Computing Generators and Relations for Matrix Algebras

by

Graham Y. Matthews

(Under the direction of Jon Carlson)

Abstract

We describe algorithms for computing a presentation for a matrix algebra over a finite field, and for computing the basic algebra associated to such a matrix algebra. We give correctness proofs of our algorithms, and implementations of them in the Magma computer algebra system. We use these implementations to compute several basic algebras.

Index words: Matrix Algebras, Finite Dimensional Algebras, Basic Algebras, Generators and Relations, Morita Theory, Modular Representation Theory.

Computing Generators and Relations for Matrix Algebras

by

Graham Y. Matthews

BSc Hons (First Class), The University of Auckland, 1989

MSc (With Distinction), The University of Auckland, 1991

Graduate Diploma, The Australian National University, 1997

A Dissertation Submitted to the Graduate Faculty

of The University of Georgia in Partial Fulfillment

of the

Requirements for the Degree

Doctor of Philosophy

Athens, Georgia

2004

Computing Generators and Relations for Matrix Algebras

by

Graham Y. Matthews

Approved:

Major Professor: Jon Carlson

Committee: Brian Boe

Leonard Chastkofsky

Elham Izadi

Robert Varley

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
August 2004

Preface

Modular representation theory is the study of the realizations of an algebra A over a field K of characteristic p, as a subalgebra of the endomorphisms of some K-vector space V. In computational modular representation theory we usually take K to be finite, and both V and A to be finite dimensional over K, so that endomorphisms of V can be represented as $n \times n$ matrices over K, where n is the K-dimension of V. In the computational setting A is usually implicitly defined via a sequence $\Lambda = \{\lambda_1, \ldots, \lambda_t\}$ of $n \times n$ matrices over K, so A is the subalgebra of $M_n(K)$ generated by Λ.

Two natural questions arise. First, can we compute a presentation for A in terms of generators and relations, and if so, can this be done in a somewhat canonical way? The more canonical the presentation, the more useful it becomes in answering related questions, such as whether two algebras are isomorphic. Second, can we compute the basic algebra associated to A? The basic algebra, B, is a usually much smaller dimensional algebra, with the property that A and B are Morita equivalent, i.e., the module category for A and the module category for B are categorically equivalent, and hence representations of A and of B are 'essentially the same'.

Previous approaches to these questions have mainly focussed on the second problem, and have assumed that A is the group algebra of some finite group G. The techniques employed center on finding idempotents $e \in \mathrm{KG}$, usually via special subgroups of G, such that the condensed algebra $e\mathrm{KG}e$ is Morita equivalent to the group algebra KG. There is a large body of work [16] on how to both construct and recognize such idempotents. The essential problem with this approach is that there is no guarantee that if $g_1, \ldots, g_n$ generate KG, then $eg_1e, \ldots, eg_ne$ will generate $e\mathrm{KG}e$.

This dissertation attempts to answer both problems via a somewhat different approach. We still attempt to find idempotents in A, and then condense A with respect to these idempotents, but we do not assume that A is the group algebra of some finite group. Rather we work directly with A as a subalgebra of a matrix algebra. We also treat the two problems as being intimately related – our solution to the second problem yields a natural solution to the first.

We start by proving that A can be 'split' as the direct sum of a subalgebra A' isomorphic (as an algebra) to A modulo its Jacobson radical $\mathrm{J}(A)$, and the two-sided ideal $\mathrm{J}(A)$. Next we show how to find generators and relations for A'. These generators are constructed in a canonical way, with two generators and a family of four relations per simple A-module. One of the two generators is actually in the basic algebra B for A, and the other becomes zero when we condense to form B. We then show how to construct a generating set for $\mathrm{J}(A)$ as a two-sided ideal. While this generating set is not quite canonical, it has the interesting property that it is wholly contained in B. Hence the set not only generates $\mathrm{J}(A)$, but also $\mathrm{J}(B)$. Our careful construction of generators for A' and $\mathrm{J}(A)$ via elements of A that are either in B, or condense to zero within B, allows us to both construct B as a matrix algebra, and to compute a presentation for B via generators and relations. We conclude by computing the basic algebra associated to several algebras, including the group algebra of the Mathieu group M_{11} in characteristic 2.

We provide algorithms for all our constructions, along with proofs of their correctness. Appendix A contains implementations of our algorithms in the Magma computer algebra system

Acknowledgments

I would like to thank the Department of Mathematics at the University of Georgia at Athens for the financial and professional support given to me during my studies.

I would also like to thank the members of the Magma computational algebra project at the University of Sydney, both for providing me with Magma itself, and for answering my many questions about how to make Magma perform the calculations necessary for the algorithmic content of this dissertation. A special mention goes to Allan Steel for his many helpful hints in this respect.

Thanks to Janet and India for their support and patience in the face of my continual claim that 'the dissertation is almost finished'.

To Dave Benson go my sincerest thanks. His insight and his ability to explain complicated mathematics in terms that I understand has helped me enormously during my time in Athens. I have benefited greatly from knowing Dave as a mathematician, a mentor and a friend.

Finally I would like to thank my supervisor Jon Carlson. Thanks for suggesting such a great topic! Thanks also for your financial support, your mathematical and computational insight, your guidance, and your willingness to help me whenever your door was open. Most of all Jon, huge huge thanks for your patience, your friendship, and your understanding during some difficult times.

I dedicate this dissertation to my mother, June Matthews, without whose continued emotional and financial support I would never have completed my studies.

Table of Contents

Chapter 1

Background

Abstract. We establish the conventions and notation used throughout this dissertation, and give the background results necessary to understand the subsequent chapters. We also discuss the computational tools used in the development of our later algorithms.

1.1 Basic Conventions and Notations

Throughout this chapter all definitions and theorems are given in the context of finite dimensional algebras and finitely generated modules over them. While many of our definitions and theorems are valid over more general rings, restricting to this context both matches the computational setting in which we use this material, and allows many of the main background results to be stated more concisely. For example we do not have to consider finiteness conditions on our algebras and modules, since finite dimensional algebras and finitely generated modules over them are finite dimensional vector spaces, and hence satisfy both the ascending and descending chain conditions on ideals and submodules. The proofs cited for theorems are usually proofs over more general rings, so the reader may consult these proofs to see the general setting.

Henceforth A denotes a finite dimensional algebra with identity element over a field K, and ${}_AA$ denotes the left regular representation of A – i.e., A thought of as a left A-module via the multiplication in A.

1.2 Composition Series

Definition 1.2.1. A *composition series* for an A-module M is a series of submodules,

$$0 = M_0 < M_1 < \cdots < M_r = M$$

where each factor, M_i/M_{i-1}, is a simple A-module

All finitely generated A-modules admit at least one composition series.

Theorem 1.2.2 (Jordan-Hölder). *Suppose M is an A-module. Given any two series of submodules of M*

$$0 = M_0 < M_1 < \cdots < M_r = M$$
$$0 = M'_0 < M'_1 < \cdots < M'_s = M$$

we may refine them to series of equal length

$$0 = L_0 < L_1 < \cdots < L_n = M$$
$$0 = L'_0 < L'_1 < \cdots < L_n = M$$

and find a permutation ρ of $\{1, \ldots, n\}$ such that $L_i/L_{i-1} \simeq L'_{\rho(i)}/L'_{\rho(i-1)}$. In particular then any two composition series for M have the same length and have the same factors up to permutation and isomorphism.

Proof. Theorem 1.1.4 of [4]. □

For finitely generated A-modules the length of a composition series is therefore an invariant of the module, and the factors appearing in a composition series are unique up to isomorphism.

1.3 The Radical of A Module

Definition 1.3.1. Let M be an A-module. The *radical* of M, denoted $\mathrm{Rad}(M)$, is the intersection of all the maximal submodules of M.

Definition 1.3.2. A finitely generated A-module M is called *semisimple* if M is a finite direct sum of simple modules.

The relevant properties of the radical of a module are summarized in the following theorem.

Theorem 1.3.3. *Let M be a finitely generated A-module. Then:*

(i) $\operatorname{Rad}(M/\operatorname{Rad}(M)) = 0$.

(ii) $\operatorname{Rad}(M)$ *is the smallest submodule M' of M such that* $\operatorname{Rad}(M/M') = 0$.

(iii) M is semisimple if and only if $\operatorname{Rad}(M) = 0$.

Proof. Corollary 5.2 of [11], and the discussion in section 1.2 of [4]. □

1.4 The Jacobson Radical

Definition 1.4.1. The *Jacabson radical* of A, denoted $\mathrm{J}(A)$, is the radical of the left regular module ${}_AA$ i.e., $\mathrm{J}(A) = \operatorname{Rad}({}_AA)$. We denote the factor ring $A/\mathrm{J}(A)$ by $\overline{A}$.

Definition 1.4.2. An algebra A is called *semisimple* if $\mathrm{J}(A) = 0$.

The basic properties of the Jacobson radical are summarized in the following theorem.

Theorem 1.4.3. *For an algebra A:*

(i) $\mathrm{J}(A) = \bigcap_I I = \bigcap_J J = \bigcap_K K$, *where I, J, and K range over all maximal left, right, and two sided ideals of A respectively.*

(ii) $\mathrm{J}(A) = \bigcap_X \operatorname{Ann}_A(X) = \bigcap_Y \operatorname{Ann}_A(Y)$, *where X and Y range over all simple left and right A-modules respectively.*

(iii) $\mathrm{J}(A)$ *is a two sided ideal of* A.

(iv) $\mathrm{J}(\overline{A}) = 0$, *so* $\overline{A}$ *is a semisimple algebra.*

(v) if M *is a finitely generated* A*-module, then* $\mathrm{Rad}(M) = \mathrm{J}(A) \cdot M$.

(vi) suppose A' *is a subring of* A *such that* $A' + \mathrm{J}(A)^2 = A$, *then* $A' = A$.

Proof. Propositions 5.5, 5.6, and 5.29, and Corollary 5.11 of [11], and Proposition 1.2.8 of [4]. □

We can relate semisimplicity of finitely generated A-modules to the Jacobson radical of A as follows.

Lemma 1.4.4. *If* A *is semisimple (so* $\mathrm{J}(A) = 0$*) then every finitely generated* A*-module is semisimple. Conversely if* ${}_AA$ *is a finitely generated semisimple* A*-module, then* A *is a semisimple algebra.*

Proof. Lemma 1.2.4 of [4]. □

We can also relate nilpotence in A to the Jacobson radical of A.

Definition 1.4.5. An element $x \in A$ is called *nilpotent* if there exists a positive integer k such that $x^k = 0$. A left ideal $I \subseteq A$ is called *nilpotent* if there exists a positive integer k such that $N^k = 0$ (i.e., all k-fold products in N are zero). A left ideal $I \subseteq A$ is called *nil* if all its elements are nilpotent. Clearly nilpotent ideals are nil.

Theorem 1.4.6. $\mathrm{J}(A)$ *is the largest nilpotent ideal in* A *and as such contains all nilpotent one-sided ideals (left or right), and all one-sided nil ideals.*

Proof. Proposition 5.15 of [11]. □

1.5 The Krull-Schmidt Theorem

Theorem 1.5.1 (Krull-Schmidt). *A finitely generated A-module M has the property that both*

(i) M is a finite direct sum of indecomposable A-modules.

(ii) whenever $M = \bigoplus_{i=1}^{m} M_i = \bigoplus_{i=1}^{n} M_i'$, with each M_i and M_i' a non-zero indecomposable A-module, then $m = n$, and (after reordering if necessary), $M_i \simeq M_i'$.

