalProdsOfSimplesFWithPIMsOfDecMatrix,
    SimplesForVerification, ListOfDifferences, DimensionsOfPIMsOverF, s, MatricesPIMsOverF,
    MODU, x, exp, facts, pprimefacts, f, k, PIMNumbersToSplitLater, AllPIMsOver_kAsMTXModules,
    BrauerCharsRepresentationsOfThePIMsOver_k, IdentifyingG, HOMs, OrdinaryCharsOfThePIMs,
```

```
temp__ordinary__classes, counter, Chi__PIM, temp__scalprods, temp__for__sort, temp__for__sort2,
temp__for__sort3, temp__for__sort4, MyRecord, IrrCT, List__Bildmatrizen, elt, pos, COAndMAGMAnow,
  ListCoefficients , IdempotentMatrices, V, BVS, PIMsInBlocks, BlockIdempotsOverFp, vectors,
 bas, sub, AllSimplesSortedInBlocks, v, AllBrauerCharsRepresentationsOfThePIMsOver__F,
 All__temp__scalprods, PIMsHere, rep, MatricesSimplesHere, a, b, DegreesSplittingFields,
 ListAllPIMsOverSplittingFields, AllBlockDiagonalGens, AllPIMsOver__FAsMTXModules,
 BasisGalConjugates, k__new, P__new, HomPP, A, pids, SizePids, E__M__B__i,
 List__modular__cen__prim__ids, NumberOfBlocks, CoefsOverC, e__B__j, FrobAut, r, e__j__now, flag,
 IdempotentKontrolleur, MAGMAElts, ListNEWCoefficients, FpBlock__aktuell, RelevantGroupEls,
 Dictionary, List__Factorizations, List__Factorizations__AsStrings,
 BrauerCharsRepresentationsOfThePIMsOver__F, AllOrdinaryCharsOfThePIMs__OverF,
 AllPIMsSortedInBlocks, u, Irr__As__List__Of__Lists, SimpleModulesOverF,
 SimpleModulesOverFforlater, AllBasesForGaloisConjugates, B, ConjugatedGeneratorMatrices,
 BlockDiagonalGens, E__M__B, pbs, cc, m, kG, BlockIdempotsToSumLater, NewIdempot,
 NumberOfSimplesHere, ModulesSimpleHere, OrdinaryCharsOfThePIMs__OverF, OldGens,
 BrauerCharsPIMsOverFqAsClassFunctions, OrdinaryCharsOfThePIMs__OverFq, ScalprodsPIMsOverFq,
 stdin, stdout, MyDir, LocationOfZPRAsString, LocationOfZPOAsString, LocationOfZMUAsString,
 path, rm, LocationOfCHOPAsString, LocationOfPWKONDAsString, LocationOfZSPAsString, options,
 pro, dir, files , CoefsOverFq, MM;

LoadPackage("io");
ChangeDirectoryCurrent("/home/bernhard");

MyDir:=Directory("/home/bernhard");
stdin := InputTextUser();;
stdout := OutputTextUser();;

LocationOfZPRAsString := "/home/bernhard/Schreibtisch/shared__meataxe-1.0/bin/zpr";
LocationOfZPOAsString := "/home/bernhard/Schreibtisch/shared__meataxe-1.0/bin/zpo";
LocationOfZMUAsString := "/home/bernhard/Schreibtisch/shared__meataxe-1.0/bin/zmu";
LocationOfCHOPAsString := "/home/bernhard/Schreibtisch/shared__meataxe-1.0/bin/chop";
LocationOfPWKONDAsString := "/home/bernhard/Schreibtisch/shared__meataxe-1.0/bin/pwkond";
LocationOfZSPAsString := "/home/bernhard/Schreibtisch/shared__meataxe-1.0/bin/zsp";

path := DirectoriesSystemPrograms();
rm := Filename(path,"rm");

if Order(G)= 1 then
    G:=Group(());
fi;
LoadPackage("io");
# DirOfChop:=Directory("/home/bernhard/Schreibtisch/shared__meataxe-1.0/src/");
# ChangeDirectoryCurrent("/home/bernhard/Schreibtisch/shared__meataxe-1.0");
gensG:=GeneratorsOfGroup(G);

if HasOrdinaryCharacterTable(G) then
    ctG:=CharacterTable(G);
else
    UUU:=EquivalentLibraryCharacterTableWithGroup(G);
    ctG:=CharacterTable(G);
fi;

F:=GF(p); # GF is an abbreviation for Galois field .
cc := ConjugacyClasses(ctG);
exp:=Exponent(G);
x:= X(GF(p), "x");
exp:=Exponent(G);
facts:=Factors(exp);
pprimefacts:=Filtered(facts, x-> x mod p <> 0 mod p);
m:=Product(pprimefacts);
f:=x^m - 1;
k:=SplittingField(f); # This is done in order to define a group ring which is large enough
# for the computations to come. Later, we choose the finite field as small as possible when
# dealing with modules.

RegularModuleOverF:=RegularModule(G,F);
REG:=RegularModuleOverF[2];
pbs:=PrimeBlocks(ctG,p);
NumberOfBlocks := Maximum(pbs.block);
kG:=GroupRing(k,G);

List__modular__cen__prim__ids:=[];

for j in [1.. NumberOfBlocks] do
    CoefsOverC:=CoefficientsOfOsimaIdempotent(ctG,p,j);
    CoefsOverFq:=List(CoefsOverC, x-> FrobeniusCharacterValue(x,p));
    # See https://www.gap-system.org/Manuals/doc/ref/chap72.html#X79BACBC47B4C413E.
```

```

        e_B_j := Sum( cc , U -> ElementOfMagmaRing(
                                    FamilyObj( Zero( kG ) ),
                                    Zero( k ),
                                    List(U,u->CoefsOverFq[Position(cc,U)]),
                                    AsList(U) )
                    );
        Add(List_modular_cen_prim_ids,e_B_j);
    od;

    # Hence, we have computed all centrally primitive idempotents of the group ring kG. We are
    # now interested in the calculation of all centrally primitive idempotents of the group
    # ring GF(p)G.
    # In order to compute these, we only have to do the following: let e_1 be the first
    # idempotent in the list List_modular_cen_prim_ids. Apply the Frobenius automorphism to all
    # coefficients of e_1 and denote the resulting idempotent by e_2.
    # Continue this process until e_1 is reached again. The sum of all obtained different
    # idempotents is a centrally primitive idempotents of the group ring GF(p)G.

    FrobAut:=FrobeniusAutomorphism(k);

    BlockIdempotsToSumLater:=[];

    IdempotentKontrolleur:=[];

    for j in [1.. NumberOfBlocks] do
        r:=1;
        FpBlock_aktuell:=[];
        flag:=false;
        e_j_now:=List_modular_cen_prim_ids[j];
        if not e_j_now in IdempotentKontrolleur then
            Add(IdempotentKontrolleur,e_j_now);
            Add(FpBlock_aktuell,e_j_now);
            COAndMAGMAnow:=CoefficientsAndMagmaElements(e_j_now);
            MAGMAElts:=[];
            for i in [1.. Size(COAndMAGMAnow)/2] do
                Add(MAGMAElts,COAndMAGMAnow[2*i-1]);
            od;

            while flag=false do
                ListNEWCoefficients:=[];
                for i in [1.. Size(COAndMAGMAnow)/2] do
                    Add(ListNEWCoefficients,COAndMAGMAnow[2*i]^(FrobAut^r));
                od;

NewIdempot := ElementOfMagmaRing(FamilyObj(Zero(kG)), Zero(k), ListNEWCoefficients, MAGMAElts);
                if NewIdempot in FpBlock_aktuell then
                    flag:=true;
                else
                    Add(FpBlock_aktuell,NewIdempot);
                    Add(IdempotentKontrolleur,NewIdempot);
                    r:=r+1;
                fi;
            od;
            Add(BlockIdempotsToSumLater,FpBlock_aktuell);
        fi;
    od;

    BlockIdempotsOverFp:=List(BlockIdempotsToSumLater, x -> Sum(x));

    files := Filtered(DirectoryContents(MyDir), f -> Length(f)>1 and f[1] = 'M');
    for f in files do
        if f[2] <> '.' and not ForAll(f{[2..Length(f)]}, IsDigitChar) then
            continue;
        fi;
        f := Filename(MyDir, f);
        RemoveFile(f);
    od;

    M:=Filename(DirectoryCurrent(), "M");

    for i in [1.. Size(REG.generators)] do
        CMtxBinaryFFMatOrPerm(REG.generators[i],p,Concatenation(M,String(i)));
    od;

    RelevantGroupEls:=[];
    # We find out which elements of G occur as coefficients of the block idempotents.
```

```
    for j in [1.. Size(BlockIdempotsOverFp)] do
        e_j_now:=BlockIdempotsOverFp[j];
        COAndMAGMAnow:=CoefficientsAndMagmaElements(e_j_now);
        for i in [1.. Size(COAndMAGMAnow)/2] do
            if (COAndMAGMAnow[2*i-1] in RelevantGroupEls)=false then
                Add(RelevantGroupEls,COAndMAGMAnow[2*i-1]);
            fi;
        od;
    od;

Dictionary:=[];
for i in [1.. Size(RelevantGroupEls)] do
    Add(Dictionary,[]);
od;

for i in [1.. Size(RelevantGroupEls)] do
    Add(Dictionary[i],RelevantGroupEls[i]);
od;

# It is important for the next command that the generators of G have already been computed.

List_Factorizations:=List(RelevantGroupEls, x -> Factorization(G,x));

List_Factorizations_AsStrings:=List(List_Factorizations, x -> String(x));

# We replace the string "<identity ...>" by the string "x1*x1^-1".

for j in [1.. Size(List_Factorizations_AsStrings)] do
    if 'i' in List_Factorizations_AsStrings[j] then # i.e. if we have "<identity ...>" here
        List_Factorizations_AsStrings[j] := "x1*x1^-1";
    fi;
od;

for i in [1.. Size(RelevantGroupEls)] do
    STr:=List_Factorizations_AsStrings[i];
    Add(Dictionary[i],FromStringToMatrix(G,gensG,p,STr));
od;

    IdempotentMatrices:=[];

    for j in [1.. Size(BlockIdempotsOverFp)] do
        e_j_now := BlockIdempotsOverFp[j];
        COAndMAGMAnow := CoefficientsAndMagmaElements(e_j_now);
        ListCoefficients :=[];
        for i in [1.. Size(COAndMAGMAnow)/2] do
            Add(ListCoefficients,COAndMAGMAnow[2*i]);
        od;
        List_Bildmatrizen := [];
        for i in [1.. Size(COAndMAGMAnow)/2] do
        # We search for the position of the element COAndMAGMAnow[2*i-1] in the list Dictionary
        # and add the corresponding matrix to the list List_Bildmatrizen.
            elt := COAndMAGMAnow[2*i-1];
            pos := Position(RelevantGroupEls,elt);
            Add(List_Bildmatrizen,Dictionary[pos][2]);
        od;

        temp:=[];
        for i in [1.. Size(COAndMAGMAnow)/2] do
            Add(temp, ListCoefficients[i]*List_Bildmatrizen[i]);
        od;

        Add(IdempotentMatrices, Sum(temp));

    od;

    # Evaluating the GF(p)-linearly extended regular representation at the block idempotents
    # yields idempotent matrices E_i. They decompose our
    # vector space k^n accordingly: k^n = E_1 * k^n + ... + E_r * k^n (direct sum).

    V:=FullRowSpace(REG.field,REG.dimension);
    BVS:=BasisVectors(Basis(V));

    PIMsInBlocks:=[];

    for j in [1.. Size(BlockIdempotsOverFp)] do
        vectors:=[];
        MM:=IdempotentMatrices[j];
```

```
        for s in BVS do
            Add(vectors, s*MM); # GAP acts from the right.
        od;

        bas:=MTX.SpinnedBasis(vectors,REG.generators,REG.field);
        sub:=MTX.InducedActionSubmodule(REG,bas);
        Add(PIMsInBlocks,sub); # This list gives the block decomposition of the regular module
        # into projective direct summands.
    od;

    # this deletes some old files :

    files := Filtered
    (DirectoryContents(MyDir), f -> Length(f)>1 and f[1] = 'm' and f[2] = 'a' and f[3] = 't' );
    for f in files do
        if f[4] <> '.' and not ForAll(f{[4..Length(f)]}, IsDigitChar) then
            continue;
        fi ;
        f := Filename(MyDir, f);
        RemoveFile(f);
    od;

    files := Filtered(DirectoryContents(MyDir), f -> Length(f)>1 and f[1] =
    'r' and f[2] = 'e' and f[3] = 'p' );
    for f in files do
        if f[4] <> '.' and not ForAll(f{[4..Length(f)]}, IsDigitChar) then
            continue;
        fi ;
        f := Filename(MyDir, f);
        RemoveFile(f);
    od;

    AllSimplesSortedInBlocks:=[];

    for v in [1.. Size(BlockIdempotsOverFp)] do
        Add(AllSimplesSortedInBlocks,[]);
    od;

    AllPIMsSortedInBlocks:=[];

    for v in [1.. Size(BlockIdempotsOverFp)] do
        Add(AllPIMsSortedInBlocks,[]);
    od;

    AllBrauerCharsRepresentationsOfThePIMsOver_F:=[];
    AllOrdinaryCharsOfThePIMs_OverF:=[];
    All_temp_scalprods:=[];

    ctGmodp:=ctG mod p;
    Display(ctGmodp);
    dec:=DecompositionMatrix(ctGmodp);

    cclsG:=ConjugacyClasses(ctG);

    # TO DO: Hier jetzt das mit dem neuen Plan für die p'-classes einfügen !!! also mit ClassNamesBrauerTableG,
      ClassNamesOrdinaryTableG und Position(ClassNamesOrdinary, ClassNamesBrauer[3]); !!! und an ALLEN Parallelstellen !!!!
    # UND DANN NOT TO FORGET: Die database "löschen", damit nicht mit den alten, zT falschen Ergebnissen gerechnet wird (z.B
      . bei einer Probe mit SmallGroup(44,1) für p=2 !)!


    p_prime_cclsG:=[];
    ClassNamesOrdinaryTableG:=ClassNames(ctG);
    ClassNamesBrauerTableG:=ClassNames(ctGmodp);
    for i in [1.. Size(ClassNamesBrauerTableG)] do
        pos:=Position(ClassNamesOrdinaryTableG, ClassNamesBrauerTableG[i]);
        Add(p_prime_cclsG,cclsG[pos]);
    od;

    # ALTE, wahrscheinl. falsche Variante:
    # for i in [1.. Size(cclsG)] do
    #     if not Order(Representative(cclsG[i])) mod p = (0 mod p) then
    #         Add(p_prime_cclsG,cclsG[i]);
    #     fi ;
    # od;

    MatricesSimplesOverF:=[];

```

```
BrauerCharsOverSplittingField:=[];
for i in [1.. Size( Irr (ctGmodp))] do
    Add(BrauerCharsOverSplittingField,Irr(ctGmodp)[i]);
od;

pModularReductionsOfOrdinaryChars:=[];
for j in [1.. Size( Irr (ctG))] do
    temp:=[];
    for i in [1.. Size( Irr (ctGmodp))] do
        Add(temp,dec[j][i]*BrauerCharsOverSplittingField[i]);
    od;
    Add(pModularReductionsOfOrdinaryChars,Sum(temp));
od;

BrauerCharsPIMsOverSplittingField:=[];
for i in [1.. Size( Irr (ctGmodp))] do
    temp:=[];
    for j in [1.. Size( Irr (ctG))] do
        Add(temp, dec[j][i]*pModularReductionsOfOrdinaryChars[j]);
    od;
    Add(BrauerCharsPIMsOverSplittingField,Sum(temp));
od; # Hence, the list BrauerCharsPIMsOverSplittingField and the columns of the decomposition
# matrix dec are sorted in the same fahion.

# ##################################################################
#
# Next, we begin with the construction of a list consisting
# of pairs (Simples, complex IBr). This is needed and finished later .
#
# ##################################################################

# The aim is that the orderings of p_prime_classes_IBr
# and p_prime_cclsG coincide.

# Recall: p_prime_cclsG is the list of p'-classes of G.

Representatives_p_prime_cclsG := List(p_prime_cclsG, x -> Representative(x));


ListComplexIBrs:=List([1..Size(Irr(ctGmodp))], x -> []);

for a in [1.. Size( Irr (ctGmodp))] do
    for b in [1.. Size( Irr (ctGmodp))] do
        Add(ListComplexIBrs[a], Representatives_p_prime_cclsG[b]^Irr(ctGmodp)[a]);
    od;
od;

ListFrobeniusCharValsOfComplexIBrs:=List([1..Size(Irr(ctGmodp))], x -> []);

for a in [1.. Size( Irr (ctGmodp))] do
    for b in [1.. Size( Irr (ctGmodp))] do
Add(ListFrobeniusCharValsOfComplexIBrs[a],FrobeniusCharacterValue(Representatives_p_prime_cclsG[b]^Irr(ctGmodp)[a],p));
    # This makes sense due to Curtis-Reiner (1962) page 587 (82.5) Corollary
    od;
od;



List_Factorizations_p_prime_cclsG :=
List(Representatives_p_prime_cclsG, x -> Factorization(G,x));

List_Factorizations_AsStrings_p_prime_cclsG :=
List(List_Factorizations_p_prime_cclsG, x -> String(x));

        # We replace the string "<identity ...>" by the string "x1*x1^-1".

        for ww in [1..Size(List_Factorizations_AsStrings_p_prime_cclsG)] do
            if 'i' in List_Factorizations_AsStrings_p_prime_cclsG[ww] then
            # i.e. if we have "<identity ...>" here
                List_Factorizations_AsStrings_p_prime_cclsG[ww] := "x1*x1^-1";
            fi ;
        od;

##################################################################

# The following loop computes the PIMs of GF(p)G.
# We proceed along the following steps:
```

```
    # a) Collect information from the decomposition matrix, such like the ordinary character
    # of the PIMs of GF(p)G which lie in the currently considered block.
    # b) Compute a composition series of the projective GF(p)G-module Q under consideration,
    # i.e. belonging to the current p-block B, save both the irreducible GF(p)G-modules and
    # their stable peakword kernels.
    # c) Fix an irreducible B-module S. Compute the dimension of the corresponding projective
    # cover P of S character- theoretically.
    # d) Choose a basis C of the stable peakword kernel of S. Repeat computing the submodule
    # spinned by a basis vector v of C until a submodule sub having correct dimension is found.
    # The module sub is automatically isomorphic to P due to the following reason: the module
    # sub is S-local and, therefore, it is an epimorphic image of P. Hence it is enough to
    # compare their respective dimensions.
    # d) In the end, we assign to the PIMs of GF(p)G their respective ordinary characters when
    # considered as kG-modules, where k is a splitting field for them. This is done by
    # considering the respective Brauer-characters.

    for j in [1.. Size(BlockIdempotsOverFp)] do
        PIMsHere:=[];
        for i in [1.. Size(gensG)] do
            Add(PIMsHere, PIMsInBlocks[j].generators[i]);
            # This gives us the generators of the matrix representations of the direct sum of
            # all PIMs of this block, but with multiplicities.
        od;
        rep:=Filename(DirectoryCurrent(), Concatenation("rep","_",String(j),"_"));

        for i in [1.. Size(gensG)] do
            CMtxBinaryFFMatOrPerm(PIMsHere[i],p,Concatenation(rep,Concatenation(".",String(i))));
        od;

        options:=["-g",String(Size(gensG)), rep];

        pro := Process(MyDir, LocationOfCHOPAsString, stdin, stdout, options);
        if not IsZero(pro) then
            Print("The last process did not return zero!");
            return(fail);
        fi;

        Read(Concatenation(rep,".cfinfo"));

        NumberOfSimplesHere:=Size(CFInfo.ConstituentNames);

        options:=["-n", "-k", rep];

        pro := Process(MyDir, LocationOfPWKONDAsString, stdin, stdout, options);
        if not IsZero(pro) then
            Print("The last process did not return zero!");
            return(fail);
        fi;

        MatricesSimplesHere:=[];
        for i in [1.. Size(CFInfo.ConstituentNames)] do
Add(MatricesSimplesHere,ReadRepFrom(Concatenation(rep,CFInfo.ConstituentNames[i]),Size(gensG),F));
        od;

        ModulesSimpleHere := List(MatricesSimplesHere, x -> GModuleByMats(x,F));

        BrauerCharsSimplesOverFasClassFunctions:=[];
        for i in [1.. NumberOfSimplesHere] do
            Print("Nun sind wir bei Brauer-Charakter Nr. "); Print(i); Print(" von insgesamt ");
            Print(NumberOfSimplesHere); Print("\n");

            Print(MTX.IsAbsolutelyIrreducible(ModulesSimpleHere[i]));

            k_max_here := GF(p^MTX.DegreeSplittingField(ModulesSimpleHere[i]));
            # this is fine in that particular case, since we only consider composition factors
            # of S tensor Fq (and not of P tensor Fq).

            COLLECTEDFACSimplesshere :=
            MTX.CollectedFactors(GModuleByMats(ModulesSimpleHere[i].generators,k_max_here));
            # it's a list of the form [[sim1, anzahl1], [sim2, anzahl2], ...]

            COMPFACSSimplesFq:=List(COLLECTEDFACSimplesshere, x -> x[1]);

            ListFrobeniusCharValsOfBrauerCharValsOfSIM := [];

            for c in [1.. Size(COMPFACSSimplesFq)] do
                FrobCharValsTemp:=[];
```

```
                for yy in [1.. Size(p__prime__cclsG)] do
                # conjugacy class number yy yields string number yy
                    STr:=List__Factorizations__AsStrings__p__prime__cclsG[yy];
                    for q in [1.. Size(COMPFACSSimplesFq[c].generators)] do
                        STr:=ReplacedString(STr, Concatenation
("x",String(q)),Concatenation("COMPFACSSimplesFq[",String(c),"].generators","[",String(q),"]"));
                    od;

                    Add(FrobCharValsTemp, TraceMat(EvalString(STr)));
                od;

                Add(ListFrobeniusCharValsOfBrauerCharValsOfSIM, FrobCharValsTemp);

            od;

            TeMp:=[];

            for v in [1.. Size(ListFrobeniusCharValsOfBrauerCharValsOfSIM)] do
                for w in [1.. Size(ListFrobeniusCharValsOfComplexIBrs)] do
                    if IsZero(ListFrobeniusCharValsOfBrauerCharValsOfSIM[v] −
                    ListFrobeniusCharValsOfComplexIBrs[w]) then
                        Add(TeMp, COLLECTEDFACSimplesshere[v][2]*ListComplexIBrs[w]);
                    fi;
                od;
            od;

            Print("TeMp ist gerade gleich:"); Print(TeMp);

            phi:= ClassFunction( ctGmodp, Sum(TeMp) );
            Add(BrauerCharsSimplesOverFasClassFunctions,phi);
        od;

        Append(MatricesSimplesOverF,MatricesSimplesHere);

        for i in [1.. Size(CFInfo.ConstituentNames)] do
            Add(AllSimplesSortedInBlocks[j],ModulesSimpleHere[i]);
        od;

        ScalProdsOfSimplesFWithPIMsOfDecMatrix:=[];
        for i in [1.. Size(BrauerCharsSimplesOverFasClassFunctions)] do
            temp:=[];
            for v in [1.. Size(BrauerCharsPIMsOverSplittingField)] do
                Add(temp, ScalarProduct(BrauerCharsPIMsOverSplittingField[v],
                BrauerCharsSimplesOverFasClassFunctions[i]));
            od;
            Add(ScalProdsOfSimplesFWithPIMsOfDecMatrix,temp);
        od;

        DimensionsOfPIMsOverF:=[];
        for i in [1.. Size(ScalProdsOfSimplesFWithPIMsOfDecMatrix)] do
            temp:=[];
            for v in [1.. Size(Irr(ctGmodp))] do
Add(temp,ScalProdsOfSimplesFWithPIMsOfDecMatrix[i][v]*Degree(BrauerCharsPIMsOverSplittingField[v]));
# this is the correct ordering, since above i also runs through the simples (over F).
            od;
            Add(DimensionsOfPIMsOverF,Sum(temp));
        od;

        s:=Size(gensG);
        MatricesPIMsOverF:=[];

        if Gcd(Order(G),p)=1 then
            MatricesPIMsOverF:=ShallowCopy(MatricesSimplesHere);
        else
            for a in [1.. Size(CFInfo.ConstituentNames)] do
                b:=1;
                repeat
                    temp:=[];

                    options:=["−g", String(s), "−s", "sub", "−n", String(b), rep, Concatenation(rep,
                    CFInfo.ConstituentNames[a],".k") ];

                    pro := Process(MyDir, LocationOfZSPAsString, stdin, stdout, options);
                    if not IsZero(pro) then
                        Print("The last process did not return zero!");
                        return(fail);
                    fi;
```

```
# Exec(Concatenation("zsp -g ",String(s)," -s sub -n ",String(b)," ",rep," ",rep,
# CFInfo.ConstituentNames[a],".k"));

# The command -s sub saves the subspace sub. Moreover, rep is the representation of the regular
# B-module, and rep5.k, say, is the representation of the stable peakword kernel no. 5, i.e.
# of the stable peakword kernel which belongs to the composition factor having dimension 5.
# If there had been different composition factors of dimension 5, then it would instead have
# been denoted by *.5a, *.5b, *.5c, et cetera.

                    Append(temp,ReadRepFrom("sub",Size(gensG),F));
                    MODU:=GModuleByMats(temp,F);
                    b:=b+1;
                until IsZero(MTX.Dimension(MODU)-DimensionsOfPIMsOverF[a]);
                Add(MatricesPIMsOverF,temp);
            od;
        fi;

        for i in [1.. Size(MatricesPIMsOverF)] do
            Add(AllPIMsSortedInBlocks[j],GModuleByMats(MatricesPIMsOverF[i],F));
        od;

        BrauerCharsRepresentationsOfThePIMsOver_F:=[];

        for i in [1.. Size(ScalProdsOfSimplesFWithPIMsOfDecMatrix)] do
            temp:=[];
                for v in [1.. Size(ScalProdsOfSimplesFWithPIMsOfDecMatrix[i])] do
Add(temp, ScalProdsOfSimplesFWithPIMsOfDecMatrix[i][v]*BrauerCharsPIMsOverSplittingField[v]);
                od;
            Add(BrauerCharsRepresentationsOfThePIMsOver_F,Sum(temp));
        od;

        OrdinaryCharsOfThePIMs_OverF:=[];

        for i in [1.. Size(BrauerCharsRepresentationsOfThePIMsOver_F)] do
            temp_ordinary_classes:=[];
            counter:=1;
            for v in [1.. Size(cclsG)] do
                if not Order(Representative(cclsG[v])) mod p = (0 mod p) then
Add(temp_ordinary_classes,BrauerCharsRepresentationsOfThePIMsOver_F[i][counter]);
                    counter:=counter + 1;
                else
                    Add(temp_ordinary_classes,0);
                fi;
            od;
            Chi_PIM:=ClassFunction(ctG, temp_ordinary_classes);
            Add(OrdinaryCharsOfThePIMs_OverF,Chi_PIM);
        od;

        temp_scalprods:=[];
        for i in [1.. Size(OrdinaryCharsOfThePIMs_OverF)] do
            Add(temp_scalprods,MatScalarProducts(ctG,[OrdinaryCharsOfThePIMs_OverF[i]],Irr(ctG)));
        od;

Append(AllBrauerCharsRepresentationsOfThePIMsOver_F,BrauerCharsRepresentationsOfThePIMsOver_F);
        Append(AllOrdinaryCharsOfThePIMs_OverF,OrdinaryCharsOfThePIMs_OverF);
        Append(All_temp_scalprods,temp_scalprods);
    od;

    NumberOfSimplesOverF:=Size(Flat(AllSimplesSortedInBlocks));

    AllPIMsOver_FAsMTXModules:=Flat(AllPIMsSortedInBlocks);

    if IdGroupsAvailable(Order(G)) then
        IdentifyingG:=IdSmallGroup(G);
    else
        IdentifyingG:=["could not identify G !!!"];
    fi;

    IrrCT:=Irr(ctG);

    Irr_As_List_Of_Lists:=[];

    for u in [1.. Size(IrrCT)] do
        v:=ShallowCopy(IrrCT[u]);
        Add(Irr_As_List_Of_Lists,v);
    od;
```

```
    # Next, we define the irreducible modules over F=GF(p), since we would like to return them
    # later in order to help calculating a splitting field for the PIMs.

    SimpleModulesOverF:=Flat(AllSimplesSortedInBlocks);
    SimpleModulesOverFforlater:=[];

    for a in [1.. Size(SimpleModulesOverF)] do
        Add(SimpleModulesOverFforlater,
        ShallowCopy(GModuleByMats(SimpleModulesOverF[a].generators,SimpleModulesOverF[a].field)));
    od;

    MyRecord:=rec();
    MyRecord.gensG := gensG;
    MyRecord.OrderG := Order(G);
    MyRecord.G := G;
    MyRecord.IdentifyingG := IdentifyingG;
    MyRecord.Field := GF(p);
    MyRecord.Characteristic := p;
    MyRecord.AllPIMsOver__FAsMTXModules := AllPIMsOver__FAsMTXModules;
    MyRecord.OrdinaryCharsOfThePIMs__OverF := AllOrdinaryCharsOfThePIMs__OverF;
    MyRecord.DecompositionMatrix := dec;
    MyRecord.ConjugacyClasses := cclsG;
    MyRecord.pPrimeClasses := p__prime__cclsG;
    MyRecord.temp__scalprods:=All__temp__scalprods;
    MyRecord.Irr__As__List__Of__Lists:=Irr__As__List__Of__Lists;
    MyRecord.SimpleModulesOverF:=SimpleModulesOverF;

# Now, we compute the PIMs of kG, where k is a splitting field for the irreducible GF(p)G-modules.

    ListAllPIMsOverSplittingFields:=[];

    LoadPackage("qpa", "=1.34");

    DegreesSplittingFields :=[];

    for i in [1.. Size(MyRecord.SimpleModulesOverF)] do
        Print(MTX.IsAbsolutelyIrreducible(MyRecord.SimpleModulesOverF[i]));
        Add(DegreesSplittingFields,MTX.DegreeSplittingField(MyRecord.SimpleModulesOverF[i]));
    od;

    AllBasesForGaloisConjugates:=[];
    AllBlockDiagonalGens:=[];

    k__max:=GF(p^Lcm(DegreesSplittingFields)); # Lcm stands for least common multiple

    for i in [1.. Size(DegreesSplittingFields)] do
        Print("i ist jetzt gleich: ");Print(i);Print(" von ");Print(Size(DegreesSplittingFields));
        Print("\n");
        if DegreesSplittingFields[i] = 1 then
            Add(ListAllPIMsOverSplittingFields,AllPIMsOver__FAsMTXModules[i]);
        else
            BasisGalConjugates:=[];
            k__new:=GF(p^DegreesSplittingFields[i]);
            P__new:=GModuleByMats(AllPIMsOver__FAsMTXModules[i].generators,k__new);
            HomPP:=MTX.BasisModuleHomomorphisms(P__new,P__new);
            A:=FullMatrixAlgebra(k__new, Size(P__new.generators[1]));
            B:=Subalgebra(A,HomPP);
            # Next, we consider HommPP as matrix algebra and compute primitive idempotents for
            # its decomposition.
            pids:=IdempotentsForDecomposition(B);

            SizePids:=Size(pids);

            for j in pids do
                j__new:=ShallowCopy(j);
                bas := MTX.SpinnedBasis(j__new,P__new.generators,P__new.field);
                bas__new:=ShallowCopy(bas);
                Append(BasisGalConjugates,bas__new);
                sub := MTX.InducedActionSubmodule(P__new,bas);
                Add(ListAllPIMsOverSplittingFields,sub);
            od;

            Add(AllBasesForGaloisConjugates,[i,BasisGalConjugates,k__new,SizePids]);
            # This list collects information about which PIMs do not remain indecomposable after
            # tensoring with a splitting field and saves the matrix for the base change.
```

```
        ConjugatedGeneratorMatrices:=[];

        for a in [1.. Size(P_new.generators)] do
            OldGens:=P_new.generators[a];
            E_M_B:=BasisGalConjugates;
            E_M_B_i:=BasisGalConjugates^-1;
            BlockDiagonalGens:=E_M_B*OldGens*E_M_B_i;
            Add(ConjugatedGeneratorMatrices,BlockDiagonalGens);
        od;
        Add(AllBlockDiagonalGens,[i,ConjugatedGeneratorMatrices]);
    fi;
od;


#################################################################
# Here is the continuation for the trick with the dictionary.
# In order to determine the Brauer characters of the PIMs, we compute the composition factors,
# then the FrobeniusCharacterValues of the occurring simple modules, then use the dictionary
# such that we obtain the Brauer characters of the simple composition factors without computing
# Brauer character values at this stage of the program... and last, but not least, we just have
# to sum the Brauer characters of the composition factors in order to obtain their
# desired Brauer character value of the projective indecomposable module(s) in question
# This uses the Corollary about traces of kG-modules from one of the books of Curtis and Reiner,
# see Chapter 5 of the thesis.

        BrauerCharsPIMsOverFqAsClassFunctions:=[];
        for i in [1.. Size(ListAllPIMsOverSplittingFields)] do
            Print("Nun sind wir bei Brauer-Charakter Nr. "); Print(i);
            Print(" von den PIMs over Fq von insgesamt ");
            Print(Size(ListAllPIMsOverSplittingFields)); Print("\n");

            ModuleForCollectionOfFacs :=
            GModuleByMats(ListAllPIMsOverSplittingFields[i].generators,k_max);

            COLLECTEDFACSPIMsFq := MTX.CollectedFactors(ModuleForCollectionOfFacs);
            # it is a list of this form: [[sim1, anzahl1], [sim2, anzahl2], ...]

            COMPFACSPIMsFq:=List(COLLECTEDFACSPIMsFq, x -> x[1]);

            ListFrobeniusCharValsOfBrauerCharValsOfPIM := [];

            for c in [1.. Size(COMPFACSPIMsFq)] do
                FrobCharValsTemp:=[];
                for yy in [1.. Size(p_prime_cclsG)] do
                # ccls number yy yields string number yy (i.e.: the yy-th string)
                    STr:=List_Factorizations_AsStrings_p_prime_cclsG[yy];
                    for q in [1.. Size(COMPFACSPIMsFq[c].generators)] do
                        STr:=ReplacedString(STr, Concatenation
("x",String(q)),Concatenation("COMPFACSPIMsFq[",String(c),"].generators","[",String(q),"]"));
                    od;
                    Add(FrobCharValsTemp, TraceMat(EvalString(STr)));
                od;
                Add(ListFrobeniusCharValsOfBrauerCharValsOfPIM, FrobCharValsTemp);
            od;

            TeMp:=[];

            for v in [1.. Size(ListFrobeniusCharValsOfBrauerCharValsOfPIM)] do
                for w in [1.. Size(ListFrobeniusCharValsOfComplexIBrs)] do
                    if IsZero(ListFrobeniusCharValsOfBrauerCharValsOfPIM[v] -
                    ListFrobeniusCharValsOfComplexIBrs[w]) then
                        Add(TeMp, COLLECTEDFACSPIMsFq[v][2]*ListComplexIBrs[w]);
                    fi;
                od;
            od;

               Print("TeMp ist gerade gleich:"); Print(TeMp);

            phi:= ClassFunction( ctGmodp, Sum(TeMp) );
            Add(BrauerCharsPIMsOverFqAsClassFunctions,phi);
        od;

        MyRecord.ListAllPIMsOverSplittingFields:=ListAllPIMsOverSplittingFields;

        MyRecord.AllBasesForGaloisConjugates:=AllBasesForGaloisConjugates;

        MyRecord.AllBlockDiagonalGens:=AllBlockDiagonalGens;
```

```
# Here, we compute the ordinary characters of the PIMs of kG where k is a splitting field .