Proof. Theorem 1.4.6 of [4]. □

1.6 Idempotents

Definition 1.6.1. An element $e \in A$ is called *idempotent* if $e \neq 0$ and $e^2 = e$. Two idempotents e_1 and e_2 are said to be *orthogonal* if $e_1e_2 = e_2e_1 = 0$. An idempotent e is said to be *primitive* if we cannot express e in the form $e = e_1 + e_2$ with e_1 and e_2 orthogonal idempotents.

The idempotent stucture of an algebra A is intimately related to decompositions of ${}_AA$ into finite direct sums of modules.

Definition 1.6.2. Let M be a non-zero indecomposable A-module. M is called a *projective indecomposable* module if M is isomorphic to a direct summand of the (free) A-module ${}_AA$.

Lemma 1.6.3. *If $A = \bigoplus_{i=1}^{n} J_i$, with the J_i non-zero left ideals of A, and $1 = e_1 + \cdots + e_n$, with $e_i \in J_i$, then the e_i are pairwise orthogonal idempotents with $J_i = Ae_i$. Conversely, if $e_1, \ldots, e_n$ are pairwise orthogonal idempotents then $e_1 + \cdots + e_n$ is an idempotent, and $A(e_1 + \cdots + e_n) = \bigoplus_{i=1}^{n} Ae_i$.*

Proof. Lemma 41.1 of [12]. □

Corollary 1.6.4. *Let $e \in A$ be an idempotent. Then the following are equivalent:*

(i) e is primitive.

(ii) Ae is an indecomposable left A-module.

(iii) eA is an indecomposable right A-module.

Moreover if e is primitive then Ae is a projective indecomposable A-module.

Proof. The three main points are proved in Corollary 41.3 of [12]. The fact that Ae is a projective indecomposable module when e is primitive follows from the fact that ${}_AA = Ae \oplus A(1-e)$. □

The point here is that in a finite dimensional algebra, the identity of A can be expressed as a finite sum of pairwise orthogonal primitive idempotents, and every such expression then gives a decomposition of A into a direct sum of projective indecomposable A-modules. Moreover, since ${}_AA$ satisfies the Krull-Schmidt theorem, every projective indecomposable A-module is isomorphic to Ae for some primitive idempotent $e \in A$.

The correspondence between expressions for the identity in terms of primitive idempotents, and decompositions of ${}_AA$ into projective indecomposables, can be taken further via the idea of conjugacy of idempotents.

Definition 1.6.5. Two idempotents e_1 and e_2 are called *conjugate* in A, if there exists an invertible element $r \in A$ such that $r^{-1}e_1r = e_2$.

Lemma 1.6.6. *Two idempotents e_1 and e_2 are conjugate in A if and only if (as A-modules) $Ae_1 \simeq Ae_2$ and $A(1-e_1) \simeq A(1-e_2)$.*

Proof. Proposition 1.7.2 of [4]. □

Summarizing we have the following theorem.

Theorem 1.6.7. *In a finite dimensional algebra A the identity of A can be expressed as a finite sum of primitive idempotents, $1_A = \sum e_k$. Such an expression then gives a decomposition, ${}_AA = \bigoplus Ae_k$, of ${}_AA$ into projective indecomposables. Moreover every projective indecomposable A-module is isomorphic to Ae_k for some choice of k, and $Ae_k \simeq Ae_j$ if and only if e_k and e_j are conjugate,*

Thus the primitive idempotents characterize the projective indecomposable modules for a finite dimensional algebra. They can also be used to characterize the simple modules for a finite dimensional algebra.

Theorem 1.6.8. *Let e_1 and e_2 be primitive idempotents in A. Then:*

(i) $\mathrm{J}(A)e_1$ *is the unique maximal submodule in* Ae_1.

(ii) $Ae_1/\mathrm{J}(A)e_1$ *is a simple A-module.*

(iii) $Ae_1 \simeq Ae_2$ *if and only if* $Ae_1/\mathrm{J}(A)e_1 \simeq Ae_2/\mathrm{J}(A)e_2$.

Proof. Theorem 45.7 of [12]. □

Theorem 1.6.9. *There is a 1-1 correspondence between isomorphism classes of projective indecomposable A-modules, and isomorphism classes of simple A-modules given by $P \leftrightarrow P/\operatorname{Rad}(P)$. Specifically this correspondence works as follows: if P is a projective indecomposable A-module, then $P \simeq Ae$ for some primitive idempotent $e \in A$. By the previous lemma $S = Ae/J(A)e$ is simple, and*

$$P/\operatorname{Rad}(P) \simeq P/\mathrm{J}(A)P \simeq Ae/J(A)Ae = Ae/J(A)e = S$$

Conversely if S is a simple A-module then $S \simeq Ae/J(A)e$ for some primitive idempotent $e \in A$. Taking $P = Ae$ then gives a projective indecomposable A-module such that $P/\operatorname{Rad}(P) \simeq S$.

Proof. Corollary 45.8 of [12]. □

Until now we have used expressions for the identity of A in terms of primitive idempotents to produce decompositions of ${}_AA$ into finite direct sums of projective indecomposables. We can in fact say more, specifically for suitable choices of ideals $I \subseteq A$, expressing the identity of A/I as a finite sum of idempotents, is sufficient to decompose ${}_AA$ into finite sums of A-modules.

Theorem 1.6.10 (Idempotent Lifting Theorem). *Suppose I is a two sided nilpotent ideal of A. Then:*

(i) if $\overline{e}$ is an idempotent in A/I, then there exists an idempotent $e \in A$ (called a lift of $\overline{e}$), such that $\overline{e} = e + I$.

(ii) if $\overline{e}$ is primitive in A/I, then any lift e is primitive in A.

(iii) if $\overline{e}_1$ and $\overline{e}_2$ are orthogonal in A/I, then any lifts e_1 and e_2 are orthogonal in A.

(iv) if $\overline{e}_1$ and $\overline{e}_2$ are conjugate in A/I, then any lifts e_1 and e_2 are conjugate in A.

(v) if $e_1, \ldots, e_n$ are pairwise orthogonal (primitive) idempotents in A, then $e_1 + I, \ldots, e_n + I$, are pairwise orthogonal (primitive) idempotents in A/I.

(vi) if $\overline{e}_1, \ldots, \overline{e}_n$ are pairwise orthogonal idempotents in A/I such that $1_{A/I} = \sum \overline{e}_i$, then there exist pairwise orthogonal idempotents $e_1, \ldots, e_n$ in A, with $\overline{e}_i = e_i + I$, such that $1_A = \sum e_i$. Moreover if $\overline{e}_i$ is primitive, then so is e_i, and if $\overline{e}_i$ is conjugate to $\overline{e}_j$, then e_i is conjugate to e_j.

Proof. Theorem 1.7.3 and Corollary 1.7.4 of [4]. □

This theorem will form the cornerstone of several later algorithms. It will be most often used in the special case where I is actually equal to $\mathrm{J}(A)$, rather than simply contained in it.

We finish this section with a technical result.

Lemma 1.6.11. *If e is an idempotent in A, then eAe is a subalgebra of A with identity e. Moreover $\mathrm{J}(eAe) = eAe \cap \mathrm{J}(A) = e\,\mathrm{J}(A)e$, and $eAe/\,\mathrm{J}(eAe) \simeq \overline{e}\overline{A}\overline{e}$, where $\overline{e} = e + \mathrm{J}(A)$.*

Proof. The first claim is obvious. The second claim is proved in Proposition 5.13 of [11]. □

1.7 Wedderburn's Theorems

Idempotents in a finite dimensional algebra A are closely connected to endomorphism rings over A, and hence to matrix rings. The connection is via the opposite algebra to A.

Definition 1.7.1. A^{op} denotes the opposite algebra to A, i.e., the algebra whose underlying set is A but whose multiplication is reversed. So if '$\cdot$' denotes the product operator in A^{op}, then by definition $a \cdot b = ba$.

The basic properties of opposite algebras are as follows.

Lemma 1.7.2. *For an algebra A:*

(i) $A^{op} \simeq \mathrm{End}_A({}_AA)$.

(ii) $(A^{op})^{op} \simeq A$.

(iii) $M_n(A)^{op} \simeq M_n(A^{op})$.

For a family A_i of algebras, $(\bigoplus_i A_i)^{op} \simeq \bigoplus_i (A_i^{op})$.

Proof. Lemmas 11 and 15 of [2]. □

The connection between opposite algebras and endomorphism rings is given in the following lemmas.

Lemma 1.7.3. *If M is an A-module, and e is an idempotent in A, then as abelian groups*

$$eM \simeq \mathrm{Hom}_A(Ae, M)$$

Moreover we have an isomorphism of algebras $eAe \simeq \mathrm{End}_A(Ae)^{op}$.

Proof. Lemma 1.3.3 of [4]. □

Lemma 1.7.4. *Suppose e is a primitive idempotent in A. Then* $\mathrm{End}_A(Ae/\,\mathrm{J}(A)e) \simeq \mathrm{End}_A(Ae)/\,\mathrm{J}\left(\mathrm{End}_A(Ae)\right)$.

Proof. Lemma 45.9 of [12]. □

Corollary 1.7.5. *Suppose that P is a projective indecomposable A-module with corresponding simple S. Then* $\mathrm{End}_A(S) \simeq \mathrm{End}_A(P/\,\mathrm{Rad}(P)) \simeq \mathrm{End}_A(P)/\,\mathrm{J}(\mathrm{End}_A(P))$.

Proof. Since P is a projective indecomposable, $P \simeq Ae$ for some primitive idempotent $e \in A$, and $S \simeq P/\,\mathrm{Rad}(P)$. Hence

$$\begin{aligned}
\mathrm{End}_A(S) &\simeq \mathrm{End}_A(P/\,\mathrm{Rad}(P)) \\
&\simeq \mathrm{End}_A(Ae/\,\mathrm{J}(A)Ae) \\
&\simeq \mathrm{End}_A(Ae/\,\mathrm{J}(A)e) \\
&\simeq \mathrm{End}_A(Ae)/\,\mathrm{J}(\mathrm{End}_A(Ae)) \\
&\simeq \mathrm{End}_A(P)/\,\mathrm{J}(\mathrm{End}_A(P))
\end{aligned}$$

□

We now come to a theorem which will form the backbone of much of theoretical material developed later. This theorem provides a complete description of finite dimensional semisimple algebras over a field.

Definition 1.7.6. An algebra A is called *simple* if A contains no non-trivial two sided ideals.

Theorem 1.7.7 (Wedderburn's Structure Theorem). *If A is a finite dimensional semisimple algebra over a field K then*

$$A \simeq \bigoplus_{i=1}^{r} A_i$$

where $A_i = M_{n_i}(D_i)$, with D_i a division ring containing K in its center, and the A_i uniquely determined. Moreover A has exactly r isomorphism classes of simple modules S_i, $1 \leq i \leq r$, $D_i \simeq \mathrm{End}_A(S_i)^{op}$, and $\dim_{D_i{}^{op}}(S_i) = n_i$. If A is simple then $A \simeq M_n(D)$ (i.e., $r = 1$ in the direct sum decomposition).

Proof. Theorem 1.3.5 of [4] □

The converse of this theorem is also true – i.e., every matrix algebra over a division ring containing a field K in its center is a simple finite dimensional K-algebra, and any finite direct sum of matrix algebras over division rings containing a field K in their centers is a semisimple finite dimensional K-algebra. Here we are of course assuming that the matrix algebras have finite degree.

Our main use of the Wedderburn Structure theorem is in the special case when K is a finite field. In this case the following two theorems are critical.

Theorem 1.7.8. *Every finite division ring is a field.*

Proof. Theorem 68.9 of [10]. □

Corollary 1.7.9. *Let A be a semisimple algebra over a finite field K. Then*

$$A \simeq \bigoplus_{i=1}^{r} M_{n_i}(K_i)$$

where K_i is a finite extension of K.

Proof. By the Wedderburn Structure theorem

$$A \simeq \bigoplus_{i=1}^{r} M_{n_i}(D_i)$$

where $D_i \simeq \mathrm{End}_A(S_i)^{\mathrm{op}}$, and contains K in its center. Each S_i is a finite dimensional K-vector space, and since K is finite, each S_i, and hence D_i, is also finite. The result now follows from the previous theorem. □

The Idempotent Lifting theorem, the Wedderburn Structure theorem, together with 1.6.9 and some knowledge of idempotents in a full matrix algebra, provide a methodology for understanding an algebra A by examining its semisimple quotient $\overline{A}$. As we shall make heavy use of this methodology in later chapters, we detail how this approach works here.

By the Wedderburn Structure theorem

$$\overline{A} \simeq \bigoplus_{i=1}^{r} A_i$$

where $A_i = M_{n_i}(D_i)$, and $D_i = \mathrm{End}_A(S_i)^{\mathrm{op}}$ (S_i is the i-th simple A-module, and has dimension n_i over ${D_i}^{\mathrm{op}}$).

Take $\overline{e}_{ij} \in A_i$ to be the matrix with a 1 in the j-th diagonal entry, and zero elsewhere. Then

$$1_{A/J(A)} = \sum_{i=1}^{r} \sum_{j=1}^{n_i} \overline{e}_{ij}$$

is an expression for the identity of $\overline{A}$ in terms of pairwise orthogonal primitive idempotents in $\overline{A}$ (here we are identifying $\overline{A}$ and $\bigoplus_{i=1}^{r} A_i$ via the isomorphism between them). By the Idempotent Lifting theorem there is a corresponding expression,

$$1_A = \sum_{i=1}^{r} \sum_{j=1}^{n_i} e_{ij}$$

with the e_{ij} pairwise orthogonal primitive idempotents in A.

Since $\overline{e}_{ij}$ and $\overline{e}_{ik}$ are conjugate in A_i, e_{ij} and e_{ik} are conjugate in A, and so by 1.6.7, Ae_{ij} and Ae_{ik} are isomorphic projective indecomposable A-modules. Hence if we define $P_i = Ae_{i1}$, we have a decomposition

$${}_AA \simeq \bigoplus_{i=1}^{r} P_i^{n_i}$$

of ${}_AA$ into projective indecomposable A-modules. By 1.6.9 each P_i corresponds to a simple module S_i, namely

$$S_i \simeq P_i/\operatorname{Rad}(P_i) \simeq Ae_i/J(A)Ae_i \simeq Ae_i/J(A)e_i$$

Finally observe that by 1.7.3 and 1.7.5

$$\begin{aligned} D_i^{\mathrm{op}} &= (\operatorname{End}_A(S_i)^{\mathrm{op}})^{\mathrm{op}} \\ &\simeq \operatorname{End}_A(Ae_i/\operatorname{J}(A)e_i) \\ &\simeq \operatorname{End}_A(Ae_i)/\operatorname{J}(\operatorname{End}_A(Ae_i)) \\ &\simeq e_iAe_i/\operatorname{J}(e_iAe_i) \end{aligned}$$

These observations lead to the final theorem in this section.

Theorem 1.7.10. *Using the notation above, suppose that* $f_i = \sum_{k=1}^{n_i} e_{ik}$. *Then*

$$f_iAf_i/\operatorname{J}(f_iAf_i) \simeq A_i$$

Proof. Since $Ae_{ik} \simeq Ae_{ij}$ we have that

$$\operatorname{End}_A(Af_i) = \operatorname{End}_A\big(A\big(\sum_{k=1}^{n_i} e_{ik}\big)\big) \simeq M_{n_i}\big(\operatorname{End}_A(Ae_{i1})\big)$$

By 1.7.3 and 1.7.2 it then follows that

$$\begin{aligned} f_iAf_i/\operatorname{J}(f_iAf_i) &\simeq \operatorname{End}_A(Af_i)^{\mathrm{op}}/\operatorname{J}\big(\operatorname{End}_A(Af_i)^{\mathrm{op}}\big) \\ &\simeq \operatorname{End}_A\big(A\big(\sum_{k=1}^{n_i} e_{ik}\big)\big)^{\mathrm{op}}/\operatorname{J}\Big(\operatorname{End}_A\big(A\big(\sum_{k=1}^{n_i} e_{ik}\big)\big)^{\mathrm{op}}\Big) \\ &\simeq M_{n_i}\big(\operatorname{End}_A(Ae_{i1})\big)^{\mathrm{op}}/\operatorname{J}\Big(M_{n_i}\big(\operatorname{End}_A(Ae_{i1})\big)^{\mathrm{op}}\Big) \\ &\simeq M_{n_i}\big(\operatorname{End}_A(Ae_{i1})^{\mathrm{op}}\big)/\operatorname{J}\Big(M_{n_i}\big(\operatorname{End}_A(Ae_{i1})^{\mathrm{op}}\big)\Big) \\ &\simeq M_{n_i}\big(\operatorname{End}_A(S_i)^{\mathrm{op}}\big)/\operatorname{J}\Big(M_{n_i}\big(\operatorname{End}_A(S_i)^{\mathrm{op}}\big)\Big) \\ &\simeq M_{n_i}(D_i)/\operatorname{J}\big(M_{n_i}(D_i)\big) \\ &\simeq M_{n_i}(D_i) \end{aligned}$$

The last step here follows from the fact that full matrix algebras are simple, and so have no non-trivial two sided ideals. □

1.8 Finite Fields

We remind the reader of three results from finite field theory which we shall make essential use of.

Definition 1.8.1. Suppose L is an extension field of K, such that $L = K(\alpha)$ for some $\alpha \in L$. Then α is called a *primitive element* of L over K.

Theorem 1.8.2 (Primitive Element Theorem). *Suppose L is a finite extension of K. Then there exists a primitive element α of L over K, so that $L = K(\alpha)$.*

Proof. Theorem 4.6 of [15]. □

Theorem 1.8.3. *Let K be a finite field of size $q = p^d$, and let $K^* = K \setminus 0$ be the multiplicative group of units of K. Then:*

(i) K^* is cyclic of order $q - 1$.

(ii) the map that sends every element to its p-th power in K (the Frobenius map) is an automorphism of K leaving the ground field $\mathbb{F}_p$ fixed.

(iii) the map that sends every element to its q-th power in K is the identity map on K.

Proof. Theorem 5.3 of [15]. □

Theorem 1.8.4. *Let m and n be positive integers. Then $\mathbb{F}_{p^n}$ is contained in $\mathbb{F}_{p^m}$ if and only if n divides m.*

Proof. Theorem 5.5 of [15]. □

1.9 Vector Spaces

We give some standard results on vector spaces which we shall use in our later algorithms.

Lemma 1.9.1 (Fitting's Lemma). *Let M be a finitely generated A-module, and $f \in \mathrm{End}_A(M)$. Then $M = \mathrm{im}(f^n) \oplus \ker(f^n)$ for some n.*

Proof. Lemma 1.4.4 of [4]. □

Lemma 1.9.2. *Suppose U is a vector space over K with basis $\{u_1, \ldots, u_n\}$, and L is an extension field of K of degree d with K-basis $\{b_1, \ldots, b_d\}$. Then $V = L \otimes_K U$ is a vector space over K of dimension dn, with K-basis (called the standard basis) $\{b_i \otimes u_j\}$. Moreover the action of L on V is given by $l(b_i \otimes u_j) = (lb_i) \otimes u_j$, for all $l \in L$.*

Proof. Corollary 2.4 of [15]. □

Lemma 1.9.3. *Suppose V is a vector space of dimension n over L, where L is an extension field of K of degree d with K-basis $\{b_1, \ldots, b_d\}$. Let $\{v_1, \ldots, v_n\}$ be a basis for V over L. Let W be the K-span of $\{v_1, \ldots, v_n\}$. Then $L \otimes_K W \simeq V$ via the isomorphism that maps $b_i \otimes v_j \mapsto b_i v_j$.*

Proof. The map $b_i \otimes v_j \mapsto b_i v_j$ is onto. The result now follows by the previous lemma, since $\dim_K(L \otimes_K W) = dn = \dim_K(V)$. □

Lemma 1.9.4. *Suppose L is an extension field of K of degree d. Then every element of $M_n(L)$ can be written in the form*

$$\begin{pmatrix} \alpha_{11} & \cdots & \alpha_{1n} \\ \alpha_{21} & \cdots & \alpha_{2n} \\ \vdots & \vdots & \vdots \\ \alpha_{n1} & \cdots & \alpha_{nn} \end{pmatrix}_{m \times m}$$

where $m = nd$, and each α_{ij} is a $d \times d$ matrix over K representing an element $l \in L$ with respect to some basis $\{b_1, \ldots, b_d\}$ of L over K.

Proof. Let $\alpha \in M_n(L)$. Then α acts on the natural vector space V of dimension n over L. Let $\{v_1, \ldots, v_n\}$ be a basis for V over L. By the previous lemma $V \simeq L \otimes_K W$. Order the standard basis for $L \otimes_K W$ as follows

$$\{b_1 \otimes v_1, b_2 \otimes v_1, \ldots, b_d \otimes v_1, b_1 \otimes v_2, b_2 \otimes v_2, \ldots, b_d \otimes v_2, \ldots\}$$

With respect to this basis α has the required form. □

Remark. As $\dim_K(M_n(L)) = n^2 d$, it is clear that not every $d \times d$ matrix over K can appear as an α_{ij}.

1.10 Splitting Fields

Given a K-algebra, A, and an extension field L of K, the vector space $A^L = L \otimes_K A$ becomes an L-algebra by defining the scalar action of L on A^L to be

$$l_1 \cdot (l_2 \otimes a_1) = (l_1 \cdot l_2) \otimes a_1$$

and the algebra multiplication to be

$$(l_1 \otimes a_1) \cdot (l_2 \otimes a_2) = (l_1 \cdot l_2) \otimes (a_1 \cdot a_2)$$

for all $l_1, l_2 \in L$, and $a_1, a_2 \in A$. The unit in A^L is $1_L \otimes 1_A$. Note that A can be embedded in A^L as a K-subalgebra via the K-embedding $a \mapsto 1 \otimes a$, and then A^L is simply the set of L-linear combinations of elements of the image of A under this embedding. It follows then that if A is generated as a K-algebra by $a_1, \ldots, a_t$, then A^L is generated as an L-algebra by $1 \otimes a_1, \ldots, 1 \otimes a_t$.

Given an A-module M the vector space $M^L = L \otimes_K M$ becomes an A^L-module under the action

$$(l_1 \otimes a)(l_2 \otimes m) = (l_1 \cdot l_2) \otimes (a \cdot m)$$

for all $l_1, l_2 \in L$, and $a \in A$, and $m \in M$. We can embed M into M^L as a K-subspace via the K-embedding $m \mapsto 1 \otimes m$, and then M^L is simply the set of L-linear combinations of elements of the image of M under this embedding. Note that $\dim_L(M^L) = \dim_K(M)$. Indeed if $B = \{v_1, \ldots, v_n\}$ is a K-basis for M, then B (under the embedding) is also an L-basis for M^L. Moreover if $\rho : A \to \mathrm{End}_K(M)$ is a matrix representation of A with respect to the K-basis B, and $\rho^L : A^L \to \mathrm{End}_L(M^L)$ is a matrix representation of A^L with respect to the L-basis B, then ρ^L agrees with ρ on A, and

$$\rho^L(\sum_i l_i a_i) = \sum_i l_i \rho(a_i)$$

for all $l_i \in L$ and $a_i \in A$.