            OrdinaryCharsOfThePIMs_OverFq:=[];

            for i in [1.. Size(BrauerCharsPIMsOverFqAsClassFunctions)] do
                temp_ordinary_classes:=[];
                counter:=1;
                for v in [1.. Size(cclsG)] do
                    if not Order(Representative(cclsG[v])) mod p = (0 mod p) then
Add(temp_ordinary_classes,BrauerCharsPIMsOverFqAsClassFunctions[i][counter]);
                        counter:=counter + 1;
                    else
                        Add(temp_ordinary_classes,0);
                    fi ;
                od;
                Chi_PIM:=ClassFunction(ctG, temp_ordinary_classes);
                Add(OrdinaryCharsOfThePIMs_OverFq,Chi_PIM);
            od;

            MyRecord.OrdinaryCharsOfThePIMs_OverFq:=OrdinaryCharsOfThePIMs_OverFq;

            # Here, we compute the scalar products of the latter ordinary characters with each
            # ordinary irreducible character:

            ScalprodsPIMsOverFq:=[];
            for i in [1.. Size(OrdinaryCharsOfThePIMs_OverFq)] do
Add(ScalprodsPIMsOverFq,MatScalarProducts(ctG,[OrdinaryCharsOfThePIMs_OverFq[i]],Irr(ctG)));
            od;

            ListForScalProdsTest := Flat(ScalprodsPIMsOverFq);

            for i in ListForScalProdsTest do
                if not IsInt(i) then
                    Print("DAS MIT SCALPRODS hat net geklappt...sind net alles Integers!!!");
                    return(fail);
                fi ;
            od;

            MyRecord.ScalprodsPIMsOverFq:=ScalprodsPIMsOverFq;

            MyRecord.SimpleModulesOverF:=0;

            MyRecord.SimpleModulesOverF:=SimpleModulesOverFforlater;

            return MyRecord;
end;

# Example: G:=AlternatingGroup(6); p:=2; U:=PIMsFqG(G,p);
```

7.2 An algorithm for the computation of trivial source character tables using the Shared C MeatAxe and GAP

```
WriteOrGetPIMsDataOverFqViaDatabase:=function(G,p)

    local str, str0, file, GroupsSameOrder, psi, i, psi_test, dataPIMs, H, U;

    if Order(G) < 101 then
        str:="PIMsDatabaseOverFq1to100.txt";
    elif Order(G) < 201 then
        str:="PIMsDatabaseOverFq101to200.txt";
    elif Order(G) < 301 then
        str:="PIMsDatabaseOverFq201to300.txt";
    elif Order(G) < 401 then
        str:="PIMsDatabaseOverFq301to400.txt";
    elif Order(G) < 501 then
        str:="PIMsDatabaseOverFq401to500.txt";
    elif Order(G) < 601 then
        str:="PIMsDatabaseOverFq501to600.txt";
    elif Order(G) < 701 then
        str:="PIMsDatabaseOverFq601to700.txt";
    elif Order(G) < 801 then
        str:="PIMsDatabaseOverFq701to800.txt";
    elif Order(G) < 901 then
        str:="PIMsDatabaseOverFq801to900.txt";
    elif Order(G) < 1001 then
        str:="PIMsDatabaseOverFq901to1000.txt";
    elif Order(G) < 1101 then
        str:="PIMsDatabaseOverFq1001to1100.txt";
    elif Order(G) < 1201 then
        str:="PIMsDatabaseOverFq1101to1200.txt";
    elif Order(G) < 1301 then
        str:="PIMsDatabaseOverFq1201to1300.txt";
    elif Order(G) < 1401 then
        str:="PIMsDatabaseOverFq1301to1400.txt";
    elif Order(G) < 1501 then
        str:="PIMsDatabaseOverFq1401to1500.txt";
    else
        str:="PIMsDatabaseOverFqGroupOrdersLargerThan1500.txt";
    fi;

    Read("/home/bernhard/Schreibtisch/GAP_Database/GAP_Database_Over_Fq/PIMsOverFq.txt");

    str0:="/home/bernhard/Schreibtisch/GAP_Database/GAP_Database_Over_Fq/";

    file:=Concatenation(str0,str);

    Read(file);

    psi:=0;

    if SmallGroupsAvailable( Order(G) ) then
        GroupsSameOrder:=Filtered(databasePIMsFq, x -> x.IdentifyingG=IdSmallGroup(G));
    else
        GroupsSameOrder:=Filtered(databasePIMsFq, x -> x.OrderG=Order(G));
    fi;

    for i in [1..Size(GroupsSameOrder)] do
        if p = GroupsSameOrder[i].Characteristic then
            psi_test:=IsomorphismGroups(G, GroupsSameOrder[i].G);
            if psi_test <> fail then
                psi:=ShallowCopy(psi_test);
                dataPIMs:=GroupsSameOrder[i];
            fi;
        fi;
    od;

    if psi <> 0 then
        return([psi,dataPIMs]);
    else
        psi:=IsomorphismPermGroup(G);
        H:=Image(psi);
        U:=PIMsFqG(H,p);
        Add(databasePIMsFq,U);
        PrintTo(file, "databasePIMsFq:=");
```

```
        AppendTo(file, databasePIMsFq);
        AppendTo(file,";");
        return([psi,U]);
    fi;
end;
```

```
LoadPackage("ctbllib");

Read("/home/bernhard/Schreibtisch/GAP__Database/GAP__Database_Over_Fq/WriteOrGetPIMsOverFqViaDatabase.txt");
Read("/home/bernhard/Schreibtisch/StripOffOneCopyOfNFromMIfPossible.txt");

# The function EquivalentLibraryCharacterTableWithGroup is written by Thomas Breuer

#############################################################################
##
#F  EquivalentLibraryCharacterTableWithGroup( <G> )
##
EquivalentLibraryCharacterTableWithGroup:= function( G )
    local init, Gcopy, name, attr, Gtbl, tbl, trans, compat, ccl, new, i;

    # If the group stores already an ordinary character table
    # then we cannot set the attributes consistently.
    if HasOrdinaryCharacterTable( G ) then
      Error( "<G> has already a character table" );
    fi;

    # Test cheap attributes first, and exclude duplicates.
    init:= AllCharacterTableNames( Size, Size( G ),
                 NrConjugacyClasses, NrConjugacyClasses( G ),
                 IsDuplicateTable, false );
    if Length( init ) = 0 then
      # No expensive tests are needed.
      # In particular, do not compute a character table.
      return fail;
    fi;

    # Create a copy of the group, in order to compute its character table
    # without storing it.
    # (Note that calling 'AttributeValueNotSet' for 'OrdinaryCharacterTable'
    # does not help, since 'Irr' etc. would appear silently.)
    # Store the known attributes of 'G' in the copy,
    # in particular 'Gcopy' and 'G' have the same ordering of conj. classes.
    Gcopy:= GroupWithGenerators( GeneratorsOfGroup( G ) );
    for name in KnownAttributesOfObject( G ) do
      attr:= ValueGlobal( name );
      Setter( attr )( Gcopy, attr( G ) );
    od;

    # Compute the character table of the copy.
    Gtbl:= OrdinaryCharacterTable( Gcopy );
    for name in init do
      tbl:= CharacterTable( name );
      trans:= TransformingPermutationsCharacterTables( tbl, Gtbl );
      if trans <> fail then
        # Take this library table:
        # - Permute the classes stored in the group.
        compat:= ListPerm( trans.columns, NrConjugacyClasses( tbl ) );
        ccl:= ConjugacyClasses( G ){ compat };

        # - Copy the contents of the library table.
        new:= ConvertToLibraryCharacterTableNC(
                  rec( UnderlyingCharacteristic := 0 ) );

        # - Set the supported attribute values except 'Irr'.
        for i in [ 3, 6 .. Length( SupportedCharacterTableInfo ) ] do
          if Tester( SupportedCharacterTableInfo[ i-2 ] )( tbl )
             and SupportedCharacterTableInfo[ i-1 ] <> "Irr" then
            Setter( SupportedCharacterTableInfo[ i-2 ] )( new,
                SupportedCharacterTableInfo[ i-2 ]( tbl ) );
          fi;
        od;

        # - Set the irreducibles.
        SetIrr( new, List( Irr( tbl ),
            chi -> Character( new, ValuesOfClassFunction( chi ) ) ) );

        # - Set the group in the table.
        SetUnderlyingGroup( new, G );
        SetConjugacyClasses( new, ccl );
        SetIdentificationOfConjugacyClasses( new, compat );

        # - Set the table in the group.
        SetOrdinaryCharacterTable( G, new );
```

```
            return new;
        fi ;
    od;

    # No library table fits .
    # However, we set the computed character table , since we know it.
    SetOrdinaryCharacterTable( G, Gtbl );
    return fail;
    end;

FromStringToMatrix:=function(G,gensG,p,fAsString)

    local DirOfChop, M, i, RES, ERGEBNIS, MAT, StringNow, z, PositionsOpenParentheses,
    PositionsCloseParentheses, KLAMMER, r, SSS, STR, SPLITnow, KlammerAuf, KlammerZu, u,
    StringNow1, StringNow2, StringNow3, StringYNow, StringToChange, SPLIT, INP, ergebnis_to_return,
    stdin, stdout, MyDir, LocationOfZPRAsString, LocationOfZPOAsString, LocationOfZMUAsString, path,
    rm, options, pro, dir, files , f;

    LoadPackage("io");
    # DirOfChop:=Directory("/home/bernhard/Schreibtisch/shared_meataxe-1.0/src/");

    ChangeDirectoryCurrent("/home/bernhard");

    MyDir:=Directory("/home/bernhard");
    stdin := InputTextUser();;
    stdout := OutputTextUser();;
    LocationOfZPRAsString := "/home/bernhard/Schreibtisch/shared_meataxe-1.0/bin/zpr";
    LocationOfZPOAsString := "/home/bernhard/Schreibtisch/shared_meataxe-1.0/bin/zpo";
    LocationOfZMUAsString := "/home/bernhard/Schreibtisch/shared_meataxe-1.0/bin/zmu";
    path := DirectoriesSystemPrograms();
    rm := Filename(path,"rm");

    RES:=Filename(MyDir, "RES");

    ERGEBNIS:=Filename(MyDir, "ERGEBNIS");
    MAT:=Filename(DirectoryCurrent(), "MAT");
    KLAMMER:=Filename(DirectoryCurrent(), "KLAMMER");

    dir := Directory("/home/bernhard");

    files := Filtered(DirectoryContents(MyDir), f -> Length(f)>3 and f[1] = 'R' and f[2] = 'E' and f[3] = 'S');
    for f in files do
        if f[4] <> '.' and not ForAll(f{[4..Length(f)]}, IsDigitChar) then
            continue;
        fi ;
        f := Filename(MyDir, f);
        RemoveFile(f);
    od;

    files := Filtered(DirectoryContents(MyDir), f -> Length(f)>8 and f[1] = 'E' and f[2] = 'R'
    and f[3] = 'G' and f[4] = 'E' and f[5] = 'B' and f[6] = 'N' and f[7] = 'I' and f[8] = 'S');
    for f in files do
        if f[9] <> '.' and not ForAll(f{[9..Length(f)]}, IsDigitChar) then
            continue;
        fi ;
        f := Filename(MyDir, f);
        RemoveFile(f);
    od;

    files := Filtered(DirectoryContents(MyDir), f -> Length(f)>3 and f[1] = 'M' and f[2] = 'A' and f[3] = 'T');
    for f in files do
        if f[4] <> '.' and not ForAll(f{[4..Length(f)]}, IsDigitChar) then
            continue;
        fi ;
        f := Filename(MyDir, f);
        RemoveFile(f);
    od;

    files := Filtered(DirectoryContents(MyDir), f -> Length(f)>7 and f[1] = 'K' and f[2] = 'L'
    and f[3] = 'A' and f[4] = 'M' and f[5] = 'M' and f[6] = 'E' and f[7] = 'R');
    for f in files do
        if f[8] <> '.' and not ForAll(f{[8..Length(f)]}, IsDigitChar) then
            continue;
        fi ;
        f := Filename(MyDir, f);
        RemoveFile(f);
```

```
od;

StringNow:=ShallowCopy(fAsString);
z:=0;

# The first step is to eliminate all occurring parentheses in the string fAsString.
# We search for the innermost pair of parentheses and evaluate the expression enclosed by them.
# Iteratively, after finitely many steps we are done.

while '(' in StringNow do
    PositionsOpenParentheses := Positions(StringNow,'(');
    PositionsCloseParentheses := Positions(StringNow,')');
    z:=z+1;
    KlammerZu:=PositionsCloseParentheses[1]; # find first closing parenthesis
    u:=PositionsCloseParentheses[1];
    while (u in PositionsOpenParentheses)=false do
        u:=u-1;
    od;
    KlammerAuf:=u;
    StringToChange := StringNow{ [KlammerAuf+1..KlammerZu-1] } ;
    SPLIT:=SplitString(StringToChange, "*");

    for i in [1.. Size(SPLIT)] do
        STR:=SPLIT[i];
        SPLITnow:=SplitString(STR,"^");
        if Size(SPLITnow)=2 then # In this case the expression contains an exponent.
            SSS:=ReplacedString(SPLITnow[1],"x","M");
            SSS:=ReplacedString(SSS,"y","KLAMMER.");

            options:=[SSS, String(SPLITnow[2]), Concatenation("MAT.",String(i))];

            pro := Process(MyDir, LocationOfZPOAsString, stdin, stdout, options);
            if not IsZero(pro) then
                Print("The last process did not return zero!");
                return(fail);
            fi;
        else
            SSS:=ReplacedString(SPLITnow[1],"x","M");
            SSS:=ReplacedString(SSS,"y","KLAMMER.");

            options:=[SSS, String(1), Concatenation("MAT.",String(i))];

            pro := Process(MyDir, LocationOfZPOAsString, stdin, stdout, options);
            if not IsZero(pro) then
                Print("The last process did not return zero!");
                return(fail);
            fi;
        fi;
    od;

    options:=["MAT.1","MAT.2","RES.2"];

    pro := Process(MyDir, LocationOfZMUAsString, stdin, stdout, options);
    if not IsZero(pro) then
        Print("The last process did not return zero!");
        return(fail);
    fi;

    for i in [2.. Size(SPLIT)-1] do
options:=[Concatenation("RES.",String(i)),Concatenation("MAT.",String(i+1)),Concatenation("RES.",String(i+1))];
        pro := Process(MyDir, LocationOfZMUAsString, stdin, stdout, options);
        if not IsZero(pro) then
            Print("The last process did not return zero!");
            return(fail);
        fi;
    od;

    r:=Maximum(Size(SPLIT),2);

    options:=[Concatenation("RES.",String(r)), "1", Concatenation("KLAMMER.",String(z))];

    pro := Process(MyDir, LocationOfZPOAsString, stdin, stdout, options);
    if not IsZero(pro) then
        Print("The last process did not return zero!");
        return(fail);
    fi;

```

```
        if KlammerAuf > 1 then
            StringNow1 := StringNow{ [1..KlammerAuf−1] } ;
        else
            StringNow1 := "";
        fi ;

        StringNow2 := Concatenation("KLAMMER.",String(z));

        if KlammerZu < Size(StringNow) then
            StringNow3 := StringNow{ [KlammerZu+1..Size(StringNow)] } ;
        else
            StringNow3 := "";
        fi ;
        StringYNow := Concatenation("y",String(z));
        StringNow:=Concatenation(StringNow1,StringYNow,StringNow3);
    od;

    # We have finally eliminated all parentheses.

    StringToChange := StringNow;
    # This string does not contain any parentheses anymore, but it can still contain
    # exponents or multiplication symbols.

    SPLIT:=SplitString(StringToChange, "*");

    for i in [1.. Size(SPLIT)] do
        STR:=SPLIT[i];
        SPLITnow:=SplitString(STR,"^");
        if Size(SPLITnow)=2 then # hence, there are exponents involved
            SSS:=ReplacedString(SPLITnow[1],"x","M");
            SSS:=ReplacedString(SSS,"y","KLAMMER.");

            options:=[SSS, String(SPLITnow[2]), Concatenation("MAT.",String(i))];

            pro := Process(MyDir, LocationOfZPOAsString, stdin, stdout, options);
            if not IsZero(pro) then
                Print("The last process did not return zero!");
                return(fail);
            fi ;
        else
            SSS:=ReplacedString(SPLITnow[1],"x","M");
            SSS:=ReplacedString(SSS,"y","KLAMMER.");

            options:=[SSS, String(1), Concatenation("MAT.",String(i))];

            pro := Process(MyDir, LocationOfZPOAsString, stdin, stdout, options);
            if not IsZero(pro) then
                Print("The last process did not return zero!");
                return(fail);
            fi ;
        fi ;
    od;

    if Size(SPLIT) = 1 then

        options:=[Concatenation(MAT,".",String(1)), Concatenation(ERGEBNIS,".text")];

        pro := Process(MyDir, LocationOfZPRAsString, stdin, stdout, options);
        if not IsZero(pro) then
            Print("The last process did not return zero!");
            return(fail);
        fi ;
        INP:=
InputTextString(Concatenation("ergebnis"," := ScanMeatAxeFile(","\"",Concatenation(ERGEBNIS,".text"),"\"",");" ));;
        Read(INP);
    else

        options:=["MAT.1","MAT.2","RES.2"];

        pro := Process(MyDir, LocationOfZMUAsString, stdin, stdout, options);
        if not IsZero(pro) then
            Print("The last process did not return zero!");
            return(fail);
        fi ;

        for i in [2.. Size(SPLIT)−1] do
```

```
options:=[Concatenation("RES.",String(i)), Concatenation("MAT.",String(i+1)), Concatenation("RES.",String(i+1))];

            pro := Process(MyDir, LocationOfZMUAsString, stdin, stdout, options);
            if not IsZero(pro) then
                Print("The last process did not return zero!");
                return(fail);
            fi;
        od;

        r:=Maximum(Size(SPLIT),2);

        options:=[Concatenation("RES.",String(r)), "1",  Concatenation("ERGEBNIS.",String(1))];

        pro := Process(MyDir, LocationOfZPOAsString, stdin, stdout, options);
        if not IsZero(pro) then
            Print("The last process did not return zero!");
            return(fail);
        fi;

        options:=[Concatenation(ERGEBNIS, ".", String(1)), Concatenation(ERGEBNIS, ".text")];

        pro := Process(MyDir, LocationOfZPRAsString, stdin, stdout, options);
        if not IsZero(pro) then
            Print("The last process did not return zero!");
            return(fail);
        fi;

        INP :=
InputTextString( Concatenation("ergebnis"," := ScanMeatAxeFile(","\"",Concatenation(ERGEBNIS,".text"),"\"",");" ));;
        Read(INP);
    fi;

    ergebnis_to_return := ShallowCopy(ergebnis);

    return ergebnis_to_return;
end;

#################################################################
##
##
#F MyBaseChangeMat(s,a,m,p)
##

# s := size of a small block within alpha
# a = anz := total number of blocks of alpha
# m := number of alphas in Alpha_BIG

# => number of sxs-blocks = m*anz

MyBaseChangeMat := function(s,a,m,p)

    local MyOneMatrix, LISTE, i, j, BKM, BLMAsMat, BasChMat;

    MyOneMatrix := IdentityMat(s,GF(p));
    LISTE := [];

    for i in [1..a] do
        for j in [1..m] do
            Add(LISTE,[(i-1)*m+j,(j-1)*a+1+(i-1),MyOneMatrix]);
        od;
    od;

    # begin with (1,1), then consider (2,a+1), then (3,2a+1) ... till (m,(m-1)*a+1) ... then
    # (m+1,2), then (m+2, a+2), then (m+3, 2a+2) ... till (2m, (m-1)*a + 2) ... then
    # (2m+1, 3) , then (2m+2, a+3), then (2m+3, 2a+3) ... till (3m, (m-1)*a + 3) ... then
    #                   ...
    # then finally:
    # ((a-1)*m+1 , a) and ((a-1)*m+2, 2*a) and ((a-1)*m+3, 3*a) ... till (a*m , a*m).
    # Recall that blocks in the block matrix and ot only single rows/columns are involved.

    BKM:=BlockMatrix(LISTE, m*a, m*a);

    BLMAsMat := MatrixByBlockMatrix(BKM);

    return BLMAsMat; # Mind that this is already the transposed matrix!
end;
```

```
TSModulesAndLiftsOverFq:=function(G,p)

    local x, exp, facts, pprimefacts, m, f, k, n, q, Syl, ccsSyl, i, j, temp, c, pSubgroupsUpToConjugacy,
    N, P, hom, FAC, V, TheRecord, ccF, CompleteList_V_M_Chi, List_N_i_hom_i_FAC_i, R, PSI,
    PSI_TO_THE_MINUS_ONE, gensFAC, OldMatScalProds, OldCharacterTable, PermutationsOldAndNewCharTable,
    PermRows, temp_sizes, rho_N_bars_for_all_N, rho_N_bars, rho_N_bar, rho_Ns, rho_N, ctG, gensG, HOMs,
    gensPSIofFAC, PSI_AsGroupHomom, PSI_TO_THE_MINUS_ONE_AsGroupHomom, OldIrrCT, NewIrrCT,
    ChiProjsNewTable, a, Chi, t, PIMsG, w, b, MODUgenerators, MODU, v, ctN, ctFAC, gensOfN,
    Inflated_Characters_N, Inflated_Modules_N, InducedCharacters, InducedModules,
    CompleteList_V_M_Chi_copy, Chi_G, Chi_N, M_N, M_G, dirM_G, s, r, IndecSummands, temp_list_j,
    u, d, AllTSModulesAsHoms, a2, ScalProdsTSModules, cclsG, MyTSModulesAndLiftsRecord, IdentifyingG,
    Subgroups_Pi, UUU, OldConjugacyClasses, Alpha_i_s_G, OldMatScalProdsOverFq,
    List_Preimages_OldConjugacyClasses, TranspMatOldIrr, NewOldIrrCT, PermColumns, ChiProjsNewTableOverFq,
    G_Auxiliary, ListGensAsStrings, ps, fac, facAsString, TSModulesOverFpWithCorrectMatrices,
    PIMsOverFp_In_Database, MatricesModuleNow, CompleteList_V_M_Chi_Over_Fq, PIMsOverFqCorrectModules,
    counter, ModuleOverFpNowOverFq, AlphaNow, AlphaNow_i, BlockDiagonalGens, OldGens,
    ListOfListsNewGeneratorMatrices, ConjugatedMatricesOverFq, matt, SmallerModulesFromTheDiagonal,
    IRR_CTG, AuxiliaryAllToChopAndMultMatrices, AllBasesGalConjugatesForGreen,
    CounterNumberOfDirSummandsGreenCorrForAllPGroups, cclsFAC, Inflated_Characters_N_OverFq,
    N_bar_Auxiliary, NaturalHomOfGenN, PsiOfNatHomOfGenN, Alpha_i_s_N, TSModulesNOverFpWithCorrectMatrices,
    MatricesForConjugationStillToChopAndMultiply, DIFFERENZ, MAX_Abspalt, StripErgebnis, M_Ind,
    ListMatricesForConjugationWithCorrectDimensions, diff, IdentityMatrixToAddOnTheTopLeftCorner, MatNew,
    MATR, ModuleMIndInBlockDiagonalForm, TSModulesNOverFqCorrectModules, CounterNumberOfDirSummandsGreenCorr,
    AllSubmodulesOfGREENOverFq, BasisGalConjugatesForGreen, LOverFq, y, z, Alpha_BIG, ss, aa, mm, BasChMat,
    ConjugationMatrixForInducedModuleTransitionFromFpToFq, uu, RelevantVecs_FqBackToFp_AsListOfLists, ff,
    k_now, ModuleMIndInBlockDiagonalFormOverFq, genMatricesGreenCorresp, g, DimGreenCorresp,
    GreenCorrespondAtLevelG, EinheitsMat, RelevantVecs_Green, V1, BAS, BS, V2, V3, S, h, bas, bb,
    ScalprodsPIMsOverFq, GensOfGr, H, IrrCT, Irr_As_List_Of_Lists, L, ListeCopiesOfAlphaNow,
    RelevantVecs_FqBackToFp, BSnew, l, submod;

    LoadPackage("io");

    ChangeDirectoryCurrent("/home/bernhard");

    MyDir:=Directory("/home/bernhard");
    stdin := InputTextUser();;
    stdout := OutputTextUser();;
    LocationOfZPRAsString := "/home/bernhard/Schreibtisch/shared_meataxe-1.0/bin/zpr";
    LocationOfZPOAsString := "/home/bernhard/Schreibtisch/shared_meataxe-1.0/bin/zpo";
    LocationOfZMUAsString := "/home/bernhard/Schreibtisch/shared_meataxe-1.0/bin/zmu";
    path := DirectoriesSystemPrograms();
    rm := Filename(path,"rm");

    LoadPackage("PERMUT");

    k:=GF(p);

    Syl:=SylowSubgroup(G,p);
    ccsSyl:=ConjugacyClassesSubgroups(Syl);
    temp:=[];
    for i in [1.. Size(ccsSyl)] do
        Append(temp,[ccsSyl[i][1]]) ;
    od;
    ccsSyl:=ShallowCopy(temp);
    pSubgroupsUpToConjugacy:=[];
    # pSubgroupsUpToConjugacy later gives us the p-subgroups of the Sylow subgr. of G up to conjug. in G

    for j in [1.. Size(ccsSyl)] do
        c:=ConjugateSubgroups(G,ccsSyl[j]);
        if Size( Intersection (AsSet(c),AsSet(pSubgroupsUpToConjugacy)))=0 then
            Append(pSubgroupsUpToConjugacy,[ccsSyl[j]]);
    # if no conjugate subgr. of ccsSyl[j] is in pSubgroupsUpToConjugacy then
    # add ccsSyl[j] to pSubgroupsUpToConjugacy
        fi ;
    od;

    temp_sizes:=[];
    for i in [1.. Size(pSubgroupsUpToConjugacy)] do
        Add(temp_sizes,Size(pSubgroupsUpToConjugacy[i]));
    od;
    SortParallel (temp_sizes,pSubgroupsUpToConjugacy);

    CompleteList_V_M_Chi:=[];
    List_N_i_hom_i_FAC_i:=[];
    rho_N_bars_for_all_N:=[];

    if HasOrdinaryCharacterTable(G) then
```

```
        ctG:=CharacterTable(G);
    else
        UUU:=EquivalentLibraryCharacterTableWithGroup(G);
        ctG:=CharacterTable(G);
    fi;

    Display(ctG);

    gensG:=GeneratorsOfGroup(G);

    ########################################

    V:=WriteOrGetPIMsDataOverFqViaDatabase(G,p);
    PSI:=V[1]; # PSI is the map IsomorphismPermGroup from the program WriteOrGetPIMs
    TheRecord:=V[2];
    gensPSIofFAC:=TheRecord.gensG; # i.e. the generators of the image of PSI
    PSI_TO_THE_MINUS_ONE:=InverseGeneralMapping(PSI);
    gensFAC:=List(gensPSIofFAC, x -> ImagesRepresentative(PSI_TO_THE_MINUS_ONE,x));
    PSI_AsGroupHomom:=GroupHomomorphismByImages(G,Image(PSI),gensFAC,gensPSIofFAC);
    PSI_TO_THE_MINUS_ONE_AsGroupHomom:=GroupHomomorphismByImages(Image(PSI),G,gensPSIofFAC,gensFAC);
    OldMatScalProds:=TheRecord.temp_scalprods;
    OldMatScalProdsOverFq:=TheRecord.ScalprodsPIMsOverFq;
    OldIrrCT:=TheRecord.Irr_As_List_Of_Lists;
    OldConjugacyClasses := List(TheRecord.ConjugacyClasses, xxx-> Representative(xxx));

    List_Preimages_OldConjugacyClasses:=
    List(OldConjugacyClasses, yyy -> ImagesRepresentative(PSI_TO_THE_MINUS_ONE,yyy));

    cclsG:=ConjugacyClasses(ctG);

    TranspMatOldIrr:=[];

    for v in [1.. Size(cclsG)] do
        for w in [1.. Size(List_Preimages_OldConjugacyClasses)] do
            if IsConjugate(G,Representative(cclsG[v]),List_Preimages_OldConjugacyClasses[w]) then
                Add(TranspMatOldIrr,TransposedMat(OldIrrCT)[w]);
            fi;
        od;
    od;

    NewIrrCT:=Irr(ctG);

    NewOldIrrCT:=TransposedMat(TranspMatOldIrr);
    # hence, the labelling of the columns corresponds to the representatives of the classes of ctG

    PermutationsOldAndNewCharTable:=TransformingPermutations(NewOldIrrCT,NewIrrCT);
    PermRows:=PermutationsOldAndNewCharTable.rows;

    PermColumns:=PermutationsOldAndNewCharTable.columns;
    if not IsZero(Order(PermColumns)-1) then
        Print("Columns war nicht die leere Permutation !!!");
        return fail;
    else
        Print("Das mit PermColumns hat nun beim ersten Mal bei G geklappt!!!   ;-) ");
    fi;

    ChiProjsNewTable:=[];
    for a in [1.. Size(OldMatScalProds)] do
        Chi:=0;
        for j in [1.. Size(OldMatScalProds[a])] do
            Chi := Chi + OldMatScalProds[a][j][1]*Irr(ctG)[OnPoints(j,PermRows)];
        od;
        Chi := ClassFunction(ctG,Chi);
        Add(ChiProjsNewTable,Chi);
    od;

     # do the same for the ordinary characters of the PIMs over Fq:

    ChiProjsNewTableOverFq:=[];
    for a in [1.. Size(OldMatScalProdsOverFq)] do
        Chi:=0;
        for j in [1.. Size(OldMatScalProdsOverFq[a])] do
            Chi := Chi + OldMatScalProdsOverFq[a][j][1]*Irr(ctG)[OnPoints(j,PermRows)];
        od;
        Chi := ClassFunction(ctG,Chi);
        Add(ChiProjsNewTableOverFq,Chi);
    od;
```

```
rho_N_bars:=[];

G_Auxiliary:=GroupByGenerators(gensPSIofFAC); # we still consider the group G; therefore, here, FAC=G/<1>.
ListGensAsStrings:=[];
for a in [1.. Size(gensG)] do
    ps:=Image(PSI_AsGroupHomom,gensG[a]);
    fac:=Factorization(G_Auxiliary,ps);
    facAsString:=String(fac);

    if 'i' in facAsString then # i.e. if we have "<identity ...>" here
        facAsString := "x1*x1^-1";
    fi;

    Add(ListGensAsStrings,facAsString);
od;

# Next, we collect the underlying representations of all the PIMs over Fp from the database,
# and, if available, also alpha.

PIMsOverFp_In_Database:=TheRecord.AllPIMsOver_FAsMTXModules;
# This is a list of records.

Alpha_i_s_G := TheRecord.AllBasesForGaloisConjugates;

TSModulesOverFpWithCorrectMatrices:=[];

for b in [1.. Size(PIMsOverFp_In_Database)] do
    M:=Filename(DirectoryCurrent(), "M");

    files := Filtered(DirectoryContents(MyDir), f -> Length(f)>1 and f[1] = 'M');
    for f in files do
        if f[2] <> '.' and not ForAll(f{[2..Length(f)]}, IsDigitChar) then
            continue;
        fi;
        f := Filename(MyDir, f);
        RemoveFile(f);
    od;

    for s in [1.. Size(gensPSIofFAC)] do
        CMtxBinaryFFMatOrPerm(PIMsOverFp_In_Database[b].generators[s],p,Concatenation(M,String(s)));
    od;

    MatricesModuleNow:=[];

    for t in [1.. Size(gensG)] do # here, we are adding the matrices corresponding to gensG
        Add(MatricesModuleNow,FromStringToMatrix(G,gensG,p,ListGensAsStrings[t]));
    od;
    Add(TSModulesOverFpWithCorrectMatrices,GModuleByMats(MatricesModuleNow,GF(p)));
od;

# It is important which generators of which group are expressed by which other generators
# of which other group.

# First, we save the matrices, then, we express the images of Gens_FAC by the generators
# from the database group.

for v in [1.. Size(ChiProjsNewTable)] do
    Add(CompleteList_V_M_Chi, [pSubgroupsUpToConjugacy[1],
    TSModulesOverFpWithCorrectMatrices[v],ChiProjsNewTable[v]]);
od;

Print("completelistvmchi ist"); Print(CompleteList_V_M_Chi);

# now over the field Fq:
# in order to do that we conjugate the matrices (with entries in Fp) with the alpha_i's,
# but only if this is necessary

CompleteList_V_M_Chi_Over_Fq:=[];

PIMsOverFqCorrectModules:=[];

# 'correct' means that we consider the actual group now and
# not the group from the database any longer

for i in [1.. Size(ChiProjsNewTable)] do
    counter:=0;
```

```
        for j in [1.. Size(Alpha_i_s_G)] do
            if Alpha_i_s_G[j][1] = i then
                counter:=counter + 1;
                ModuleOverFpNowOverFq :=
                GModuleByMats(TSModulesOverFpWithCorrectMatrices[i].generators,Alpha_i_s_G[j][3]);
                # recall that [3] gives the field and [4] gives the number of direct summands
                # hence we can multiply the gens (formerly over Fp) with alpha
                ConjugatedMatricesOverFq:=[];
                for a in [1.. Size(ModuleOverFpNowOverFq.generators)] do
                    OldGens:=ModuleOverFpNowOverFq.generators[a];
                    AlphaNow := Alpha_i_s_G[j][2];
                    AlphaNow_i := Alpha_i_s_G[j][2]^-1;
                    BlockDiagonalGens:=AlphaNow*OldGens*AlphaNow_i;
                    Add(ConjugatedMatricesOverFq,BlockDiagonalGens);
                od;

                # Now, we extract the matrix blocks (and define the smaller representations).

                t:=Size(ConjugatedMatricesOverFq[1])/Alpha_i_s_G[j][4];
                # Hence the variable t equals the number of rows of the old matrix divided by
                # the number of direct summands. This is equal to the number of
                # rows of the new, smaller matrices.

                ListOfListsNewGeneratorMatrices:=List([1..Alpha_i_s_G[j][4]], x -> []);

                for b in [1.. Size(ConjugatedMatricesOverFq)] do
                    for u in [1.. Alpha_i_s_G[j][4]] do
                        matt:=ExtractSubMatrix(ConjugatedMatricesOverFq[b], [(u-1)*t+1..u*t], [(u-1)*t+1..u*t]);
                        Add(ListOfListsNewGeneratorMatrices[u], matt);
                    od;
                od;

                SmallerModulesFromTheDiagonal:=[];

                for c in [1.. Alpha_i_s_G[j][4]] do
Add(SmallerModulesFromTheDiagonal,GModuleByMats(ListOfListsNewGeneratorMatrices[c],Alpha_i_s_G[j][3]));
                od;

                Append(PIMsOverFqCorrectModules,SmallerModulesFromTheDiagonal);
            fi;
        od;

        if counter = 0 then
            Add(PIMsOverFqCorrectModules, TSModulesOverFpWithCorrectMatrices[i]);
        fi;
    od;

    # next, we create the list V_M_Chi (i.e.: vertex, module, character) over Fq:

    for w in [1.. Size(ChiProjsNewTableOverFq)] do
        Add(CompleteList_V_M_Chi_Over_Fq, [pSubgroupsUpToConjugacy[1],
        PIMsOverFqCorrectModules[w],ChiProjsNewTableOverFq[w]]);
    od;

    IRR_CTG:=Irr(ctG);

    AuxiliaryAllToChopAndMultMatrices := [];
    AllBasesGalConjugatesForGreen:=[];

    CounterNumberOfDirSummandsGreenCorrForAllPGroups := [];
    # this will be WITHOUT the PIMs later, too!

    for i in pSubgroupsUpToConjugacy do
        if Order(i) > 1 then
            Print("i ist gerade gleich: "); Print(i); Print("von insgesamt");
            Print(Size(pSubgroupsUpToConjugacy)); Print("p-Untergruppen (bis auf Konjugation in G).\n");
            N:=Normalizer(G,i);

            ctN:=CharacterTable(N); Display(ctN);

            P:=AsSubgroup(N,i);
            hom:=NaturalHomomorphismByNormalSubgroupNC(N,P);
            FAC:=Image(hom);
            ctFAC:=CharacterTable(FAC); Display(ctFAC);

            if not IsIdenticalObj( PreImagesRange( hom ), N ) then
                Print("ES GAB PROBLEME BEI INFLATION !!!");
```

```
            return(fail);
        fi;

        gensOfN:=GeneratorsOfGroup(N);

        Print("Jetzt faengt die Berechnung von V an...");
        V:=WriteOrGetPIMsDataOverFqViaDatabase(FAC,p);
        PSI:=V[1];
        TheRecord:=V[2];
        gensPSIofFAC:=TheRecord.gensG; # i.e. the generators of the image of PSI
        PSI_TO_THE_MINUS_ONE:=InverseGeneralMapping(PSI);
        gensFAC:=List(gensPSIofFAC, x -> ImagesRepresentative(PSI_TO_THE_MINUS_ONE,x));
        PSI_AsGroupHomom:=GroupHomomorphismByImages(FAC,Image(PSI),gensFAC,gensPSIofFAC);
        PSI_TO_THE_MINUS_ONE_AsGroupHomom:=GroupHomomorphismByImages(Image(PSI),FAC,gensPSIofFAC,
    gensFAC);
        OldMatScalProds:=TheRecord.temp_scalprods;
        OldMatScalProdsOverFq:=TheRecord.ScalprodsPIMsOverFq;
        OldIrrCT:=TheRecord.Irr_As_List_Of_Lists;

        OldConjugacyClasses := List(TheRecord.ConjugacyClasses, xxx-> Representative(xxx));
        List_Preimages_OldConjugacyClasses :=
        List(OldConjugacyClasses, yyy -> ImagesRepresentative(PSI_TO_THE_MINUS_ONE,yyy));

        cclsFAC:=ConjugacyClasses(ctFAC);

        TranspMatOldIrr:=[];

        for v in [1.. Size(cclsFAC)] do
            for w in [1.. Size(List_Preimages_OldConjugacyClasses)] do
                if IsConjugate(FAC,Representative(cclsFAC[v]),
                List_Preimages_OldConjugacyClasses[w]) then
                    Add(TranspMatOldIrr,TransposedMat(OldIrrCT)[w]);
                fi;
            od;
        od;

        NewIrrCT:=Irr(ctFAC);
        NewOldIrrCT:=TransposedMat(TranspMatOldIrr);

        PermutationsOldAndNewCharTable:=TransformingPermutations(NewOldIrrCT,NewIrrCT);
        PermRows:=PermutationsOldAndNewCharTable.rows;
        PermColumns:=PermutationsOldAndNewCharTable.columns;

        if not IsZero(Order(PermColumns)-1) then
            Print("Columns war nicht die leere Permutation BEI FAC !!!");
            return fail;
        else
            Print("Das mit PermColumns hat nun bei irgend so einem FAC geklappt!!!   ;-) ");
        fi;

        ChiProjsNewTable:=[];
        for a in [1.. Size(OldMatScalProds)] do
            Chi:=0;
            for j in [1.. Size(OldMatScalProds[a])] do
                Chi := Chi + OldMatScalProds[a][j][1]*Irr(ctFAC)[OnPoints(j,PermRows)];
            od;
            Chi := ClassFunction(ctFAC,Chi);
            Add(ChiProjsNewTable,Chi);
        od;

        ChiProjsNewTableOverFq:=[];
        for a in [1.. Size(OldMatScalProdsOverFq)] do
            Chi:=0;
            for j in [1.. Size(OldMatScalProdsOverFq[a])] do
                Chi := Chi + OldMatScalProdsOverFq[a][j][1]*Irr(ctFAC)[OnPoints(j,PermRows)];
            od;
            Chi := ClassFunction(ctFAC,Chi);
            Add(ChiProjsNewTableOverFq,Chi);
        od;