Definition 1.10.1. A simple A-module M is called *absolutely irreducible* if M^L is a simple A^L-module for every extension L of K. If M is an A-module such that the composition factors of M^L are absolutely irreducible, then L is called a *splitting field* for M. If L is a splitting field for every simple A-module, then A is called an *L-split* algebra. L is called a *splitting field* for A if A^L is an L-split algebra.

The following two theorems characterize absolutely irreducible modules, establish the existence of splitting fields, and tell us how splitting fields behave under field extensions.

Theorem 1.10.2. *A simple A-module M is absolutely irreducible if and only if* $\mathrm{End}_A(M) \simeq K$.

Proof. Theorem 29.13 of [10]. □

Corollary 1.10.3. *Every simple module over an algebraically closed field is absolutely irreducible, and every algebra over an algebraically closed field, L, is L-split. Hence the algebraic closure of K is always a splitting field for any A-module and for any K-algebra.*

Corollary 1.10.4. *If L is a splitting field for a simple A-module S, then every extension L' of L is also a splitting field for S.*

Theorem 1.10.5. *If L is a splitting field for A, then every extension L' of L is also a splitting field for A.*

Proof. Theorem 29.21 of [10]. □

We will also need to know how changing scalars from K to a splitting field L interacts with the Jacobson radical of A. We have the following two facts.

Theorem 1.10.6. *L is a splitting field for an algebra A if and only if L is a splitting field for $\overline{A}$.*

Proof. Proposition 7.9 of [14]. □

Theorem 1.10.7. *Suppose L is a splitting field for A. Then $\mathrm{J}(A^L) = \mathrm{J}(A)^L$.*

Proof. Proposition 29.22 of [10]. □

We can give very concrete descriptions of splitting fields for modules and algebras over finite fields. Specifically we have the following theorem.

Theorem 1.10.8. *Let K be a finite field, A be a K-algebra, and S be a simple A-module. Then $\mathrm{End}_A(S)^{op}$ is the minimal splitting field for S. In particular, in the notation of 1.7.9, if S_i is the i-th simple A-module, then K_i is the minimal splitting field for S_i.*

Proof. Theorem 1.3.3 of [19]. □

Corollary 1.10.9. *Let A be an algebra over a finite field $K = \mathbb{F}_{p^c}$. Suppose that*

$$\overline{A} \simeq \bigoplus_{i=1}^{r} M_{n_i}(K_i)$$

where K_i is a degree d_i extension field of K. Let $l = c \cdot \mathrm{LCM}(d_1, \ldots, d_r)$. Then $L = \mathbb{F}_{p^l}$ is the minimal splitting field for A.

Proof. By 1.10.6 a minimal splitting field for $\overline{A}$ will be a minimal splitting field for A. By 1.10.8, K_i is a splitting field for the i-th simple module S_i. By 1.8.4, L is the smallest field containing all the K_i. Finally by 1.10.5, L is a splitting field for each S_i, and hence for A. □

We close this section with an important point, namely that there may be more simple A^L modules than there are simple A-modules. A single simple A-module may split into several simple A^L modules when we extend scalars to L.

1.11 Morita Equivalence and Basic Algebras

Definition 1.11.1. Two categories C and D are called *equivalent* if there are covariant functors $F : C \to D$, and $G : D \to C$ such that GF (i.e., $G \circ F$) is naturally isomorphic to the identity functor on C, and FG is naturally isomorphic to the identity functor on D. Specifically for each object M of C, and N of D there are isomorphisms $\alpha_M : M \to GF(M)$ and $\beta_N : N \to FG(N)$, such that the following diagrams,

$$\begin{array}{ccc} GF(M) & \xrightarrow{GF(f)} & GF(M') \\ {\scriptstyle \alpha_M}\downarrow & & \downarrow{\scriptstyle \alpha_{M'}} \\ M & \xrightarrow{f} & M' \end{array} \qquad \begin{array}{ccc} FG(N) & \xrightarrow{FG(g)} & FG(N') \\ {\scriptstyle \beta_N}\downarrow & & \downarrow{\scriptstyle \beta_{N'}} \\ N & \xrightarrow{g} & N' \end{array}$$

commute for all morphisms $f : M \to M'$ in C, and $g : N \to N'$ in D.

Properties of objects or maps in a category are considered to be 'categorical' if these properties are preserved by arbitrary equivalences of categories.

Definition 1.11.2. The category of finitely generated left A-modules is denoted A-mod. The category of all left A-modules is denoted A-Mod.

Definition 1.11.3. Two algebras A and B are called *Morita equivalent* if A-Mod and B-Mod are equivalent.

The following theorem gives a list of categorical properties for module categories over algebras.

Theorem 1.11.4. *Let A and B be Morita equivalent algebras, with the equivalence realized by the functors $F : A\text{-}Mod \to B\text{-}Mod$, and $G : A\text{-}Mod \to B\text{-}Mod$. Then:*

(i) *A-modules M and N are isomorphic if and only if $F(M)$ and $F(N)$ are isomorphic B-modules.*

(ii) *S is a simple A-module if and only if $F(S)$ is a simple B-module.*

(iii) *P is a projective A-module if and only if $F(P)$ is a projective B-module.*

(iv) *M is an indecomposable A-module if and only if $F(M)$ is a indecomposable B-module.*

(v) *M is a finitely generated A-module if and only if $F(M)$ is a finitely generated B-module.*

(vi) *if the finitely generated A-module M has composition factors $S_1, \ldots, S_t$, with multiplicities $n_1, \ldots, n_t$, then $F(M)$ has composition factors $F(S_1), \ldots, F(S_t)$, with the same multiplicities.*

(vii) *the submodule lattices of M and $F(M)$ are isomorphic for all A-modules M.*

(viii) $\mathrm{Hom}_A(M, N)$ *and* $\mathrm{Hom}_B(F(M), F(N))$ *are isomorphic for all A-modules M. In addition* $\mathrm{End}_A(M)$ *and* $\mathrm{End}_B(F(M))$ *are isomorphic as rings for all A-modules M.*

Proof. Section 3D of [11]. □

Observe here that if A and B are Morita equivalent then by (v), A-mod is equvialent to B-mod. Observe also that by (iii) and (v), $F({}_AA)$ is a finitely generated

projective B-module. Moreover combining 1.7.2 and (viii) gives us that

$$A \simeq \mathrm{End}_A({}_AA)^{\mathrm{op}} \simeq \mathrm{End}_B(F({}_AA))^{\mathrm{op}}$$

Hence we can consider $F({}_AA)$ as a B-A bimodule by letting $A \simeq \mathrm{End}_B(F({}_AA))^{\mathrm{op}}$ act on the right of $F({}_AA)$. Similarly we can consider $G({}_BB)$ as an A-B bimodule.

Definition 1.11.5. An A-module P is called a *progenerator* of A-Mod if:

(i) P is projective.

(ii) P is finitely generated.

(iii) every A-module is a homomorphic image of a direct sum of copies of P.

The obvious example of a progenerator for A-Mod is ${}_AA$. The connection between progenerators and Morita equivalences is given by the following theorem.

Theorem 1.11.6 (Morita's Theorem). *Let A and B be Morita equivalent algebras, with the equivalence realized by the functors $F : A\text{-}Mod \to B\text{-}Mod$, and $G : A\text{-}Mod \to B\text{-}Mod$. Let $P = F({}_AA)$ considered as a B-A bimodule, and let $Q = G({}_BB)$ considered as an A-B bimodule. Then for any A-module M*

$$F(M) \simeq {}_BP \otimes_A M$$

and for any B-module N

$$G(N) \simeq {}_AQ \otimes_B N$$

Moreover $P \otimes_A Q \simeq B$ as B-B bimodules, and $Q \otimes_B P \simeq A$ as A-A bimodules.

Conversely algebras A and B are Morita equivalent if there exist bimodules ${}_BP_A$ and ${}_AQ_B$, which are projective as both left and right modules, such that as a B-B bimodule

$${}_BP \otimes_A Q_B \simeq B$$

and as an A-A bimodule

$$_AQ\otimes_B P_A \simeq A$$

In this case the Morita equivalence is realized by the functors $F = P\otimes_A - : A\text{-}Mod \to B\text{-}Mod$ *and* $G = Q\otimes_B - : B\text{-}Mod \to A\text{-}Mod$, *and* P *and* Q *are progenerators for* B*-Mod and* A*-Mod respectively.*

Proof. Theorem 3.54 of [11]. □

Given two algebras A and B it is usually extremely difficult to tell if A and B are Morita equivalent. However, starting with an algebra A we can easily construct algebras B which are Morita equivalent to A as follows.

Theorem 1.11.7. *Let* P *be a progenerator of* A*-Mod, and* $B = \mathrm{End}_A(P)^{op}$. *Then* A *and* B *are Morita equivalent, with the equivalence given by the bimodules* $_AP_B$ *and* $Q = \mathrm{Hom}_A(P,_A A)$ *(as a* B*-*A *bimodule).*

Proof. Theorem 2.2.3 of [4]. □

Corollary 1.11.8. A *is Morita equivalent to* $M_n(A)$ *for every value of* n.

Proof. For any n take $P = {}_AA^n$ (i.e., P is the free left A-module with finite basis $\{x_1, \ldots, x_n\}$. Then P is a progenerator of A-Mod, so by 1.11.7, A is Morita equivalent to $B = \mathrm{End}_A(P)^{\mathrm{op}}$. But $B \simeq M_n(A)$ via the map

$$b \mapsto (a_{ij})$$

where $x_i b = \sum_j a_{ij} x_j$. □

We can also use 1.11.7, together with the Wedderburn Structure theorem, to construct algebras Morita equivalent to a given finite dimensional algebra A. Suppose

$\overline{A} \simeq \bigoplus_{i=1}^{r} M_{n_i}(D_i)$. Corresponding to this decomposition are projective indecomposables P_i such that ${}_AA = \bigoplus_{i=1}^{r} P_i^{n_i}$. Let

$$P = \bigoplus_{i=1}^{r} P_i^{m_i}$$

for some choice of $m_i \neq 0$. Then P is a progenerator of A-Mod, so A is Morita equivalent to $B = \mathrm{End}_A(P)^{\mathrm{op}}$. Note that by the Wedderburn Structure theorem

$$B/\operatorname{J}(B) \simeq \bigoplus_{i=1}^{r} M_{m_i}(D_i)$$

If we take each $m_i = 1$ (i.e., P is equal to a direct sum of precisely one copy of each projective indecomposable A-module), then P is still a progenerator of A-Mod, so A is still Morita equivalent to B, and

$$B/\operatorname{J}(B) \simeq \bigoplus_{i=1}^{r} M_1(D_i) \simeq \bigoplus_{i=1}^{r} D_i$$

So $B/\operatorname{J}(B)$ is isomorphic to a direct sum of division algebras, and every simple B-module S_i is then 1-dimensional over the corresponding division ring D_i. Clearly simple modules cannot be any smaller than 1 dimensional over the corresponding division ring, so B is the 'smallest' K-algebra Morita equivalent to A.

This discussion motivates the following definition, and provides a proof of the following theorem.

Definition 1.11.9. A finite dimensional algebra B is called a *basic algebra* if $B/\operatorname{J}(B)$ is isomorphic to a direct sum of division algebras – i.e., in a decomposition of $B/\operatorname{J}(B)$ via the Wedderburn Structure theorem, all the matrix components are 1×1 matrix rings over division algebras. Equivalently B is called basic if every simple B-module, S, is one dimensional over $\mathrm{End}_B(S)^{\mathrm{op}}$.

Theorem 1.11.10. *Every finite dimensional algebra A is Morita equivalent to some basic algebra B.*

1.12 Faithful Representations

We finish our theoretical background with a discussion of a special kind of representation which will prove useful in the development of our algorithms.

Definition 1.12.1. Let M be an A-module. A representation $\phi : A \to \mathrm{End}_A(M)$ is called a *faithful* representation of A if ϕ is injective, or equivalently if $\mathrm{Ann}_A(M) = 0$ (i.e., the only element of A that annihilates all elements of M is 0).

The canonical example of a faithful representation is when A is a matrix algebra, since a matrix $a \in A$ annihilates a vector space M if and only if $a = 0$.

Lemma 1.12.2. *Suppose P is a projective indecomposable A-module with corresponding simple module S and division algebra D. Then for any finitely generated A-module M, $dim_D \mathrm{Hom}_A(P, M)$ is the multiplicity of S as a composition factor of M.*

Proof. Lemma 1.7.6 of [4] □

Theorem 1.12.3. *Suppose M is a finitely generated A-module, and $\phi : A \to \mathrm{End}_A(M)$ is a faithful representation. Then every simple A-module appears as a composition factor of M.*

Proof. Let e be a primitive idempotent in A. Since ϕ is faithful $eM = \phi(e)M \neq 0$. Hence by 1.7.3, $\mathrm{Hom}_A(Ae, M) \simeq eM \neq 0$. Now Ae is a projective indecomposable A-module by 1.6.7, so by the previous lemma, the simple A-module, $Ae/\,\mathrm{J}(Ae)$, corresponding to Ae, appears at least once as a composition factor of M. The result now follows by 1.6.9 since all projective indecomposable A-modules, and hence all simple A-modules arise from primitive idempotents. □

The following lemma provides a simple but very useful method for creating faithful representations.

Lemma 1.12.4. *Suppose $a \in A$ is such that a annihilates all the projective indecomposable A-modules. Then $a = 0$.*

Proof. By 1.6.7, the identity of A can be decomposed as $1_A = \sum_i e_i$, where each e_i is a primitive idempotent corresponding to a projective indecomposable Ae_i. If a annihilates all the projective indecomposables, then a annihilates each e_i, and hence

$$a = a \cdot 1 = a \cdot \sum_i e_i = \sum_i ae_i = 0 \qquad \square$$

Corollary 1.12.5. *Suppose M is a finitely generated projective A-module – so $M = \bigoplus_i P_i$, with each P_i a projective indecomposable module. If every isomorphism class of projective indecomposable A-modules occurs at least once in this decomposition, then the representation $\rho : A \to \mathrm{End}_K(M)$ is faithful.*

Proof. Suppose $a \in \ker \rho$. Then a annihilates M, and hence a annihilates every projective indecomposable. By the previous lemma, $a = 0$. $\square$

Corollary 1.12.6. *The regular representation is always faithful.*

Proof. By 1.6.7, A is a direct sum of at least one copy of every isomorphism class of projective indecomposable A-modules. $\square$

1.13 Computational Background

We conclude this chapter with some background on the algorithms and computational tools we use in later chapters.

1.13.1 The MeatAxe

Suppose that A is a finite dimensional algebra over a finite field K, and that M is an A-module of K-dimension n. In computatonal problems the A-module structure of M is usually specified by a finite set of $n \times n$ matrices over K, giving the action of A with respect to some chosen basis for M.

The first question we might ask is how to determine algorithmically whether or not M is simple, and if M is not simple, how to construct a submodule of M. In 1984, Richard Parker developed a program called the Meat-Axe, which answered this question by implementing an algorithmic criterion for simplicity originally proposed by S. Norton.

Since 1984, the Meat-Axe has been expanded into a large collection of related algorithms for computing properties of A-modules specified via finite sets of matrices. For example the Meat-Axe can:

(i) compute a composition series for M.

(ii) identify isomorphic composition factors of M.

(iii) determine socle and radical series for M.

(iv) compute a splitting field for M.

Details of these algorithms, as implemented in a more modern version of the Meat-Axe called the C-Meat-Axe, can be found in [16].

1.13.2 Gröbner Bases

Suppose R is a multivariate polynomial ring over a field K, and I is an ideal in R. Theoretically speaking we have a criterion for membership of I, namely that $f \in I$ if and only $f \equiv 0 \mod I$. Buchberger showed that this criterion can be made effective. He devised an algorithm called multivariate reduction (with respect to a set of generators for I), and showed that if f reduces under this algorithm to zero, then $f \in I$. He then defined a Gröbner basis [17] for I (with respect to some variable ordering) to be a finite subset $G \subseteq I$ such that:

(i) G generates I as an ideal.

(ii) if $f \in I$ then, under the reduction algorithm with respect to G, f reduces to zero.

Buchberger proved that every ideal contains a Gröbner basis, and that, given a finite set of generators for an ideal I, there is a terminating algorithm, called Buchberger's algorithm, to compute a Gröbner basis for I. Having a Gröbner basis for I ensures that computation in the quotient ring R/I is decidable, and that the word problem for R is solvable. In theory Buchberger's algorithm for computing a Gröbner basis can be terribly inefficient (doubly exponential in the number of variables in the worst case). In practice, however, it usually performs well.

The theory behind Gröbner bases has since been extended to many different contexts, one of which is when R is a free non-commutative algebra over a field K, and I is an ideal in R. In this case a non-commutative extension of Buchberger's algorithm can be used to compute a Gröbner basis for I. There is a caveat however, namely that the basis may not be finite, and thus the algorithm may not terminate.

1.13.3 Magma

In later chapters we develop a number of programs. These programs are all written using the Magma programming language. Magma [8, 7] is a computer algebra system designed to provide a software environment for computing with the structures which arise in algebra, geometry, number theory and (algebraic) combinatorics. Magma contains high performance implementations of many of the algorithms used in computational representation theory. Specifically we make essential use of Magma's implementation of the basic operations for linear algebra over finite fields, the non-commutative Gröbner Basis algorithms, and the Meat-Axe algorithms for computing composition series for finite dimensional modules over finite dimensional algebras over finite fields.

Chapter 2

Theory

Abstract. We present the main theoretical developments of this dissertation. We prove that there is a presentation for a full matrix algebra over a finite field given by two generators, and four families of relations. Using this presentation we show that a finite dimensional algebra over a finite field can be 'split' into the direct sum of the Jacobson radical and the algebra modulo the radical. We then prove the existence of an 'algorithmically constructible' generating set for the Jacobson radical as a two sided ideal.

2.1 A Presentation for a Full Matrix Algebra

Let K be a finite field, and L be a finite extension of K of degree d. Let A denote a full $n \times n$ matrix algebra over L.

For $1 \leq j, k \leq n$, let e_{jk} denote the jk-th elementary matrix in A. For $1 \leq j \leq n$, define $e_j = e_{jj}$ and $z_j = e_{j(j+1)}$ (all such indices should be taken modulo n, with the representatives of the n equivalence classes being 1 to n – in particular, $z_n = e_{n(n+1)} = e_{n1}$). Let ε be any primitive element of L (so $L = K(\varepsilon)$) and let $h = x^d + c_{d-1}x^{d-1} + \cdots + c_1 x + c_0 \in K[x]$ be the minimal polynomial for ε over K (so $h(\varepsilon) = 0$ and $c_0 \neq 0$). Let $\alpha = \varepsilon 1_A$. Let q denote the size of L.

Observe that A has a basis over L consisting of the e_{jk}, and a basis over K consisting of products $b_i e_{jk}$, where $b_1, \ldots, b_d$ is a basis for L over K. Hence $\dim_K(A) = dn^2$. Recall that L^* is a cyclic group of order $q-1$, so taking q-th powers in L is the identity map.

Lemma 2.1.1. *We have the following obvious facts concerning the e_{jk}, the e_j, the z_j and α:*

(i) $e_{jk}e_{lm} = \delta_{kl}e_{jm}$ *and hence* $e_je_k = \delta_{jk}e_j$.

(ii) $1_A = \sum_{j=1}^{n} e_{jj} = \sum_{j=1}^{n} e_j = \alpha^{q-1}$.

(iii) α *commutes with the* e_{jk}, *the* e_j, *and the* z_j.

(iv) $h(\alpha) = 0$.

(v) $z_j \in e_jAe_{j+1}$, *and hence* $e_kz_j = \delta_{kj}z_j$ *and* $z_je_k = \delta_{j(k+1)}z_j$.

(vi) $z_jz_k = \delta_{(j+1)k}e_{j(j+2)}$, *and hence* $z_jz_{j+1}\cdots z_nz_1\cdots z_{k-1} = e_{jk}$. *In particular* $z_1z_2\cdots z_n = e_1$, $z_2z_3\cdots z_nz_1 = e_2$, *etc.*

The next lemma states a fairly obvious fact for A, but in a quite general context. We will need this generality later.

Lemma 2.1.2. *Let B be a K-algebra. Suppose $d_1, \ldots, d_s$ are mutually orthogonal idempotents in B. Suppose also that $b_1, \ldots, b_s$ are elements of B such that for all $1 \leq j \leq s$ we have $b_j \in d_jBd_{j+1}$ (as above indices should be taken modulo s, with the representatives of the s equivalence classes being 1 to s, so $d_{s+1} = d_1$, and $b_{s+1} = b_1$). Then for $k \leq s$*

$$(\sum_{j=1}^{s} b_j)^k = \sum_{m=1}^{s} b_m b_{m+1} \cdots b_{m+(k-1)}$$

Moreover if $\sum_{j=1}^{s} d_j = 1_B$, *and* $b_jb_{j+1}\cdots b_nb_1\cdots b_{j-1} = d_j$ *for all* $1 \leq j \leq s$, *then*

$$(\sum_{j=1}^{s} b_j)^s = 1_B$$

Proof. An arbitrary term t in the expansion of $(\sum_{j=1}^{s} b_j)^k$ has the form

$$t = b_{m_1}b_{m_2}\cdots b_{m_k}$$

where each b_{m_l} is equal to one of the b_j. Now suppose that the m_l indices are not consecutive – i.e., for some l we have $m_{(l+1)} \neq m_l + 1$. Then, since $b_j = d_j b_j d_{j+1}$, we have

$$\begin{aligned} b_{m_l} b_{m_{l+1}} &= d_{m_l} b_{m_l} d_{(m_l+1)} d_{m_{(l+1)}} b_{m_{(l+1)}} d_{(m_{(l+1)}+1)} \\ &= d_{m_l} b_{m_l} \delta_{((m_l+1))(m_{(l+1)})} d_{(m_l+1)} b_{m_{(l+1)}} d_{(m_{(l+1)}+1)} \\ &= d_{m_l} b_{m_l} \cdot 0 \cdot d_{(m_l+1)} b_{m_{(l+1)}} d_{(m_{(l+1)}+1)} \\ &= 0 \end{aligned}$$

and so $t = 0$. Hence the only non-zero terms in the expansion of $(\sum_{j=1}^{s} b_j)^k$ are those with consecutive indices, and every such term occurs exactly once. Taking $k = s$ then gives

$$\begin{aligned} (\sum_{j=1}^{s} b_j)^s &= \sum_{m=1}^{s} b_m b_{m+1} \cdots b_{m+(s-1)} \\ &= b_1 b_2 \cdots b_s + b_2 b_3 \cdots b_s b_1 + \cdots + b_s b_1 b_2 \cdots b_{s-1} \\ &= d_1 + d_2 + \cdots + d_s \\ &= 1_B \end{aligned}$$

□

Corollary 2.1.3. *For $1 \leq k \leq n$ we have*

$$(\sum_{j=1}^{n} z_j)^k = \sum_{m=1}^{n} z_m z_{m+1} \cdots z_{m+(k-1)}$$

In particular then $(\sum_{j=1}^{n} z_j)^n = 1_A$.