        # next, we inflate the characters Chi_i and then we also inflate the modules

        Inflated_Characters_N:=[];

        for j in [1.. Size(ChiProjsNewTable)] do
            Add(Inflated_Characters_N,RestrictedClassFunction(ctFAC,ChiProjsNewTable[j],hom));
        od;
```

```
Inflated_Characters_N_OverFq:=[];

for j in [1.. Size(ChiProjsNewTableOverFq)] do
    Add(Inflated_Characters_N_OverFq,RestrictedClassFunction(ctFAC,ChiProjsNewTableOverFq[j],hom));
od;

N_bar_Auxiliary:=GroupByGenerators(gensPSIofFAC); # this is the group from the database
ListGensAsStrings:=[];

for a in [1.. Size(gensOfN)] do
    NaturalHomOfGenN := Image(hom,gensOfN[a]);
    PsiOfNatHomOfGenN := Image(PSI_AsGroupHomom,NaturalHomOfGenN);

    fac:=Factorization(N_bar_Auxiliary,PsiOfNatHomOfGenN);
    facAsString:=String(fac);

    if 'i' in facAsString then # i.e. if we have "<identity ...>" here
        facAsString := "x1*x1^-1";
    fi;
    Add(ListGensAsStrings,facAsString);
od;

PIMsOverFp_In_Database:=TheRecord.AllPIMsOver_FAsMTXModules;

Alpha_i_s_N := TheRecord.AllBasesForGaloisConjugates;

TSModulesNOverFpWithCorrectMatrices:=[];

for b in [1.. Size(PIMsOverFp_In_Database)] do
    files := Filtered(DirectoryContents(MyDir), f -> Length(f)>1 and f[1] = 'M');

    for f in files do
        if f[2] <> '.' and not ForAll(f{[2..Length(f)]}, IsDigitChar) then
            continue;
        fi;
        f := Filename(MyDir, f);
        RemoveFile(f);
    od;

    for s in [1.. Size(gensPSIofFAC)] do
        CMtxBinaryFFMatOrPerm(PIMsOverFp_In_Database[b].generators[s],p,Concatenation(M,String(s)));
    od;

    MatricesModuleNow:=[];

    for t in [1.. Size(gensOfN)] do # here, we want to have the matrices corresponding to gensOfN
        Add(MatricesModuleNow,FromStringToMatrix(N,gensOfN,p,ListGensAsStrings[t]));
    od;

    Add(TSModulesNOverFpWithCorrectMatrices,GModuleByMats(MatricesModuleNow,GF(p)));
od;

Inflated_Modules_N := ShallowCopy(TSModulesNOverFpWithCorrectMatrices);

# Next, we induce the characters and modules from N (the present normaliser) to G !

InducedCharacters:=[];
InducedModules:=[];

CompleteList_V_M_Chi_copy:=ShallowCopy(CompleteList_V_M_Chi);
for t in [1.. Size(Inflated_Characters_N)] do
# the inflated characters N are belonging to the modules over Fp

    Chi_N := Inflated_Characters_N[t];
    Chi_G := InducedClassFunction(Chi_N,ctG);

    M_N := Inflated_Modules_N[t];
    M_G := InducedGModule(G,N,M_N);

    # At the moment, we have fixed: the p-subgroup (i.e.: the vertex) and the induced module
    # M_G over Fp. We want to keep track of the stripped off modules from the list
    # CompleteListVMChi. Then we have the matrix StripErgebnis[3], which equals basComplete
    # and that consists of first BasisImPHI and then BasisKerPHI.
    # That way, the matrix which is stripped off first , i.e. L with my GAP notation,
    # is on the top left when it comes to the block diagonal matrices ( after base change).
```

```
    M_Ind := ShallowCopy(M_G);

    MatricesForConjugationStillToChopAndMultiply:=[];

    for j in [1.. Size(CompleteList_V_M_Chi_copy)] do
        # next, by comparing ordinary characters, we test, if it is possible at all
        # that the present module occurs as a direct summand
        DIFFERENZ := Chi_G - CompleteList_V_M_Chi_copy[j][3];
        if ForAll( IRR_CTG, x -> ScalarProduct(x,DIFFERENZ) > -1 ) then
            MAX_Abspalt:=0;
            while ForAll( IRR_CTG, x -> ScalarProduct(x,DIFFERENZ) > -1 ) do
                MAX_Abspalt:=MAX_Abspalt+1;
                DIFFERENZ := DIFFERENZ - CompleteList_V_M_Chi_copy[j][3];
            od;

            if Size(ContainingConjugates( G, i, CompleteList_V_M_Chi_copy[j][1] )) > 0 then
            # i.e.: if i contains a G-conjugate of the vertex of our module that we test now
                Print("Now, the Shared C MeatAxe is executed.");
                # Now, we strip off as many copies of the first candidate as possible,
                # since after that the index j is increased by one.

                counter:=0;
                repeat
                    StripErgebnis := StripOffOneCopyOfNFromMIfPossible(M_G,CompleteList_V_M_Chi_copy[j
][2]);

                    M_G := StripErgebnis[2];
                    if StripErgebnis[1]=1 then
# thus we can at least strip off one indec. t.s. module from M_G, namely the module CompleteList_V_M_Chi_copy[j][2]
                        Chi_G := Chi_G - CompleteList_V_M_Chi_copy[j][3];
                        counter := counter + 1;
                        Add(MatricesForConjugationStillToChopAndMultiply, StripErgebnis[3]);
# the representation which is stripped off (i.e.: L with my GAP notation)
# is put to the top left in the block diagonal matrix after base change
                        Print("the matrix in question ist: "); Print(StripErgebnis[3]); Print("\n");
                    fi;
                until StripErgebnis[1]=0 or counter > MAX_Abspalt;
            fi;
        fi;
    od;

    Add(AuxiliaryAllToChopAndMultMatrices,MatricesForConjugationStillToChopAndMultiply);
    # this is only for debugging purposes

    # Now, MatricesForConjugationStillToChopAndMultiply contains the base change matrices
    # which yield block diagonal matrices. But we still have to add dxd-identity matrices
    # before multiplying the base change matrices.
    # Here, d = diff = n - Dimension of the present matrix
    # and n = Dim of the matrices of M_Ind.

    n := Size(M_Ind.generators[1]); # this is correct, since M_Ind is defined via ShallowCopy

    if not IsZero(Size(gensG)-Size(M_Ind.generators)) then
        Print("number of gensG is equal to the number of gens of M_Ind!"); return(fail);
    fi;

    ListMatricesForConjugationWithCorrectDimensions := [];

    for a in [1.. Size(MatricesForConjugationStillToChopAndMultiply)] do
        m := Size(MatricesForConjugationStillToChopAndMultiply[a]);
        diff := n - m;
        IdentityMatrixToAddOnTheTopLeftCorner := IdentityMat(diff,GF(p));
        if diff > 0 then
            MatNew := DirectSumMat(IdentityMatrixToAddOnTheTopLeftCorner,
            MatricesForConjugationStillToChopAndMultiply[a]);
        else
            MatNew := MatricesForConjugationStillToChopAndMultiply[a];
        fi;
        Add(ListMatricesForConjugationWithCorrectDimensions,MatNew);
        # this looks as follows: [M1,M2,M3,...,Mr]
    od;

    # in the end, we would like to have: Mr*Mr-1*...*M2*M1*M*M1^-1*M2^-1*...*Mr^-1
    # more precisely: I'd like to define the matrix MATR given by Mr*...*M1

    MATR := M_Ind.generators[1]^0;
```

```
        if Size(ListMatricesForConjugationWithCorrectDimensions) > 0 then
            MATR := ListMatricesForConjugationWithCorrectDimensions[1];

            for b in [1.. Size(MatricesForConjugationStillToChopAndMultiply)−1] do
                MATR := ListMatricesForConjugationWithCorrectDimensions[b+1] ∗ MATR;
                # we multiply Id from the left by M1, then by M2, ...
            od;
        fi ;

        # We remark that GAP can directly multiply matrices with entries in GF(p)
        # by matrices with entries in F_q.

        ModuleMIndInBlockDiagonalForm :=
GModuleByMats(List([1..Size(M_Ind.generators)], x −> MATR ∗ M_Ind.generators[x] ∗ MATR^−1),GF(p));

        # We have described how we obtain the Green correspondent(s) over Fp that way.
        # But, of course, we are also interested in the trivial source modules over Fq.

        # In order to obtain them we conjugate the induced module by Alpha_Big and then by
        # MyBaseChangeMatrix

        TSModulesNOverFqCorrectModules:=[];

        counter:=0;
        CounterNumberOfDirSummandsGreenCorr := [];
        AllSubmodulesOfGREENOverFq := [];
        BasisGalConjugatesForGreen := [];
        for w in [1.. Size(Alpha_i_s_N)] do
            if Alpha_i_s_N[w][1] = t then
                counter:=counter + 1;

                L := ShallowCopy(M_Ind);
                LOverFq := GModuleByMats(L.generators,Alpha_i_s_N[w][3]);
                y := L.dimension;
                AlphaNow := Alpha_i_s_N[w][2];
                z := y/Size(AlphaNow);
                ListeCopiesOfAlphaNow := List([1..z], x −> AlphaNow);
                Alpha_BIG := DirectSumMat(ListeCopiesOfAlphaNow);

                ss := Size(AlphaNow) / Alpha_i_s_N[w][4];
                # this is equal to Size(AlphaNow) divided by the number of summands in Alpha
                aa := Alpha_i_s_N[w][4];
                mm := ShallowCopy(z);

                BasChMat := MyBaseChangeMat(ss,aa,mm,p);

                ConjugationMatrixForInducedModuleTransitionFromFpToFq := BasChMat ∗ Alpha_BIG;

                # Strategy: start with L , conjugate it , then:
                # consider (for all Galois conjugates separately ) the vectors [1,0,0,0,0,0,...] till , say,
    [0,0,1,0,0,...],
                # and so forth. Then compute the corresponding vectors after base change, such
                # that they are in the same basis as the module L.
                # The last vectors correspond to the Green correspondent over Fp.
                # Finally, compute the intersection and the induced submodule of L.

                # After transition from Fp to Fq the induced module has in the end as many
                # direct summands as one little alpha has blocks , i.e. aa many.

                # From these aa many summands, each has the following dimension
                # in the big matrix:
                # Dim(Alpha_BIG) / aa ... and this equals y / aa.

                uu := y/aa; # this is the number of basis vectors to be considered per summand

                RelevantVecs_FqBackToFp_AsListOfLists:=[];

                for ff in [1.. aa] do
                    RelevantVecs_FqBackToFp := List([(ff−1)∗uu+1..(ff−1)∗uu+uu],
                    x −> ConjugationMatrixForInducedModuleTransitionFromFpToFq[x]);
                    Add(RelevantVecs_FqBackToFp_AsListOfLists,RelevantVecs_FqBackToFp);
                    # these sublists generate submodules of L over Fq.
                od;
                # Recall from linear algebra : the columns of a base change matrix
                # are the images of the basis vectors , but GAP acts from the right.
                # Hence, here, the rows are the images of the basis vectors .

```

```
if Size(ListMatricesForConjugationWithCorrectDimensions) > 0 then
    DimGreenCorresp := M_G.dimension;
else
    DimGreenCorresp := Size(MATR);
fi;

k_now := Alpha_i_s_N[w][3];
ModuleMIndInBlockDiagonalFormOverFq :=
GModuleByMats(ModuleMIndInBlockDiagonalForm.generators, k_now);

genMatricesGreenCorresp := [];

for g in [1.. Size(ModuleMIndInBlockDiagonalFormOverFq.generators)] do
    Add(genMatricesGreenCorresp,
    ExtractSubMatrix(ModuleMIndInBlockDiagonalFormOverFq.generators[g],
    [Size(MATR) - DimGreenCorresp+1..Size(MATR)],
    [Size(MATR) - DimGreenCorresp+1..Size(MATR)]));
od;

GreenCorrespondAtLevelG :=
GModuleByMats(genMatricesGreenCorresp,ModuleMIndInBlockDiagonalFormOverFq.field);
Print("TheGreencorresp is:\n"); Print(GreenCorrespondAtLevelG); Print("\n");

CopyOfGreenOverFq := ShallowCopy(GreenCorrespondAtLevelG);
# and later we change this copy

matricesforconjugationstilltochopandmultiply := [];

for ff in [1.. aa] do

    BasIndFq := MTX.SpinnedBasis(RelevantVecs_FqBackToFp_AsListOfLists[ff],
    LOverFq.generators,k_now);

    IndFqAsModule := MTX.InducedActionSubmodule(LOverFq,BasIndFq);

    StripResultAsList := MaxCommonDirectSummandFq(CopyOfGreenOverFq,IndFqAsModule);
    # Order within the list : [PHIasModule,RestOfGREEN,ConjugationMatr]

    Add(matricesforconjugationstilltochopandmultiply, StripResultAsList[3]);

    submod := StripResultAsList[1];
    Add(AllSubmodulesOfGREENOverFq,submod);

        # Replace Copy of Green:
    CopyOfGreenOverFq := ShallowCopy(StripResultAsList[2]);
od;

# test, if sum dimenisons = dim green over Fq:

if IsVectorSpace(CopyOfGreenOverFq)=false then
    Print("the sum of the dimensions is not equal to the DIM of GREEN");
    Print("or there was a mistake in the function MaxCommonDirectSummand");
    return(fail);
fi;

# again, we keep track of the base change matrices
# we still have to chop and multiply them

listmatricesforconjugationwithcorrectdimensions := [];

nnn := GreenCorrespondAtLevelG.dimension;

for aaa in [1.. Size(matricesforconjugationstilltochopandmultiply)] do
    mmm := Size(matricesforconjugationstilltochopandmultiply[aaa]);
    diff := nnn - mmm;
    identitymatrixtoaddonthetopleftcorner := IdentityMat(diff,k_now);
    if diff > 0 then
        matnew := DirectSumMat(identitymatrixtoaddonthetopleftcorner,
        matricesforconjugationstilltochopandmultiply[aaa]);
    else
        matnew := matricesforconjugationstilltochopandmultiply[aaa];
    fi;
    Add(listmatricesforconjugationwithcorrectdimensions,matnew);
    # this looks as follows: [M1,M2,M3,...,Mr]
od;

# Now we want to define the matrix matr given by
```

```
                        # Mr*...*M2*M1*M*M1^-1*M2^-1*...*Mr^-1

                        matr := matricesforconjugationstilltochopandmultiply[1]^0;

                        if Size(listmatricesforconjugationwithcorrectdimensions) > 0 then
                            matr := listmatricesforconjugationwithcorrectdimensions[1];

                            for bbb in [1.. Size(matricesforconjugationstilltochopandmultiply)-1] do
                                matr := listmatricesforconjugationwithcorrectdimensions[bbb+1] * matr;
                        # we multiply Id from the left by M1, then by M2, ...
                            od;
                        fi;
                        Append(BasisGalConjugatesForGreen,matr);
                    fi;
                od;

                if counter = 0 then
                    Add(CompleteList_V_M_Chi_Over_Fq, [i,M_G,Chi_G]);
                    Add(CompleteList_V_M_Chi, [i,M_G,Chi_G]);
                    Add(AllBasesGalConjugatesForGreen,[]);
                    Add(CounterNumberOfDirSummandsGreenCorrForAllPGroups,1);
                else
                    for bb in [1.. Size(AllSubmodulesOfGREENOverFq)] do
                        Add(CompleteList_V_M_Chi_Over_Fq,[i, AllSubmodulesOfGREENOverFq[bb],"character to do !!!"]);
                    od;
                    Add(CompleteList_V_M_Chi, [i,M_G,Chi_G]);
                    Add(AllBasesGalConjugatesForGreen,BasisGalConjugatesForGreen);
                    Add(CounterNumberOfDirSummandsGreenCorrForAllPGroups,Size(AllSubmodulesOfGREENOverFq));
                fi;
            od;
        fi;
    od;

    Print("CompleteList_V_M_Chi ist: "); Print(CompleteList_V_M_Chi);

    ScalProdsTSModules:=[];
    for m in [1.. Size(CompleteList_V_M_Chi)] do
        Add(ScalProdsTSModules,MatScalarProducts(ctG,[CompleteList_V_M_Chi[m][3]],Irr(ctG)));
    od;

    # It remains to compute the scalar products over Fq.

    ScalprodsPIMsOverFq := [];
    for m in [1.. Size(CompleteList_V_M_Chi_Over_Fq)] do
        if Order(CompleteList_V_M_Chi_Over_Fq[m][1]) = 1 then
            Add(ScalprodsPIMsOverFq,MatScalarProducts(ctG,[CompleteList_V_M_Chi_Over_Fq[m][3]],Irr(ctG)));
        fi;
    od;

    if IdGroupsAvailable(Order(G)) then
        IdentifyingG:=IdSmallGroup(G);
    else
        IdentifyingG:=["could not identify G !!!"];
    fi;

    # We now substitute all t.s. modules with a copy of themselves with less information.

    for m in [1.. Size(CompleteList_V_M_Chi)] do
        a2:=ShallowCopy(CompleteList_V_M_Chi[m][2].generators);
        CompleteList_V_M_Chi[m][2]:= GModuleByMats(a2,k);
    od;

    # We collect the subgroups P_i to use them in a later program (here: TSCT).
    # Note that we do that in the end to obtain the correct generators of the (factor) groups.

    Subgroups_Pi:=[];

    for i in pSubgroupsUpToConjugacy do
        GensOfGr := GeneratorsOfGroup(i);
        if Order(i) > 1 then
            Add(Subgroups_Pi,GensOfGr);
        else
            H := Group([()]);
            GensOfGr := GeneratorsOfGroup(H);
            Add(Subgroups_Pi,GensOfGr);
        fi;
    od;
```

```
    IrrCT:=Irr(ctG);

    Irr__As__List__Of__Lists:=[];

    for u in  [1.. Size(IrrCT)] do
        v:=ShallowCopy(IrrCT[u]);
        Add(Irr__As__List__Of__Lists,v);
    od;

    MyTSModulesAndLiftsRecord :=  rec();

    MyTSModulesAndLiftsRecord.gensG := gensG;
    MyTSModulesAndLiftsRecord.OrderG := Order(G);
    MyTSModulesAndLiftsRecord.G := G;
    MyTSModulesAndLiftsRecord.IdentifyingG := IdentifyingG;
    MyTSModulesAndLiftsRecord.Field := k;
    MyTSModulesAndLiftsRecord.Characteristic := Characteristic(k);
    MyTSModulesAndLiftsRecord.CompleteList__V__M__Chi__Over__Fp := CompleteList__V__M__Chi;
    MyTSModulesAndLiftsRecord.CompleteList__V__M__Chi__Over__Fq := CompleteList__V__M__Chi__Over__Fq;
    MyTSModulesAndLiftsRecord.IrrCT := Irr__As__List__Of__Lists;
    MyTSModulesAndLiftsRecord.ScalProdsTSModules__Over__Fp := ScalProdsTSModules;
    MyTSModulesAndLiftsRecord.cclsG := cclsG;
    MyTSModulesAndLiftsRecord.SubgroupsPi := Subgroups__Pi;
    MyTSModulesAndLiftsRecord.ScalprodsPIMsOverFq := ScalprodsPIMsOverFq;

    MyTSModulesAndLiftsRecord.AllBasesGalConjugatesForGreen__WITHOUT__PIMS__AT__THE__BEGINNING:=
    AllBasesGalConjugatesForGreen;

    MyTSModulesAndLiftsRecord.CounterNumberOfDirSummandsGreenCorrForAllPGroups:=
    CounterNumberOfDirSummandsGreenCorrForAllPGroups;

    MyTSModulesAndLiftsRecord.AuxiliaryAllToChopAndMultMatrices := AuxiliaryAllToChopAndMultMatrices;

    MyTSModulesAndLiftsRecord.AllBasesGalConjugates__ONLY__PIMS__AT__THE__BEGINNING := Alpha__i__s__G;

    return MyTSModulesAndLiftsRecord;
end;
```

```
WriteOrGetTSModulesAndLiftsOverFqViaDatabase:=function(G,p)

    local str, str0, fileTS, GroupsSameOrder, psi, i, psi__test, dataTSModulesAndLifts, H, U;

    if Order(G) < 101 then
        str:="TSLiftsDatabaseOverFq1to100.txt";
    elif Order(G) < 201 then
        str:="TSLiftsDatabaseOverFq101to200.txt";
    elif Order(G) < 301 then
        str:="TSLiftsDatabaseOverFq201to300.txt";
    elif Order(G) < 401 then
        str:="TSLiftsDatabaseOverFq301to400.txt";
    elif Order(G) < 501 then
        str:="TSLiftsDatabaseOverFq401to500.txt";
    elif Order(G) < 601 then
        str:="TSLiftsDatabaseOverFq501to600.txt";
    elif Order(G) < 701 then
        str:="TSLiftsDatabaseOverFq601to700.txt";
    elif Order(G) < 801 then
        str:="TSLiftsDatabaseOverFq701to800.txt";
    elif Order(G) < 901 then
        str:="TSLiftsDatabaseOverFq801to900.txt";
    elif Order(G) < 1001 then
        str:="TSLiftsDatabaseOverFq901to1000.txt";
    elif Order(G) < 1101 then
        str:="TSLiftsDatabaseOverFq1001to1100.txt";
    elif Order(G) < 1201 then
        str:="TSLiftsDatabaseOverFq1101to1200.txt";
    elif Order(G) < 1301 then
        str:="TSLiftsDatabaseOverFq1201to1300.txt";
    elif Order(G) < 1401 then
        str:="TSLiftsDatabaseOverFq1301to1400.txt";
    elif Order(G) < 1501 then
        str:="TSLiftsDatabaseOverFq1401to1500.txt";
    else
       str:="TSLiftsDatabaseOverFqGroupOrdersLargerThan1500.txt";
    fi;

    Read(
"/home/bernhard/Schreibtisch/GAP__Database/GAP__Database__Over__Fq/TSModulesAndLiftsOverFq.txt");

    str0:="/home/bernhard/Schreibtisch/GAP__Database/GAP__Database__Over__Fq/";

    fileTS:=Concatenation(str0,str);

    Read(fileTS);

    GroupsSameOrder:=Filtered(databaseTSFq, x -> x.OrderG=Order(G));

    psi:=0;

    for i in [1.. Size(GroupsSameOrder)] do
        if p = GroupsSameOrder[i].Characteristic then
            psi__test:=IsomorphismGroups(G, GroupsSameOrder[i].G);
            if psi__test <> fail then
                psi:=ShallowCopy(psi__test);
                dataTSModulesAndLifts:=GroupsSameOrder[i];
            fi;
        fi;
    od;
    if psi <> 0 then
        return([psi,dataTSModulesAndLifts]);
    else
        psi:=IsomorphismPermGroup(G);
        H:=Image(psi);
        U:=TSModulesAndLiftsOverFq(H,p);
        Add(databaseTSFq,U);
        PrintTo(fileTS, "databaseTSFq:=");
        AppendTo(fileTS, databaseTSFq);
        AppendTo(fileTS,";");
        return([psi,U]);
    fi;
end;
```

```
LoadPackage("ctbllib");
LoadPackage("io");
LoadPackage("qpa", "=1.34");

Read(
"/home/bernhard/Schreibtisch/GAP__Database/GAP__Database__Over__Fq/WriteOrGetTSModulesAndLiftsOverFqViaDatabase.txt
    ");
Read("/home/bernhard/Schreibtisch/StripOffOneCopyOfNFromMIfPossible.txt");


# The program EquivalentLibraryCharacterTableWithGroup is written by Thomas Breuer:

#############################################################################
##
#F  EquivalentLibraryCharacterTableWithGroup( <G> )
##
EquivalentLibraryCharacterTableWithGroup:= function( G )
    local init, Gcopy, name, attr, Gtbl, tbl, trans, compat, ccl, new, i;

    # If the group stores already an ordinary character table
    # then we cannot set the attributes consistently.
    if HasOrdinaryCharacterTable( G ) then
      Error( "<G> has already a character table" );
    fi;

    # Test cheap attributes first, and exclude duplicates.
    init:= AllCharacterTableNames( Size, Size( G ),
                NrConjugacyClasses, NrConjugacyClasses( G ),
                IsDuplicateTable, false );
    if Length( init ) = 0 then
      # No expensive tests are needed.
      # In patricular, do not compute a character table.
      return fail;
    fi;

    # Create a copy of the group, in order to compute its character table
    # without storing it.
    # (Note that calling 'AttributeValueNotSet' for 'OrdinaryCharacterTable'
    # does not help, since 'Irr' etc. would appear silently.)
    # Store the known attributes of 'G' in the copy,
    # in particular 'Gcopy' and 'G' have the same ordering of conj. classes.
    Gcopy:= GroupWithGenerators( GeneratorsOfGroup( G ) );
    for name in KnownAttributesOfObject( G ) do
      attr:= ValueGlobal( name );
      Setter( attr )( Gcopy, attr( G ) );
    od;

    # Compute the character table of the copy.
    Gtbl:= OrdinaryCharacterTable( Gcopy );
    for name in init do
      tbl:= CharacterTable( name );
      trans:= TransformingPermutationsCharacterTables( tbl, Gtbl );
      if trans <> fail then
        # Take this library table:
        # - Permute the classes stored in the group.
        compat:= ListPerm( trans.columns, NrConjugacyClasses( tbl ) );
        ccl:= ConjugacyClasses( G ){ compat };

        # - Copy the contents of the library table.
        new:= ConvertToLibraryCharacterTableNC(
                  rec( UnderlyingCharacteristic := 0 ) );

        # - Set the supported attribute values except 'Irr'.
        for i in [ 3, 6 .. Length( SupportedCharacterTableInfo ) ] do
          if Tester( SupportedCharacterTableInfo[ i-2 ] )( tbl )
             and SupportedCharacterTableInfo[ i-1 ] <> "Irr" then
            Setter( SupportedCharacterTableInfo[ i-2 ] )( new,
                SupportedCharacterTableInfo[ i-2 ]( tbl ) );
          fi;
        od;

        # - Set the irreducibles.
        SetIrr( new, List( Irr( tbl ),
            chi -> Character( new, ValuesOfClassFunction( chi ) ) ) );

        # - Set the group in the table.
        SetUnderlyingGroup( new, G );
```

```
          SetConjugacyClasses( new, ccl );
          SetIdentificationOfConjugacyClasses( new, compat );

          # - Set the table in the group.
          SetOrdinaryCharacterTable( G, new );

          return new;
        fi ;
    od;

    # No library table fits .
    # However, we set the computed character table , since we know it.
    SetOrdinaryCharacterTable( G, Gtbl );
    return fail;
    end;

# The following function DoesVtxContainQ is an auxiliary program that determines
# if the p-group Vtx contains an N-conjugate of Q.

DoesVtxContainQ:= function( N, Vtx, Q )
    local ccsSubgroups, ccsSubgroupsReps, ccsSubgroups_as_Subgroups_Of_N, i, U, U_in_N, Q_in_N, flag;

    ccsSubgroups := ConjugacyClassesSubgroups(Vtx);
    ccsSubgroupsReps := List(ccsSubgroups, x -> Representative(x));
    ccsSubgroups := Filtered(ccsSubgroupsReps, x -> Order(x) = Order(Q));

    ccsSubgroups_as_Subgroups_Of_N:=[];
    for i in [1.. Size(ccsSubgroups)] do
        U:=ccsSubgroups[i];
        U_in_N:=AsSubgroup(N,U);
        Add(ccsSubgroups_as_Subgroups_Of_N, U_in_N);
    od;

    Q_in_N:=AsSubgroup(N,Q);

    flag := false;

    for i in [1.. Size(ccsSubgroups)] do
        flag:= IsConjugate(N,ccsSubgroups_as_Subgroups_Of_N[i],Q_in_N);
        if flag <> false then
            return flag;
        fi ;
    od;

    return flag;
    end;

# The following function converts a GAP expression to tex code.
# Example: x squared is translated to "x^2".

GAPStringToTex:=function(str)

    local list1 , i , list2 , PlusOrMinus;

    list1 :=SplitString(str ,"^");

    if Size( list1 ) > 1 then
        for i in [2.. Size( list1 )] do
            if Size(SplitString ( list1 [i ], "-"))=2 then
                list2 :=SplitString( list1 [i ], "-");
                PlusOrMinus:="-";
            elif Size(SplitString ( list1 [i ], "+"))=2 then
                list2 :=SplitString( list1 [i ], "+");
                PlusOrMinus:="+";
            else
                list2 :=[ list1 [i ]];
            fi ;
            if Size( list2 )=2 then
                list1 [i-1]:=Concatenation(list1[i-1],"^{", list2 [1], "}",PlusOrMinus);
                list1 [i]:= list2 [2];
            else
                list1 [i-1]:=Concatenation(list1[i-1],"^{", list2 [1], "}");
                list1 [i]:="";
            fi ;
        od;
    fi ;
    return Concatenation(list1);
```

```
end;

# Now we define a function that allows us to evaluate a class function at a group
# element in GAP.

EvaluationOfClassFunctionAtElement := function(chi, elm, grp, ListConjugacyClasses)
# Remark: ListConjugacyClasses must be a list of elements, NOT a list of classes .

    local pos, flag , i , ClassNow;

    pos:=0;
    flag:=false;
    i:=0;
    while flag = false do
        i:=i+1;
        ClassNow := ListConjugacyClasses[i];

        if IsConjugate(grp,ClassNow, elm) then
            pos:=i;
            flag:=true;
        fi ;
    od;

    return ValuesOfClassFunction(chi)[pos];
end;

# Again, we need the program FromStringToMatrix.

FromStringToMatrix:=function(G,gensG,p,fAsString)

    local DirOfChop, M, i, RES, ERGEBNIS, MAT, StringNow, z, PositionsOpenParentheses,
    PositionsCloseParentheses, KLAMMER, r, SSS, STR, SPLITnow, KlammerAuf, KlammerZu, u,
    StringNow1, StringNow2, StringNow3, StringYNow, StringToChange, SPLIT, INP, ergebnis_to_return,
    stdin, stdout, MyDir, LocationOfZPRAsString, LocationOfZPOAsString, LocationOfZMUAsString,
    path, rm, options, pro, dir , files , f ;

    LoadPackage("io");
    # DirOfChop:=Directory("/home/bernhard/Schreibtisch/shared_meataxe-1.0/src/");
    ChangeDirectoryCurrent("/home/bernhard");

    MyDir:=Directory("/home/bernhard");
    stdin := InputTextUser();;
    stdout := OutputTextUser();;
    LocationOfZPRAsString := "/home/bernhard/Schreibtisch/shared_meataxe-1.0/bin/zpr";
    LocationOfZPOAsString := "/home/bernhard/Schreibtisch/shared_meataxe-1.0/bin/zpo";
    LocationOfZMUAsString := "/home/bernhard/Schreibtisch/shared_meataxe-1.0/bin/zmu";
    path := DirectoriesSystemPrograms();
    rm := Filename(path,"rm");

    RES:=Filename(MyDir, "RES");

    ERGEBNIS:=Filename(MyDir, "ERGEBNIS");
    MAT:=Filename(DirectoryCurrent(), "MAT");
    KLAMMER:=Filename(DirectoryCurrent(), "KLAMMER");

    dir := Directory("/home/bernhard");

    files := Filtered(DirectoryContents(MyDir), f -> Length(f)>3 and f[1] = 'R' and f[2] = 'E'
    and f[3] = 'S');
    for f in files do

        if f [4] <> '.' and not ForAll(f{[4..Length(f)]}, IsDigitChar) then
            continue;
        fi ;
        f := Filename(MyDir, f);
        RemoveFile(f);
    od;

    files := Filtered(DirectoryContents(MyDir), f -> Length(f)>8 and f[1] = 'E' and f[2] = 'R'
    and f[3] = 'G' and f[4] = 'E' and f[5] = 'B' and f[6] = 'N' and f[7] = 'I' and f[8] = 'S');
    for f in files do

        if f [9] <> '.' and not ForAll(f{[9..Length(f)]}, IsDigitChar) then
            continue;
        fi ;
        f := Filename(MyDir, f);
        RemoveFile(f);
```

```
od;

 files  := Filtered(DirectoryContents(MyDir), f -> Length(f)>3 and f[1] = 'M' and f[2] = 'A'
and f[3] = 'T');
for f in files do

     if f[4] <> '.' and not ForAll(f{[4..Length(f)]}, IsDigitChar) then
         continue;
     fi;
     f := Filename(MyDir, f);
     RemoveFile(f);
od;

 files  := Filtered(DirectoryContents(MyDir), f -> Length(f)>7 and f[1] = 'K' and f[2] = 'L'
and f[3] = 'A' and f[4] = 'M' and f[5] = 'M' and f[6] = 'E' and f[7] = 'R');
for f in files do

     if f[8] <> '.' and not ForAll(f{[8..Length(f)]}, IsDigitChar) then
         continue;
     fi;
     f := Filename(MyDir, f);
     RemoveFile(f);
od;

StringNow:=ShallowCopy(fAsString);
z:=0;

  while '(' in StringNow do
     PositionsOpenParentheses := Positions(StringNow,'(');
     PositionsCloseParentheses := Positions(StringNow,')');
     z:=z+1;
     KlammerZu:=PositionsCloseParentheses[1];
     u:=PositionsCloseParentheses[1];
     while (u in PositionsOpenParentheses)=false do
         u:=u-1;
     od;
     KlammerAuf:=u;
     StringToChange := StringNow{ [KlammerAuf+1..KlammerZu-1] } ;
     SPLIT:=SplitString(StringToChange, "*");

     for i in [1.. Size(SPLIT)] do
         STR:=SPLIT[i];
         SPLITnow:=SplitString(STR,"^");
         if Size(SPLITnow)=2 then
             SSS:=ReplacedString(SPLITnow[1],"x","M");
             SSS:=ReplacedString(SSS,"y","KLAMMER.");

             options:=[SSS, String(SPLITnow[2]), Concatenation("MAT.",String(i))];

             pro := Process(MyDir, LocationOfZPOAsString, stdin, stdout, options);
             if not IsZero(pro) then
                 Print("The last process did not return zero!");
                 return(fail);
             fi;
         else
             SSS:=ReplacedString(SPLITnow[1],"x","M");
             SSS:=ReplacedString(SSS,"y","KLAMMER.");

             options:=[SSS, String(1), Concatenation("MAT.",String(i))];

             pro := Process(MyDir, LocationOfZPOAsString, stdin, stdout, options);
             if not IsZero(pro) then
                 Print("The last process did not return zero!");
                 return(fail);
             fi;
         fi;
     od;

     options:=["MAT.1","MAT.2","RES.2"];

     pro := Process(MyDir, LocationOfZMUAsString, stdin, stdout, options);
     if not IsZero(pro) then
         Print("The last process did not return zero!");
         return(fail);
     fi;

     for i in [2.. Size(SPLIT)-1] do
```

```
            options:=[Concatenation("RES.",String(i)),
            Concatenation("MAT.",String(i+1)),Concatenation("RES.",String(i+1))];

            pro := Process(MyDir, LocationOfZMUAsString, stdin, stdout, options);
            if not IsZero(pro) then
                Print("The last process did not return zero!");
                return(fail);
            fi;
        od;

        r:=Maximum(Size(SPLIT),2);

        options:=[Concatenation("RES.",String(r)), "1", Concatenation("KLAMMER.",String(z))];

        pro := Process(MyDir, LocationOfZPOAsString, stdin, stdout, options);
        if not IsZero(pro) then
            Print("The last process did not return zero!");
            return(fail);
        fi;

        if KlammerAuf > 1 then
            StringNow1 := StringNow{ [1..KlammerAuf-1] } ;
        else
            StringNow1 := "";
        fi;

        StringNow2 := Concatenation("KLAMMER.",String(z));

        if KlammerZu < Size(StringNow) then
            StringNow3 := StringNow{ [KlammerZu+1..Size(StringNow)] } ;
        else
            StringNow3 := "";
        fi;
        StringYNow := Concatenation("y",String(z));
        StringNow:=Concatenation(StringNow1,StringYNow,StringNow3);
    od;

    StringToChange := StringNow;

    SPLIT:=SplitString(StringToChange, "*");

    for i in [1.. Size(SPLIT)] do
        STR:=SPLIT[i];
        SPLITnow:=SplitString(STR,"^");
        if Size(SPLITnow)=2 then
            SSS:=ReplacedString(SPLITnow[1],"x","M");
            SSS:=ReplacedString(SSS,"y","KLAMMER.");

            options:=[SSS, String(SPLITnow[2]), Concatenation("MAT.",String(i))];

            pro := Process(MyDir, LocationOfZPOAsString, stdin, stdout, options);
            if not IsZero(pro) then
                Print("The last process did not return zero!");
                return(fail);
            fi;
        else
            SSS:=ReplacedString(SPLITnow[1],"x","M");
            SSS:=ReplacedString(SSS,"y","KLAMMER.");

            options:=[SSS, String(1), Concatenation("MAT.",String(i))];

            pro := Process(MyDir, LocationOfZPOAsString, stdin, stdout, options);
            if not IsZero(pro) then
                Print("The last process did not return zero!");
                return(fail);
            fi;
        fi;
    od;

    if Size(SPLIT) = 1 then

        options:=[Concatenation(MAT,".",String(1)), Concatenation(ERGEBNIS,".text")];

        pro := Process(MyDir, LocationOfZPRAsString, stdin, stdout, options);
        if not IsZero(pro) then
            Print("The last process did not return zero!");
```

```
                return(fail);
            fi;