Proof. Take $s = n$, $d_j = e_j$, and $b_j = z_j$, $1 \leq j \leq n$ in the previous lemma. □

Define the following two elements of A

$$\beta = \begin{pmatrix} \varepsilon & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \cdots & 0 \end{pmatrix}$$

and

$$\tau = \begin{pmatrix} 0 & 1 & 0 & \cdots & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & \cdots & \cdots & \cdots & 0 & 1 \\ 1 & 0 & \cdots & \cdots & \cdots & 0 \end{pmatrix}$$

Observe that τ is a permutation matrix. Multiplying $a \in A$ on the left by τ cyclically permutes the rows of a, while multiplying a on the right by τ cyclically permutes the columns of a.

Lemma 2.1.4. *We have the following basic facts concerning β and τ:*

(i) $\tau = \sum_{j=1}^{n} z_j$, *and hence* $\tau^n = 1_A$.

(ii) $e_1 = \beta^{q-1}$.

(iii) $e_{jk} = \tau^{n-(j-1)} e_1 \tau^{k-1} = \tau^{n-(j-1)} \beta^{q-1} \tau^{k-1}$. *In particular* $e_j = \tau^{n-(j-1)} \beta^{q-1} \tau^{j-1}$.

(iv) $1_A = \sum_{j=1}^{n} \tau^{n-(j-1)} \beta^{q-1} \tau^{j-1} = \sum_{j=1}^{n} \tau^{n-j} \beta^{q-1} \tau^{j}$.

(v) $\beta = e_1 \alpha e_1 = e_1 \alpha = \alpha e_1$.

(vi) $\beta^k = (e_1 \alpha e_1)^k = e_1 \alpha^k e_1$. *In particular* $\beta^q = \beta$.

(vii) $\beta h(\beta) = 0$.

(viii) $\beta \tau^k \beta = 0$ *for* $1 \leq k \leq n-1$.

(ix) $(\beta^{q-1})^2 = \beta^{q-1}$.

(x) $\alpha^k = \sum_{j=1}^{n} \tau^{n-(j-1)} \beta^k \tau^{j-1}$.

Proof. Statement (i) follows from 2.1.1. Statements (ii) and (iii) are obvious, and (iv) follows from (iii) since

$$1_A = \sum_{j=1}^{n} e_j = \sum_{j=1}^{n} \tau^{n-(j-1)} \beta^{q-1} \tau^{j-1}$$

(v) and (vi) are obvious. (vii) follows from (vi) since

$$\begin{aligned} h(\beta) &= c_n\beta^m + \cdots + c_1\beta + c_0 \cdot I \\ &= c_n(e_1\alpha e_1)^m + \cdots + c_1(e_1\alpha e_1) + c_0 \cdot I \\ &= c_n(e_1\alpha^m e_1) + \cdots + c_1(e_1\alpha e_1) + c_0 \cdot I \\ &= e_1(c_n\alpha^m + \cdots + c_1\alpha + c_0)e_1 - e_1c_0e_1 + c_0 \cdot I \\ &= e_1h(\alpha)e_1 - e_1c_0e_1 + c_0 \cdot I \\ &= -c_0e_1 + c_0 \cdot I \end{aligned}$$

and hence

$$\beta h(\beta) = (e_1\alpha e_1)(-c_0e_1 + c_0 \cdot I) = -c_0(e_1\alpha e_1) + c_0(e_1\alpha e_1) = 0$$

$(viii)$ follows from 2.1.1 and 2.1.3, since for $1 \leq k \leq n-1$ we have

$$\begin{aligned} \beta\tau^k\beta &= (e_1\alpha e_1)\tau^k(e_1\alpha e_1) \\ &= (e_1\alpha e_1)(\sum_{j=1}^{n} z_j)^k(e_1\alpha e_1) \\ &= (e_1\alpha e_1)(z_1z_2\cdots z_k + z_2z_3\cdots z_kz_1 + \cdots + z_kz_1z_2\cdots z_{k-1})(e_1\alpha e_1) \\ &= (e_1\alpha e_1)(z_1z_2\cdots z_k)(e_1\alpha e_1) + (e_1\alpha e_1)(z_2z_3\cdots z_kz_1)(e_1\alpha e_1) + \\ &\quad \cdots + (e_1\alpha e_1)(z_kz_1z_2\cdots z_{k-1})(e_1\alpha e_1) \\ &= (e_1\alpha e_1)(z_1z_2\cdots(z_ke_1))(\alpha e_1) + (e_1\alpha)(e_1z_2)(z_3\cdots z_kz_1)(e_1\alpha e_1) + \\ &\quad \cdots + (e_1\alpha e_1)(z_kz_1z_2\cdots z_{k-2})(z_{k-1}e_1)(\alpha e_1) \\ &= 0 \end{aligned}$$

Finally, (ix) follows from (ii), and (x) from (vi). □

Motivated by (i), (iv), (vii), and $(viii)$ we have the following theorem.

Theorem 2.1.5. *A has a presentation as a K-algebra as F/I, where $F = K\langle B, T\rangle$ is the free K-algebra in non-commuting variables B and T, and I is the ideal in F generated by the elements:*

(i) $T^n - 1_F$.

(ii) $\sum_{j=1}^{n} T^{n-j}B^{q-1}T^j - 1_F$.

(iii) $Bh(B)$

(iv) BT^kB *for all* $1 \leq k \leq n-1$.

Proof. Define a map $\vartheta : F \to A$ as follows:

$$B \mapsto \beta$$

$$T \mapsto \tau$$

It follows from 2.1.4 that $I \subseteq \ker \vartheta$, so ϑ induces a well defined algebra homomorphism ϑ^* from $F/I \to A$. It also follows from 2.1.4 that ϑ, and hence ϑ^*, is onto, since

$$\vartheta(T^{n-(j-1)}B^{q-1}T^{k-1}) = \tau^{n-(j-1)}\beta^{q-1}\tau^{k-1} = e_{jk}$$

and

$$\vartheta(\sum_{j=1}^{n} T^{n-(j-1)}B^lT^{j-1}) = \sum_{j=1}^{n} \tau^{n-(j-1)}\beta^l\tau^{j-1} = \alpha^l$$

and products of the e_{jk} and the α^l span A as a K-vector space.

Let m be a monomial in B and T. By (i) we can assume that if T^j is a factor of m, then $0 \leq j \leq n-1$. By (iii) we have $B^{d+1} = -(c_{d-1}B^d + \cdots + c_0B)$, and hence we can assume that if B^k is a factor of m, then $1 \leq k \leq d$.

Suppose that m starts with B^k. Then the possibilities for m are as follows (all congruences are modulo I):

- $m = B^k \equiv T^0 B^k T^0$
- $m = B^k T^j \equiv T^0 B^k T^j$

These are the only possiblities for m, since if m starts with $B^k T^j B^{k'}$, then $m \equiv 0$ by (i) and (iv).

Suppose that m starts with T^j. Then the possibilities for m are as follows:

- $m = T^j$, in which case

$$\begin{aligned} m &\equiv 1_F \cdot m \\ &\equiv (\sum_{i=1}^{n} T^{n-i} B^{q-1} T^i) T^j \\ &\equiv \sum_{i=1}^{n} T^{n-i} B^{q-1} T^{k_i} \text{ where } k_i = (i+j) \bmod n \end{aligned}$$

- $m = T^j B^k \equiv T^j B^k T^0$
- $m = T^j B^k T^{j'}$

Again these are the only possiblities for m, since if m starts with $T^j B^k T^{j'} B^{k'}$, then $m \equiv 0$ by (iv).

Hence any non-zero element of F/I can be written as a K-linear combination of products of the form $T^j B^k T^l$ with $0 \leq j, l \leq n-1$ and $1 \leq k \leq d$. Hence F/I has K-dimension at most dn^2. Since ϑ^* is onto a space of K-dimension exactly dn^2, we conclude that ϑ^* is an algebra isomorphism. □

Corollary 2.1.6. *A is generated as a K-algebra by the elements β and τ.*

Proof. By 2.1.4, β and τ satisfy the relations of 2.1.5 (identifying β with B and τ with T). □

2.2 Splitting the Radical: Part One

Let K be a finite field of characteristic p, and L be a finite extension of K of degree d. Let $\overline{A} = A/\operatorname{J}(A)$, and suppose that $\overline{A} \simeq M_n(L)$.

For $1 \leq j, k \leq n$, let $\overline{e}_{jk}$ denote the jk-th elementary matrix in $\overline{A}$. For $1 \leq j \leq n$, define $\overline{e}_j = \overline{e}_{jj}$ and $\overline{z}_j = \overline{e}_{j(j+1)}$ (all such indices should be taken modulo n, with the representatives of the n equivalence classes being 1 to n – in particular $\overline{z}_n = \overline{e}_{n(n+1)} = \overline{e}_{n1}$). Taking I and $F = K\langle B, T\rangle$ as they are in 2.1.5, we know that $F/I \overset{\vartheta^*}{\simeq} \overline{A}$. Let $\overline{B}$ and $\overline{T}$ denote the images in F/I of B and T respectively, and let $\overline{E}_j = \vartheta^{*-1}(\overline{e}_j)$, and $\overline{Z}_j = \vartheta^{*-1}(\overline{z}_j)$.

By 2.1.5, $\overline{A}$ is generated by elements $\overline{\beta} = \vartheta(\overline{B})$ and $\overline{\tau} = \vartheta(\overline{T})$, with associated polynomial $h = x^d + c_{d-1}x^{d-1} + \cdots + c_1 x + c_0 \in K[x]$ (so $\overline{\beta}h(\overline{\beta}) = 0$). Note that $\overline{B}$, $\overline{T}$, the $\overline{E}_j$ and the $\overline{Z}_j$, satisfy the same relations in F/I that $\overline{\beta}$, $\overline{\tau}$, and the $\overline{e}_j$ and $\overline{z}_j$ do in $\overline{A}$.

We can express the identity of F/I as a sum of mutually orthogonal primitive idempotents as

$$1_{F/I} = \sum_{i=1}^{n} \overline{E}_j$$

By the idempotent lifting theory, there exists a lift of this sum to an expression for the identity of A as

$$1_A = \sum_{i=1}^{n} e_j$$

where the e_j are mutually orthogonal primitive idempotents in A.

Theorem 2.2.1. *Let K be a finite field of characteristic p, and A be a finite dimensional K-algebra. Suppose that $\overline{A} = A/\operatorname{J}(A)$ is isomorphic to $M_n(L)$, where L is a finite extension of K. Then the exact sequence*

$$0 \to J(A) \to A \xrightarrow{\mu} \overline{A} \to 0$$

is right split by an algebra homomorphism. That is, there is an algebra homorphism $\psi : \overline{A} \to A$ *such that* $\mu \circ \psi = 1_{\overline{A}}$*, and as vector spaces*

$$A = A' \oplus \mathrm{J}(A)$$

where $A' = \operatorname{im} \psi$ *is a subalgebra of* A *isomorphic to* $\overline{A}$*.*

Proof. By 2.1.5 we have the exact sequence

$$0 \to J(A) \to A \xrightarrow{\mu'} F/I \to 0$$

where μ' is the composition

$$A \xrightarrow{\mu} \overline{A} \xrightarrow{\vartheta^{*-1}} F/I$$

We will define $\omega : F \to A$ by giving images for B and T, and show that it induces a well defined algebra homomorphism $\omega^* : F/I \to A$ that splits μ'. Taking $\psi = \omega^* \circ \vartheta^{*-1} : \overline{A} \to A$ will then give us the required splitting for μ.

Let $\gamma \in A$ be any inverse image under μ' of $\overline{B}$. Then

$$\mu'(\gamma h(\gamma)) = \mu'(\gamma) h(\mu'(\gamma)) = \overline{B} h(\overline{B}) = 0$$

Hence $\gamma h(\gamma) \in \mathrm{J}(A)$, and thus for some j we have $(\gamma h(\gamma))^j = 0$. Define

$$\beta = \gamma^{q^k}$$

where k is any integer such that $q^k \geq j$.

Let $z_j \in A$, $1 \leq j \leq n-1$, be any inverse image under μ' of $\overline{Z}_j$. Let $z'_n \in A$ be any inverse image under μ' of $\overline{Z}_n$. We can take z_j to be in $e_j A e_{j+1}$, so $e_j z_j = z_j$ and $z_j e_{j+1} = z_j$. Similarly we can take z'_n to be in $e_n A e_1$, so $e_n z'_n = z'_n$ and $z'_n e_1 = z'_n$. Let $x = z_1 z_2 \cdots z_{n-1}$. Observe that

$$\mu'(x z'_n - e_1) = \overline{Z}_1 \overline{Z}_2 \cdots \overline{Z}_{n-1} \overline{Z}_n - \overline{E}_1 = \overline{E}_1 - \overline{E}_1 = 0$$

Hence $xz'_n - e_1 \in \mathrm{J}(A)$, and so for some l we have that $(xz'_n - e_1)^{p^l} = 0$. Define

$$z_n = z'_n(xz'_n)^{p^l-1}$$

and

$$\tau = \sum_{j=1}^{n} z_j$$

Observe that $z_n \in e_nAe_1$, and z_n is an inverse image under μ' of $\overline{Z}_n$, since

$$\mu'(z_n) = \mu'(z'_n(xz'_n)^{p^l-1}) = \mu'(z'_n)\mu'(xz'_n)^{p^l-1} = \overline{Z}_n\overline{E}_1{}^{p^l-1} = \overline{Z}_n\overline{E}_1 = \overline{Z}_n$$

Define the algebra homomorphism $\omega : F \to A$ by

$$B \mapsto \beta$$

$$T \mapsto \tau$$

To prove that ω induces a well defined algebra homomorphism $\omega^* : F/I \to A$, we must show that $I \subseteq \ker\omega$, or equivalently that β and τ satisfy the relations implied by 2.1.5.

To prove that β and τ satisfy (iii) of 2.1.5, we first observe that since h divides $x^{q-1}-1$, and x divides x^{q-1}, we have that xh divides $(x^{q-1}-1)x^{q-1} = (x^{q-1})^2 - x^{q-1}$. Since $\beta h(\beta) = 0$ by construction, it then follows that $(\beta^{q-1})^2 - \beta^{q-1} = 0$, and thus that β^{q-1} is idempotent. Thus for any j

$$(\tau^{n-j}\beta^{q-1}\tau^j)^2 = \tau^{n-j}\beta^{q-1}\tau^j\tau^{n-j}\beta^{q-1}\tau^j = \tau^{n-j}(\beta^{q-1})^2\tau^j = \tau^{n-j}\beta^{q-1}\tau^j$$

and so $\tau^{n-j}\beta^{q-1}\tau^j$ is idempotent. Since

$$\tau^{n-j}\beta^{q-1}\tau^j e_{j+1} = e_{j+1}\tau^{n-j}\beta^{q-1}\tau^j = e_{j+1}$$

it follows that $\tau^{n-j}\beta^{q-1}\tau^j - e_{j+1}$ is also idempotent, since,

$$(\tau^{n-j}\beta^{q-1}\tau^j - e_{j+1})^2 = (\tau^{n-j}\beta^{q-1}\tau^j)^2 - \tau^{n-j}\beta^{q-1}\tau^j e_{j+1}$$

$$\begin{aligned} &- e_{j+1}\tau^{n-j}\beta^{q-1}\tau^{j} + {e_{j+1}}^{2} \\ &= (\tau^{n-j}\beta^{q-1}\tau^{j}) - e_{j+1} - e_{j+1} + e_{j+1} \\ &= \tau^{n-j}\beta^{q-1}\tau^{j} - e_{j+1} \end{aligned}$$

Finally observe that

$$\mu'(\tau^{n-j}\beta^{q-1}\tau^{j} - e_{j+1}) = \overline{T}^{n-j}\overline{B}^{q-1}\overline{T}^{j} - \overline{E}_{j+1} = \overline{E}_{j+1} - \overline{E}_{j+1} = 0$$

Thus $\tau^{n-j}\beta^{q-1}\tau^{j} - e_{j+1}$ is both idempotent and in $\mathrm{J}(A)$, which is only possible if $\tau^{n-j}\beta^{q-1}\tau^{j} - e_{j+1} = 0$, and thus $\tau^{n-j}\beta^{q-1}\tau^{j} = e_{j+1}$. Consequently we have that

$$\sum_{j=1}^{n} \tau^{n-j}\beta^{q-1}\tau^{j} = \sum_{j=1}^{n} e_{j+1} = 1_A$$

To prove that β satisfies (iii) of 2.1.5, observe that taking q-th powers is the identity map in L, so by the usual characteristic p binomial expansion we have

$$\begin{aligned} (h(\gamma))^{q^k} &= (\gamma^{q^k})^d + (c_{d-1})^{q^k}(\gamma^{q^k})^{d-1} + \cdots + {c_1}^{q^k}(\gamma^{q^k}) + {c_0}^{q^k} \\ &= (\gamma^{q^k})^d + c_{d-1}(\gamma^{q^k})^{d-1} + \cdots + c_1(\gamma^{q^k}) + c_0 \\ &= h(\gamma^{q^k}) \end{aligned}$$

Hence

$$\beta h(\beta) = \gamma^{q^k} h(\gamma^{q^k}) = \gamma^{q^k} h(\gamma)^{q^k} = (\gamma h(\gamma))^{q^k} = 0$$

To prove that β satisfies (iv) of 2.1.5, note that since $\beta \in e_1 A e_1$ and $z_k \in e_k A e_{k+1}$, $\beta z_k = 0$ if $k \neq 1$ and $z_k \beta = 0$ if $k \neq n$. Thus for $1 \leq k \leq n-1$

$$\begin{aligned} \beta\tau^k\beta &= \beta(\sum_{j=1}^{n} z_j)^k\beta \\ &= \beta\left(\sum_{m=1}^{n} z_m z_{m+1}\cdots z_{m+(k-1)}\right)\beta \\ &= \sum_{m=1}^{n} \beta z_m z_{m+1}\cdots z_{m+(k-1)}\beta \end{aligned}$$

$$= 0$$

The second step follows from 2.1.2. The last step follows from that fact that if $m \neq 1$ then $\beta z_m = 0$, and if $m = 1$, then $m + (k-1) < n$, and hence $z_{m+(k-1)}\beta = 0$.

To prove that τ satisfies (i) of 2.1.5, first observe that $xz_n' \in e_1 A e_1$, so xz_n' and e_1 commute. Hence

$$0 = (xz_n' - e_1)^{p^l} = (xz_n')^{p^l} - e_1^{p^l} = (xz_n')^{p^l} - e_1$$

and thus $(xz_n')^{p^l} = e_1$. It then follows that

$$z_1 z_2 \cdots z_{n-1} z_n = xz_n = xz_n'(xz_n')^{p^l - 1} = (xz_n')^{p^l} = e_1$$

and therefore

$$z_n(z_1 z_2 \cdots z_{n-1} z_n) = z_n e_1 = z_n$$

For $1 \leq i \leq n-1$ define $u_i = z_i \cdots z_n z_1 \cdots z_{i-1}$. Observe that $u_i \neq 0$ since u_i is an inverse image under μ' of $\overline{Z}_i \cdots \overline{Z}_n \overline{Z}_1 \cdots \overline{Z}_{i-1} = \overline{E}_i$. Observe also that u_i is idempotent, since

$$\begin{aligned} u_i &= z_i \cdots z_n z_1 \cdots z_{i-1} \\ &= z_i \cdots z_n (z_1 \cdots z_n) z_1 \cdots z_{i-1} \\ &= (z_i \cdots z_n z_1 \cdots z_{i-1})(z_i \cdots z_n z_1 \cdots z_{i-1}) \\ &= {u_i}^2 \end{aligned}$$

Since $e_i u_i = u_i$ and $u_i e_i = u_i$, it then follows that $e_i - u_i$ is also idempotent, since,

$$(e_i - u_i)^2 = {e_i}^2 - e_i u_i - u_i e_i + {u_i}^2 = e_i - u_i - u_i + u_i = e_i - u_i$$

Now $e_i = (e_i - u_i) + u_i$, and

$$(e_i - u_i)u_i = e_i u_i - {u_i}^2 = u_i - u_i = 0$$

If $e_i \neq u_i$ then we have a decomposition of e_i into mutually orthogonal idempotents $e_i - u_i$ and u_i, contradicting the primitivity of e_i. Hence $e_i = u_i$, and thus for $1 \leq i \leq n-1$

$$z_i \cdots z_n z_1 \cdots z_{i-1} = u_i = e_i$$

Since we have previously established that $z_1 z_2 \cdots z_{n-1} z_n = e_1$, we therefore have that $\tau^n = 1$ by 2.1.2.

Since β and τ satisfy the relations implied by 2.1.5, ω induces a well defined homomorphism $\omega^* : F/I \to A$. Finally

$$\mu'(\omega^*(\overline{B})) = \mu'(\beta) = \mu'(\gamma^{q^k}) = \mu'(\gamma)^{q^k} = \overline{B}^{q^k} = \overline{B}$$

and

$$\mu'(\omega^*(\overline{T})) = \mu'(\tau) = \mu'(\sum_{j=1}^{n} z_j) = \sum_{j=1}^{n} \mu'(z_j) = \sum_{j=1}^{n} \overline{Z}_j = \overline{T}$$

Hence $\mu' \circ \omega^* = 1_{F/I}$, from which it immediately follows that $\mu \circ \psi = 1_{\overline{A}}$. □

To put the previous theorem in perspective, note that if A is a K-algebra, then we can always split A as a vector space direct sum $A = A' \oplus \mathrm{J}(A)$ – so $A' \simeq \overline{A}$ as vector spaces. As long as K is perfect, such a vector space splitting in fact gives us an algebra isomorphism between A' and $\overline{A}$, which then allows us to realize $\overline{A}$ as a subalgebra of A (see Corollary 4.1.11 of [4] for details). The point of the above theorem is that when K is a finite field we can realize this splitting constructively (modulo finding inverse images for $\overline{B}$ and the $\overline{Z}_j$ under μ').

Remark. Henceforth we will identify a matrix algebra $M_n(L)$ with its presentation F/I as given in 2.1.5.

2.3 Splitting the Radical: Part Two

Let A be a finite dimensional algebra over a field K of characteristic p. Let $S_1, \ldots, S_r$ denote representatives for the isomorphism classes of simple A-modules.

Let $\overline{A} = A/\operatorname{J}(A)$, and let μ denote the canonical map from A to $A/\operatorname{J}(A)$. Since $\overline{A}$ is semisimple, by Wedderburn's theorems we know that

$$\overline{A} \simeq \bigoplus_{i=1}^{r} A_i$$

where $A_i = M_{n_i}(K_i)$ and K_i is a finite extension of K of degree d_i. Let q_i denote the size of K_i, and $\pi_i : \overline{A} \to A_i$ denote the projection map.