            INP :=InputTextString
( Concatenation("ergebnis"," := ScanMeatAxeFile(","\"",Concatenation(ERGEBNIS,".text"),"\"",");"));;
            Read(INP);
        else

            options:=["MAT.1","MAT.2","RES.2"];

            pro := Process(MyDir, LocationOfZMUAsString, stdin, stdout, options);
            if not IsZero(pro) then
                Print("The last process did not return zero!");
                return(fail);
            fi;

            for i in [2.. Size(SPLIT)−1] do

                options:=[Concatenation("RES.",String(i)), Concatenation("MAT.",String(i+1)),
                Concatenation("RES.",String(i+1))];

                pro := Process(MyDir, LocationOfZMUAsString, stdin, stdout, options);
                if not IsZero(pro) then
                    Print("The last process did not return zero!");
                    return(fail);
                fi;
            od;

            r:=Maximum(Size(SPLIT),2);

            options:=[Concatenation("RES.",String(r)), "1", Concatenation("ERGEBNIS.",String(1))];

            pro := Process(MyDir, LocationOfZPOAsString, stdin, stdout, options);
            if not IsZero(pro) then
                Print("The last process did not return zero!");
                return(fail);
            fi;

            options:=[Concatenation(ERGEBNIS, ".", String(1)), Concatenation(ERGEBNIS, ".text")];

            pro := Process(MyDir, LocationOfZPRAsString, stdin, stdout, options);
            if not IsZero(pro) then
                Print("The last process did not return zero!");
                return(fail);
            fi;

            INP :=InputTextString
( Concatenation("ergebnis"," := ScanMeatAxeFile(","\"",Concatenation(ERGEBNIS,".text"),"\"",");"));;
            Read(INP);
        fi;

        ergebnis_to_return := ShallowCopy(ergebnis);

        return ergebnis_to_return;
end;

# The following auxiliary program evaluates the Brauer character corresponding to an (Fp)N−module
# at the p'−conjugacy classes of N.
# Input: (Fp)N−Modul M, group N, gensOfN, ListPPrimeClassesN, p, alpha, AnzSummands
# Output: BrauerCharacterValues of M at the PPrimeClasses of N

BrauerCharValuesOfMAtPPrimeClassesOfN_neue_Version :=
function(ModuleOverN,N,gensOfN,ListPPrimeClassesN,p,alpha, AnzSummands)

        local N_Auxiliary, ListPPrimeClassesNAsStrings, a, ps, fac, facAsString, M, r, s, MatricesModuleNow, t;

        ChangeDirectoryCurrent("/home/bernhard");

        MyDir:=Directory("/home/bernhard");
        stdin := InputTextUser();;
        stdout := OutputTextUser();;
        LocationOfZPRAsString := "/home/bernhard/Schreibtisch/shared_meataxe−1.0/bin/zpr";
        LocationOfZPOAsString := "/home/bernhard/Schreibtisch/shared_meataxe−1.0/bin/zpo";
        LocationOfZMUAsString := "/home/bernhard/Schreibtisch/shared_meataxe−1.0/bin/zmu";
        path := DirectoriesSystemPrograms();
        rm := Filename(path,"rm");

```

```
    N__Auxiliary := GroupByGenerators(gensOfN); # this is the same order as the matrices of ModuleOverN
    ListPPrimeClassesNAsStrings:=[];

    for a in [1.. Size(ListPPrimeClassesN)] do
        ps:=ListPPrimeClassesN[a];
        fac:=Factorization(N__Auxiliary,ps);
        facAsString:=String(fac);

        if 'i' in facAsString then # i.e. if we have "<identity ...>" here
            facAsString := "x1*x1^-1";
        fi;

        Add(ListPPrimeClassesNAsStrings,facAsString);
    od;

    M:=Filename(DirectoryCurrent(), "M");

    files := Filtered(DirectoryContents(MyDir), f -> Length(f)>1 and f[1] = 'M');
    for f in files do
        # Skip all files with names not starting with M. or having the form M<zahl>
        if f[2] <> '.' and not ForAll(f{[2..Length(f)]}, IsDigitChar) then
            continue;
        fi;
        f := Filename(MyDir, f);
        RemoveFile(f);
    od;

    for s in [1.. Size(gensOfN)] do
        CMtxBinaryFFMatOrPerm(ModuleOverN.generators[s],p,Concatenation(M,String(s)));
    od;

    MatricesModuleNow:=[];

    # here we want the matrices corresponding to ListPPrimeClassesN
    for t in [1.. Size(ListPPrimeClassesN)] do
        Add(MatricesModuleNow,FromStringToMatrix(N,gensOfN,p,ListPPrimeClassesNAsStrings[t]));
    od;

    # It is only left to conjugate with alpha and collect the determined values in a list .
    # We collect all AnzSummands lists in ListAllBrauerEvaluationsNow.
    # jetzt müssen wir nur noch mit alpha konjugieren und AnzSummands viele Fälle/Auswertungen als kleine Listen in die große Liste packen:
    ListAllBrauerEvaluationsNow:=[];

    if AnzSummands > 1 then
        ListMatricesWithBlockDiagonalEntries := List(MatricesModuleNow, x -> alpha*x*alpha^-1);
        DimOfDirSummand:=Size(ListMatricesWithBlockDiagonalEntries[1])/AnzSummands;

        for i in [1.. AnzSummands] do
            ListeSmallModuleMatricesOverFq :=
            List(ListMatricesWithBlockDiagonalEntries, x -> ExtractSubMatrix(x,
            [(i-1)*DimOfDirSummand + 1 .. i*DimOfDirSummand],
            [(i-1)*DimOfDirSummand + 1 .. i*DimOfDirSummand]));
            Add(ListAllBrauerEvaluationsNow, List(ListeSmallModuleMatricesOverFq,
            x -> BrauerCharacterValue(x)));
        od;
    else
        Add(ListAllBrauerEvaluationsNow, List(MatricesModuleNow, x -> BrauerCharacterValue(x)));
    fi;
    return ListAllBrauerEvaluationsNow;
end;


# The next function restricts a FpG-module to N:

Restriction := function(G,gensG,N,gensOfN,ModuleOverG,p) # ModuleOverG should be a record

    local G__Auxiliary, ListGensAsStrings, a, ps, fac, facAsString, M, r, s,
            MatricesModuleNow, t, RestrictedModule;

    ChangeDirectoryCurrent("/home/bernhard");

    MyDir:=Directory("/home/bernhard");
```

```
        stdin := InputTextUser();;
    stdout := OutputTextUser();;
    LocationOfZPRAsString := "/home/bernhard/Schreibtisch/shared__meataxe−1.0/bin/zpr";
    LocationOfZPOAsString := "/home/bernhard/Schreibtisch/shared__meataxe−1.0/bin/zpo";
    LocationOfZMUAsString := "/home/bernhard/Schreibtisch/shared__meataxe−1.0/bin/zmu";
        path := DirectoriesSystemPrograms();
    rm := Filename(path,"rm");

    G__Auxiliary := GroupByGenerators(gensG);
    ListGensAsStrings:=[];
    for a in [1.. Size(gensOfN)] do
        ps:=gensOfN[a];
        fac:=Factorization(G__Auxiliary,ps);
        facAsString:=String(fac);

        if 'i' in facAsString then # i.e. if we have "<identity ...>" here
            facAsString := "x1*x1^−1";
        fi ;

        Add(ListGensAsStrings,facAsString);
    od;

    M:=Filename(DirectoryCurrent(), "M");

    files := Filtered(DirectoryContents(MyDir), f −> Length(f)>1 and f[1] = 'M');
    for f in files do
        # Skip all files with names not starting with M. or having the form M<zahl>
        if f [2] <> '.' and not ForAll(f{[2..Length(f)]}, IsDigitChar) then
            continue;
        fi ;
        f := Filename(MyDir, f);
        RemoveFile(f);
    od;

    for s in [1.. Size(gensG)] do
        CMtxBinaryFFMatOrPerm(ModuleOverG.generators[s],p,Concatenation(M,String(s)));
    od;

    MatricesModuleNow:=[];

    for t in [1.. Size(gensOfN)] do # now we want the matrices corresponding to gensOfN
        Add(MatricesModuleNow,FromStringToMatrix(N,gensOfN,p,ListGensAsStrings[t]));
    od;

    RestrictedModule := GModuleByMats(MatricesModuleNow,GF(p));

    return RestrictedModule;
end;

# In order to compute the off−diagonal entries in the trivial source character table , we apply the following method. We restrict
      the present module M (with vertex P, say) to the present
# normaliser N(Q). Then, we compute which direct summands have vertices containing Q and dircard the other modules from the
      direct sum decomposition. After that , we compute the Brauer
# character values of the remaining summands at the p'−classes of N(Q). We would like to avoid multiplying matrices over Fq.
      Therefore, we do not compute a direct sum decomposition
# directly , but with the help of previously computed information. Our Strategy is analogous to that durig the computation of t.s.
      kG−modules (i.e.: Green correspondents) over Fq.
# However, this time, in order to do computations over Fq, we have to keep track of which modules we get rid of. That way,
#
# 1) den eingerahmten Modul erst über N, dann für alle pprime classes definieren Wichtige Bemerkung: zuerst Blockmatrizen geht
      net, da man dann ja wieder über Fq multiplizieren müsste !!!
# 2) Endvektoren der Vektorraumschnitte für alle Summands of alpha__now berechnen
# 3) InducedSubmodule mit 2)
# 4) BrauerCharValues berechnen und returnen

GetBrauerCharacterValuesOffDiagonalEntries__neue__Version := function(N, gensOfN, ModuleOverN,
DIM__after__boese__Lste, List__PPrimeClassesN, alpha__G, NumberOfSummands__alpha__G, MATR, p)

local ListAllBrauerEvaluationsNow, N__Auxiliary, ListPPrimeClassesNAsStrings, a, ps, fac,
        facAsString, M, r, s, MatricesModuleWithPPrimeClasses, t,
        MatricesModuleWithPPrimeClassesInBlocksAndOverFq, MatricesSubModuleGuteListe,
        k__now, BraverEingerahmterModul, EinheitsMat, RelevantVecs__GuteListe, V1, BAS, BS,
        RelevantVecs__FqBackToFp__AsListOfLists, uu, ff, RelevantVecs__FqBackToFp, V2, V3, S,
        BSnew, h, hh, ll, bas, submod, temp, b, c;

    ChangeDirectoryCurrent("/home/bernhard");

    MyDir:=Directory("/home/bernhard");
```

```
stdin := InputTextUser();;
stdout := OutputTextUser();;
LocationOfZPRAsString := "/home/bernhard/Schreibtisch/shared__meataxe-1.0/bin/zpr";
LocationOfZPOAsString := "/home/bernhard/Schreibtisch/shared__meataxe-1.0/bin/zpo";
LocationOfZMUAsString := "/home/bernhard/Schreibtisch/shared__meataxe-1.0/bin/zmu";
path := DirectoriesSystemPrograms();
rm := Filename(path,"rm");

ListAllBrauerEvaluationsNow:=[];
N__Auxiliary := GroupByGenerators(gensOfN); # the matrices of ModuleOverN are given in the same order
ListPPrimeClassesNAsStrings:=[];

for a in [1.. Size(List__PPrimeClassesN)] do
    ps:=List__PPrimeClassesN[a];
    fac:=Factorization(N__Auxiliary,ps);
    facAsString:=String(fac);

    if 'i' in facAsString then # i.e. if we have "<identity ...>" here
        facAsString := "x1*x1^-1";
    fi;

    Add(ListPPrimeClassesNAsStrings,facAsString);
od;

M:=Filename(DirectoryCurrent(), "M");

 files := Filtered(DirectoryContents(MyDir), f -> Length(f)>1 and f[1] = 'M');
for f in files do
    # Skip all files with names not starting with M. or having the form M<zahl>
    if f[2] <> '.' and not ForAll(f{[2..Length(f)]}, IsDigitChar) then
        continue;
    fi;
    f := Filename(MyDir, f);
    RemoveFile(f);
od;

for s in [1.. Size(gensOfN)] do
    CMtxBinaryFFMatOrPerm(ModuleOverN.generators[s],p,Concatenation(M,String(s)));
od;

# now we want the matrices corresponding to List__PPrimeClassesN
MatricesModuleWithPPrimeClasses:=[];
for t in [1.. Size(List__PPrimeClassesN)] do
Add(MatricesModuleWithPPrimeClasses,FromStringToMatrix(N,gensOfN,p,ListPPrimeClassesNAsStrings[t]));
od;

# next, we conjugate with MATR and focus only on certain blocks of the (block) matrix representation

MatricesModuleWithPPrimeClassesInBlocksAndOverFq :=
List(MatricesModuleWithPPrimeClasses, x -> MATR * x * MATR^-1);

# now we extract the submodule corresponding to a dir. sum of 'good' modules, i.e. modules whose vertices
# contain the p-subgroup in question .... it is called 'BraverEingerahmterModul' and located on the bottom right
# of the block diagonal matrix representation MatricesModuleWithPPrimeClassesInBlocksAndOverFq.

MatricesSubModuleGuteListe := List(MatricesModuleWithPPrimeClassesInBlocksAndOverFq,
    x -> ExtractSubMatrix(x, [Size(MATR) - DIM__after__boese__Lste+1..Size(MATR)],
    [Size(MATR) - DIM__after__boese__Lste+1..Size(MATR)] ) );

k__now := DefaultField(Flat(alpha__G));

BraverEingerahmterModul := GModuleByMats(MatricesSubModuleGuteListe,k__now);

EinheitsMat := IdentityMat(Size(MATR) , k__now );
RelevantVecs__GuteListe := List([1..DIM__after__boese__Lste],
x -> EinheitsMat[Size(MATR) - DIM__after__boese__Lste + x]);

V1 := VectorSpace(k__now, RelevantVecs__GuteListe);

# We now compute in the good/bad basis, but define it over Fq:
BAS:=Basis(V1);
BS:=BasisVectors(BAS);

RelevantVecs__FqBackToFp__AsListOfLists:=[];

uu := Size(alpha__G)/NumberOfSummands__alpha__G; # this is the number of rows for each summand
```

```
    for ff in [1.. NumberOfSummands_alpha_G] do
        RelevantVecs_FqBackToFp := List([(ff−1)∗uu+1..(ff−1)∗uu+uu], x −> (alpha_G[x])∗MATR^−1);
        Add(RelevantVecs_FqBackToFp_AsListOfLists,RelevantVecs_FqBackToFp);
    od;

    for ff in [1.. NumberOfSummands_alpha_G] do
        V2 := VectorSpace(k_now,RelevantVecs_FqBackToFp_AsListOfLists[ff]);
        V3 := Intersection( V1, V2 );
        BAS:=Basis(V3);
        BS:=BasisVectors(BAS);

        # take only the vectors in question and then write new basis which fits for the intersection :

        S:=Size(MATR);
        BSnew:=[];
        for h in [1.. Size(BS)] do
            Add(BSnew,[]);
        od;

        for hh in [1.. Size(BS)] do
            for ll in [(S−DIM_after_boese_Lste+1)..S] do
                Add(BSnew[hh],BS[hh][ll]);
            od;
        od;

        bas := MTX.SpinnedBasis(BSnew,BraverEingerahmterModul.generators,k_now);

        submod := MTX.InducedActionSubmodule(BraverEingerahmterModul,bas);
        # This is the module we wanted to determine. It remains to determine the Brauer
        # character and its values on the p’−classes of N.
        temp:=[];

        for r in [1.. Size(submod.generators)] do
            Add(temp,BrauerCharacterValue(submod.generators[r]));
        od;

        Add(ListAllBrauerEvaluationsNow, temp);
    od;

    return ListAllBrauerEvaluationsNow;
end;

# We now give an implemenation of the algorithm in Bezout’s lemma.
# Input: two natural numbers u, v
# Output: [s1,t1] where s1∗u + t1∗v = 1
# If u and v are not coprime, the program returns ’fail’.

Bezout := function(u,v)

local x, y, r1, r2, s1, s2, t1, t2, s2_old, t2_old, r, q;

if u<v then
x := v;
y := u;
else
x := u;
y := v;
fi;

r1 := x;
r2 := y;
s1 := 1;
s2 := 0;
t1 := 0;
t2 := 1;

while r2 > 0 do
r := ShallowCopy(r1 mod r2);
        q := ShallowCopy((r1−r)/r2);
        r1 := ShallowCopy(r2);
        r2 := ShallowCopy(r);
        s2_old := ShallowCopy(s2);
        s2 := ShallowCopy(s1−q∗s2);
        s1 := ShallowCopy(s2_old);
        t2_old := ShallowCopy(t2);
        t2 := ShallowCopy(t1−q∗t2);
        t1 := ShallowCopy(t2_old);
```

```
od;

if not IsZero(1-r1) then
Print("The gcd of the two numbers is not equal to 1 or you have not entered (a) non-negative integer(s).");
    return(fail);
fi;

if u<v then
    return([t1,s1]);
fi;

return([s1,t1]);
end;

# The following computes the p-part resp. the p'-part of a group element g of a finite group.
# INPUT: (g,p) where g is a group element of a finite permutation group and p is a prime number.
# OUTPUT: a list that contains the p-part and the p'-part of g as elements.

PPartAndPPrimePartOfGroupElement := function(g,p)

    local ORD, facs, facs_filtered, n__p, n__q, BEZ, m, n;

    ORD:=Order(g);
    facs := Factors(ORD);
    facs_filtered := Filtered(facs, x -> IsZero(x-p));
    n__p := Product(facs_filtered);
    n__q := ORD/n__p;

    if n__q = 1 then # g is a p-element in this case
        return([g,g^ORD]);
    fi;

    BEZ := Bezout(n__p,n__q);

    m := BEZ[2] * n__q; # hence, g^m is the p-part of g

    n := BEZ[1] * n__p; # hence, g^n is the p'-part of g

return([g^m, g^n]);
end;

#
# INPUT: L=Res^G__{N_j}
# conjugation matrix turning L into a direct sum of L1,L2,L3,...
# number of summands of L over Fq
# multiplicity list over FpNj (i.e.: how many t.s. Fp N_j - modules occur how often in GOOD__LIST (for the present t.s. FpN_j-module L)
# number of Fq-summands per FpNj-module
# the conjugation matrices in order to go from FpNj to FqNj
# group generators of Nj
# present p'-classes

FromRestrictionToOffDiagonalEntries :=
        function(V__M__Chi__Over__Fp__Nj, GeneratorsOfNj, L, ConjugMatrixL, NumberOfSummandsL, F__q__max,
         MultiplicityListFpNj, NumberOfFqSummandsPerFpNjModule, ConjugMatricesFromFpNjToFqNjAsList,
         PresentPPrimeClasses)

# L is already an FpNj-module. MultiplicityListFpNj refers to the complete list vmchiNjoverFp,
# but therein only to GOOD modules, i.e. the modules with vertices (conjugate to) Qj where
# Qj is such that Nj=N__G(Qj).
#
# Strategy:
# 1) Create List__Z := NumberOfSummands-list * V__M__Chi__singleFqModules__in__the__correct__order
# 2) Create list temp__L__i
# 3) test the following two modules for maximal common direct summands: Lnow (see below) and the modules in the innermost lists of ListZ;
# then, insert the integer j at the right place ... e.g.: if Lnow=L2 and it has a non-zero common direct summand with ListZ [5][6][7], then insert the number 2 at position [5][6][7] of the list tempLi and
# overwrite L2 with 'RestOfL2', i.e. L2 after having stripped off the common direct summand
# 4) Since Brauer characters are additive, there are no further problems
# we can just conjugate all modules with matrices such that we have all matrices w.r.t. a common basis... after that we obtain the intersection / common dir. summands by intersecting vector spaces;
# Lastly, we compute the BrauerCharacterValues und return the result

    local ListZ, ListY, ListZ__BrauerChars, ListY__BrauerChars, i, PresentModuleOverFp,
    PresentModuleInBlockDiagonalForm, s, ANZ, t, ListWithDirSummandsFqNjRelevantGoodModules,
    j, ModuleGensForJthModule, ListX__BrauerChars, u, ListTemp__L__i, k, ModuleLInBlockDiagonalForm,
```

```
    ListModulesLi, Lnow, a, b, c, MaxComSum, ListWithBrauerCharValuesOfTheL_i_s_StillToBeAddedUp;

    ListZ := [];
    ListY := [];
    ListZ_BrauerChars := [];
    ListY_BrauerChars := [];

    for i in [1.. Size(V_M_Chi_Over_Fp_Nj)] do
        if MultiplicityListFpNj[i] > 0 then

            Print("MultiplicityListFpNj ist gerade: "); Print(MultiplicityListFpNj);
            Print("MultiplicityListFpNj[i] ist gerade: "); Print(MultiplicityListFpNj[i]);

            PresentModuleOverFp := V_M_Chi_Over_Fp_Nj[i][2];

            if Size(ConjugMatricesFromFpNjToFqNjAsList[i]) > 0 then

PresentModuleInBlockDiagonalForm := GModuleByMats(List(PresentModuleOverFp.generators,
x -> ConjugMatricesFromFpNjToFqNjAsList[i]*x*ConjugMatricesFromFpNjToFqNjAsList[i]^-1),F_q_max);

            else
                PresentModuleInBlockDiagonalForm := ShallowCopy(PresentModuleOverFp);
            fi;
            s := Size(PresentModuleOverFp.generators[1]);
            ANZ := NumberOfFqSummandsPerFpNjModule[i];
            t := s/ANZ;
            ListWithDirSummandsFqNjRelevantGoodModules := [];
            for j in [1.. ANZ] do
                ModuleGensForJthModule := List(PresentModuleInBlockDiagonalForm.generators,
                x -> ExtractSubMatrix(x, [(j-1)*t+1..j*t], [(j-1)*t+1..j*t]));

                Add(ListWithDirSummandsFqNjRelevantGoodModules,
                ShallowCopy(GModuleByMats(ModuleGensForJthModule,F_q_max)));
            od;

            if Size(ConjugMatricesFromFpNjToFqNjAsList[i]) > 0 then
                ListX_BrauerChars := BrauerCharValuesOfMAtPPrimeClassesOfN_neue_Version(PresentModuleOverFp,
                Group(GeneratorsOfNj), GeneratorsOfNj, PresentPPrimeClasses, Characteristic(F_q_max),
                ConjugMatricesFromFpNjToFqNjAsList[i], ANZ);
            else
                ListX_BrauerChars := BrauerCharValuesOfMAtPPrimeClassesOfN_neue_Version(PresentModuleOverFp,
                Group(GeneratorsOfNj), GeneratorsOfNj, PresentPPrimeClasses, Characteristic(F_q_max),
                IdentityMatrix(F_q_max,s), ANZ);
            fi;

            # We add as many isomorphic copies to the list ListY as the multiplicitieslist tells us
            for u in [1.. MultiplicityListFpNj[i]] do
                Add(ListY,ListWithDirSummandsFqNjRelevantGoodModules);
                Add(ListY_BrauerChars, ListX_BrauerChars);
            od;
            Add(ListZ, ListY);
            Add(ListZ_BrauerChars, ListY_BrauerChars);
        fi;
    od;

    Print("ListZ ist gerade: "); Print(ListZ);
    Print("ListZ_BrauerChars ist gerade: "); Print(ListZ_BrauerChars);

    # Step 2):

    ListTemp_L_i := [];

    for i in [1.. Size(ListZ_BrauerChars)] do
        Add(ListTemp_L_i, []);
    od;

    for i in [1.. Size(ListZ)] do
        for j in [1.. Size(ListZ[i])] do
            Add(ListTemp_L_i[i], []);
        od;
    od;

    for i in [1.. Size(ListZ)] do
        for j in [1.. Size(ListZ[i])] do
            for k in [1.. Size(ListZ[i][j])] do
                Add(ListTemp_L_i[i][j], []);
            od;
```

```
    od;
od;

# Step 3)

# we suppose that the FpNj-module L is part of the input

if Size(ConjugMatrixL)>0 then
    ModuleLInBlockDiagonalForm := GModuleByMats(List(L.generators,
    x -> ConjugMatrixL * x * ConjugMatrixL^-1), F_q_max);
else
    ModuleLInBlockDiagonalForm := ShallowCopy(L);
fi;

ListModulesLi:=[];

for j in [1.. NumberOfSummandsL] do
    t := Size(L.generators[1]) /NumberOfSummandsL;
    ModuleGensForJthModule := List(ModuleLInBlockDiagonalForm.generators,
    x -> ExtractSubMatrix(x, [(j-1)*t+1..j*t], [(j-1)*t+1..j*t]));
    Add(ListModulesLi, ShallowCopy(GModuleByMats(ModuleGensForJthModule,F_q_max)));
od;

Print("ListModulesLi ist gerade gleich: "); Print(ListModulesLi);

#
# Now, we have the modules L1, L2, L3,... and next:
# if MaxCommonDirectSummand yields true, then put a "1" at the correct place ot ListTemp_L_i (later, when L2 is considered: put a 2, ...)
# and replace L1 (later: L2, L3, ...)


List_Boese_Tripels:=[];
List_all_abcs:=[];

ListZmuh:=[];
for a in [1.. Size(ListZ)] do
    Add(ListZmuh,[]);
od;
for a in [1.. Size(ListZ)] do
    for b in [1.. Size(ListZ[a])] do
        Add(ListZmuh[a],[]);
    od;
od;

for a in [1.. Size(ListZ)] do
    for b in [1.. Size(ListZ[a])] do
        for c in [1.. Size(ListZ[a][b])] do
            ListZmuh[a][b][c]:="muh";
        od;
    od;
od;

Print("Jetzt, bevor wir angefangen haben, muss ich noch sagen: ListZ ist gerade: ");
Print(ListZ);

for j in [1.. Size(ListModulesLi)] do
    Lnow := ShallowCopy(ListModulesLi[j]);
    for a in [1.. Size(ListZ)] do
        for b in [1.. Size(ListZ[a])] do
            for c in [1.. Size(ListZ[a][b])] do
            Add(List_all_abcs,[a,b,c]);
                Print("Lnow ist gerade: "); Print(Lnow);
                Print(" und [a,b,c] ist gerade: ");Print([a,b,c]);
                if not IsVectorSpace(Lnow) then
                    if not IsInt(ListZmuh[a][b][c]) then
                        if not [a,b,c] in List_Boese_Tripels then
                        Dim_Lnow_vorher:=ShallowCopy(Lnow.dimension);
                        MaxComSum := MaxCommonDirectSummandFq(Lnow,ListZ[a][b][c]);

                        Print("MaxComSum ist geradeMUH: "); Print(MaxComSum);

                        if not IsVectorSpace(MaxComSum[2]) then
                            if not IsZero(MaxComSum[2].dimension - Dim_Lnow_vorher) then
                            # this means that Lnow got replaced by a module
                            # which has smaller k-dimension

```

```
                                    Add(ListTemp_L_i[a][b][c], j);
                                    Print("ListTemp_L_i ist gerade: "); Print(ListTemp_L_i);
                                    Print("[a,b,c] ist gerade: "); Print([a,b,c]);
                                    Print("Wir sind gerade im Fall MUH1.\n\n");

                                    ListZmuh[a][b][c]:=-1;
                                    Add(List_Boese_Tripels,[a,b,c]);
                                    Print("Jetzt ist die -1 passiert!!!");
                                    Print("ListZmuh ist gerade: "); Print(ListZmuh);
                                    Lnow := ShallowCopy(MaxComSum[2]);
                                fi;
                            else
                                if not IsZero(Dim_Lnow_vorher) then
                                # this means that Lnow got replaced by a module
                                # which has smaller k-dimension
                                    Add(ListTemp_L_i[a][b][c], j);
                                    Print("ListTemp_L_i ist geradeMUHI: "); Print(ListTemp_L_i);
                                    Print("[a,b,c] ist gerade: "); Print([a,b,c]);
                                    Print("Wir sind gerade im Fall MUH2.\n\n");

                                    Print("1.ListZmuh ist gerade: "); Print(ListZmuh);
                                    if j=2 then
                                        ListZmuh[a][b][c]:=-3;
                                        Print("Jetzt ist die -3 passiert!!!");
                                    elif j=1 then
                                        ListZmuh[a][b][c]:=-4;
                                    else
                                        ListZmuh[a][b][c]:=-5;
                                    fi;
                                    Print("2.ListZmuh ist gerade: "); Print(ListZmuh);
                                    Print("List_all_abcs ist gerade: "); Print(List_all_abcs);
                                    Add(List_Boese_Tripels,[a,b,c]);
                                    Lnow := ShallowCopy(MaxComSum[2]);
# Hence, at this stage, Lnow is the nullspace ... and does not lie in the Filter MTX-module any longer...this is on purpose.
                                fi;
                            fi;

                            fi;
                        fi;
                    fi;
                od;
            od;
        od;
    od;
    Print("List_all_abcs ist gerade: "); Print(List_all_abcs);

    ListWithBrauerCharValuesOfTheL_i_s_StillToBeAddedUp := List(ListModulesLi, x -> []);

# For example, if G= A4 and p=2 then it can happen that the bad module is gone but the good module
# decomposes into a direct sum with multiplicities ... this happens in the line 6 0 0 2 0 0 0 in the
# tsct and 2 means 2 times the triviial module

    #for j in [1.. Size(ListModulesLi)] do
        for a in [1.. Size(ListTemp_L_i)] do
            for b in [1.. Size(ListTemp_L_i[a])] do
                for c in [1.. Size(ListTemp_L_i[a][b])] do
                # c runs through all boxes of the form [2], [1], [3] , ... in the current list
                    if Size(ListTemp_L_i[a][b][c])>0 then
                        u:= ListTemp_L_i[a][b][c][1];
                        Add(ListWithBrauerCharValuesOfTheL_i_s_StillToBeAddedUp[u],
                        ListZ_BrauerChars[a][b][c]);
                    fi;
                od;
            od;
        od;
    #od;

    Print("ListWithBrauerCharValuesOfTheL_i_s_StillToBeAddedUp ist gerade gleich: ");
    Print(ListWithBrauerCharValuesOfTheL_i_s_StillToBeAddedUp);

    Print("ListTemp_L_i ist gerade: "); Print(ListTemp_L_i);

    return List(ListWithBrauerCharValuesOfTheL_i_s_StillToBeAddedUp, x -> Sum(x));
end;

#################################################################
#################################################################
```

```
# The following program is the main program.
# The strategy is as follows:
# 1) the tsct is computed block columnwise;
# 2) we start with the PIMs;
# 3) we collect and use data from the database(s) (i.e. PIMsdatabase and TSLiftsdatabase)
# and must not forget to express the generators, ordinary characters, etc.
# via the generators, ordinary characters, etc. of the group entered by the user;
# 4) we compute the t.s. modules over Fp first and can then use the same
# conjugation matrices as those that were used for the groups from the database;
# 5) the list GuteListe and BoeseListe means the following:
# if a triv. s. kN-module (for some normaliser N<G) is in GuteListe, then its Brauer
# construction w.r.t. the p-subgroup in question is equal to 0;
# 6) we often use our implementation of the algorithm by Brooksbank and Luks that finds
# maximal common direct summands of two given modules due to the following reason:
# the direct summands of a module are often given w.r.t. completely different k-bases (rather
# than in a uniform way);
# 7) once the t.s.c.t. is computed, we determine the ordinary characters by using the
# Corollary of Section 3.1;
# 8) we collect some further data (for producing the tex-file later)
# and return the record containing the data.
######################################################################
######################################################################

TSCTFq:=function(G,p)

    local x, exp, facts, pprimefacts, m, f, k, W, PSI, TheOldRecord, HOMs, OldCompleteList_V_M_Chi,
    OldIrrCT, OldScalProdsTSModules, OldCclsG, SubgroupsPiOld, PSI_TO_THE_MINUS_ONE, gensPSIofFAC,
    gensFAC, PSI_AsGroupHomom, PSI_TO_THE_MINUS_ONE_AsGroupHomom, Subgroups_Pi, j, tempPi, l, ctG,
    gensG, NewIrrCT, PermutationsOldAndNewCharTable, PermRows, ChiTSModulesNewTable, a, Chi,
    rho_TSModules, t, rho_M, TSModulesNEU, w, b, MODUgenerators, MODU, VerticesNEU, gensGRP, tempVTX,
    List_Normalisers, List_FactorGroups, List_NbarEpis, U, N, UU, homNbarEpi, FAC,
    List_All_p_prime_Classes, ccFAC, List_p_prime_Classes_Of_N, RNK, P_PRIME_ORDER, List_Test_Now,
    Gesamtliste_Characters_Chi_All, gensOfN, Ncopy, ctN, W_N, PSI_N, TheOldRecord_N, HOMs_N,
    OldCompleteList_V_M_Chi_N, OldIrrCT_N, OldScalProdsTSModules_N, OldCclsN, SubgroupsPiOld_N,
    PSI_TO_THE_MINUS_ONE_N, gensPSIofFAC_N, gensFAC_N, PSI_AsGroupHomom_N,
    PSI_TO_THE_MINUS_ONE_AsGroupHomom_N, NewIrrCT_N, PermutationsOldAndNewCharTable_N, PermRows_N,
    ChiTSModulesNewTable_N, c, rho_TSModules_N, TSModulesNEU_N, VerticesNEU_N, d, GRP,
    CompleteList_V_M_Chi_N, u, AllTSCTMatCharactersForN, ResM_G_N, IdentifyingG,
    OrdinaryCTAsListOfLists, MyTSCTRecord, ListAllBrauerEvaluations,
    GrosseGesamtliste_V_M_Chi_Over_Fp_All_Normalisers, OldCompleteList_V_M_Chi_Over_Fq,
    OldScalProdsPIMsOverFq, UUU, OldConjugacyClasses, List_Preimages_OldConjugacyClasses, cclsG,
    TranspMatOldIrr, v, counter, Restr, temp_multiplicities, p_prime_classes_recent,
    DIM_after_boese_Lste, MAX_Abspalt, StripErgebnis, NumberOfSummands_alpha_aktuell, GuteListe,
    BoeseListe, DIFFERENZ, Chi_N, MatricesForConjugationStillToChopAndMultiply, NewListV_M_Chi,
    RestrCopy, Fq_RecentModule, AllBasesGalConjugates_BOTH_PIMS_AND_GREEN_N, NumberOfPIMsInN,
    NumberOfSummandsOverFqPerFpNjModuleToEnterInAuxProg, NewListV_M_Chi_N, Pnow, Q,
    ListPostitionsGuteListe, CounterNumberOfDirSummandsOf_The_Alpha_i_s_G,
    TSModulesOverFp_N_WithCorrectMatrices, MatricesModuleNow, alpha_aktuell, NumberOfPIMsInGOverFp,
    cc, TSCTMAT_As_List_With_n_Sublists, fac, ps, N_Auxiliary, dd, SizeTSCT,
    ListOrdinaryCharacterValuesOfAllTSModulesOverFq, ListOrdinaryCharsAllTSModules, ee,
    OrdinaryClasses_Names, List_All_p_prime_Classes_Names, ScalProdsTSModules, OrdinaryClasses,
    flag, ListAllBrauerEvaluationsAsListOfBlockColumns, ListOrdinaryCharOfTSModuleNow,
    FinalVertexPosition, PreliminaryVertexOfInterest, List_p_prime_Classes_Names_Of_N, zzz, gg,
    pos, DIM_per_Summand_Recent_Module_OverFqN, temp_multiplicities_for_huge_List_V_M_Chi_N, vv,
    uu, tt, LISCHDEEE_ModulDIMs, FLAGGE, s, TSModulesOverFpWithCorrectMatrices, pprimeclassesnow,
    temp, List_p_prime_Classes_Names_now, gensOfN_vorlaeufig, PreliminaryPPrimePartOfInterest,
    AllBasesGalConjugatesForGreen_WITHOUT_PIMS_AT_THE_BEGINNING_N, r, pprimePart, ppart, ListZ;

    LoadPackage("io");
    ChangeDirectoryCurrent("/home/bernhard");

    MyDir:=Directory("/home/bernhard");
    stdin := InputTextUser();;
    stdout := OutputTextUser();;
    LocationOfZPRAsString := "/home/bernhard/Schreibtisch/shared_meataxe-1.0/bin/zpr";
    LocationOfZPOAsString := "/home/bernhard/Schreibtisch/shared_meataxe-1.0/bin/zpo";
    LocationOfZMUAsString := "/home/bernhard/Schreibtisch/shared_meataxe-1.0/bin/zmu";
    path := DirectoriesSystemPrograms();
    rm := Filename(path,"rm");

    LoadPackage("PERMUT");

    x:= X(GF(p), "x");
    exp:=Exponent(G);
    facts:=Factors(exp);
    pprimefacts:=Filtered(facts, x-> x mod p <> 0 mod p);
    m:=Product(pprimefacts);
```

```
f:=x^m - 1;
k:=SplittingField(f);

ListAllBrauerEvaluations:=[];

GrosseGesamtliste_V_M_Chi_Over_Fp_All_Normalisers := [];

W:=WriteOrGetTSModulesAndLiftsOverFqViaDatabase(G,p);

HasOrdinaryCharacterTable( G );

PSI:=W[1]; # PSI denotes the map IsomorphismPermGroup from the progr. WriteOrGetTSModulesAndLiftsOverFqViaDatabase
TheOldRecord:=W[2];
OldCompleteList_V_M_Chi:=TheOldRecord.CompleteList_V_M_Chi_Over_Fp;
OldCompleteList_V_M_Chi_Over_Fq:=TheOldRecord.CompleteList_V_M_Chi_Over_Fq;
OldIrrCT:=TheOldRecord.IrrCT;
OldScalProdsTSModules:=TheOldRecord.ScalProdsTSModules_Over_Fp;
OldCclsG:=TheOldRecord.cclsG;
SubgroupsPiOld:=TheOldRecord.SubgroupsPi;
PSI_TO_THE_MINUS_ONE:=InverseGeneralMapping(PSI);
gensPSIofFAC:=TheOldRecord.gensG; # i.e. the generators of the image of psi
gensFAC:=List(gensPSIofFAC, x -> ImagesRepresentative(PSI_TO_THE_MINUS_ONE,x));
PSI_AsGroupHomom:=GroupHomomorphismByImages(G,Image(PSI),gensFAC,gensPSIofFAC);
PSI_TO_THE_MINUS_ONE_AsGroupHomom:=GroupHomomorphismByImages(Image(PSI),G,gensPSIofFAC,gensFAC);


OldScalProdsPIMsOverFq := TheOldRecord.ScalprodsPIMsOverFq;

Subgroups_Pi:=[];
for j in [1.. Size(SubgroupsPiOld)] do
    tempPi:=[];
    if Size(SubgroupsPiOld[j])=0 then
        Add(Subgroups_Pi,Group(()));
    else
        for l in [1.. Size(SubgroupsPiOld[j])] do
            Add(tempPi, ImagesRepresentative(PSI_TO_THE_MINUS_ONE,SubgroupsPiOld[j][l]));
        od;
        Add(Subgroups_Pi,Group(tempPi));
    fi;
od;

if HasOrdinaryCharacterTable(G) then
    ctG:=CharacterTable(G);
else
    UUU:=EquivalentLibraryCharacterTableWithGroup(G);
    ctG:=CharacterTable(G);
fi;

Display(ctG); # without this command only the head of the character table would be computed in some cases.

gensG:=GeneratorsOfGroup(G);

NewIrrCT:=Irr(ctG);

OldConjugacyClasses := List(TheOldRecord.cclsG, xxx-> Representative(xxx));

List_Preimages_OldConjugacyClasses :=
List(OldConjugacyClasses, yyy -> ImagesRepresentative(PSI_TO_THE_MINUS_ONE,yyy));

cclsG:=ConjugacyClasses(ctG);

TranspMatOldIrr:=[];

for v in [1.. Size(cclsG)] do
    for w in [1.. Size(List_Preimages_OldConjugacyClasses)] do
        if IsConjugate(G,Representative(cclsG[v]),List_Preimages_OldConjugacyClasses[w]) then
            Add(TranspMatOldIrr,TransposedMat(OldIrrCT)[w]);
        fi;
    od;
od;

NewOldIrrCT:=TransposedMat(TranspMatOldIrr);

PermutationsOldAndNewCharTable:=TransformingPermutations(NewOldIrrCT,NewIrrCT);
PermRows:=PermutationsOldAndNewCharTable.rows;

PermColumns:=PermutationsOldAndNewCharTable.columns;
```

```
    if not IsZero(Order(PermColumns)-1) then
        Print("Columns war nicht die leere Permutation !!!");
        return fail;
    else
        Print("Das mit PermColumns hat nun beim ersten Mal bei G geklappt!!!   ;-) ");
    fi;

    # Now, we collect the data (concerning modules, characters, vertices) from the database ... over the field Fp

    ChiTSModulesNewTable:=[];
    for a in [1.. Size(OldScalProdsTSModules)] do
        Chi:=0;
        for j in [1.. Size(OldScalProdsTSModules[a])] do
            Chi := Chi + OldScalProdsTSModules[a][j][1]*NewIrrCT[OnPoints(j,PermRows)];
        od;
        Chi := ClassFunction(ctG,Chi);
        Add(ChiTSModulesNewTable,Chi);
    od;



ChiPIMsNewTableOverFq:=[];

for a in [1.. Size(OldScalProdsPIMsOverFq)] do
    Chi:=0;
    for j in [1.. Size(OldScalProdsPIMsOverFq[a])] do
        Chi := Chi + OldScalProdsPIMsOverFq[a][j][1]*NewIrrCT[OnPoints(j,PermRows)];
    od;
    Chi := ClassFunction(ctG,Chi);
    Add(ChiPIMsNewTableOverFq,Chi);
od;

##################################################################

    # A) finish collecting the remaining data from list_V_M_Chi

    G_Auxiliary := GroupByGenerators(gensPSIofFAC); # here: FAC = G/<1>
    ListGensAsStrings:=[];
    for a in [1.. Size(gensG)] do # here, G is the group that was entered by the user, i.e.: as input.
        ps:=Image(PSI_AsGroupHomom,gensG[a]);
        fac:=Factorization(G_Auxiliary,ps);
        facAsString:=String(fac);

        if 'i' in facAsString then # i.e. if we have "<identity ...>" here
            facAsString := "x1*x1^-1";
        fi;

        Add(ListGensAsStrings,facAsString);
    od;

    ListAllTSModulesFpFromTheDatabase := List(OldCompleteList_V_M_Chi, x -> x[2]);

    Alpha_i_s_G := ShallowCopy(TheOldRecord.AllBasesGalConjugatesForGreen_WITHOUT_PIMS_AT_THE_BEGINNING);

    CounterNumberOfDirSummandsOf_The_Alpha_i_s_G :=
    ShallowCopy(TheOldRecord.CounterNumberOfDirSummandsGreenCorrForAllPGroups);

    NumberOfPIMsInGOverFp := Size(Filtered(OldCompleteList_V_M_Chi, x -> Order(x[1]) = 1 ));
    NumberOfPIMsInGOverFq := Size(Filtered(OldCompleteList_V_M_Chi_Over_Fq, x -> Order(x[1]) = 1 ));

    TSModulesOverFpWithCorrectMatrices:=[];

    for b in [1.. Size(ListAllTSModulesFpFromTheDatabase)] do
        M:=Filename(DirectoryCurrent(), "M");

        files := Filtered(DirectoryContents(MyDir), f -> Length(f)>1 and f[1] = 'M');
        for f in files do
            if f[2] <> '.' and not ForAll(f{[2..Length(f)]}, IsDigitChar) then
                continue;
            fi;
            f := Filename(MyDir, f);
            RemoveFile(f);
        od;

        for s in [1.. Size(gensPSIofFAC)] do
            CMtxBinaryFFMatOrPerm(ListAllTSModulesFpFromTheDatabase[b].generators[s],p,
            Concatenation(M,String(s)));
```

```
        od;

        # we want the matrices corresponding to gensG
        MatricesModuleNow:=[];
        for t in [1.. Size(gensG)] do
            Add(MatricesModuleNow,FromStringToMatrix(G,gensG,p,ListGensAsStrings[t]));
        od;
        Add(TSModulesOverFpWithCorrectMatrices,GModuleByMats(MatricesModuleNow,GF(p)));
    od;

    # Now we take care of the vertices :

    VerticesNEU:=[];

    for j in [1.. Size(OldCompleteList_V_M_Chi)] do
        GRP:=OldCompleteList_V_M_Chi[j][1];
        gensGRP:=GeneratorsOfGroup(GRP);
        if Size(gensGRP)=0 then
            Add(VerticesNEU,Group(()));
        else
            tempVTX:=[];
            for a in [1.. Size(gensGRP)] do
                Add(tempVTX,Image(PSI_TO_THE_MINUS_ONE_AsGroupHomom,gensGRP[a]));
            od;
            Add(VerticesNEU,Group(tempVTX));
        fi ;
    od;

    NewListV_M_Chi := []; # this list will also contain the data about the PIMs
    for j in [1.. Size(OldCompleteList_V_M_Chi)] do
        Add(NewListV_M_Chi, [VerticesNEU[j],TSModulesOverFpWithCorrectMatrices[j],ChiTSModulesNewTable[j]]);
    od;

    Add(GrosseGesamtliste_V_M_Chi_Over_Fp_All_Normalisers, NewListV_M_Chi);









    # hier dann weitermachen!!!









    # B) take care of the normalisers N_i and the factor groups and p'-classes with the 'correct' representatives :

    List_Normalisers := [];
    List_FactorGroups := [];
    List_NbarEpis := [];

    for j in [1.. Size(Subgroups_Pi)] do
        U:=AsSubgroup(G,Subgroups_Pi[j]);
        N:=Normaliser(G,U);
        UU:=AsSubgroup(N,U);
        homNbarEpi:=NaturalHomomorphismByNormalSubgroupNC(N,UU);
        FAC:=Image(homNbarEpi);
        Add(List_Normalisers, N);
        Add(List_FactorGroups,FAC);
        Add(List_NbarEpis,homNbarEpi);
    od;

    # find out the p'-classes of the 'correct' representatives :

    List_All_p_prime_Classes:=[];

    for j in [1.. Size(List_FactorGroups)] do
        ccFAC:=ConjugacyClasses(List_FactorGroups[j]);
        List_p_prime_Classes_Of_N:=[];
        for a in [1.. Size(ccFAC)] do
```

```
                if  ((Order(Representative(ccFAC[a])) mod p) = (0 mod p)) = false then
                    RNK:= PreImages( List__NbarEpis[j], Representative(ccFAC[a]) );
                    P__PRIME__ORDER:=Filtered(RNK, x -> ((Order(x) mod p) = (0 mod p)) = false);
                    Add(List__p__prime__Classes__Of__N,P__PRIME__ORDER[1]);
                fi ;
            od;
            Add(List__All__p__prime__Classes, List__p__prime__Classes__Of__N);
        od;


# Next, we add something to the complete  list  of  all  TSCT-matrix entries, namely something from the PIMs over Fq:

pprimeclassesnow := List__All__p__prime__Classes[1];

for t  in  [1.. Size(ChiPIMsNewTableOverFq)] do
    temp := [];
    for y in  [1.. Size(pprimeclassesnow)] do
        Add(temp, pprimeclassesnow[y]^ChiPIMsNewTableOverFq[t]);
        # ALT:
        # Add(temp, EvaluationOfClassFunctionAtElement(ChiPIMsNewTableOverFq[t],
        # pprimeclassesnow[y],  G,  List(ConjugacyClasses(ctG), x -> Representative(x)) ));
    od;
    Add(ListAllBrauerEvaluations, temp);
od;

List__All__p__prime__Classes__Names := [];
List__p__prime__Classes__Names__now := [];

for zzz  in pprimeclassesnow do
    for gg in  [1.. Size(ConjugacyClasses(ctG))] do
        if  IsConjugate(G, Representative(ConjugacyClasses(ctG)[gg]), zzz) then
            pos:=gg;
        fi ;
    od;
    Add(List__p__prime__Classes__Names__now, ClassNames(ctG)[pos]);
od;

Add(List__All__p__prime__Classes__Names, List__p__prime__Classes__Names__now);

# It remains to  collect  the BrauerCharacterValues of all  t.s. kG-modules at the p'-conjugacy classes
# in the  correct  order.
# Recall that  the  list  Alpha__i__s__G is without PIMs.
# Recall that  CounterNumberOfDirSummandsOf__The__Alpha__i__s__G is without PIMs.

for i  in  [1.. Size(NewListV__M__Chi)-NumberOfPIMsInGOverFp] do
    DAS := BrauerCharValuesOfMAtPPrimeClassesOfN__neue__Version(NewListV__M__Chi[i+NumberOfPIMsInGOverFp][2],
    G,gensG,pprimeclassesnow,p,Alpha__i__s__G[i], CounterNumberOfDirSummandsOf__The__Alpha__i__s__G[i]);
    Append(ListAllBrauerEvaluations, DAS);
od;
# now, we are done with the  first  block  column of the  TSCT

temp:=[];

################################################################

# Next, we consider  all  normalisers  except  for  N1 = N_G(<1>)

################################################################

    SizeTSCT := Sum(CounterNumberOfDirSummandsOf__The__Alpha__i__s__G) + NumberOfPIMsInGOverFq;

    for j  in  [2.. Size(List__Normalisers)] do
        N:=List__Normalisers[j];
        Q := ShallowCopy(Subgroups__Pi[j]);
        if  IsZero(Order(G)-Order(N)) then
            NewListV__M__Chi__N := ShallowCopy(NewListV__M__Chi);
            Add(GrosseGesamtliste__V__M__Chi__Over__Fp__All__Normalisers, NewListV__M__Chi__N);

            List__p__prime__Classes__Names__Of__N := [];
            # This is done only now, since  only now the  table  ctN is  fixed

            p__prime__classes__recent := List__All__p__prime__Classes[j];

            for zzz  in  p__prime__classes__recent do
                for gg in  [1.. Size(ConjugacyClasses(ctG))] do
                # notice:  ctG instead  of  ctN
                    if  IsConjugate(G, Representative(ConjugacyClasses(ctG)[gg]), zzz) then
```

```
                    # again: G instead of N
                        pos:=gg;
                    fi ;
                od;
                Add(List_p_prime_Classes_Names_Of_N, ClassNames(ctG)[pos]); # again: ctG instead of ctN
            od;

            Add(List_All_p_prime_Classes_Names, List_p_prime_Classes_Names_Of_N);

            # next, we do the following:
            # if the function DoesVtxContainQ returns false, then we add zeros; otherwise: compute and save the BrauerCharValues
            # of the [j]-th p'-class (we consider N[j]... hence it is important not to take the p'-classes of G)

            for v in [1.. Size(NewListV_M_Chi)] do
                if v in [1.. NumberOfPIMsInGOverFp-1] then
                elif v = NumberOfPIMsInGOverFp then
                    for y in [1.. NumberOfPIMsInGOverFq] do
                        Add(ListAllBrauerEvaluations, List([1..Size(p_prime_classes_recent)], x -> 0));
                    od;
                else # i.e. v > NumberOfPIMsInGOverFp and j > 1
                    if DoesVtxContainQ(G,NewListV_M_Chi[v][1],Q) then

                        # add copies of NumberOfSummands_alpha_aktuell many rows to the right place:

    SumSoFar := Sum(CounterNumberOfDirSummandsOf_The_Alpha_i_s_G{[1..v-NumberOfPIMsInGOverFp-1]})
    + NumberOfPIMsInGOverFq;

for u in [SumSoFar+1..SumSoFar+CounterNumberOfDirSummandsOf_The_Alpha_i_s_G[v-NumberOfPIMsInGOverFp]] do
                            temp:=[];
                            for y in [1.. Size(p_prime_classes_recent)] do
                                for z in [1.. Size(List_All_p_prime_Classes[1])] do
                                    if IsConjugate(G,p_prime_classes_recent[y],
                                    List_All_p_prime_Classes[1][z]) then
                                        Add(temp,ListAllBrauerEvaluations[u][z]); # EVTL. : hier der Fehler, da die Reihenfolge
      der p'-classes von N_1=G und N_2 cong G net gleich sein muss!!!...habe aus [u][y] nun [u][z] genacht!
                                    fi ;
                                od;
                            od;
                            Add(ListAllBrauerEvaluations,temp);
                        od;
                    else
                        # add NumberOfSummands_alpha_aktuell many rows with zeros
                    for ss in [1.. CounterNumberOfDirSummandsOf_The_Alpha_i_s_G[v-NumberOfPIMsInGOverFp]] do
                            Add(ListAllBrauerEvaluations, List([1..Size(p_prime_classes_recent)], x -> 0));
                    od;
                    fi ;
                fi ;
            od;
        else    # i.e. Order(G)>Order(N)
            gensOfN_vorlaeufig:=GeneratorsOfGroup(N);
            Ncopy := GroupWithGenerators( gensOfN_vorlaeufig );
            ctN:=CharacterTable(N); Display(ctN);


            List_p_prime_Classes_Names_Of_N:=[];

            p_prime_classes_recent := List_All_p_prime_Classes[j];

            for zzz in p_prime_classes_recent do
                for gg in [1.. Size(ConjugacyClasses(ctN))] do
                    if IsConjugate(N, Representative(ConjugacyClasses(ctN)[gg]), zzz) then
                        pos:=gg;
                    fi ;
                od;
                Add(List_p_prime_Classes_Names_Of_N, ClassNames(ctN)[pos]);
            od;

            Add(List_All_p_prime_Classes_Names, List_p_prime_Classes_Names_Of_N);

            W_N:=WriteOrGetTSModulesAndLiftsOverFqViaDatabase(Ncopy,p); #
            PSI_N := W_N[1];
            TheOldRecord_N := W_N[2];
            OldCompleteList_V_M_Chi_N:=TheOldRecord_N.CompleteList_V_M_Chi_Over_Fp;

            AllBasesGalConjugates_ONLY_THE_PIMS_N :=
                TheOldRecord_N.AllBasesGalConjugates_ONLY_PIMS_AT_THE_BEGINNING;
            CounterNumberSummands_OF_ONLY_THE_Green_Correspondents_N :=
```

```
    TheOldRecord_N.CounterNumberOfDirSummandsGreenCorrForAllPGroups;
AllBasesGalConjugatesForGreen_WITHOUT_PIMS_AT_THE_BEGINNING_N :=
    TheOldRecord_N.AllBasesGalConjugatesForGreen_WITHOUT_PIMS_AT_THE_BEGINNING;

OldIrrCT_N:=TheOldRecord_N.IrrCT;
OldScalProdsTSModules_N:=TheOldRecord_N.ScalProdsTSModules_Over_Fp;
OldCclsN:=TheOldRecord_N.cclsG;
PSI_TO_THE_MINUS_ONE_N:=InverseGeneralMapping(PSI_N);
gensPSIofFAC_N:=TheOldRecord_N.gensG; # i.e. the generators of the image of PSI; here: FAC=N despite of the name
gensFAC_N:=List(gensPSIofFAC_N, x -> ImagesRepresentative(PSI_TO_THE_MINUS_ONE_N,x));
gensOfN := ShallowCopy(gensFAC_N);

PSI_AsGroupHomom_N:=GroupHomomorphismByImages(N,Image(PSI_N),gensFAC_N,gensPSIofFAC_N);
PSI_TO_THE_MINUS_ONE_AsGroupHomom_N:=
    GroupHomomorphismByImages(Image(PSI_N),N,gensPSIofFAC_N,gensFAC_N);

# next, we construct complete_list_v_m_chi of N

NewIrrCT_N :=Irr(ctN);

OldConjugacyClasses_N := List(OldCclsN, xxx-> Representative(xxx));

List_Preimages_OldConjugacyClasses_N := List(OldConjugacyClasses_N,
    yyy -> ImagesRepresentative(PSI_TO_THE_MINUS_ONE_N,yyy));

cclsN:=ConjugacyClasses(ctN);

TranspMatOldIrr:=[];

for v in [1.. Size(cclsN)] do
    for w in [1.. Size(List_Preimages_OldConjugacyClasses_N)] do
        if IsConjugate(N,Representative(cclsN[v]),List_Preimages_OldConjugacyClasses_N[w]) then
            Add(TranspMatOldIrr,TransposedMat(OldIrrCT_N)[w]);
        fi;
    od;
od;

NewOldIrrCT_N:=TransposedMat(TranspMatOldIrr);

PermutationsOldAndNewCharTable:=TransformingPermutations(NewOldIrrCT_N,NewIrrCT_N);
PermRows:=PermutationsOldAndNewCharTable.rows;

PermColumns:=PermutationsOldAndNewCharTable.columns;
if not IsZero(Order(PermColumns)-1) then
    Print("Columns war nicht die leere Permutation !!!");
    return fail;
else
    Print("Das mit PermColumns hat nun beim ersten Mal bei N geklappt!!!  ;-) ");
fi;

PermutationsOldAndNewCharTable_N := ShallowCopy(PermutationsOldAndNewCharTable);
PermRows_N := PermutationsOldAndNewCharTable_N.rows;

ChiTSModulesNewTable_N:=[];
for a in [1.. Size(OldScalProdsTSModules_N)] do # we are working over Fp
    Chi:=0;
    for c in [1.. Size(OldScalProdsTSModules_N[a])] do
        Chi := Chi + OldScalProdsTSModules_N[a][c][1]*NewIrrCT_N[OnPoints(c,PermRows_N)];
    od;
    Chi := ClassFunction(ctN,Chi);
    Add(ChiTSModulesNewTable_N,Chi);
od;



N_Auxiliary := GroupByGenerators(gensPSIofFAC_N);
# now we have the group from the databse
ListGensAsStrings_N:=[];
for a in [1.. Size(gensOfN)] do
    ps:=Image(PSI_AsGroupHomom_N,gensOfN[a]);
    fac:=Factorization(N_Auxiliary,ps);
    facAsString:=String(fac);

    if 'i' in facAsString then # i.e. if we have "<identity ...>" here
        facAsString := "x1*x1^-1";
    fi;

```

```
            Add(ListGensAsStrings_N,facAsString);
        od;

        ListAllTSModulesFp_N_FromTheDatabase := List(OldCompleteList_V_M_Chi_N , x -> x[2]);

        TSModulesOverFp_N_WithCorrectMatrices:=[];

        for b in [1.. Size(ListAllTSModulesFp_N_FromTheDatabase)] do
            M:=Filename(DirectoryCurrent(), "M");

            files := Filtered(DirectoryContents(MyDir), f -> Length(f)>1 and f[1] = 'M');
            for f in files do
                if f[2] <> '.' and not ForAll(f{[2..Length(f)]}, IsDigitChar) then
                    continue;
                fi;
                f := Filename(MyDir, f);
                RemoveFile(f);
            od;

            for s in [1.. Size(gensPSIofFAC_N)] do
                CMtxBinaryFFMatOrPerm(ListAllTSModulesFp_N_FromTheDatabase[b].generators[s],
                p,Concatenation(M,String(s)));
            od;

            MatricesModuleNow:=[];

            for t in [1.. Size(gensOfN)] do
                Add(MatricesModuleNow,FromStringToMatrix(N,gensOfN,p,ListGensAsStrings_N[t]));
            od;

            Add(TSModulesOverFp_N_WithCorrectMatrices,GModuleByMats(MatricesModuleNow,GF(p)));
        od;

# Next, we compute the actual vertices for the trivial source modules of the normaliser N

        VerticesNEU_N:=[];

        for w in [1.. Size(OldCompleteList_V_M_Chi_N)] do
            GRP:=OldCompleteList_V_M_Chi_N[w][1];
            gensGRP:=GeneratorsOfGroup(GRP);
            if Size(gensGRP)=0 then
                Add(VerticesNEU_N,Group(()));
            else
                tempVTX:=[];
                for a in [1.. Size(gensGRP)] do
                    Add(tempVTX,Image(PSI_TO_THE_MINUS_ONE_AsGroupHomom_N,gensGRP[a]));
                od;
                Add(VerticesNEU_N,Group(tempVTX));
            fi;
        od;

        NewListV_M_Chi_N := [];
        for w in [1.. Size(OldCompleteList_V_M_Chi_N)] do
            Add(NewListV_M_Chi_N, [VerticesNEU_N[w],TSModulesOverFp_N_WithCorrectMatrices[w],
            ChiTSModulesNewTable_N[w]]);
        od;

        # now we finally have computed NewListV_M_Chi_N

        Add(GrosseGesamtliste_V_M_Chi_Over_Fp_All_Normalisers, NewListV_M_Chi_N);

        for v in [1.. Size(NewListV_M_Chi)] do
            if v in [1.. NumberOfPIMsInGOverFp-1] then
            elif v = NumberOfPIMsInGOverFp then
                for y in [1.. NumberOfPIMsInGOverFq] do
                    Add(ListAllBrauerEvaluations, List([1..Size(p_prime_classes_recent)], x -> 0));
                od;
            else
                L := NewListV_M_Chi[v][2];
                Pnow := NewListV_M_Chi[v][1]; # the corresponding vertex
                Q := ShallowCopy(Subgroups_Pi[j]);
                BoeseListe := [];
                GuteListe := [];
                ListPostitionsGuteListe:=[];
                for gg in [1.. Size(NewListV_M_Chi_N)] do
                    if DoesVtxContainQ(N,NewListV_M_Chi_N[gg][1],Q) then
                        Add(GuteListe, NewListV_M_Chi_N[gg]);
```

```
                Add(ListPostitionsGuteListe, gg);
            else
                Add(BoeseListe,NewListV_M_Chi_N[gg]);
            fi;
        od;

        if IsZero(Size(GuteListe)) then
            NumberOfSummands_alpha_aktuell := ShallowCopy(
            CounterNumberOfDirSummandsOf_The_Alpha_i_s_G[v-NumberOfPIMsInGOverFp]);
            for rr in [1.. NumberOfSummands_alpha_aktuell] do
                Add(ListAllBrauerEvaluations, List([1..Size(p_prime_classes_recent)], x -> 0));
            od;
        else

            # we need the correct alpha_i of the present t.s. FpG-module

            alpha_aktuell := ShallowCopy(Alpha_i_s_G[v-NumberOfPIMsInGOverFp]);
            NumberOfSummands_alpha_aktuell := ShallowCopy(
            CounterNumberOfDirSummandsOf_The_Alpha_i_s_G[v-NumberOfPIMsInGOverFp]);

            # the present alpha is still correct/valid at the level of N;
            # next, we need the restriction to N of the present FpG-module;

            Restr := Restriction(G,gensG,N,gensOfN,NewListV_M_Chi[v][2],p); # over Fp
            RestrCopy := ShallowCopy(Restr);
            Chi_N := RestrictedClassFunction( ctG, NewListV_M_Chi[v][3], ctN );

            MatricesForConjugationStillToChopAndMultiply:=[];

            for a in [1.. Size(BoeseListe)] do
                DIFFERENZ := Chi_N - BoeseListe[a][3]; # possible over Fp
                if ForAll( Irr(ctN), x -> ScalarProduct(x,DIFFERENZ) > -1 ) then
                    MAX_Abspalt:=0;
                    while ForAll( Irr(ctN), x -> ScalarProduct(x,DIFFERENZ) > -1 ) do
                        MAX_Abspalt:=MAX_Abspalt+1;
                        DIFFERENZ := DIFFERENZ - BoeseListe[a][3];
                    od;

                    counter:=0;
                    repeat
                        StripErgebnis :=
                            StripOffOneCopyOfNFromMIfPossible(Restr,BoeseListe[a][2]);
                        Restr := StripErgebnis[2];
                        if StripErgebnis[1]=1 then
                        # this means that we can strip off one indec. t.s. module
                        # from Restr, namely the module BoeseListe[a][2]
                            Chi_N := Chi_N - BoeseListe[a][3];
                            counter := counter + 1;
                            Add(MatricesForConjugationStillToChopAndMultiply, StripErgebnis[3]);
                            Print("Le Matrix in question ist: ");
                            Print(StripErgebnis[3]); Print("\n");
                        fi;
                    until StripErgebnis[1]=0 or counter > MAX_Abspalt;
                fi;
            od;

            if IsVectorSpace(Restr) then

                DIM_after_boese_Lste := 0;

                # now, we insert Anz_Fq many zeros into the tsct

                for uu in [1.. NumberOfSummands_alpha_aktuell] do
                    Add(ListAllBrauerEvaluations, List(p_prime_classes_recent, x -> 0));
                od;
            else

                DIM_after_boese_Lste := Restr.dimension; # still over Fp

                temp_multiplicities := [];

                for a in [1.. Size(GuteListe)] do
                DIFFERENZ := Chi_N - GuteListe[a][3];
                    if ForAll( Irr(ctN), x -> ScalarProduct(x,DIFFERENZ) > -1 ) then
                        MAX_Abspalt := 0;
                        while ForAll( Irr(ctN), x -> ScalarProduct(x,DIFFERENZ) > -1 ) do
                            MAX_Abspalt:=MAX_Abspalt+1;
```

```
                    DIFFERENZ := DIFFERENZ − GuteListe[a][3];
                od;

                counter:=0;
                repeat
                    StripErgebnis :=
                        StripOffOneCopyOfNFromMIfPossible(Restr,GuteListe[a][2]);
                    Restr := StripErgebnis[2];
                    if StripErgebnis[1]=1 then
                        Chi_N := Chi_N − GuteListe[a][3];
                        counter := counter + 1;
                        Print("Le Matrix in question ist : ");
                        Print(StripErgebnis[3]) ; Print("\n");
                    fi ;
                until StripErgebnis[1]=0 or counter > MAX_Abspalt;
                Add(temp_multiplicities, [a,counter]) ;
                # this means: first entry = position of this module in GuteListe;
                # second entry = multiplicity of this module
            fi ;
        od;

        # Remark:
        # ListPostitionsGuteListe is referring to NewListV_M_Chi_N (over Fp, including PIMs);
        # temp_multiplicities however is referring to 1.. Size(GuteListe)

        NumberOfPIMsInN := Size(Filtered(NewListV_M_Chi_N, x −> Size(x[1])<2 ));

        ListGoodModulesOverFpWithCorrectMultiplicities := [];

        temp_multiplicities_for_huge_List_V_M_Chi_N:=[];

        for hh in [1.. Size(NewListV_M_Chi_N)] do
            FLAGGE:=false;
            for jj in [1.. Size(ListPostitionsGuteListe)] do
                if hh=ListPostitionsGuteListe[jj] then # here, the vertex does contain a subgroup but we still have to
                # check if we add zeros or not
                    for ll in [1.. Size(temp_multiplicities)] do
                        if temp_multiplicities[ ll ][1] = jj then
                            Add(temp_multiplicities_for_huge_List_V_M_Chi_N,
                            temp_multiplicities[ ll ][2]) ;
                            FLAGGE:=true;
                        fi ;
                    od;
                fi ;
            od;
            if FLAGGE = false then
                Add(temp_multiplicities_for_huge_List_V_M_Chi_N,0); # still over Fp
            fi ;
        od;

        LISCHDEEE_ModulDIMs := List(NewListV_M_Chi_N, x −> x[2].dimension);

        if IsZero(temp_multiplicities_for_huge_List_V_M_Chi_N∗LISCHDEEE_ModulDIMs −
            DIM_after_boese_Lste ) then
            Print("test hat geklappt");
        else
            Print("FEHLER");
        fi ;

        # we define some of the modules in question over smaller fields :

DIM_per_Summand_Recent_Module_OverFqN := Size(alpha_aktuell) / NumberOfSummands_alpha_aktuell;

        if Size(Flat(alpha_aktuell))>0 then
            Fq_RecentModule := Field(Flat(alpha_aktuell));
        else
            Fq_RecentModule := GF(p);
        fi ;

    AllBasesGalConjugates_BOTH_PIMS_AND_GREEN_N := List([1..NumberOfPIMsInN], x −> []);

    CounterNumberSummandsOfThePIMsInN := List([1..NumberOfPIMsInN], x −> []);

    for tt in [1.. NumberOfPIMsInN] do
        for uu in [1.. Size(AllBasesGalConjugates_ONLY_THE_PIMS_N)] do
            if Size(AllBasesGalConjugates_ONLY_THE_PIMS_N[uu])>0 and
```

```
                        AllBasesGalConjugates_ONLY_THE_PIMS_N[uu][1] = tt then

                        AllBasesGalConjugates_BOTH_PIMS_AND_GREEN_N[tt] :=
                            ShallowCopy(AllBasesGalConjugates_ONLY_THE_PIMS_N[uu][2]);
                        CounterNumberSummandsOfThePIMsInN[tt] :=
                            ShallowCopy(AllBasesGalConjugates_ONLY_THE_PIMS_N[uu][4]);
                    fi;
                od;
            od;

            for vv in [1.. Size(CounterNumberSummandsOfThePIMsInN)] do
                if CounterNumberSummandsOfThePIMsInN[vv]=0 then
                    CounterNumberSummandsOfThePIMsInN[vv] := 1;
                fi;
            od;

            Append(AllBasesGalConjugates_BOTH_PIMS_AND_GREEN_N,
            AllBasesGalConjugatesForGreen_WITHOUT_PIMS_AT_THE_BEGINNING_N);

                    # example: let G=A4 and p= 2. Then, matrices of the underlying representations of the PIMs are given as follows:
                    # V.AllBasesGalConjugates_ONLY_PIMS_AT_THE_BEGINNING;
                    #[ [ 2, [ [ Z(2)^0, 0*Z(2), Z(2)^0, Z(2)^0, Z(2^2)^2, Z(2^2), Z(2^2), Z(2^2) ], [ 0*Z(2), Z(2)^0, Z(2)^0, 0*Z(2), Z(2)^0, Z(2)^0, Z(2)^0, Z(2^2) ], [ 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2), Z(2^2)^2, 0*Z(2), Z(2)^0, Z(2)^0 ],
                    #[ 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2), Z(2^2)^2, 0*Z(2), Z(2^2) ], [ Z(2)^0, Z(2)^0, Z(2)^0, 0*Z(2), Z(2)^0, Z(2)^0, Z(2)^0, Z(2^2) ], [ 0*Z(2), Z(2)^0, Z(2^2), Z(2)^0, 0*Z(2), Z(2)^0, Z(2)^0, Z(2^2)^2 ],
                    #[ 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2), Z(2)^0, Z(2^2)^2, 0*Z(2) ], [ 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0, Z(2^2), 0*Z(2), Z(2^2)^2, Z(2^2) ] ], GF(2^2), 2 ] ]

                    Fieldq1 := Fq_RecentModule;

                    if Size(Flat(AllBasesGalConjugates_BOTH_PIMS_AND_GREEN_N))>0 then
                        Fieldq2 := Field(Flat(AllBasesGalConjugates_BOTH_PIMS_AND_GREEN_N));
                    else
                        Fieldq2:=GF(p);
                    fi;

                    Fqmax := GF(Maximum([Size(Fieldq1),Size(Fieldq2)]));

                    NumberOfSummandsOverFqPerFpNjModuleToEnterInAuxProg := [];
                    Append(NumberOfSummandsOverFqPerFpNjModuleToEnterInAuxProg,
                    ShallowCopy(CounterNumberSummandsOfThePIMsInN));

                    Append(NumberOfSummandsOverFqPerFpNjModuleToEnterInAuxProg,
                    ShallowCopy(CounterNumberSummands_OF_ONLY_THE_Green_Correspondents_N));

                    ZZZZ := FromRestrictionToOffDiagonalEntries(NewListV_M_Chi_N, gensOfN,
                    RestrCopy, alpha_aktuell, NumberOfSummands_alpha_aktuell, Fqmax,
                    temp_multiplicities_for_huge_List_V_M_Chi_N,
                    NumberOfSummandsOverFqPerFpNjModuleToEnterInAuxProg,
                    AllBasesGalConjugates_BOTH_PIMS_AND_GREEN_N, p_prime_classes_recent);

                    Append(ListAllBrauerEvaluations,ZZZZ);

                fi;
            fi;
        fi;
    od;
  fi;
od;


# Next, we construct the ordinary characters of the trivial source modules using Rickard's formula;
# then, we save the whole triv. s. c. t. as a big matrix.


ListAllBrauerEvaluationsAsListOfBlockColumns := [];

SizeAuxiliary := Size(ListAllBrauerEvaluations)/SizeTSCT;

for i in [1.. SizeAuxiliary] do
    Add(ListAllBrauerEvaluationsAsListOfBlockColumns,
    ListAllBrauerEvaluations{[(i-1)*SizeTSCT+1..(i-1)*SizeTSCT+SizeTSCT]});
od;

TSCTMAT_As_List_With_n_Sublists:=[];
```

```
for i in [1.. SizeTSCT] do
    temp:=[];
    for j in [1.. SizeAuxiliary] do
        Append(temp, ListAllBrauerEvaluationsAsListOfBlockColumns[j][i]);
    od;
    Add(TSCTMAT__As__List__With__n__Sublists,temp);
od;

ListOrdinaryCharacterValuesOfAllTSModulesOverFq := [];

for s in [1.. SizeTSCT] do
    ListOrdinaryCharOfTSModuleNow := [];
    for x in ConjugacyClasses(ctG) do
        g := Representative(x);
        PPartAndPPrimePart := PPartAndPPrimePartOfGroupElement(g,p);
        ppart := PPartAndPPrimePart[1];
        pprimePart := PPartAndPPrimePart[2];
        PreliminaryVertexOfInterest := GroupByGenerators([ppart]);
        r := fail ;
        t := 0;
        while r = fail do
            t := t+1;
            PSubGroupNow := Subgroups__Pi[t];
            r := RepresentativeAction(G,Subgroups__Pi[t],PreliminaryVertexOfInterest);
            # if H := Subgroups__Pi[t] and K := PreliminaryVertexOfInterest are conjugate in G,
            # then we have H = r*K*r^-1.
        od;

        FinalVertexPosition := ShallowCopy(t);
        PreliminaryPPrimePartOfInterest := r*pprimePart*r^-1;
        # conjugating the old p'-part into the correct p-subgroup of G

        N:=List__Normalisers[FinalVertexPosition];

        u := 0;
        flag := false;
        while flag = false do
            u := u+1;
            if IsConjugate(N,List__All__p__prime__Classes[FinalVertexPosition][u],
                PreliminaryPPrimePartOfInterest) then
                flag := true;
            fi ;
        od;
        Add(ListOrdinaryCharOfTSModuleNow, ListAllBrauerEvaluationsAsListOfBlockColumns[t][s][u]);
    od;
    Add(ListOrdinaryCharacterValuesOfAllTSModulesOverFq, ListOrdinaryCharOfTSModuleNow);
od;

if IdGroupsAvailable(Order(G)) then
    IdentifyingG:=IdSmallGroup(G);
else
    IdentifyingG:=["could not identify G !!!"];
fi ;

OrdinaryClasses := List(ConjugacyClasses(ctG), x -> Representative(x));
OrdinaryClasses__Names := ClassNames(ctG);

OrdinaryCTAsListOfLists:=[];
for m in [1.. Size(Irr(ctG))] do
    Add(OrdinaryCTAsListOfLists, ShallowCopy(Irr(ctG)[m]));
od;

ListOrdinaryCharsAllTSModules := List(ListOrdinaryCharacterValuesOfAllTSModulesOverFq, x -> ClassFunction(ctG,x));

ScalProdsTSModules:=[];
for m in [1.. Size(ListOrdinaryCharsAllTSModules)] do
    Add(ScalProdsTSModules,Flat(MatScalarProducts(ctG,[ListOrdinaryCharsAllTSModules[m]],Irr(ctG))));
od;

MyTSCTRecord := rec();

MyTSCTRecord.G := G;
MyTSCTRecord.gensG := gensG;
MyTSCTRecord.OrderG := Order(G);
MyTSCTRecord.IdentifyingG := IdentifyingG;
MyTSCTRecord.Gesamtliste__Characters__Chi__All := ListOrdinaryCharsAllTSModules;
MyTSCTRecord.TSCTMAT__As__List__With__n__Sublists := TSCTMAT__As__List__With__n__Sublists;
```

```
    MyTSCTRecord.Field := k;
    MyTSCTRecord.Characteristic := Characteristic(k);
    MyTSCTRecord.List__All__p__prime__Classes := List__All__p__prime__Classes;
    MyTSCTRecord.Pis := Subgroups__Pi;
    MyTSCTRecord.Nis := List__Normalisers;

    MyTSCTRecord.OrdinaryClasses := OrdinaryClasses;
    MyTSCTRecord.OrdinaryClasses__Names := OrdinaryClasses__Names;
    MyTSCTRecord.OrdinaryCTAsListOfLists := OrdinaryCTAsListOfLists;
    MyTSCTRecord.ScalProdsTSModules := ScalProdsTSModules;
    MyTSCTRecord.List__All__p__prime__Classes__Names := List__All__p__prime__Classes__Names;

    # we insert two tests here:
    # 1) Do the degrees of the ordinary characters of the t.s. modules coincide with the degrees of
    # the underlying modular representations of the triv. s. modules?
    # 2) Is Triv__p(G) invertible?

    cc:=0;

    dd:=SizeTSCT;

    for ee in [1.. dd] do
        if not IsZero(TSCTMAT__As__List__With__n__Sublists[ee][1] − ListOrdinaryCharsAllTSModules[ee][1]) then
            cc:=cc+1;
        fi;
    od;

    if not IsZero(cc) then
Print("The computed character degrees do probably not coincide with the dimensions of the trivial source modules.");
        return(fail);
    fi;


    if IsZero(Determinant(TSCTMAT__As__List__With__n__Sublists)) then
        Print("The determinant is equal to zero. Hence, the trivial source characters are linearly dependend.",
        "Thus, there is a mistake somewhere.");
        return(fail);
    fi;

    FlatScalProds := Flat(ScalProdsTSModules);
    FlagInt := true;

    for i in FlatScalProds do
        if not IsInt(i) then
            FlagInt := false;
        fi;
    od;

    if FlagInt=false then
        return("ERROR!");
    fi;

    return MyTSCTRecord;

end;

###########################

# We now define an auxiliary function that converts the computed
# entries of the t.s.c.t., as well as the characters and the conjugacy classes
# from GAP to tex

TSCTDataToTexFile:=function(G,p,file)
# Here, 'file' is the name of a file given as a string, e.g. "SmallGroup__36__3__p__is__3.tex".
# The output of the function will be saved in this file.

    local f, TSCT__Record, Classes__OrdinaryTable, OrdinaryTableAsList, m, i,
    beginning__of__chi__as__string, j, ClassesNamesOfctG;

    if IsString( file ) then f := OutputTextFile(file,false);
        SetPrintFormattingStatus(f,false);
    else
        f:= file ;
    fi;

    TSCT__Record := TSCTFq(G,p);
```

```
Classes_OrdinaryTable := TSCT_Record.OrdinaryClasses;

ClassesNamesOfctG := TSCT_Record.OrdinaryClasses_Names;

OrdinaryTableAsList := TSCT_Record.OrdinaryCTAsListOfLists;

m := Size(Classes_OrdinaryTable);

# We now start with the LaTeX-file.

PrintTo(f, "\\documentclass[varwidth=\\maxdimen,border=10]{standalone}\n");
PrintTo(f, "\\begin{document}\n");
PrintTo(f, "The group \$G\$ is isomorphic to the group labelled by\\ ");
PrintTo(f, TSCT_Record.IdentifyingG); PrintTo(f, "\\ ");
PrintTo(f, "in the Small Groups library."); PrintTo(f,"\\");PrintTo(f,"\\"); PrintTo(f, "\n");
PrintTo(f, "Ordinary character table of "); PrintTo(f, "\$G\$");
PrintTo(f, "\\ \$\\cong\$\\ "); PrintTo(f, StructureDescription(G)); PrintTo(f, ":");
PrintTo(f,"\\");PrintTo(f,"\\"); PrintTo(f, "\n");

# Now we generate a LaTex-code for the ordinary character table of G.

PrintTo(f, "\\begin{center}");PrintTo(f, "\n");

PrintTo(f, "\\begin{tabular}{\@{}l\@{}l\@{}l\@{}");
PrintTo(f, "}\n");

PrintTo(f,    "\\hline\n");

PrintTo(f, "\\(\\begin{array}{|l");
PrintTo(f, "|");
PrintTo(f, ListWithIdenticalEntries(m, 'c'));
PrintTo(f, "|}\n");
PrintTo(f, " ");

for i in [1.. m] do
   PrintTo(f, " & ");
   PrintTo(f, ClassesNamesOfctG[i]);
od;
PrintTo(f,    "\\\\ \\hline\n");

beginning_of_chi_as_string:="\\chi_{";

for i in [1.. m] do
    PrintTo(f, Concatenation(beginning_of_chi_as_string,String(i),"}"));
    for j in [1.. m] do
        PrintTo(f, " & ");
        PrintTo(f, GAPStringToTex(String(OrdinaryTableAsList[i][j])));
    od;
    PrintTo(f, "\\\\"); PrintTo(f, "\n");
od;

PrintTo(f,    "\\hline\n");
PrintTo(f,"\\end{array}\\)\\\\");
PrintTo(f, "\n");
PrintTo(f,"\\end{tabular}\n");

PrintTo(f, "\\end{center}"); PrintTo(f, "\n");

# Next, we collect some data for the creation of the t.s.c.t., the p'-classes, the normalisers, etc.

Pis := TSCT_Record.Pis;
Nis := TSCT_Record.Nis;

p := TSCT_Record.Characteristic;

List_All_p_prime_Classes := TSCT_Record.List_All_p_prime_Classes;

TSCTMAT_As_List_With_n_Sublists := TSCT_Record.TSCTMAT_As_List_With_n_Sublists;

PrintTo(f, "\\begin{tabular}{\@{}l\@{}l\@{}l\@{}");
stri := "l\@{}";
len:=1+Size(Pis)+Size(Pis);
li := ListWithIdenticalEntries(len, stri);

PrintTo(f, Concatenation(li));
PrintTo(f, "}\n");
```

```
PrintTo(f, "Trivial source character table of "); PrintTo(f, "\$G\$");
PrintTo(f, "\\ \$\\cong\$\\ "); PrintTo(f, StructureDescription(G)); PrintTo(f, " at");
PrintTo(f, "\\ \$p="); PrintTo(f, p); PrintTo(f, "\$:");
PrintTo(f, "\\\\"); PrintTo(f, "\n");
PrintTo(f, "\\(\\begin{array}{|l");

for i in [1.. Size(List_All_p_prime_Classes)] do
    j:=Size(List_All_p_prime_Classes[i]);
    PrintTo(f, "|");
    PrintTo(f, ListWithIdenticalEntries(j, 'c'));
od;
PrintTo(f, "|}\n");

List_ccls_flat:=Flat(List_All_p_prime_Classes);

PrintTo(f,    "\\hline\n");
PrintTo(f, "\\textup{Normalisers}\\ N_i");

for i in [1.. Size(List_All_p_prime_Classes)] do
    PrintTo(f, " & ");
    PrintTo(f, "\\multicolumn{"); PrintTo(f, Size(List_All_p_prime_Classes[i]));
    PrintTo(f, "}{c|}{"); PrintTo(f,Concatenation("N_{",String(i),"}"));PrintTo(f,"}");
od;

PrintTo(f,    "\\\\ \\hline\n");

PrintTo(f, "p\\textup{-subgroups\\ of\\ } G\\ \\textup{up\\ to\\ conjugacy\\ in\\ } G");
for i in [1.. Size(List_All_p_prime_Classes)] do
    PrintTo(f, " & ");
    PrintTo(f, "\\multicolumn{"); PrintTo(f, Size(List_All_p_prime_Classes[i]));
    PrintTo(f, "}{c|}{"); PrintTo(f,Concatenation("P_{",String(i),"}"));PrintTo(f,"}");
od;
PrintTo(f,    "\\\\ \\hline\n");

List_Representatives_Names := [];
for b in [1.. Size(TSCT_Record.List_All_p_prime_Classes_Names)] do
    Append(List_Representatives_Names,TSCT_Record.List_All_p_prime_Classes_Names[b]);
od;

PrintTo(f, "\\textup{Representatives}\\ n_j\\ \\in\\ N_i");
for i in [1.. Size(List_ccls_flat)] do
    PrintTo(f, " & ");
    PrintTo(f, List_Representatives_Names[i]);
od;
PrintTo(f,    "\\\\ \\hline\n");

nsq:=Size(Flat(TSCTMAT_As_List_With_n_Sublists));

if IsSquareInt(nsq) then
    sqrt:=RootInt(nsq);
else
    Print("The number of the matrix entries is not a square!");
    return(fail);
fi;

List_numbers_for_grid:=[];

u:=0;
for i in [1.. Size(List_All_p_prime_Classes)] do
    j:=Size(List_All_p_prime_Classes[i]);
    Append(List_numbers_for_grid,[u+j]);
    u:=u+j;
od;

Remove(List_numbers_for_grid);;

my_chi_string:="";

beginning_of_chi_as_string:="\\chi_{";

plus_as_string:="+";

times_as_string:="\\cdot\ ";

new_string_lists_for_characters:=[];

for i in [1.. sqrt] do
```

```
        Append(new_string_lists_for_characters,[[]]);
    od;

    The_TS_characters := TSCT_Record.ScalProdsTSModules;

    for i in [1.. Size(The_TS_characters)] do
        for j in [1.. Size(The_TS_characters[i])] do
            my_chi_string:= Concatenation(my_chi_string,"{");
            my_chi_string:= Concatenation(my_chi_string,String(The_TS_characters[i][j]));
            my_chi_string:= Concatenation(my_chi_string,"}");
            my_chi_string:= Concatenation(my_chi_string, times_as_string);
            my_chi_string:= Concatenation(my_chi_string, beginning_of_chi_as_string);
            my_chi_string:= Concatenation(my_chi_string, String(j));
            my_chi_string:= Concatenation(my_chi_string, "}");
            Append(new_string_lists_for_characters[i],[my_chi_string]);
            my_chi_string:="";
        od;
    od;

    The_TS_characters_new:=[];

    String_TS_Char:="";

for i in [1.. Size(new_string_lists_for_characters)] do
       for j in [1.. Size(new_string_lists_for_characters[i])] do
               String_TS_Char:=Concatenation(String_TS_Char, new_string_lists_for_characters[i][j]);
               if j < Size(new_string_lists_for_characters[i]) then
                       String_TS_Char:=Concatenation(String_TS_Char, plus_as_string);
               fi ;
       od;
       Append(The_TS_characters_new,[String_TS_Char]);
       String_TS_Char:="";
od;

for i in [1.. sqrt] do
       PrintTo( f, The_TS_characters_new[i]);
       for j in [1.. sqrt] do
               PrintTo( f, " & ");
               PrintTo( f, GAPStringToTex(String(Flat(TSCTMAT_As_List_With_n_Sublists)[(i-1)*sqrt + j])));
       od;
       PrintTo(f,    "\\\\");
       PrintTo(f, "\n");
       if i in List_numbers_for_grid then
               PrintTo(f,    " \\hline\n");
       fi ;
od;

    PrintTo(f,    "\\hline\n\n");
    PrintTo(f,"\\end{array}\\)\\\\");
    PrintTo(f, "\n");

    PrintTo(f,"\\ \\\\"); PrintTo(f, "\n");

    The_groups_P_i:=TSCT_Record.Pis;
    The_groups_N_i:=TSCT_Record.Nis;

    for i in [1.. Size(The_groups_P_i)] do
        PrintTo(f,"\\ \\\\"); PrintTo(f, "\n");
        my_P_i_string:= Concatenation("\$P","_","{",String(i),"}");
        PrintTo(f, my_P_i_string);
        PrintTo(f,"\ =\ ");
        PrintTo(f, The_groups_P_i[i]);
        PrintTo(f,"\\cong\$\ ");
        PrintTo(f, StructureDescription(The_groups_P_i[i]));
    od;

    PrintTo(f,"\\ \\\\"); PrintTo(f, "\n");

    for i in [1.. Size(The_groups_N_i)] do
        PrintTo(f,"\\ \\\\"); PrintTo(f, "\n");
        my_N_i_string:= Concatenation("\$N","_","{",String(i),"}");
        PrintTo(f, my_N_i_string);
        PrintTo(f,"\ =\ ");
        PrintTo(f, The_groups_N_i[i]);
        PrintTo(f,"\\cong\$\ ");
        PrintTo(f, StructureDescription(The_groups_N_i[i]));
    od;
```

```
    PrintTo(f,"\\end{tabular}\n");

    PrintTo(f,"\\end{document}\n");

    if IsString( file ) then
        CloseStream(f);
    fi;
    return(TSCT_Record);
end;






# The function TSCT_pdf_producer has as input a positive integer t and
# produces the trivial source character tables (pdf's) of all small groups
# with order t at once.

TSCT_pdf_producer := function(t)

    local StrFolder, ALL, primedivs, i, iso, G, IIDD, p, file, V, STR, z;

    ChangeDirectoryCurrent("/home/bernhard");

    StrFolder:=Concatenation("mkdir ",String(t));
    Exec(StrFolder);
    ChangeDirectoryCurrent(Concatenation("/home/bernhard","/",String(t)));

    ALL:=AllSmallGroups(t);
    primedivs:=PrimeDivisors(t);
    for i in ALL do
        iso:=IsomorphismPermGroup(i);
        G:=Image(iso);
        IIDD:=IdSmallGroup(G); # a list with two entries
        for p in primedivs do
            file:=Concatenation("SmallGroup","_",String(IIDD[1]),"_",String(IIDD[2]),
            "_for_the_prime_",String(p),".tex");
            ChangeDirectoryCurrent(Concatenation("/home/bernhard","/",String(t)));
            V:=TSCTDataToTexFile(G,p,file);
        od;
    od;

    ChangeDirectoryCurrent(Concatenation("/home/bernhard","/",String(t)));
    STR:=Concatenation("for i in ","/home/bernhard","/",String(t),"/","\*",".tex; do pdflatex ",
    "\"","\$","i","\"",";done");

    Exec(STR);

    z:=DirectoryCurrent();

    return(z);
end;

#################################################################
# Example:
#################################################################


#   p:=11;
#   GG:=SmallGroup(22,2);
#   iso:=IsomorphismPermGroup(GG);
#   G:=Image(iso);
#
#   IIDD:=IdSmallGroup(G);
#
#   # Creation of the tex-file
#   file:=Concatenation("SmallGroup","_",String(IIDD[1]),"_",String(IIDD[2]),"_for_the_prime_",String(p),".tex");
#
#
#   # p:=7;
#   # G:=AtlasGroup("L2(8).3");
#   # IIDD:=IdSmallGroup(G);
#   # file:=Concatenation("L2(8)_DOT_3","_for_the_prime_",String(p),".tex");
#
#
```

```
#    # Producing the pdf-file of the trivial source character table
#    V:=TSCTDataToTexFile(G,p,file);
```

7.3 A MAGMA algorithm for the computation of trivial source character tables

```
// The program that finds the vertices is taken from R. Zimmermann's
// PhD thesis, see R. Zimmermann, Vertizes einfacher Moduln Symmetrischer
// Gruppen, PhD thesis (German), University of Jena, Jena, 2004.
// Cf. https://users.fmi.uni-jena.de/~susanned/vertex.html


RTEx := function(endos,delta,d,F,A,B)
      rt := RightTransversal(A,B);
      tr := [];
      for f in endos do Append(~tr,ScalarMatrix(d,Zero(F))); end for;
      for g in rt do
           bild := delta(g);
           bildi := bild^-1;
           for i:=1 to #tr do
                tr[i] +:= bildi*endos[i]*bild;
           end for;
           end for;
      return tr;
end function;

// The following function computes the image of the trace map Tr^G_H of M.
RelTr := function(endos,M,sgl)
      F := BaseRing(M);
      d := Dimension(M);
      delta := Representation(M);
      s := #sgl;
      while s gt 1 do
       bilder := RTEx(endos,delta,d,F,sgl[s-1],sgl[s]);
       endos := [];
       for phi in bilder do
            if phi ne ScalarMatrix(d,Zero(F))
                    then Append(~endos,phi);
                    end if;
                 end for;
            if endos eq [] then return endos; end if;
            s -:= 1;
            end while;
      return endos;
end function;

// This function tests whether the G-module M is relatively H-projective
IsProjective := function(M,H,sgl) // sgl is a descending chain of subgroups, starting with G
      d := Dimension(M);
basis := Basis(EndomorphismAlgebra(Restriction(M,H)));
      for i in RelTr(basis,M,Append(sgl,H)) do
           if Rank(i) eq d then return true; end if;
           end for;
      return false;
end function;

// The following function determines a set of representatives for the G-conjugacy classes
// of the subgroups that are contained in the elements of the list sub
CCReps := function(sub,G)
    groups := [];
    for i in sub do
       H := i`subgroup;
       need := true;
       for K in groups do
            if IsConjugate(G,H,K) then need:=false; break; end if;
       end for;
       if need then Append(~groups,H); end if;
    end for;
    return groups;
end function;

// The next function computes a minimal subgroup V of H w.r.t. the property that
// M is V-projective and |V| is greater than or equal to min
function Vx(M,G,H,sgl,min) // sgl is a descending chain of subgroups of G, starting with G
    if #H gt min then
       Append(~sgl,H);
       for K in CCReps(MaximalSubgroups(H),G) do
           if IsProjective(M,K,sgl) then return Vx(M,G,K,sgl,min); end if;
```

```
            end for;
        end if;
    return H;
end function;

// This function computes a vertex of M, if M is H−projective and indecomposable
function VxStart(M, H)
    G := Group(M);
    ssyl := #SylowSubgroup(H,Characteristic(BaseRing(M)));
    V := SymmetricGroup(1);
    checked := [];
    for U in IndecomposableSummands(Restriction(M,H)) do
         need := true;
         for W in checked do
              if IsIsomorphic(U,W) then need:=false; break; end if;
              end for;
         if need then
             Append(~checked,U);
             VU := Vx(U,H,H,[],Maximum(#V,ssyl/Gcd(ssyl,Dimension(U))));
             if VU eq H then
                 print("The vertex is: ");
                 return H;
                 end if;
             if #VU gt #V then
                 V:=VU;
                 print("New lower bound: ");
                 print(V);
                 print("\n");
                 end if;
             end if;
         end for;
    print("The vertex is: ");
    return V;
end function;

// This short program tests, whether a vertex of the entered indecomposable KN−module M
// contains (a conjugate in N of) Q. K is assumed to be a splitting field for N and all
// of its subgroups.
DoesVxContainQ:= function(M,Q);
N := Group(M);
Vtx:=VxStart(M,N);
ccsSubgroups := [s`subgroup : s in Subgroups(Vtx)];
ccsSubgroups := [H : H in ccsSubgroups | Order(H) eq Order(Q)];

ccsSubgroups_as_Subgroups_of_N:=[];

for i in [1..#ccsSubgroups] do
     U := ccsSubgroups[i];
     U_in_N:=sub<N|[U.i: i in [1..Ngens(U)]]>;
     Append(~ccsSubgroups_as_Subgroups_of_N,U_in_N);
end for;

Q_in_N := sub<N|[Q.i: i in [1..Ngens(Q)]]>;

flag := false;

// We test if Q is N−conjugate to a subgroup of the vertex of M.
// We only consider subgroups of N with the same order as Q.
for i in [1..#ccsSubgroups] do
     flag := IsConjugate(N,ccsSubgroups_as_Subgroups_of_N[i], Q_in_N);
          if flag ne false then
          return flag;
     end if;
end for;

return flag;

end function;


// The program PermutationModulesForSylow first computes a list of subgroups
// of P up to conjugacy in P...then, for every such subgroup Q, the induced module
// k_Q^P is computed and added to the list PermModules.
// The list PermModules is returned.
// We remark that the present program is later used in the case that P is a
// Sylow p−subgroup of the group G in question.

```

```
PermutationModulesForSylow:= function(P,K)
    ccsSyl:= [s`subgroup : s in Subgroups(P)];
    ccsSyl_without_Syl:=[Q: Q in ccsSyl | Order(Q) lt Order(P)];
    temp:=[];
    while #ccsSyl_without_Syl gt 0 do
        u:=ccsSyl_without_Syl[1];
        Append(~temp,u);
        inter:=[Q : Q in ccsSyl_without_Syl | IsConjugate(P, Q, u)];
        ccsSyl_without_Syl:=[Q : Q in ccsSyl_without_Syl | Q notin inter];
    end while;
    Append(~temp,P);
    temp_sizes:=[];
    for i in [1..#temp] do
        Append(~temp_sizes,#temp[i]);
    end for;
    ParallelSort(~temp_sizes,~temp);
    PermModules:=[];
    for i in [1..#temp] do
        T:=TrivialModule(temp[i],K);
        I:=Induction(T,P);
        Append(~PermModules,I);
    end for;
    return PermModules;
end function;


// The following is the main program. It computes a list with many entries.
// The eighth entry gives Triv_p(G) as a matrix.
// The tenth entry gives a list whose entries are of the form
// [vertex of the trivial source module M, module M, ordinary character of M]
// The fourth entry gives the list of all p'-conjugacy classes of all
// normaliser quotients.
// The other entries do not play an important role. They were used in order
// to convert the computed data into a tex file.

OrdinaryCharactersAndProjectiveNBarModules:= function(G,p)
    m:=Exponent(G);
    K:=GF(p);
    Kx<x>:=PolynomialRing(K);
    f:=x^m -1;
    L:=SplittingField(f);
// By a theorem of Brauer, L is a splitting field for G. From the definition of the exponent of
// a group we see that it is also a splitting field for all subgroups of G.
    u:=#L;
    K:=GF(u);
// We work over K (and not L) to be sure that we work over a Galois field where the elements
// are consistent with the MAGMA command BrauerCharacter.
    R:=CyclotomicField(m: Sparse := true);
    ct:=CharacterTable(G);
    IrrG:=[];
    for j in [1..#ct] do
        Append(~IrrG, ct[j]);
    end for;
    PIMs_G:=ProjectiveIndecomposableModules(G,K);
    CompleteList_V_M_Chi:=[**];
    Syl:=SylowSubgroup(G,p);
    ccsSyl:= [s`subgroup : s in Subgroups(Syl)];
    ccsSyl_without_Syl:=[P: P in ccsSyl | Order(P) lt Order(Syl)];
    temp:=[];
    while #ccsSyl_without_Syl gt 0 do
        u:=ccsSyl_without_Syl[1];
        Append(~temp,u);
        inter:=[P : P in ccsSyl_without_Syl | IsConjugate(G, P, u)];
        ccsSyl_without_Syl:=[P : P in ccsSyl_without_Syl | P notin inter];
    end while;
    Append(~temp,Syl);
    temp_sizes:=[];
    for i in [1..#temp] do
        Append(~temp_sizes,#temp[i]);
    end for;
    ParallelSort(~temp_sizes,~temp);

//Now we produce the partial subgroup lattice :
    List_For_Partial_Subgroup_Lattice:=[];
    for i in [1..#temp] do
        MOG:=MinimalOvergroups(G,temp[i]);
        for j in [1..#temp] do
```

```
            if temp[j] in MOG then
                Append(~List_For_Partial_Subgroup_Lattice,[i,j]);
            end if;
        end for;
    end for;

    PMS:=PermutationModulesForSylow(Syl,K);

// At first , we look at the indec. projective kG-modules:
    ccs_G:=[c[3] : c in Classes(G)];
    TG:=sub<G|[]>;
    List_Chi_Proj:=[];
    for m in [1..#PIMs_G] do
        cfs:=CompositionFactors(PIMs_G[m]);
        list_br :=[];
        for a in cfs do
            Append(~list_br,BrauerCharacter(a));
        end for;
        chi:=&+list_br;
        Append(~List_Chi_Proj, chi);
        print("MEIN AKTUELLER BRAUERCHARACTER IST"); print(chi);
    end for;
    for m in [1..#PIMs_G] do
        Append(~CompleteList_V_M_Chi, [*TG, PIMs_G[m], List_Chi_Proj[m]*]);
    end for;

// We keep track of all the p'-conjugacy classes:
    List_All_p_prime_Classes:=[**];
    Reps_ccs_N_bar:=[**];
    List_All_p_prime_Classes_Sizes:=[**];
    for i in [1..#ccs_G] do
        if not (IsZero(Order(ccs_G[i]) mod p)) then
                Append(~List_All_p_prime_Classes,ccs_G[i]);
                Append(~Reps_ccs_N_bar,ccs_G[i]);
        end if;
    end for;
    Append(~List_All_p_prime_Classes_Sizes,#List_All_p_prime_Classes);

List_Normalizers:=[];
Append(~List_Normalizers,G);

// Now, we look at all the other indecomposable trivial source kG-modules:
for H in temp do
    if #H gt 1 then
        N:=Normaliser(G,H); Append(~List_Normalizers,N);
        H_in_N:=sub<N|H>;
        FAC,f:=quo<N|H_in_N>;
        Ker:=Kernel(f);
        PIMs:=ProjectiveIndecomposableModules(FAC,K);
        s:=#PIMs;
        ccs_N_bar:=[c[3] : c in Classes(FAC)];
        w:=0;
        for i in ccs_N_bar do
            if not (IsZero(Order(i) mod p)) then
// Note that o(n_bar) coprime to p does not always imply that o(n) is coprime to p.
// Thus, we first search for a representative of N with this property.
                Append(~Reps_ccs_N_bar,i);
                RNK:=Ker * i@@f;
                ELS_RNK:=[x: x in N | x in RNK];
                P_PRIME_ORDER:=[y: y in ELS_RNK | not IsZero(Order(y) mod p)];
                Append(~List_All_p_prime_Classes,P_PRIME_ORDER[1]);
                w:=w+1;
            end if;
        end for;
        Append(~List_All_p_prime_Classes_Sizes,w);
        ccs_N:=[c[3] : c in Classes(N)];
        Brauer_Characters_N_bar:=[];
        Brauer_Characters_N:=[];
        Vertices_and_Brauer_Characters_Induced_Modules_G:=[];
        Brauer_Characters_TS_Modules_G:=[];
        List_Chi_Proj:=[];
        for m in [1..#PIMs] do
            cfs:=CompositionFactors(PIMs[m]);
            list_br :=[];
            for a in cfs do
                Append(~list_br,BrauerCharacter(a));
            end for;
```

```
            chi:=&+list_br;
// chi is treated as a class function by MAGMA with values set to be 0 for p−elements.
// Thus, this is already the correct ordinary character!
            Append(~List_Chi_Proj, chi);
            print("List_chi_proj_PIMs_FAC ist im Moment: "); print(List_Chi_Proj);
        end for;
        Inflated_Characters:=[];
        Inflated_Modules:=[];
        for i in [1.. s] do
            Append(~Inflated_Characters,LiftCharacter(List_Chi_Proj[i], f, N));
            phi:=Representation(PIMs[i]);
                u:=f*phi;
                M:=GModule(N,[u(N.i): i in [1..Ngens(N)]]);
            Append(~Inflated_Modules,M);
        end for;
        q:=0;
        ccs_N_p_prime:=[y: y in [1..#ccs_N] | not IsZero(Order(ccs_N[y]) mod p)];

        print("The inflated characters are:"); print(Inflated_Characters); print("Now without test!");

        Induced_Characters:=[];
        Induced_Modules:=[];

// Next, we compute the induced modules and the induced characters ...
// ... after that, we compute the trivial source kG−modules; this is
// achieved by viewing them as Green correspondents and
// using the MAGMA command SummandIsomorphism in order to determine a
// direct sum decomposition of the kG−modules induced from the normaliser N.
// Note that the MAGMA command SummandIsomorphism is the implemented
// version of the pseudocode from the article of Brooksbank and Luks.

        CompleteList_V_M_Chi_copy:= CompleteList_V_M_Chi;
        print("CompleteList_V_M_Chi_copy ist jetzt gleichA");
        print(CompleteList_V_M_Chi_copy);
        for i in [1.. s] do
            Chi_N:=Inflated_Characters[i];
            Chi_G:=Induction(Chi_N,G);
            M_N:=Inflated_Modules[i];
            M_G:=Induction(M_N,G);
            M_G_neu:=Induction(M_N,G);
            q:=0;
            for j in [1..#CompleteList_V_M_Chi] do
                L:=CompleteList_V_M_Chi[j][2];
                DIM_Summand:=Dimension(L);
                while DIM_Summand gt 0 do
                    if forall{ ch: ch in IrrG | InnerProduct(ch, Chi_G − CompleteList_V_M_Chi[j][3]) gt −1} then
                        SumIso, V, f := SummandIsomorphism(L,M_G_neu);
                        if Dimension(V) gt 0 then
                            SUB:=sub<M_G_neu|V>;
                            A,C:=HasComplement(M_G_neu,SUB);
                            M_G_neu:=C;
                            Chi_G:=Chi_G − CompleteList_V_M_Chi[j][3];
                            q:=q+1;
                        else
                            DIM_Summand:=0;
                        end if;
                    else
                        DIM_Summand:=0;
                    end if;
                end while;
            end for;

            print("Das Herausfinden von Chi hat geklappt fuer PIM Nummer"); print(i);
            print("und der geliftete Character is"); print(Chi_G);
            print("und q ist gleich"); print(q);

            Append(~CompleteList_V_M_Chi_copy, [*H, M_G_neu, Chi_G*]);
            print("CompleteList_V_M_Chi_copy ist jetzt gleichB"); print(CompleteList_V_M_Chi_copy);

        end for;
        CompleteList_V_M_Chi:= CompleteList_V_M_Chi_copy;
    end if;
end for;
print("Die endgueltige Groesse der Liste CompleteList_V_M_Chi ist "); print(#CompleteList_V_M_Chi);

// Now we compute all the remaining entries of the trivial source character table.
// At first, we produce a list whose entries are representatives of the p'−conjugacy classes
```

```
// of N_bar for all N.
List_All_p_prime_Classes_NOT_FLAT:=[];
j:=0;
for n in [1..#List_Normalizers] do
    temp95:=[];
    for i in [1.. List_All_p_prime_Classes_Sizes[n]] do
        Append(~temp95,List_All_p_prime_Classes[j+i]);
    end for;
    Append(~List_All_p_prime_Classes_NOT_FLAT,temp95);
    j:=j+List_All_p_prime_Classes_Sizes[n];
end for;


MATRIX_gesamt:=[];
for n in [1..#List_Normalizers] do
    N:=List_Normalizers[n];
    H:=temp[n];
    H_in_N:=sub<N|[H.i: i in [1..Ngens(H)]]>;
    Q_in_Syl:=sub<Syl|[H.i: i in [1..Ngens(H)]]>;
    MATRIX:=[];
    for j in CompleteList_V_M_Chi do
// Here, we search for a p-permutation basis of M with respect to our Sylow p-subgroup Syl.
// We restrict M to Syl and check which summands of PMS are IsIsomorphic
// to which summands of the restriction of M to Syl.
        res_M_Syl:=Restriction(j[2],Syl);
        dir_res_M_Syl:=DirectSumDecomposition(res_M_Syl);
        temp_M_neu:=[];
        for r in [1..#dir_res_M_Syl] do
            for s in PMS do
                if IsIsomorphic(s,dir_res_M_Syl[r]) then
                    Append(~temp_M_neu,s);
                end if;
            end for;
        end for;
        M_neu:=DirectSum(temp_M_neu);
        dimM_neu:=Dimension(M_neu);
        repM_neu:=Representation(M_neu);
        elsQ:=[x:x in Q_in_Syl];
        Positions_Of_Fixed_Points:=[];
        for i in [1.. dimM_neu] do
            if forall{ q: q in elsQ | IsZero(1-repM_neu(q)[i][i])} then
                Append(~Positions_Of_Fixed_Points,i);
            end if;
        end for;
        DIM_Fix := #Positions_Of_Fixed_Points;
        print("Die Groesse der Fixpunktbasis ist"); print(DIM_Fix);

        elsH:=[x:x in H_in_N];


// temp_Brauer collects the modules whose Brauer characters get summed to the entries
// in the t.s.c.t.
// We have to find out which modules (i.e.: direct summands of the restriction to N)
// can be discarded.
        MATRIX_i:=[**];
        j_N:=Restriction(j[2],N); print("j_N ist:"); print(j_N);
        print("Der Brauercharakter von j_N ist:");
        cfs:=CompositionFactors(j_N);
        list_br:=[];
        for a in cfs do
            Append(~list_br,BrauerCharacter(a));
        end for;
        psi_j_N:=&+list_br;
        print(psi_j_N);

        temp_Brauer:=[];
        if IsZero(Dimension(j_N)-DIM_Fix) then
// in this case, the dimension of the Brauer construction is equal to the number
// of fixed points;
// hence, every direct summand has to be taken into account in this case
            temp_Brauer:=DirectSumDecomposition(j_N);
        else
            dir:=DirectSumDecomposition(j_N);
            for s in dir do
                DIM_s:=Dimension(s);
                if DIM_s lt 1 + DIM_Fix then
                    Res_s_H:=Restriction(s,H_in_N);
```

```
                    if IsZero(1−#Group(Res_s_H)) then
                        Append(~temp_Brauer,s);
                    else // Now, use the Proof of Lemma 4.10.11 in the book of Lux and Pahlings
                        dir_Res_s_H:=DirectSumDecomposition(Res_s_H);
            if forall{ i: i in dir_Res_s_H | IsIsomorphic(i,TrivialModule(H_in_N,K))} then
                            Append(~temp_Brauer,s);
                        end if;
                    end if;
                end if;
            end for;
        end if;

        print("ccs_N_bar ist jetzt bei der Eischränkung gleich"); print(ccs_N_bar);
        for i in List_All_p_prime_Classes_NOT_FLAT[n] do
            if #temp_Brauer gt 1 then
                    j_neu:=DirectSum(temp_Brauer);
                    Append(~MATRIX_i,[*BrauerCharacter(j_neu)(i)*]);
                    print("P_PRIME_ORDER ist in diesem FALL1:"); print(i);
            elif #temp_Brauer eq 1 then
                    j_neu:=temp_Brauer[1];
                    Append(~MATRIX_i,[*BrauerCharacter(j_neu)(i)*]);
                    print("P_PRIME_ORDER ist in diesem FALL2:"); print(i);
            else
                    Append(~MATRIX_i,[*0*]); print("hier hat Fall 3 eine 0 angefügt");
            end if;
        end for;

        print("MATRIX_i ist gerade gleich"); print(MATRIX_i); print("!");
        MATRIX_i_new:=[MATRIX_i[j][1]: j in [1..#MATRIX_i]];
        print("MATRIX_i_new sieht so aus:"); print(MATRIX_i_new);
        MATRIX_i_new_as_Matrix:=Matrix(R, [MATRIX_i_new]);
        Append(~MATRIX, MATRIX_i_new_as_Matrix);
    end for;
    Append(~MATRIX_gesamt, [*VerticalJoin(MATRIX)*]);
end for;

print("MATRIX_gesamt sieht nun so aus:"); print(MATRIX_gesamt);

// With this, we obtained all block column matrices. We put them together in a list.

MATRIX_gesamt_new:=[* MATRIX_gesamt[j][1] : j in [1..#MATRIX_gesamt] *];

b:=#MATRIX_gesamt_new;

Join_aktuell:= MATRIX_gesamt_new[1];

for i in [2..b] do
    Join_aktuell := HorizontalJoin(Join_aktuell,MATRIX_gesamt_new[i]);
end for;

Matrix_gesamt_as_Matrix:=Join_aktuell;

Matrix_As_Array:=[**];
for i in [1..NumberOfRows(Matrix_gesamt_as_Matrix)] do
    for j in [1..NumberOfRows(Matrix_gesamt_as_Matrix)] do
        Append(~Matrix_As_Array,Matrix_gesamt_as_Matrix[i][j]);
    end for;
end for;

All_Ordinary_Character_Values_As_Array:=[**];

for i in [1..#CompleteList_V_M_Chi] do
    W:=CompleteList_V_M_Chi[i][3];
    for w in [1..#W] do
        Append(~All_Ordinary_Character_Values_As_Array, W[w]);
    end for;
end for;

return [*List_All_p_prime_Classes_NOT_FLAT, Reps_ccs_N_bar, List_Normalizers,
List_All_p_prime_Classes, List_All_p_prime_Classes_Sizes, b, MATRIX_gesamt,Matrix_gesamt_as_Matrix,
temp, CompleteList_V_M_Chi,Matrix_As_Array,ct,ccs_G,All_Ordinary_Character_Values_As_Array,
List_For_Partial_Subgroup_Lattice*];
end function;


// Example:
// G:=Alt(4); p:=2;
```

```
// U:=OrdinaryCharactersAndProjectiveNBarModules(G,p);
// U;


// We mention that we have written a program that converts this MAGMA output
// into a tex-file. As sage and GAP are used to achieve this, we do not
// present this program here.
```

7.4 A MAGMA algorithm for the computation of p-permutation equivalences

```
RTEx := function(endos,delta,d,F,A,B)
      rt := RightTransversal(A,B);
      tr := [];
      for f in endos do Append(~tr,ScalarMatrix(d,Zero(F))); end for;
      for g in rt do
           bild := delta(g);
           bildi := bild^-1;
           for i:=1 to #tr do
                tr[i] +:= bildi*endos[i]*bild;
     end for;
           end for;
      return tr;
end function;

RelTr := function(endos,M,sgl)
      F := BaseRing(M);
      d := Dimension(M);
      delta := Representation(M);
      s := #sgl;
      while s gt 1 do
 bilder := RTEx(endos,delta,d,F,sgl[s-1],sgl[s]);
 endos := [];
 for phi in bilder do
      if phi ne ScalarMatrix(d,Zero(F))
                  then Append(~endos,phi);
                  end if;
              end for;
            if endos eq [] then return endos; end if;
            s -:= 1;
            end while;
      return endos;
end function;

IsProjective := function(M,H,sgl)
      d := Dimension(M);
basis := Basis(EndomorphismAlgebra(Restriction(M,H)));
      for i in RelTr(basis,M,Append(sgl,H)) do
           if Rank(i) eq d then return true; end if;
           end for;
      return false;
end function;

CCReps := function(sub,G)
    groups := [];
    for i in sub do
        H := i`subgroup;
        need := true;
        for K in groups do
             if IsConjugate(G,H,K) then need:=false; break; end if;
        end for;
        if need then Append(~groups,H); end if;
    end for;
    return groups;
end function;

function Vx(M,G,H,sgl,min)
    if #H gt min then
        Append(~sgl,H);
        for K in CCReps(MaximalSubgroups(H),G) do
            if IsProjective(M,K,sgl) then return Vx(M,G,K,sgl,min); end if;
            end for;
        end if;
    return H;
end function;

function VxStart(M, H)
    G := Group(M);
    ssyl := #SylowSubgroup(H,Characteristic(BaseRing(M)));
    V := SymmetricGroup(1);
    checked := [];
    for U in IndecomposableSummands(Restriction(M,H)) do
         need := true;
```

```
        for W in checked do
            if IsIsomorphic(U,W) then need:=false; break; end if;
            end for;
        if need then
            Append(~checked,U);
            VU := Vx(U,H,H,[],Maximum(#V,ssyl/Gcd(ssyl,Dimension(U))));
            if VU eq H then
                print("The vertex is: ");
                return H;
                end if;
            if #VU gt #V then
                V:=VU;
                print("New lower bound: ");
                print(V);
                print("\n");
                end if;
            end if;
        end for;
    print("The vertex is: ");
    return V;
end function;

DoesVxContainQ:= function(M,Q);
N := Group(M);
Vtx:=VxStart(M,N);
ccsSubgroups := [s`subgroup : s in Subgroups(Vtx)];
ccsSubgroups := [H : H in ccsSubgroups | Order(H) eq Order(Q)];

ccsSubgroups_as_Subgroups_of_N:=[];

for i in [1..#ccsSubgroups] do
    U := ccsSubgroups[i];
    U_in_N:=sub<N|[U.i: i in [1..Ngens(U)]]>;
    Append(~ccsSubgroups_as_Subgroups_of_N,U_in_N);
end for;

Q_in_N := sub<N|[Q.i: i in [1..Ngens(Q)]]>;

flag := false;

for i in [1..#ccsSubgroups] do
flag := IsConjugate(N,ccsSubgroups_as_Subgroups_of_N[i], Q_in_N);
        if flag ne false then
return flag;
end if;
end for;

return flag;

end function;

PermutationModulesForSylow:= function(P,K)
ccsSyl:= [s`subgroup : s in Subgroups(P)];
ccsSyl_without_Syl:=[Q: Q in ccsSyl | Order(Q) lt Order(P)];
temp:=[];
while #ccsSyl_without_Syl gt 0 do
u:=ccsSyl_without_Syl[1];
Append(~temp,u);
inter:=[Q : Q in ccsSyl_without_Syl | IsConjugate(P, Q, u)];
ccsSyl_without_Syl:=[Q : Q in ccsSyl_without_Syl | Q notin inter];
end while;
Append(~temp,P);
temp_sizes:=[];
for i in [1..#temp] do
Append(~temp_sizes,#temp[i]);
end for;
ParallelSort(~temp_sizes,~temp);
PermModules:=[];
for i in [1..#temp] do
T:=TrivialModule(temp[i],K);
I:=Induction(T,P);
Append(~PermModules,I);
end for;
return PermModules;
end function;
```

```
// In the following, m denotes the maximum of the exponents of the groups GxG and HxH.
OrdinaryCharactersAndProjectiveNBarModules_NEW:= function(G,m,p)
K:=GF(p);
Kx<x>:=PolynomialRing(K);
f:=x^m -1;
L:=SplittingField(f);
u:=#L;
K:=GF(u);
R:=CyclotomicField(m: Sparse := true);
ct:=CharacterTable(G);
IrrG:=[];
for j in [1..#ct] do
     Append(~IrrG, ct[j]);
end for;
PIMs_G:=ProjectiveIndecomposableModules(G,K);
CompleteList_V_M_Chi:=[**];
Syl:=SylowSubgroup(G,p);
ccsSyl:= [s`subgroup : s in Subgroups(Syl)];
ccsSyl_without_Syl:=[P: P in ccsSyl | Order(P) lt Order(Syl)];
temp:=[];
while #ccsSyl_without_Syl gt 0 do
     u:=ccsSyl_without_Syl[1];
     Append(~temp,u);
     inter:=[P : P in ccsSyl_without_Syl | IsConjugate(G, P, u)];
     ccsSyl_without_Syl:=[P : P in ccsSyl_without_Syl | P notin inter];
end while;
Append(~temp,Syl);
temp_sizes:=[];
for i in [1..#temp] do
     Append(~temp_sizes,#temp[i]);
end for;
ParallelSort(~temp_sizes,~temp);

List_For_Partial_Subgroup_Lattice:=[];
for i in [1..#temp] do
     MOG:=MinimalOvergroups(G,temp[i]);
     for j in [1..#temp] do
          if temp[j] in MOG then
               Append(~List_For_Partial_Subgroup_Lattice,[i,j]);
          end if;
     end for;
end for;

PMS:=PermutationModulesForSylow(Syl,K);

ccs_G:=[c[3] : c in Classes(G)];
TG:=sub<G|[]>;
List_Chi_Proj:=[];
for m in [1..#PIMs_G] do
     cfs:=CompositionFactors(PIMs_G[m]);
     list_br:=[];
     for a in cfs do
          Append(~list_br,BrauerCharacter(a));
     end for;
     chi:=&+list_br;
     Append(~List_Chi_Proj, chi);
     print("MEIN AKTUELLER BRAUERCHARACTER IST"); print(chi);
     end for;
for m in [1..#PIMs_G] do
     Append(~CompleteList_V_M_Chi, [*TG, PIMs_G[m], List_Chi_Proj[m]*]);
end for;

List_All_p_prime_Classes:=[**];
Reps_ccs_N_bar:=[**];
List_All_p_prime_Classes_Sizes:=[**];
for i in [1..#ccs_G] do
     if not (IsZero(Order(ccs_G[i]) mod p)) then
          Append(~List_All_p_prime_Classes,ccs_G[i]);
          Append(~Reps_ccs_N_bar,ccs_G[i]);
     end if;
end for;
Append(~List_All_p_prime_Classes_Sizes,#List_All_p_prime_Classes);

List_Normalizers:=[];
Append(~List_Normalizers,G);
```

```
for H in temp do
    if #H gt 1 then
        N:=Normaliser(G,H); Append(~List_Normalizers,N);
        H_in_N:=sub<N|H>;
        FAC,f:=quo<N|H_in_N>;
        Ker:=Kernel(f);
        PIMs:=ProjectiveIndecomposableModules(FAC,K);
        s:=#PIMs;
        ccs_N_bar:=[c[3] : c in Classes(FAC)];
        w:=0;
        for i in ccs_N_bar do
            if not (IsZero(Order(i) mod p)) then
                Append(~Reps_ccs_N_bar,i);
                RNK:=Ker * i@@f;
                ELS_RNK:=[x: x in N | x in RNK];
                P_PRIME_ORDER:=[y: y in ELS_RNK | not IsZero(Order(y) mod p)];
                Append(~List_All_p_prime_Classes,P_PRIME_ORDER[1]);
                w:=w+1;
            end if;
        end for;
        Append(~List_All_p_prime_Classes_Sizes,w);
        ccs_N:=[c[3] : c in Classes(N)];
        Brauer_Characters_N_bar:=[];
        Brauer_Characters_N:=[];
        Vertices_and_Brauer_Characters_Induced_Modules_G:=[];
        Brauer_Characters_TS_Modules_G:=[];
        List_Chi_Proj:=[];
        for m in [1..#PIMs] do
            cfs:=CompositionFactors(PIMs[m]);
            list_br :=[];
            for a in cfs do
                Append(~list_br,BrauerCharacter(a));
            end for;
            chi:=&+list_br;
            Append(~List_Chi_Proj, chi);
            print("List_chi_proj_PIMs_FAC ist im Moment: "); print(List_Chi_Proj);
        end for;
        Inflated_Characters:=[];
        Inflated_Modules:=[];
        for i in [1.. s] do
            Append(~Inflated_Characters,LiftCharacter(List_Chi_Proj[i], f, N));
            phi:=Representation(PIMs[i]);
            u:=f*phi;
            M:=GModule(N,[u(N.i): i in [1..Ngens(N)]]);
            Append(~Inflated_Modules,M);
        end for;
        q:=0;
        ccs_N_p_prime:=[y: y in [1..#ccs_N] | not IsZero(Order(ccs_N[y]) mod p)];

        print("The inflated characters are:"); print(Inflated_Characters); print("Now without test!");

        Induced_Characters:=[];
        Induced_Modules:=[];

        CompleteList_V_M_Chi_copy:= CompleteList_V_M_Chi;
        print("CompleteList_V_M_Chi_copy ist jetzt gleichA"); print(CompleteList_V_M_Chi_copy);
        for i in [1.. s] do
            Chi_N:=Inflated_Characters[i];
            Chi_G:=Induction(Chi_N,G);
            M_N:=Inflated_Modules[i];
            M_G:=Induction(M_N,G);
            M_G_neu:=Induction(M_N,G);
            q:=0;
            for j in [1..#CompleteList_V_M_Chi] do
                L:=CompleteList_V_M_Chi[j][2];
                DIM_Summand:=Dimension(L);
                while DIM_Summand gt 0 do
if forall{ ch: ch in IrrG | InnerProduct(ch, Chi_G - CompleteList_V_M_Chi[j][3]) gt -1} then
                    SumIso, V, f := SummandIsomorphism(L,M_G_neu);
                    if Dimension(V) gt 0 then
                        SUB:=sub<M_G_neu|V>;
                        A,C:=HasComplement(M_G_neu,SUB);
                        M_G_neu:=C;
                        Chi_G:=Chi_G - CompleteList_V_M_Chi[j][3];
                        q:=q+1;
                    else
                        DIM_Summand:=0;
```

```
                        end if;
                        else
                            DIM_Summand:=0;
end if;
                    end while;
                end for;

                print("Das Herausfinden von Chi hat geklappt fuer PIM Nummer"); print(i);
                print("und der geliftete Character is"); print(Chi_G);
                print("und q ist gleich"); print(q);

                Append(~CompleteList_V_M_Chi_copy, [*H, M_G_neu, Chi_G*]);
                print("CompleteList_V_M_Chi_copy ist jetzt gleichB"); print(CompleteList_V_M_Chi_copy);

            end for;
            CompleteList_V_M_Chi:= CompleteList_V_M_Chi_copy;
        end if;
end for;
print("Die endgueltige Groesse der Liste CompleteList_V_M_Chi ist "); print(#CompleteList_V_M_Chi);

List_All_p_prime_Classes_NOT_FLAT:=[];
j:=0;
for n in [1..#List_Normalizers] do
    temp95:=[];
    for i in [1.. List_All_p_prime_Classes_Sizes[n]] do
        Append(~temp95,List_All_p_prime_Classes[j+i]);
    end for;
    Append(~List_All_p_prime_Classes_NOT_FLAT,temp95);
    j:=j+List_All_p_prime_Classes_Sizes[n];
end for;

MATRIX_gesamt:=[];
for n in [1..#List_Normalizers] do
    N:=List_Normalizers[n];
    H:=temp[n];
    H_in_N:=sub<N|[H.i: i in [1..Ngens(H)]]>;
    Q_in_Syl:=sub<Syl|[H.i: i in [1..Ngens(H)]]>;
    MATRIX:=[];
    for j in CompleteList_V_M_Chi do
        res_M_Syl:=Restriction(j[2],Syl);
        dir_res_M_Syl:=DirectSumDecomposition(res_M_Syl);
        temp_M_neu:=[];
        for r in [1..#dir_res_M_Syl] do
            for s in PMS do
                if IsIsomorphic(s,dir_res_M_Syl[r]) then
                    Append(~temp_M_neu,s);
                end if;
            end for;
        end for;
        M_neu:=DirectSum(temp_M_neu);
        dimM_neu:=Dimension(M_neu);
        repM_neu:=Representation(M_neu);
        elsQ:=[x:x in Q_in_Syl];
        Positions_Of_Fixed_Points:=[];
        for i in [1.. dimM_neu] do
            if forall{ q: q in elsQ | IsZero(1-repM_neu(q)[i][i])} then
                Append(~Positions_Of_Fixed_Points,i);
            end if;
        end for;
        DIM_Fix := #Positions_Of_Fixed_Points;
        print("Die Groesse der Fixpunktbasis ist"); print(DIM_Fix);

        elsH:=[x:x in H_in_N];

        MATRIX_i:=[**];
        j_N:=Restriction(j[2],N); print("j_N ist:"); print(j_N);
        print("Der Brauercharakter von j_N ist:");
        cfs:=CompositionFactors(j_N);
        list_br :=[];
        for a in cfs do
            Append(~list_br,BrauerCharacter(a));
        end for;
        psi_j_N:=&+list_br;
        print(psi_j_N);

        temp_Brauer:=[];
        if IsZero(Dimension(j_N)-DIM_Fix) then
```

```
                temp__Brauer:=DirectSumDecomposition(j__N);
            else
                dir:=DirectSumDecomposition(j__N);
                for s in dir do
                    DIM__s:=Dimension(s);
                    if DIM__s lt 1 + DIM__Fix then
                        Res__s__H:=Restriction(s,H__in__N);
                        if IsZero(1−#Group(Res__s__H)) then
                            Append(~temp__Brauer,s);
                        else
                            dir__Res__s__H:=DirectSumDecomposition(Res__s__H);
                            if forall{ i: i in dir__Res__s__H | IsIsomorphic(i,TrivialModule(H__in__N,K))} then
                                Append(~temp__Brauer,s);
                            end if;
                        end if;
                    end if;
                end for;
            end if;

        print("ccs__N__bar ist jetzt bei der Eischränkung gleich"); print(ccs__N__bar);
        for i in List__All__p__prime__Classes__NOT__FLAT[n] do
            if #temp__Brauer gt 1 then
                j__neu:=DirectSum(temp__Brauer);
                Append(~MATRIX__i,[*BrauerCharacter(j__neu)(i)*]);
                print("P__PRIME__ORDER ist in diesem FALL1:"); print(i);
                elif #temp__Brauer eq 1 then
                j__neu:=temp__Brauer[1];
                Append(~MATRIX__i,[*BrauerCharacter(j__neu)(i)*]);
                print("P__PRIME__ORDER ist in diesem FALL2:"); print(i);
            else
                Append(~MATRIX__i,[*0*]); print("hier hat Fall 3 eine 0 angefügt");
            end if;
        end for;

        print("MATRIX__i ist gerade gleich"); print(MATRIX__i); print("!");
        MATRIX__i__new:=[MATRIX__i[j][1]: j in [1..#MATRIX__i]];
        print("MATRIX__i__new sieht so aus:"); print(MATRIX__i__new);
        MATRIX__i__new__as__Matrix:=Matrix(R, [MATRIX__i__new]);
        Append(~MATRIX, MATRIX__i__new__as__Matrix);
        end for;
        Append(~MATRIX__gesamt, [*VerticalJoin(MATRIX)*]);
end for;

print("MATRIX__gesamt sieht nun so aus:"); print(MATRIX__gesamt);

MATRIX__gesamt__new:=[* MATRIX__gesamt[j][1] : j in [1..#MATRIX__gesamt] *];

b:=#MATRIX__gesamt__new;

Join__aktuell:= MATRIX__gesamt__new[1];

for i in [2.. b] do
    Join__aktuell := HorizontalJoin(Join__aktuell,MATRIX__gesamt__new[i]);
end for;

Matrix__gesamt__as__Matrix:=Join__aktuell;

Matrix__As__Array:=[**];
for i in [1.. NumberOfRows(Matrix__gesamt__as__Matrix)] do
    for j in [1.. NumberOfRows(Matrix__gesamt__as__Matrix)] do
        Append(~Matrix__As__Array,Matrix__gesamt__as__Matrix[i][j]);
    end for;
end for;

All__Ordinary__Character__Values__As__Array:=[**];

for i in [1..#CompleteList__V__M__Chi] do
    W:=CompleteList__V__M__Chi[i][3];
    for w in [1..#W] do
        Append(~All__Ordinary__Character__Values__As__Array, W[w]);
    end for;
end for;

return [*List__All__p__prime__Classes__NOT__FLAT, Reps__ccs__N__bar, List__Normalizers,
List__All__p__prime__Classes, List__All__p__prime__Classes__Sizes, b, MATRIX__gesamt,Matrix__gesamt__as__Matrix,
temp, CompleteList__V__M__Chi,Matrix__As__Array,ct,ccs__G,All__Ordinary__Character__Values__As__Array,
List__For__Partial__Subgroup__Lattice*];
```

```
end function;

// Note that the program OrdinaryCharac...Neu_mit_Sumiso is the same
// as the program OrdinaryCharactersAndProjectiveNBarModules_NEW





// The function P_Permutation_Equivalences_For_Broue computes
// the Diophantine equation system described in Chapter 6 of the thesis.
// We are in Rickard's strengthening of Broué's abelian defect group
// conjecture here which we approach via p-permutation modules.
// We consider principal blocks.
//
// Note that there is no return value. We consider the principal block.



// The group G and the prime number p are supposed to be entered manually now here:
//




    ctG := CharacterTable(G);
    BlG := Blocks(ctG,p);
    for j in [1..#BlG] do
        if 1 in BlG[j] then
            pos_principal_blockG:=j;
        end if;
    end for;
    B_0_G := BlG[pos_principal_blockG]; // This has type SetEnum in MAGMA
    D := DefectGroup(ctG, B_0_G, p);
    H := Normaliser(G,D);
    ctH := CharacterTable(H);
    BlH := Blocks(ctH,p);
    for j in [1..#BlH] do
        if 1 in BlH[j] then
            pos_principal_blockH:=j;
        end if;
    end for;
    B_0_H := BlH[pos_principal_blockH];
    m:=Exponent(DirectProduct(G,G));
    K:=GF(p);
Kx<x>:=PolynomialRing(K);
f:=x^m -1;
L:=SplittingField(f);
u:=#L;
K:=GF(u);
GxH, i_GxH, p_GxH := DirectProduct(G,H);
print("Computing the TS modules for k[GxH]...this might take a while...");
U:=OrdinaryCharactersAndProjectiveNBarModules_NEW(GxH,m,p);
i_G_GxH := i_GxH[1];
i_H_GxH := i_GxH[2];
DeltaD_as_List := [];
ElementsD := [x: x in D];
for d in ElementsD do
    Append(~DeltaD_as_List,i_G_GxH(d)*i_H_GxH(d));
end for;
DeltaD := sub<GxH|DeltaD_as_List>;
TSModules_GxH:=[];
for i in [1..#U[10]] do
    Append(~TSModules_GxH,U[10][i][2]);
end for;

ccsDeltaD:= [s`subgroup : s in Subgroups(DeltaD)];
ccsDeltaD_without_DeltaD:=[Q: Q in ccsDeltaD | Order(Q) lt Order(DeltaD)];
temp:=[];
while #ccsDeltaD_without_DeltaD gt 0 do
    u:=ccsDeltaD_without_DeltaD[1];
    Append(~temp,u);
    inter:=[Q : Q in ccsDeltaD_without_DeltaD | IsConjugate(DeltaD, Q, u)];
    ccsDeltaD_without_DeltaD:=[Q : Q in ccsDeltaD_without_DeltaD | Q notin inter];
end while;
Append(~temp,DeltaD);
temp_sizes:=[];
```

```
for i in [1..#temp] do
    Append(~temp_sizes,#temp[i]);
end for;
ParallelSort (~temp_sizes,~temp);

// Now we look for the TS_K[GxH]−modules with twisted diagonal vertex:
temp_positions:=[];
for i in [1..#U[10]] do
    bool:=false;
    for j in [1..#temp] do
        if IsConjugate(GxH, temp[j], U[10][i ][1]) then
            bool:=true;
        end if;
    end for;
    if bool then
        Append(~temp_positions,i);
    end if;
end for;

TSModules_GxH_twisted:=[];
for i in temp_positions do
    Append(~TSModules_GxH_twisted,U[10][i][2]);
end for;




FromMAGMAKGxHModuleToBimoduleDual:=function(M,H1,H2)
        B:=Bimodule(H1,H2,M);
        LOG := LeftOppositeModule(B);
        RG := RightModule(B);
        C := Bimodule(Dual(RG), Dual(LOG));
        return(C);
end function;

Bimodules_GxH_twisted:=[];
    for i in [1..#TSModules_GxH_twisted] do
        Append(~Bimodules_GxH_twisted,Bimodule(G,H,TSModules_GxH_twisted[i]));
    end for;
    temp_chi_ordinary_G:=[];
    for i in [1..#ctG] do
        if i in B_0_G then
            Append(~temp_chi_ordinary_G, i);
        end if;
    end for;
    temp_chi_ordinary_H:=[];
    for i in [1..#ctH] do
        if i in B_0_H then
            Append(~temp_chi_ordinary_H, i);
        end if;
    end for;

    bad_indices_G:=[];
    for i in [1..#ctG] do
        if i notin B_0_G then
            Append(~bad_indices_G, i);
        end if;
    end for;
    bad_indices_H:=[];
    for i in [1..#ctH] do
        if i notin B_0_H then
            Append(~bad_indices_H, i);
        end if;
    end for;

    Bimodules_GxH_twisted_princ_block_positions:=[];

    for j in [1..#Bimodules_GxH_twisted] do
        B:=Bimodules_GxH_twisted[j];
        LM:=LeftOppositeModule(B);
        RM:=RightModule(B);
        if forall { z : z in bad_indices_G | IsZero(InnerProduct(ctG[z],BrauerCharacter(LM))) } then
            if forall { y : y in bad_indices_H | IsZero(InnerProduct(ctH[y],BrauerCharacter(RM))) } then
                Append(~Bimodules_GxH_twisted_princ_block_positions,j);
            end if;
        end if;
    end for;
```

```
    Bimodules_GxH_twisted_new := [];

    for i in [1..#Bimodules_GxH_twisted] do
        if i in Bimodules_GxH_twisted_princ_block_positions then
            Append(~Bimodules_GxH_twisted_new,Bimodules_GxH_twisted[i]);
        end if;
    end for;

DualBimodules_twisted:=[];
for i in [1..#TSModules_GxH_twisted] do
        Append(~DualBimodules_twisted,FromMAGMAKGxHModuleToBimoduleDual(TSModules_GxH_twisted[i],G,H));
    end for;

    DualBimodules_GxH_twisted_new := [];

    for i in [1..#DualBimodules_twisted] do
        if i in Bimodules_GxH_twisted_princ_block_positions then
            Append(~DualBimodules_GxH_twisted_new,DualBimodules_twisted[i]);
        end if;
    end for;


    if IsZero(#DualBimodules_GxH_twisted_new - #Bimodules_GxH_twisted_new) then
        print("Bisher passt alles mit den Anzahlen!");
    end if;

TensorProducts:=[]; print("Computing the tensorproducts of bimodules...this might take a lot of time.");
for i in [1..#Bimodules_GxH_twisted_new] do
        for j in [1..#Bimodules_GxH_twisted_new] do
        Append(~TensorProducts,TensorProduct(Bimodules_GxH_twisted_new[i],DualBimodules_GxH_twisted_new[j]));
        end for;
    end for;

// We compute the trivial source char. table of k[GxG] now in order to determine
// the (kG,kG)-bimodule which corresponds to the regular (kG,kG)-bimodule
GxG:=DirectProduct(G,G); print("Computing the TS modules for k[GxG]...this might take a while.");
V:=OrdinaryCharactersAndProjectiveNBarModules_NEW(GxG,m,p);

RegularBimodule := function(G,K)
        eG := [ g: g in G ];
        mats := [];
        for i in [1.. Ngens(G)] do
            perm := Sym(#G)![Position(eG, g*G.i) : g in eG ];
            Append(~mats, PermutationMatrix(K,perm));
        end for;
        RM := GModule(G,mats);
        mats := [];
        for i in [1.. Ngens(G)] do
            perm := Sym(#G)![Position(eG, G.i*g) : g in eG ];
            Append(~mats, PermutationMatrix(K,perm)^-1);
        end for;
        LM := GModule(G,mats);
        B := Bimodule(LM,RM);
        return(B);
    end function;

B_reg := RegularBimodule(G,K);
dir_B_reg := DirectSumDecomposition(B_reg);

pos:=0;
for i in [1..#dir_B_reg] do
        Bimod_now := dir_B_reg[i];
        RM_now := RightModule(Bimod_now);
        LM_now := LeftOppositeModule(Bimod_now);
        if forall{ z : z in bad_indices_G | IsZero(InnerProduct(ctG[z],BrauerCharacter(LM_now))) } then
            if forall{ y : y in bad_indices_G | IsZero(InnerProduct(ctG[y],BrauerCharacter(RM_now))) } then
                pos:=i;
            end if;
        end if;
    end for;

pos;

PrincipalBlock_GxG_As_Bimodule := dir_B_reg[pos];

// We extract the list of all trivial source k[GxG]-modules
```

```
TSModules_GxG := [];
for i in [1..#V[10]] do
        Append(~TSModules_GxG,V[10][i][2]);
    end for;

Bimodules_GxG:=[];
for i in [1..#V[10]] do
        Append(~Bimodules_GxG,Bimodule(G,G,TSModules_GxG[i]));
    end for;

pos := 0;

for i in [1..#V[10]] do
        if IsIsomorphic(Bimodules_GxG[i],PrincipalBlock_GxG_As_Bimodule) then
            pos:=i;
        end if;
    end for;

print("The principal block of GxG is the bimodule with the number "); print(pos);

    TensorDecompositions:=[];
    // This list eventually has r^2 entries, if r denotes the
    // number of t.s. k[GxH] modules with twisted diag. vertices

    for i in [1..#TensorProducts] do
        dirT:=DirectSumDecomposition(TensorProducts[i]);
        Append(~TensorDecompositions,dirT);
    end for;

TensorNumbers:=[];
    for i in [1..#TensorDecompositions] do
        Append(~TensorNumbers,[]);
    end for;

    for i in [1..#TensorDecompositions] do
        if #TensorDecompositions[i] gt 0 then
            for j in [1..#TensorDecompositions[i]] do
                for a in [1..#Bimodules_GxG] do
                    if IsIsomorphic(TensorDecompositions[i][j],Bimodules_GxG[a]) then
                        Append(~TensorNumbers[i],a);
                    end if;
                end for;
            end for;
        end if;
    end for;

// Note that the list TensorNumbers has lists as entries. Often, a lot of these lists are empty.
// The size of the list TensorNumbers ist r^2 the entries of the sublists are numbers between 1 and s, where s is equal to the
      number of indec. TS kGxG modules
// Example: G= A5, p=2: we have a list consisting of 169 entries which are lists themselves. These sublists are either empty or
      have entries from {1,...,59}

TensorPositions:=[];
    for i in [1..#Bimodules_GxG] do
        Append(~TensorPositions,[]);
    end for;

    for j in [1..#Bimodules_GxG] do
        for i in [1..#TensorNumbers] do
            for b in TensorNumbers[i] do
                if j eq b then
                    Append(~TensorPositions[j],i);
                end if;
            end for;
        end for;
    end for;

    TensorPositions;
    // The list TensorPositions hat s lists as entries. For each sublist l_i (i =1,..., s) the following is done: each integer from
       the set {1,..., r^2} is added to l_i as entry as many times as
    // the module occurs as a direct summand of the tensor product.
    // Example: l_1=[1,3,3,5]. means that the first t.s. k[GxG]-module TS_1GxG occurs as a summand in the (here: 13x13=169)
       tensor products,
    // namely: once in the FIRST tensor product, twice in the THIRD tensor product, and one time in the FIFTH tensor product

Variables_as_Lists:=function(n)
        m:=#Bimodules_GxH_twisted_new;
```

```
            u:=n mod m;
            if u gt 0 then
                v:=(n-u)/m;
                return [v+1,u];
            else
                v:=(n-u)/m;
                return [v,m];
            end if;
        end function;

TensorPositionsWithVariables:=[];

        for i in [1..#TensorPositions] do
            Append(~TensorPositionsWithVariables,[]);
        end for;

        for i in [1..#TensorPositions] do
            for j in [1..#TensorPositions[i]] do
                Append(~TensorPositionsWithVariables[i],Variables_as_Lists(TensorPositions[i][j]));
            end for;
        end for;

TensorPositionsWithVariables;

LISTE:=TensorPositionsWithVariables;

// IMPORTANT: the user is supposed to manually safe the objects
// LISTE, pos, the vertices of the t.s. bimodules, the positions of the x_i inside the B_i (counters are helpful),
// U, and V into a .txt-file.

// IMPORTANT: the user is supposed to read the above objects into GAP.



// Now, we need GAP and qpa.
// (continuing our example (G:=AlternatingGroup(5), p:=2):)
```

```
gap> Size(LISTE);
59
ListsAsVariables:=[];
for i in [1.. Size(LISTE)] do
    Append(ListsAsVariables,[[]]);
od;

for i in [1.. Size(LISTE)] do
    for j in [1.. Size(LISTE[i])] do
        Append(ListsAsVariables[i],[[Concatenation("x",String(LISTE[i][j][1])),Concatenation("x",String(LISTE[i][j][2]))]]);
    od;
od;


VarsforQuiver:=[];
for i in [1.. Size(LISTE)] do
    Append(VarsforQuiver,[[1,1,Concatenation("x",String(i))]]);
od;

LoadPackage("qpa");
Q:=Quiver(1,VarsforQuiver);

k:=Rationals;
kQ:=PathAlgebra(k,Q);
AssignGeneratorVariables(kQ);

genskQ:=GeneratorsOfAlgebra(kQ);

NEW_Vars:=[];
for i in [2.. Size(genskQ)] do
    Add(NEW_Vars, genskQ[i]);
od;

ListsWITHNEWVariables:=[];
for i in [1.. Size(LISTE)] do
    Append(ListsWITHNEWVariables,[[]]);
od;
for i in [1.. Size(LISTE)] do
    for j in [1.. Size(LISTE[i])] do
        Append(ListsWITHNEWVariables[i],[ NEW_Vars[LISTE[i][j][1]] * NEW_Vars[LISTE[i][j][2]] ]);
```

```
    od;
od;


rel :=[];

for i in [1.. Size(LISTE)] do
    if i = 41 then
        Add(rel, Sum(ListsWITHNEWVariables[i])−One(kQ));
    else
        Add(rel,Sum(ListsWITHNEWVariables[i]));
    fi ;
od;

rel ;

# this gives us the desired Diophantine equation system.
# By the ideas of Chapter 6 of the thesis , we are
# in the following situation : if the set of solutions is empty,
# one is able to derive a contradiction from the
# Diophantine equation system. Hence, assume that there is
# at least one solution . We are already content as soon as
# we have found one solution. How do we do this?
# a) It is possible to fina a solution with a reasonably high
# probability via the Mathematica command FindInstance
# b) We mention that it is possible to
# solve the Diophantine equation system
# recursively , but we do not elaborate the details here.
# We only mention the following: one can first find
# one / all solution (s) for the equations coefficients in
# front of those bimodules which have maximal vertices and then
# of those bimodules whose vertices have the next−highest group orders,
# and so on. This is justified by the formula TOQUOTE in the article
# od Boltje and Perepelitsky TOQUOTE.
```

7.5 A MAGMA algorithm for the computation of splendid Morita equivalences

```
RTEx := function(endos,delta,d,F,A,B)
      rt := RightTransversal(A,B);
      tr := [];
      for f in endos do Append(~tr,ScalarMatrix(d,Zero(F))); end for;
      for g in rt do
           bild := delta(g);
           bildi := bild^−1;
           for i:=1 to #tr do
                tr[i] +:= bildi*endos[i]*bild;
     end for;
           end for;
      return tr;
end function;

RelTr := function(endos,M,sgl)
      F := BaseRing(M);
      d := Dimension(M);
      delta := Representation(M);
      s := #sgl;
      while s gt 1 do
 bilder := RTEx(endos,delta,d,F,sgl[s−1],sgl[s]);
 endos := [];
 for phi in bilder do
      if phi ne ScalarMatrix(d,Zero(F))
                   then Append(~endos,phi);
                   end if;
               end for;
            if endos eq [] then return endos; end if;
            s −:= 1;
            end while;
      return endos;
end function;

IsProjective := function(M,H,sgl)
      d := Dimension(M);
basis := Basis(EndomorphismAlgebra(Restriction(M,H)));
      for i in RelTr(basis,M,Append(sgl,H)) do
           if Rank(i) eq d then return true; end if;
           end for;
      return false;
end function;

CCReps := function(sub,G)
    groups := [];
    for i in sub do
        H := i`subgroup;
        need := true;
        for K in groups do
             if IsConjugate(G,H,K) then need:=false; break; end if;
        end for;
        if need then Append(~groups,H); end if;
    end for;
    return groups;
end function;

function Vx(M,G,H,sgl,min)
    if #H gt min then
        Append(~sgl,H);
        for K in CCReps(MaximalSubgroups(H),G) do
            if IsProjective(M,K,sgl) then return Vx(M,G,K,sgl,min); end if;
            end for;
        end if;
    return H;
end function;

function VxStart(M, H)
    G := Group(M);
    ssyl := #SylowSubgroup(H,Characteristic(BaseRing(M)));
    V := SymmetricGroup(1);
    checked := [];
    for U in IndecomposableSummands(Restriction(M,H)) do
```

```
        need := true;
        for W in checked do
             if IsIsomorphic(U,W) then need:=false; break; end if;
            end for;
        if need then
           Append(~checked,U);
           VU := Vx(U,H,H,[],Maximum(#V,ssyl/Gcd(ssyl,Dimension(U))));
           if VU eq H then
              print("The vertex is: ");
              return H;
              end if;
           if #VU gt #V then
              V:=VU;
              print("New lower bound: ");
              print(V);
              print("\n");
              end if;
           end if;
        end for;
    print("The vertex is: ");
    return V;
end function;

DoesVxContainQ:= function(M,Q);
N := Group(M);
Vtx:=VxStart(M,N);
ccsSubgroups := [s`subgroup : s in Subgroups(Vtx)];
ccsSubgroups := [H : H in ccsSubgroups | Order(H) eq Order(Q)];

ccsSubgroups__as__Subgroups__of__N:=[];

for i in [1..#ccsSubgroups] do
U := ccsSubgroups[i];
U__in__N:=sub<N|[U.i: i in [1..Ngens(U)]]>;
Append(~ccsSubgroups__as__Subgroups__of__N,U__in__N);
end for;

Q__in__N := sub<N|[Q.i: i in [1..Ngens(Q)]]>;

flag := false;

for i in [1..#ccsSubgroups] do
flag := IsConjugate(N,ccsSubgroups__as__Subgroups__of__N[i], Q__in__N);
          if flag ne false then
return flag;
end if;
end for;

return flag;

end function;

PermutationModulesForSylow:= function(P,K)
ccsSyl:= [s`subgroup : s in Subgroups(P)];
ccsSyl__without__Syl:=[Q: Q in ccsSyl | Order(Q) lt Order(P)];
temp:=[];
while #ccsSyl__without__Syl gt 0 do
u:=ccsSyl__without__Syl[1];
Append(~temp,u);
inter:=[Q : Q in ccsSyl__without__Syl | IsConjugate(P, Q, u)];
ccsSyl__without__Syl:=[Q : Q in ccsSyl__without__Syl | Q notin inter];
end while;
Append(~temp,P);
temp__sizes:=[];
for i in [1..#temp] do
Append(~temp__sizes,#temp[i]);
end for;
ParallelSort(~temp__sizes,~temp);
PermModules:=[];
for i in [1..#temp] do
T:=TrivialModule(temp[i],K);
I:=Induction(T,P);
Append(~PermModules,I);
end for;
return PermModules;
end function;
```

```
OrdinaryCharactersAndProjectiveNBarModules_NEW:= function(G,m,p)
K:=GF(p);
Kx<x>:=PolynomialRing(K);
f:=x^m -1;
L:=SplittingField(f);
u:=#L;
K:=GF(u);
R:=CyclotomicField(m: Sparse := true);
ct:=CharacterTable(G);
IrrG:=[];
for j in [1..#ct] do
    Append(~IrrG, ct[j]);
end for;
PIMs_G:=ProjectiveIndecomposableModules(G,K);
CompleteList_V_M_Chi:=[**];
Syl:=SylowSubgroup(G,p);
ccsSyl:= [s`subgroup : s in Subgroups(Syl)];
ccsSyl_without_Syl:=[P: P in ccsSyl | Order(P) lt Order(Syl)];
temp:=[];

while #ccsSyl_without_Syl gt 0 do
    u:=ccsSyl_without_Syl[1];
    Append(~temp,u);
    inter:=[P : P in ccsSyl_without_Syl | IsConjugate(G, P, u)];
    ccsSyl_without_Syl:=[P : P in ccsSyl_without_Syl | P notin inter];
end while;

Append(~temp,Syl);
temp_sizes:=[];
for i in [1..#temp] do
    Append(~temp_sizes,#temp[i]);
end for;
ParallelSort(~temp_sizes,~temp);

List_For_Partial_Subgroup_Lattice:=[];
for i in [1..#temp] do
    MOG:=MinimalOvergroups(G,temp[i]);
    for j in [1..#temp] do
        if temp[j] in MOG then
            Append(~List_For_Partial_Subgroup_Lattice,[i,j]);
        end if;
    end for;
end for;

PMS:=PermutationModulesForSylow(Syl,K);

ccs_G:=[c[3] : c in Classes(G)];
TG:=sub<G|[]>;
List_Chi_Proj:=[];
for m in [1..#PIMs_G] do
    cfs:=CompositionFactors(PIMs_G[m]);
    list_br :=[];
    for a in cfs do
        Append(~list_br,BrauerCharacter(a));
    end for;
    chi:=&+list_br;
    Append(~List_Chi_Proj, chi);
    print("MEIN AKTUELLER BRAUERCHARACTER IST"); print(chi);
end for;
for m in [1..#PIMs_G] do
    Append(~CompleteList_V_M_Chi, [*TG, PIMs_G[m], List_Chi_Proj[m]*]);
end for;

List_All_p_prime_Classes:=[**];
Reps_ccs_N_bar:=[**];
List_All_p_prime_Classes_Sizes:=[**];
for i in [1..#ccs_G] do
    if not (IsZero(Order(ccs_G[i]) mod p)) then
        Append(~List_All_p_prime_Classes,ccs_G[i]);
        Append(~Reps_ccs_N_bar,ccs_G[i]);
    end if;
end for;
Append(~List_All_p_prime_Classes_Sizes,#List_All_p_prime_Classes);

List_Normalizers:=[];
Append(~List_Normalizers,G);

```

```
for H in temp do
    if #H gt 1 then
        N:=Normaliser(G,H); Append(~List__Normalizers,N);
        H__in__N:=sub<N|H>;
        FAC,f:=quo<N|H__in__N>;
        Ker:=Kernel(f);
        PIMs:=ProjectiveIndecomposableModules(FAC,K);
        s:=#PIMs;
        ccs__N__bar:=[c[3] : c in Classes(FAC)];
        w:=0;
        for i in ccs__N__bar do
            if not (IsZero(Order(i) mod p)) then
                Append(~Reps__ccs__N__bar,i);
                RNK:=Ker * i@@f;
                ELS__RNK:=[x: x in N | x in RNK];
                P__PRIME__ORDER:=[y: y in ELS__RNK | not IsZero(Order(y) mod p)];
                Append(~List__All__p__prime__Classes,P__PRIME__ORDER[1]);
                w:=w+1;
            end if;
        end for;
        Append(~List__All__p__prime__Classes__Sizes,w);
        ccs__N:=[c[3] : c in Classes(N)];
        Brauer__Characters__N__bar:=[];
        Brauer__Characters__N:=[];
        Vertices__and__Brauer__Characters__Induced__Modules__G:=[];
        Brauer__Characters__TS__Modules__G:=[];
        List__Chi__Proj:=[];
        for m in [1..#PIMs] do
            cfs:=CompositionFactors(PIMs[m]);
            list__br :=[];
            for a in cfs do
                Append(~list__br,BrauerCharacter(a));
            end for;
            chi:=&+list__br;
            Append(~List__Chi__Proj, chi);
            print("List__chi__proj__PIMs__FAC ist im Moment: "); print(List__Chi__Proj);
        end for;
        Inflated__Characters:=[];
        Inflated__Modules:=[];
        for i in [1.. s] do
            Append(~Inflated__Characters,LiftCharacter(List__Chi__Proj[i], f, N));
            phi:=Representation(PIMs[i]);
            u:=f*phi;
            M:=GModule(N,[u(N.i): i in [1..Ngens(N)]]);
            Append(~Inflated__Modules,M);
        end for;
        q:=0;
        ccs__N__p__prime:=[y: y in [1..#ccs__N] | not IsZero(Order(ccs__N[y]) mod p)];

        print("The inflated characters are:"); print(Inflated__Characters); print("Now without test!");

        Induced__Characters:=[];
        Induced__Modules:=[];

        CompleteList__V__M__Chi__copy:= CompleteList__V__M__Chi;
        print("CompleteList__V__M__Chi__copy ist jetzt gleichA"); print(CompleteList__V__M__Chi__copy);
        for i in [1.. s] do
            Chi__N:=Inflated__Characters[i];
            Chi__G:=Induction(Chi__N,G);
            M__N:=Inflated__Modules[i];
            M__G:=Induction(M__N,G);
            M__G__neu:=Induction(M__N,G);
            q:=0;
            for j in [1..#CompleteList__V__M__Chi] do
                L:=CompleteList__V__M__Chi[j][2];
                DIM__Summand:=Dimension(L);
                while DIM__Summand gt 0 do
if forall{ ch: ch in IrrG | InnerProduct(ch, Chi__G - CompleteList__V__M__Chi[j][3]) gt -1} then
    SumIso, V, f := SummandIsomorphism(L,M__G__neu);
    if Dimension(V) gt 0 then
        SUB:=sub<M__G__neu|V>;
        A,C:=HasComplement(M__G__neu,SUB);
        M__G__neu:=C;
        Chi__G:=Chi__G - CompleteList__V__M__Chi[j][3];
        q:=q+1;
    else
        DIM__Summand:=0;
```

```
    end if;
else
    DIM_Summand:=0;
end if;
                end while;
            end for;

            print("Das Herausfinden von Chi hat geklappt fuer PIM Nummer"); print(i);
            print("und der geliftete Character is"); print(Chi_G);
            print("und q ist gleich"); print(q);

            Append(~CompleteList_V_M_Chi_copy, [*H, M_G_neu, Chi_G*]);
            print("CompleteList_V_M_Chi_copy ist jetzt gleichB"); print(CompleteList_V_M_Chi_copy);

        end for;
        CompleteList_V_M_Chi:= CompleteList_V_M_Chi_copy;
    end if;
end for;
print("Die endgueltige Groesse der Liste CompleteList_V_M_Chi ist "); print(#CompleteList_V_M_Chi);

return [*CompleteList_V_M_Chi*];
end function;



// The following program tries to find a splendid Morita equivalence
// between the principal p-blocks of G and H for an isomorphism psi.
// In order to discard the existence of a splendid Morita equivalence
// between the principal p-blocks of G and H in general, one has to
// precompose psi with all elements of Aut(D_G)

PuigEquivalenceTest := function(GG,HH,p)
    if #GG lt #HH then
        G:=HH;
        H:=GG;
    else
        G:=GG;
        H:=HH;
    end if;

    ctG := CharacterTable(G);
    BlG := Blocks(ctG,p);
    for j in [1..#BlG] do
        if 1 in BlG[j] then
            pos_principal_blockG:=j;
        end if;
    end for;
    B_0_G := BlG[pos_principal_blockG]; // This has type SetEnum in MAGMA
    D_G := DefectGroup(ctG, B_0_G, p);
    ctH := CharacterTable(H);
    BlH := Blocks(ctH,p);
    for j in [1..#BlH] do
        if 1 in BlH[j] then
            pos_principal_blockH:=j;
        end if;
    end for;
    B_0_H := BlH[pos_principal_blockH]; // This has type SetEnum in MAGMA
    D_H := DefectGroup(ctH, B_0_H, p);
    m:=Exponent(DirectProduct(G,G));
    K:=GF(p);
Kx<x>:=PolynomialRing(K);
f:=x^m -1;
L:=SplittingField(f);
u:=#L;
K:=GF(u);

// Now we need an isomorphism psi: D_G -> D_H:
flag, psi := IsIsomorphic(D_G, D_H);

GxH, i_GxH, p_GxH := DirectProduct(G,H);
i_G_GxH := i_GxH[1];
i_H_GxH := i_GxH[2];
DeltaD_as_List := [];
ElementsD_G := [x: x in D_G];
for d in ElementsD_G do
        Append(~DeltaD_as_List,i_G_GxH(d)*i_H_GxH(psi(d)));
    end for;
```

```
    DeltaD := sub<GxH|DeltaD__as__List>;
    print("Computing the TS modules for k[GxH]...this might take a while.");
U:=OrdinaryCharactersAndProjectiveNBarModules__NEW(GxH,m,p);
TSModules__GxH:=[];
for i in [1..#U[1]] do
        Append(~TSModules__GxH,U[1][i][2]);
    end for;

    ccsDeltaD:= [s‘subgroup : s in Subgroups(DeltaD)];
ccsDeltaD__without__DeltaD:=[Q: Q in ccsDeltaD | Order(Q) lt Order(DeltaD)];
temp:=[];
while #ccsDeltaD__without__DeltaD gt 0 do
    u:=ccsDeltaD__without__DeltaD[1];
    Append(~temp,u);
    inter:=[Q : Q in ccsDeltaD__without__DeltaD | IsConjugate(DeltaD, Q, u)];
    ccsDeltaD__without__DeltaD:=[Q : Q in ccsDeltaD__without__DeltaD | Q notin inter];
end while;
Append(~temp,DeltaD);
temp__sizes:=[];
for i in [1..#temp] do
    Append(~temp__sizes,#temp[i]);
end for;
ParallelSort(~temp__sizes,~temp);

// Now we look for the TS_K[GxH]-modules with twisted diagonal vertex:
temp__positions:=[];
for i in [1..#U[1]] do
    bool:=false;
    for j in [1..#temp] do
        if IsConjugate(GxH, temp[#temp], U[1][i][1]) then //we only take the modules with maximal vertices
            bool:=true;
        end if;
    end for;
    if bool then
        Append(~temp__positions,i);
    end if;
end for;

TSModules__GxH__twisted:=[];
for i in temp__positions do
    Append(~TSModules__GxH__twisted,U[1][i][2]);
end for;




// Input: M is a t.s. k[GxH]-module; H1, H2 are groups; often: G=H1 and H=H2.
// Output: the dual bimodule of Bimodule(H1,H2,M)
FromMAGMAKGxHModuleToBimoduleDual:=function(M,H1,H2)
        B:=Bimodule(H1,H2,M);
        LOG := LeftOppositeModule(B);
        RG := RightModule(B);
        C := Bimodule(Dual(RG), Dual(LOG));
        return(C);
end function;


Bimodules__GxH__twisted:=[];
    for i in [1..#TSModules__GxH__twisted] do
        Append(~Bimodules__GxH__twisted,Bimodule(G,H,TSModules__GxH__twisted[i]));
    end for;

// We want only those twisted bimodules which belong to the principal block as right and left module:
    temp__chi__ordinary__G:=[];
    for i in [1..#ctG] do
        if i in B__0__G then
            Append(~temp__chi__ordinary__G, i);
        end if;
    end for;
    temp__chi__ordinary__H:=[];
    for i in [1..#ctH] do
        if i in B__0__H then
            Append(~temp__chi__ordinary__H, i);
        end if;
    end for;

    bad__indices__G:=[];
```

```
    for i in [1..#ctG] do
        if i notin B_0_G then
            Append(~bad_indices_G, i);
        end if;
    end for;
    bad_indices_H:=[];
    for i in [1..#ctH] do
        if i notin B_0_H then
            Append(~bad_indices_H, i);
        end if;
    end for;

// Now we only collect those twisted (kG,kH)-bimodules with the property that as one-sided
// modules they lie in the respective principal block.
    Bimodules_GxH_twisted_princ_block_positions:=[];

    for j in [1..#Bimodules_GxH_twisted] do
        B:=Bimodules_GxH_twisted[j];
        LM:=LeftOppositeModule(B); // this is a projective kG-module
        RM:=RightModule(B); // this is a projective kH-module
        if forall{ z : z in bad_indices_G | IsZero(InnerProduct(ctG[z],BrauerCharacter(LM))) } then
            if forall{ y : y in bad_indices_H | IsZero(InnerProduct(ctH[y],BrauerCharacter(RM))) } then
                Append(~Bimodules_GxH_twisted_princ_block_positions,j);
            end if;
        end if;
    end for;

    Bimodules_GxH_twisted_new := [];

    for i in [1..#Bimodules_GxH_twisted] do
        if i in Bimodules_GxH_twisted_princ_block_positions then
            Append(~Bimodules_GxH_twisted_new,Bimodules_GxH_twisted[i]);
        end if;
    end for;

    DualBimodules_twisted:=[];
    for i in [1..#TSModules_GxH_twisted] do
        Append(~DualBimodules_twisted,FromMAGMAKGxHModuleToBimoduleDual(TSModules_GxH_twisted[i],G,H));
    end for;

    DualBimodules_GxH_twisted_new := [];

    for i in [1..#DualBimodules_twisted] do
        if i in Bimodules_GxH_twisted_princ_block_positions then
            Append(~DualBimodules_GxH_twisted_new,DualBimodules_twisted[i]);
        end if;
    end for;

    if IsZero(#DualBimodules_GxH_twisted_new - #Bimodules_GxH_twisted_new) then
        print("Bisher passt alles mit den Anzahlen!");
    end if;


TensorProducts:=[]; print("Computing the tensorproducts of bimodules...this might take a lot of time...");
for i in [1..#Bimodules_GxH_twisted_new] do
        Append(~TensorProducts,TensorProduct(Bimodules_GxH_twisted_new[i],DualBimodules_GxH_twisted_new[i]));
    end for;

// The following function computes the regular (kG,kG)-bimodule kG.
RegularBimodule := function(G,K)
        eG := [ g: g in G ];
        mats := [];
        for i in [1.. Ngens(G)] do
            perm := Sym(#G)![Position(eG, g*G.i) : g in eG ];
            Append(~mats, PermutationMatrix(K,perm));
        end for;
        RM := GModule(G,mats);
        //and as left G-module
        mats := [];
        for i in [1.. Ngens(G)] do
            perm := Sym(#G)![Position(eG, G.i*g) : g in eG ];
            Append(~mats, PermutationMatrix(K,perm)^-1);
        end for;
        LM := GModule(G,mats);
        B := Bimodule(LM,RM);
        return(B);
    end function;
```

```
    B__reg := RegularBimodule(G,K);
dir__B__reg := DirectSumDecomposition(B__reg);

// The Bimodule B__0(kG) is projective as right module. Here, B__0(kG) denotes the
// principal block of kG.
pos:=0;
for i in [1..#dir__B__reg] do
    Bimod__now := dir__B__reg[i];
    RM__now := RightModule(Bimod__now);
    LM__now := LeftOppositeModule(Bimod__now);
    if forall{ z : z in bad__indices__G | IsZero(InnerProduct(ctG[z],BrauerCharacter(LM__now))) } then
        if forall{ y : y in bad__indices__G | IsZero(InnerProduct(ctG[y],BrauerCharacter(RM__now))) } then
            pos:=i;
        end if;
    end if;
end for;
pos;

PrincipalBlock__GxG__As__Bimodule := dir__B__reg[pos];

PuigEquivalenceGHBimodules:=[];
for i in [1..#TensorProducts] do
if IsIsomorphic(TensorProducts[i],PrincipalBlock__GxG__As__Bimodule) then
Append(~PuigEquivalenceGHBimodules,[*i,Bimodules__GxH__twisted__new[i]*]);
end if;
end for;

return PuigEquivalenceGHBimodules;
end function;
```

Deutsche Zusammenfassung

Charaktertafeln der Moduln mit trivialen Quellen von kleinen endlichen Gruppen

In der Darstellungstheorie endlicher Gruppen sind sogenannte Moduln mit trivialen Quellen, auch bekannt als p-Permutationsmoduln, von weitreichendem Interesse. Moduln mit trivialen Quellen sind die unzerlegbaren direkten Summanden von Permutationsmoduln. Sie haben viele Anwendungen in der modularen Darstellungstheorie. Um mit diesen Moduln konkrete Berechnungen durchführen zu können, betrachtet man ihre "Lifts" von Charaktersitik p zu Charakteristik 0 sowie die zugehöhrigen gewöhnlichen Charaktere. Eine "Charaktertafel der Moduln mit trivialen Quellen" enthält in der ersten Blockspalte die Auswertungen der gewöhnlichen Charaktere der Moduln mit trivialen Quellen an den p-regulären Elementen. Die weiteren Blockspalten enthalten die Charakterwerte der Brauer-Konstruktionen.

Eines der Hauptziele der vorliegenden Arbeit ist die computerbasierte Bestimmung dieser Charaktertafeln. Wir entwickeln einen Algorithmus zur Berechnung der Charaktertafeln der Moduln mit trivialen Quellen für beliebige endliche Gruppen und Primzahlen. Dieser Algorithmus ist nur durch die Speicherkapazität des verwendeten Computers beschränkt. Wir stellen unsere Implementierungen in den Computeralgebrasystemen GAP und MAGMA hiervon vor. Dafür war es bei dem quelloffenen Computeralgebrasystem GAP nötig, ein Programm zur Berechnung der projektiven unzerlegbaren Moduln einer Gruppenalgebra zu schreiben. Dieses ist ebenfalls Teil dieser Arbeit. Die berechneten Charaktertafeln der Moduln mit trivialen Quellen werden ferner in einer Datenbank gespeichert.

Im theoretischen Teil dieser Arbeit bestimmen wir die gewöhnlichen Charaktere der Moduln mit trivialen Quellen für alle domestizierten Blockalgebren. Dies verwenden wir, um die Charaktertafeln der Moduln mit trivialen Quellen für die unendliche Familie der Diedergruppen D_{4v} von Ordnung $4v$ in Charakteristik 2 zu berechnen, wobei v eine ungerade natürliche Zahl ist. Weiterhin ermitteln wir die Charaktertafeln der Moduln mit trivialen Quellen für die alternierenden Gruppen $\mathfrak{A}_4$ und $\mathfrak{A}_5$ sowie für die Matrixgruppen $\mathrm{SL}_2(11)$, $\mathrm{PSL}_2(11)$, $\mathrm{SL}_2(13)$ und $\mathrm{PSL}_2(13)$ in Charakteristik 2.

Eine Anwendung von Moduln mit trivialen Quellen sind sogenannte p-Permutationsäquivalenzen, welche von Kettenkomplexen induziert werden, die nur aus Bimoduln mit trivialen Quellen bestehen. Es besteht ein interessanter Zusammenhang zwischen diesen Äquivalenzen und Broués Vermutung über abelsche Defektgruppen. Diese Vermutung sagt eine kategorielle Äquivalenz zwischen einem Block und seinem Brauerkorrespondenten mit isomorphen abelschen Defektgruppen voraus. Da es immer nur endlich viele p-Permutationsäquivalenzen zwischen zwei fest gewählten Blockalgebren gibt, ist es möglich, alle p-Permutationsäquivalenzen konkret zu berechnen. Wir präsentieren einen Ansatz, wie dies mit einem Computeralgebrasystem realisiert werden kann.

Bibliography

[Alp86] J. L. Alperin, *Local representation theory*, *Cambridge Studies in Advanced Mathematics* **11**, Cambridge University Press, Cambridge, 1986, Modular representations as an introduction to the local representation theory of finite groups.

[Ben84] D. J. Benson, *Modular representation theory: new trends and methods*, *Lecture Notes in Mathematics* **1081**, Springer-Verlag, Berlin, 1984.

[Ben98] D. J. Benson, *Representations and cohomology, I*, second ed., *Cambridge Studies in Advanced Mathematics* **30**, Cambridge University Press, Cambridge, 1998.

[BP84] D. J. Benson and R. A. Parker, The Green ring of a finite group, *J. Algebra* **87** (1984), 290–331.

[BFL22] B. Böhmler, N. Farrell, and C. Lassueur, Trivial source character tables of $\mathrm{SL}_2(q)$, *J. Algebra* **598** (2022), 308–350.

[Bol95] R. Boltje, *Mackey Functors and Related Structures in Representation Theory and Number Theory*, Habilitation thesis, University of Augsburg, 1995.

[Bol98] R. Boltje, Linear source modules and trivial source modules, in *Group representations: cohomology, group actions and topology*, *Proc. Sympos. Pure Math.* **63**, Amer. Math. Soc., Providence, RI, 1998, pp. 7–30.

[BG07] R. Boltje and A. Glesser, On p-monomial modules over local domains, *J. Group Theory* **10** (2007), 173–183.

[BKY20] R. Boltje, Ç. Karagüzel, and D. Yilmaz, Fusion systems of blocks of finite groups over arbitrary fields, *Pacific J. Math.* **305** (2020), 29–41.

[BP20] R. Boltje and P. Perepelitsky, p-permutation equivalences between blocks of group algebras, 2020. Available at https://arxiv.org/abs/2007.09253.

[BX08] R. Boltje and B. Xu, On p-permutation equivalences: Between Rickard equivalences and isotypies, *Trans. Amer. Math. Soc.* **360** (2008), 5067–5087.

[BCP97] W. Bosma, J. Cannon, and C. Playoust, The Magma algebra system. I. The user language, *J. Symbolic Comput.* **24** no. 3-4 (1997), 235–265, Computational algebra and number theory (London, 1993).

[Bou00] S. Bouc, Burnside rings, *Handbook of Algebra* **2**, Elsevier/North-Holland, Amsterdam, 2000, pp. 739–804.

[BT10] S. Bouc and J. Thévenaz, The primitive idempotents of the p-permutation ring, *J. Algebra* **323** (2010), 2905–2915.

[BY22] S. Bouc and D. Yilmaz, Diagonal p-permutation functors, semisimplicity, and functorial equivalence of blocks, *Adv. Math.* **411** (2022).

[Bra71] R. Brauer, Some applications of the theory of blocks of characters of finite groups IV, *J. Algebra* **17** (1971), 489–521.

[BL08] P. A. Brooksbank and E. M. Luks, Testing isomorphism of modules, *J. Algebra* **320** (2008), 4020–4029.

[Bro90] M. Broué, Isométries parfaites, types de blocs, catégories dérivées, *Astérisque* **181–182** (1990), 61–92.

[Bro85] M. Broué, On Scott modules and p-permutation modules: an approach through the Brauer morphism, *Proc. Amer. Math. Soc.* **93** (1985), 401–408.

[Büy13] Y. Büyükçolak, *Canonical induction for trivial source rings*, Master's thesis, Bilkent University, 2013.

[Con68] S. B. Conlon, Decompositions induced from the Burnside algebra, *J. Algebra* **10** (1968), 102–122.

[CEKL11] D. A. Craven, C. W. Eaton, R. Kessar, and M. Linckelmann, The structure of blocks with a Klein four defect group, *Math. Z.* **268** (2011), 441–476.

[CR87] C. W. Curtis and I. Reiner, *Methods of representation theory. Vol. II*, John Wiley & Sons, Inc., New York, 1987.

[CR90] C. W. Curtis and I. Reiner, *Methods of representation theory. Vol. I*, John Wiley & Sons, Inc., New York, 1990.

[CR06] C. W. Curtis and I. Reiner, *Representation theory of finite groups and associative algebras*, AMS Chelsea Publishing, Providence, RI, 2006.

[Dei97] M. Deiml, *Zur Darstellungstheorie von Darstellungsringen*, Ph.D. thesis, Friedrich-Schiller-Universität Jena, 1997.

[Fei82] W. Feit, *The representation theory of finite groups*, *North-Holland Mathematical Library* **25**, North-Holland Publishing Co., Amsterdam–NewYork, 1982.

[Fon62] P. Fong, Solvable groups and modular representation theory, *Trans. Amer. Math. Soc.* **103** (1962), 484–494.

[GAP] The GAP Group, *GAP – Groups, Algorithms, and Programming, Version 4.12.1*, 2022. Available at https://www.gap-system.org.

[Gil10] C. C. Gill, *Tensor Products, Trivial Source Modules and Related Algebras*, Ph.D. thesis, University of Oxford, 2010.

[HL21] G. Hiss and C. Lassueur, The classification of the trivial source modules in blocks with cyclic defect groups, *Algebr. Represent. Theory* **24** (2021), 673–698.

[Hof04] T. R. Hoffman, *Constructing Basic Algebras for the Principal Block of Sporadic Simple Groups*, Ph.D. thesis, University of Arizona, 2004.

[Isa06] I. M. Isaacs, *Character theory of finite groups*, AMS Chelsea Publishing, Providence, RI, 2006.

[JL01] G. James and M. Liebeck, *Representations and characters of groups*, second ed., Cambridge University Press, New York, 2001.

[Kal09] S. Kalaycioğlu, *Computing the Projective Indecomposable Modules of Large Finite Groups*, Ph.D. thesis, University of Arizona, 2009.

[Kar92] G. Karpilovsky, *Group representations. Vol. 1. Part A*, *North-Holland Mathematics Studies* **175**, North-Holland Publishing Co., Amsterdam, 1992.

[KR94] W. Kimmerle and K. W. Roggenkamp, Non-isomorphic groups with isomorphic spectral tables and Burnside matrices, *Chinese Ann. Math. Ser. B* **15** (1994), 273–282.

[Kra98] H. Krause, Representation type and stable equivalence of Morita type for finite-dimensional algebras, *Math. Z.* **229** (1998), 601–606.

[Lan83] P. Landrock, *Finite group algebras and their modules*, *London Mathematical Society Lecture Note Series* **84**, Cambridge University Press, Cambridge, 1983.

[Las21] C. Lassueur, Representation Theory, lecture notes, https://www.mathematik.uni-kl.de/~lassueur/en/teaching/DTWS2021/DT2021/SkriptMD2021.pdf, 2021.

[Lin18a] M. Linckelmann, *The block theory of finite group algebras. Vol. I*, *London Mathematical Society Student Texts* **91**, Cambridge University Press, Cambridge, 2018.

[Lin18b] M. Linckelmann, *The block theory of finite group algebras. Vol. II*, *London Mathematical Society Student Texts* **92**, Cambridge University Press, Cambridge, 2018.

[LMR94] K. Lux, J. Müller, and M. Ringe, Peakword condensation and submodule lattices: an application of the MEAT-AXE, *J. Symbolic Comput.* **17** (1994), 529–544.

[LP10] K. Lux and H. Pahlings, *Representations of groups: A computational approach*, 1st ed., Cambridge University Press, Cambridge, 2010.

[Lü23] F. Lübeck, Standard generators of finite fields and their cyclic subgroups, *J. Symbolic Comput.* **117** (2023), 51–67.

[Maa11] L. A. Maas, *Modular Spin Characters of Symmetric Groups*, Ph.D. thesis, Universität Duisburg–Essen, 2011.

[Mic75] G. O. Michler, Green correspondence between blocks with cyclic defect groups II, in *Representations of Algebras*, *Lecture Notes in Mathematics* **488**, Springer Berlin Heidelberg, 1975, pp. 210–235.

[Mül03] J. Müller, Computational Representation Theory: Remarks on Condensation, lecture notes, http://www.math.rwth-aachen.de/~Juergen.Mueller/preprints/jm102.pdf, 2003.

[NT89] H. Nagao and Y. Tsushima, *Representations of finite groups*, Academic Press, Inc., Boston, MA, 1989, Translated from the Japanese.

[Nav98] G. Navarro, *Characters and blocks of finite groups*, *London Mathematical Society Lecture Note Series* **250**, Cambridge University Press, Cambridge, 1998.

[Per14] P. Perepelitsky, *p-permutation equivalences between blocks of finite groups*, Ph.D. thesis, University of California, Santa Cruz, 2014.

[Pfe97] G. Pfeiffer, The subgroups of M_{24}, or how to compute the table of marks of a finite group, *Experimental Mathematics* **6** (1997), 247–270.

[Ric91] J. Rickard, Derived equivalences as derived functors, *J. London Math. Soc.* **43** (1991), 37–48.

[Ric96] J. Rickard, Splendid equivalences: derived categories and permutation modules, *Proc. London Math. Soc.* **72** (1996), 331–358.

[Rob89] G. R. Robinson, On projective summands of induced modules, *J. Algebra* **122** (1989), 106–111.

[Sam14] B. Sambale, *Blocks of finite groups and their invariants*, *Lecture Notes in Mathematics* **2127**, Springer, Cham, 2014.

[Sam20] B. Sambale, Survey on perfect isometries, *Rocky Mountain J. Math.* **50** (2020), 1517–1539.

[Sco73] L. L. Scott, Modular permutation representations, *Trans. Amer. Math. Soc.* **175** (1973), 101–121.

[Thé95] J. Thévenaz, *G-algebras and modular representation theory*, *Oxford Mathematical Monographs*, Clarendon Press, Oxford, 1995.

[Web16] P. Webb, *A course in finite group representation theory*, *Cambridge Studies in Advanced Mathematics* **161**, Cambridge University Press, Cambridge, 2016.

[WTP$^+$98] R. Wilson, J. Thackray, R. Parker, F. Noeske, J. Müller, F. Lübeck, C. Jansen, G. Hiss, and T. Breuer, The Modular Atlas project, http://www.math.rwth-aachen.de/~MOC, 1998.

[Wis91] R. Wisbauer, *Foundations of module and ring theory*, *Algebra, Logic and Applications* **3**, Gordon and Breach Science Publishers, Philadelphia, PA, 1991.

[Zim14] A. Zimmermann, *Representation theory*, *Algebra and Applications* **19**, Springer, Cham, 2014, A homological algebra point of view.

Index of notation

General notions

$\mathbb{C}$	set of complex numbers
$\mathbb{Q}$	set of rational numbers
$\mathbb{Z}$	set of integers
$\mathbb{Z}_{\geq n}$	set of integers with values $\geq n$
$\mathbb{P}$	set of positive prime numbers in $\mathbb{Z}$
$\mathrm{Mat}_{n\times n}(\mathbb{Q})$	set of all $n \times n$-matrices with entries in $\mathbb{Q}$
M^{T}	transpose of the matrix M
n_p	p-part of the natural number n
$n_{p'}$	p'-part of the natural number n
$\mathbb{F}_q$	field with q elements, where q denotes a power of the prime number p

Group theory

C_n	cyclic group of order n ($n \in \mathbb{Z}_{\geq 1}$)
D_{2n}	dihedral group of order $2n$ ($n \in \mathbb{Z}_{\geq 2}$)
V_4	Klein four-group
$\mathcal{Q}_8$	quaternion group of order 8
$\mathfrak{S}_n$	symmetric group on n letters
$\mathfrak{A}_n$	alternating group on n letters
$\lvert G\rvert$	order of the group G
$\exp(G)$	exponent of the group G
$g \sim h$	g is conjugate to h in a given group
g_p	p-part of the group element g
$g_{p'}$	p'-part of the group element g
$G_{p'}$	set of p'-elements of the group G
$\mathrm{O}_p(G)$	largest normal p-subgroup of the group G
$Z(G)$	centre of the group G
$H \leq G$	H is a subgroup of the group G
$H \trianglelefteq G$	H is a normal subgroup of the group G
$[G : H]$	index of the subgroup H in the group G
$C_G(H)$	centraliser of the subgroup H in G
$N_G(H)$	normaliser of the subgroup H in G
$\overline{N}_G(H)$	$N_G(H)/H$
$G \times H$	direct product of the groups G and H
$G \rtimes H$	semidirect product of the group G by the group H
c_g	automorphism induced by conjugation with a group element g
$\mathrm{Fix}_\Omega(g)$	fixed points of the action of g on Ω
$\mathrm{Fix}_\Omega(G)$, Ω^G	fixed points of the action of G on Ω
$\mathrm{GL}(V)$	general linear group on a vector space V
$\mathrm{GL}_n(R)$	general linear group with coefficients in R
$\mathrm{SL}_n(\mathbb{F}_q)$, $\mathrm{SL}_n(q)$	special linear group with coefficients in $\mathbb{F}_q$
$\mathrm{PSL}_n(\mathbb{F}_q)$, $\mathrm{PSL}_n(q)$	projective special linear group with coefficients in $\mathbb{F}_q$

Group theory (continued)

$\mathscr{S}(G)$	set of representatives of the set of conjugacy classes of subgroups of the group G
$\mathscr{S}_p(G)$	set of representatives of the set of conjugacy classes of p-subgroups of the group G
$\mathcal{M}(G)$	table of marks of the group G

Modules

A^{op}	opposite algebra of the algebra A
A^{reg}	regular A-module on the ring A
${}_AA_A$	regular bimodule on the algebra A
RG	group ring of the group G over the ring R
$\mathrm{End}_{RG}(M)$	ring of RG-endomorphisms of the module M
$\mathrm{Hom}_R(M,N)$	all R-morphisms from M to N
Id_M	identity morphism on M
$\mathrm{Ker}(f)$	kernel of the morphism f
$\mathrm{Im}(f)$	image of the morphism f
$\mathrm{Coker}(f)$	cokernel of the morphism f
$\dim_{\mathbb{F}}(M)$	dimension of the module M as an $\mathbb{F}$-vector space
$N \leq M$	N is a submodule of the module M
M/N	the quotient module of of the module M by N
$N \mid M$	N is a direct summand of the module M (up to isomorphism)
$\mathrm{Rad}(M)$	radical of the module M
$\mathrm{Soc}(M)$	socle (or bottom) of the module M
$\mathrm{Hd}(M)$	head (or top) of the module M
$\mathrm{ann}_R(M)$	annihilator of the module M
$J(M)$	Jacobson radical of the module M
$\mathrm{Ind}_H^G(M), M\uparrow_H^G$	induction of the module M from H to G
$\mathrm{Res}_H^G(M), M\downarrow_H^G$	restriction of the module M from G to H
$P(M)$	projective cover of the module M
L^K	extension of scalars $K \otimes_{\mathcal{O}} L$
$\bar{L}$	reduction of L modulo $J(\mathcal{O})$
$\mathrm{vtx}(M)$	set of all vertices of the module M
$M^{\supseteq Q}$	direct sum of all direct summands of the module M whose vertices contain a subgroup Q
${}_R\mathrm{Mod}$	category of (not necessarily finitely generated) left R-modules
${}_R\mathrm{Mod}$	category of (not necessarily finitely generated) right R-modules
${}_R\mathrm{Mod}_S$	category of (not necessarily finitely generated) (R,S)-bimodules
${}_R\mathrm{mod}$	category of finitely generated left R-modules
${}_R\mathrm{mod}$	category of finitely generated right R-modules
${}_R\mathrm{mod}_S$	category of finitely generated (R,S)-bimodules
$\underline{\mathscr{C}}$	stable category of the category $\mathscr{C}$
$\mathscr{K}(\mathscr{C})$	Grothendieck group of the additive category $\mathscr{C}$
$a(\mathbb{F}G)$	Green ring of the group G over the field $\mathbb{F}$
$\mathrm{T}(kG)$	trivial source ring of the group G over the field k
$\mathrm{TS}(G;Q)$	set of isomorphism classes of indecomposable trivial source kG-modules with vertex Q
$\mathcal{B}(G)$	Burnside ring of the group G

Character theory and blocks

$\mathrm{Irr}_K(G)$	set of all irreducible K-characters of the group G
$\mathrm{X}(G)$	ordinary character table of the group G
$\mathrm{IBr}_p(G)$	set of all irreducible Brauer characters of the group G in characteristic p
$\mathrm{BR}_p(G)$	Brauer character table of the group G in characteristic p
$\mathrm{Bl}(kG)$	set of blocks of the group algebra kG
$\mathfrak{C}(kG)$	Cartan matrix of the group algebra kG
$\mathfrak{D}(kG)$	decomposition matrix of the group algebra kG
$d(B)$	defect of the p-block B
$\mathrm{Irr}_K(B)$	set of all irreducible K-characters of the p-block B
$\mathrm{IBr}_p(B)$	set of all irreducible Brauer characters of the p-block B in characteristic p
$\mathscr{R}(KG)$	character ring of the group algebra KG
$\mathscr{R}(kG)$	Brauer character ring of the group algebra kG
$\mathrm{Triv}_p(G)$	trivial source character table for the group G at the prime number p

Curriculum vitae

Education

Since *08/2018*	Doctoral researcher at the Department of Mathematics, Technical University of Kaiserslautern, under the supervision of Jun.-Prof. Dr. Caroline Lassueur
11/2011 - 09/2016	Master of Mathematics – University of Stuttgart
10/2008 - 11/2011	Bachelor of Mathematics – University of Stuttgart
07/2008	Abitur (general qualification for university entrance) – Albert-Schweitzer-Gymnasium Leonberg

Employment

03/2022 - 10/2022	Research Assistant in the Research Initiative "SymbTools" of the State RLP and Research Assistant at the Technical University of Kaiserslautern
10/2021 - 01/2022	Four months at the University of California, Santa Cruz, scholarship granted by the German Academic Exchange Service (DAAD)
08/2018 - 10/2021	Research Assistant in the Collaborative Research Centre TRR 195 "Symbolic Tools in Mathematics and their Application" funded by the DFG and since January '21 Research Assistant at the Technical University of Kaiserslautern

Publications

06/2022	*On the extension-closed property for the subcategory* $\mathrm{Tr}(\Omega^2(\mathrm{mod}-A))$ DOI: https://doi.org/10.1007/s10468-022-10140-7 Algebr. Represent. Theory (2022) (with René Marczinzik)
05/2022	*Trivial source character tables of* $\mathrm{SL}_2(q)$ J. Algebra **598** (2022), 308-350 (with Niamh Farrell & Caroline Lassueur)
01/2022	*A cluster tilting module for a representation-infinite block of a group algebra* J. Algebra **589** (2022), 483-494 (with René Marczinzik)
01/2018	*On a conjecture about Morita algebras* J. Algebra **508** (2018), 569-574 (with René Marczinzik)

Lebenslauf

Ausbildung

Seit *08/2018*	Doktorand am Fachbereich Mathematik der TUK, Doktormutter: Jun.-Prof. Dr. Caroline Lassueur
11/2011 - 09/2016	Master im Fach Mathematik – Universität Stuttgart
10/2008 - 11/2011	Bachelor im Fach Mathematik – Universität Stuttgart
07/2008	Abitur – Albert-Schweitzer-Gymnasium Leonberg

Beschäftigungen

03/2022 - 10/2022	Mitarbeiter bei der Forschungsinitiative "SymbTools" des Landes Rheinland-Pfalz sowie wissenschaftlicher Mitarbeiter an der TU Kaiserslautern
10/2021 - 01/2022	Vier Monate an der Universität von Kalifornien, Santa Cruz, Forschungsstipendium für Doktorandinnen und Doktoranden, erhalten vom Deutschen Akademischen Austauschdienst (DAAD)
08/2018 - 10/2021	Wissenshaftlicher Mitarbeiter im DFG-Sonderforschungsbereich TRR 195 "Symbolische Werkzeuge in der Mathematik und ihre Anwendung" und seit Januar '21 wissenschaftlicher Mitarbeiter an der TU Kaiserslautern

Veröffentlichungen

06/2022	*On the extension-closed property for the subcategory* $\mathrm{Tr}(\Omega^2(\mathrm{mod}-A))$ DOI: https://doi.org/10.1007/s10468-022-10140-7 Algebr. Represent. Theory (2022) (mit René Marczinzik)
05/2022	*Trivial source character tables of* $\mathrm{SL}_2(q)$ J. Algebra **598** (2022), 308-350 (mit Niamh Farrell & Caroline Lassueur)
01/2022	*A cluster tilting module for a representation-infinite block of a group algebra* J. Algebra **589** (2022), 483-494 (mit René Marczinzik)
01/2018	*On a conjecture about Morita algebras* J. Algebra **508** (2018), 569-574 (mit René Marczinzik)