We can express $1 \in \overline{A}$ as a sum of mutually orthogonal idempotents as

$$1_{\overline{A}} = \sum_{i=1}^{r} \overline{f}_i$$

(under the identification of $\overline{A}$ with $\bigoplus_{i=1}^{r} A_i$, $\overline{f}_i$ corresponds to $(0, 0, \ldots, 1_{A_i}, 0, \ldots, 0) \in A_i$ – i.e., the identity matrix of A_i in the i-th component, and zero elsewhere). By the idempotent lifting theory, there exists a lift of this sum to an expression

$$1_A = \sum_{i=1}^{r} f_i$$

with the f_i are mutually orthogonal idempotents in A.

Lemma 2.3.1. *We have the following basic facts concerning the f_i:*

(i) *$f_i A f_i$ is a subalgebra of A with identity f_i.*

(ii) *for $i \neq j$, $f_i A f_j \subseteq \operatorname{J}(A)$.*

(iii) $\operatorname{J}(f_i A f_i) = f_i A f_i \cap \operatorname{J}(A) = f_i \operatorname{J}(A) f_i$.

(iv) $\mu(f_i A f_i) = (f_i A f_i)/\operatorname{J}(f_i A f_i) \simeq A_i$.

Proof. See 1.6.11, and 1.7.10. □

Lemma 2.3.2. *Let K be a finite field of characteristic p, and A be a finite dimensional K-algebra. Then*

$$\operatorname{J}(A) = \bigoplus_{i} \operatorname{J}(f_i A f_i) \oplus \bigoplus_{i,j,i\neq j} (f_i A f_j)$$

Proof. Clearly

$$\bigoplus_i \mathrm{J}(f_iAf_i) \oplus \bigoplus_{i,j,i\neq j} (f_iAf_j) = \bigoplus_i (f_iAf_i \cap \mathrm{J}(A)) \oplus \bigoplus_{i,j,i\neq j} (f_iAf_j)$$
$$\subseteq \mathrm{J}(A)$$

Observe that

$$\begin{aligned} A &= 1 \cdot A \cdot 1 \\ &= (\sum_{i=1}^{r} f_i) A (\sum_{j=1}^{r} f_j) \\ &= \bigoplus_{i,j} (f_iAf_j) \\ &= \bigoplus_i (f_iAf_i) \oplus \bigoplus_{i,j,i\neq j} (f_iAf_j) \end{aligned}$$

Hence if $x \in \mathrm{J}(A)$, then $x = \sum_{i,j} x_{ij}$, with $x_{ij} \in f_iAf_j$. Now

$$f_ixf_j = f_i(\sum_{i,j} x_{ij})f_j = x_{ij}$$

Since $\mathrm{J}(A)$ is a two sided ideal, $f_ixf_j \in \mathrm{J}(A)$. Hence $x_{ij} \in \mathrm{J}(A) \cap f_iAf_j$, and thus

$$\begin{aligned} \mathrm{J}(A) &\subseteq \bigoplus_{i,j} (\mathrm{J}(A) \cap f_iAf_j) \\ &= \bigoplus_i (\mathrm{J}(A) \cap f_iAf_i) \oplus \bigoplus_{i,j,i\neq j} (\mathrm{J}(A) \cap f_iAf_j) \\ &= \bigoplus_i \mathrm{J}(f_iAf_i) \oplus \bigoplus_{i,j,i\neq j} f_iAf_j \end{aligned}$$

□

Theorem 2.3.3. *Let K be a finite field of characteristic p, and A be a finite dimensional K-algebra. Then the exact sequence*

$$0 \to J(A) \to A \xrightarrow{\mu} \overline{A} \to 0$$

is right split by an algebra homomorphism. That is, there is an algebra homorphism $\psi : \overline{A} \to A$ such that $\mu \circ \psi = 1_{\overline{A}}$, and

$$A = A' \oplus \mathrm{J}(A)$$

where $A' = \operatorname{im}\psi$ is a subalgebra of A isomorphic to $\overline{A}$.

Proof. For each $1 \leq i \leq r$ we have the exact sequence

$$0 \to \mathrm{J}(f_iAf_i) \to f_iAf_i \xrightarrow{\mu_i} A_i \to 0$$

where μ_i is the composition

$$f_iAf_i \xhookrightarrow{\iota} A \xrightarrow{\mu} \overline{A} \xrightarrow{\pi_i} A_i$$

By 2.2.1 each μ_i is right split by an algebra homomorphism $\psi_i : A_i \to f_iAf_i$, so that $f_iAf_i = A_i' \oplus \mathrm{J}(f_iAf_i)$, where $A_i' \overset{\psi_i}{\simeq} (f_iAf_i)/\mathrm{J}(f_iAf_i) \simeq A_i$ as a K-algebra. Define $A' = \bigoplus_{i=1}^{r} A_i'$ and $\psi : \overline{A} \to A$ by $\psi = \bigoplus_{i=1}^{r} \psi_i$. The orthogonality of the f_i ensures that ψ splits μ, and hence that $A' \overset{\psi}{\simeq} \overline{A}$, and as vector spaces

$$\begin{aligned}
A &= \bigoplus_{i=1}^{r} (f_iAf_i) \oplus \bigoplus_{\substack{i,j=1 \\ i\neq j}}^{r} (f_iAf_j) \\
&= \bigoplus_{i=1}^{r} (A_i' \oplus \mathrm{J}(f_iAf_i)) \oplus \bigoplus_{\substack{i,j=1 \\ i\neq j}}^{r} (f_iAf_j) \\
&= \bigoplus_{i=1}^{r} A_i' \oplus \bigoplus_{i=1}^{r} \mathrm{J}(f_iAf_i) \oplus \bigoplus_{\substack{i,j=1 \\ i\neq j}}^{r} (f_iAf_j) \\
&= A' \oplus \mathrm{J}(A)
\end{aligned}$$

□

As before the point of this theorem is that when K is a finite field, a vector space splitting of A as $A = A' \oplus \mathrm{J}(A)$ allows us to constructively realize $\overline{A}$ as a subalgebra of A.

The algebra A' will henceforth be referred to as the *semisimple part of A*. We summarize its key properties in the following corollary.

Corollary 2.3.4. *$A' = \bigoplus_{i=1}^{r} A_i'$ is a subalgebra of A which is isomorphic to $\overline{A}$. Each component A_i' is a subalgebra of A' isomorphic to the corresponding Wedderburn component $A_i = M_{n_i}(K_i)$. A' is generated as a K-algebra by elements $\{\beta_1, \tau_1, \ldots, \beta_r, \tau_r\}$,*

where each pair (β_i, τ_i) generates A_i' and satisfies the relations implied by 2.1.5 (identifying β_i with B, τ_i with T, and $1_{A_i'}$ with 1_F in the theorem).

2.4 Generating the Radical of A

Let A be a finite dimensional algebra over a field K of characteristic p. We previously established that $A = A' \oplus \mathrm{J}(A)$, and gave a generating set for A' as a K-algebra. We now turn to the problem of generating the radical of A as a two-sided ideal. Throughout this section we identify A_i and $A_i' = f_i A f_i / \mathrm{J}(f_i A f_i)$. For $1 \leq i \leq r$, let $\psi_i : A_i \to f_i A f_i$ be defined as in 2.3.3.

Lemma 2.4.1. *For any i and any $\lambda \in A$, $f_i(\lambda - \psi_i(\mu(\lambda)))f_i \in \mathrm{J}(A)$. Specifically*

$$f_i(\lambda - \psi_i(\mu(\lambda)))f_i = f_i(\lambda - \psi(\mu(\lambda)))f_i$$

Proof. By 2.3.3, $\lambda - \psi(\mu(\lambda)) \in \mathrm{J}(A)$ for any k. Since $\mathrm{J}(A)$ is a two sided ideal it follows that $f_i(\lambda - \psi(\mu(\lambda)))f_i \in \mathrm{J}(A)$ for any i and k. Now for $i \neq j$, f_i and $\psi_j(\mu(\lambda))$ are in different components of A, and so $f_i \psi_j(\mu(\lambda)) f_i = 0$. Thus

$$\begin{aligned}
f_i(\lambda - \psi(\mu(\lambda)))f_i &= f_i \left(\lambda - \sum_{j=1}^{r} \psi_j(\mu(\lambda)) \right) f_i \\
&= f_i \lambda f_i - f_i \left(\sum_{j=1}^{r} \psi_j(\mu(\lambda)) \right) f_i \\
&= f_i \lambda f_i - \sum_{j=1}^{r} f_i \psi_j(\mu(\lambda)) f_i \\
&= f_i \lambda f_i - f_i \psi_i(\mu(\lambda)) f_i \\
&= f_i(\lambda - \psi_i(\mu(\lambda)))f_i
\end{aligned}$$

□

Theorem 2.4.2. *Suppose that A is a finite dimensional K-algebra generated by elements $\lambda_1, \ldots, \lambda_t$. Then $\mathrm{J}(A)$ is generated as a two sided ideal by elements of A of the form:*

(i) $f_i\lambda_k f_j$, $i \neq j$.

(ii) $f_i(\lambda_k - \psi_i(\mu(\lambda_k)))f_i$.

where $1 \leq k \leq t$.

Proof. Let I be the ideal in A generated by elements of the given form. Observe that $f_i\lambda_k f_j \in f_iAf_j$, and if $i \neq j$, then $f_iAf_j \subseteq J(A)$ by 2.3.1. Hence elements of form (i) are in $\mathrm{J}(A)$. Elements of form (ii) are in $\mathrm{J}(A)$ by the previous lemma. Thus $I \subseteq \mathrm{J}(A)$.

Now suppose $a \in \mathrm{J}(A)$. Since $a \in A$, we can write a as a K-linear combination of products of the generators

$$\lambda_k = \left(\sum_{j=1}^{r} f_i\right) \lambda_k \left(\sum_{j=1}^{r} f_i\right)$$

Subtracting from a elements of the two-sided ideal generated by elements of form (i) leaves us with a linear combination of products of the form

$$f_i\lambda_{k_1} f_i\lambda_{k_2} f_i \cdots f_i\lambda_{k_s} f_i$$

Writing λ_{k_j} as $\lambda_{k_j} - \psi(\mu(\lambda_{k_j})) + \psi(\mu(\lambda_{k_j}))$, and using the previous lemma, we can subtract from a elements of the two-sided ideal generated by elements of form (ii), and be left with a linear combination of products of the form

$$\begin{aligned} & f_i\psi(\mu(\lambda_{k_1}))f_i\psi(\mu(\lambda_{k_2}))f_i \cdots f_i\psi(\mu(\lambda_{k_s}))f_i \\ = \ & \psi(\mu(f_i\lambda_{k_1} f_i\lambda_{k_2} f_i \cdots f_i\lambda_{k_s} f_i)) \end{aligned}$$

Hence we have

$$a = x + y$$

with $x \in I$, and $y \in \operatorname{im}\psi$. Now $y = a - x \in \mathrm{J}(A)$, so $y \in \mathrm{J}(A) \cap \operatorname{im}\psi$. By 2.3.3 we have $\mathrm{J}(A) \cap \operatorname{im}\psi = 0$, so $y = 0$, and thus $a \in I$. Hence $\mathrm{J}(A) = I$. $\square$

The arguments used in the previous lemma and theorem in fact establish a more interesting generating set for $\mathrm{J}(A)$ as a two-sided ideal.

For $1 \leq j \leq n_i$ let $\overline{g}_{ij} \in A_i$ denote the jj-th elementary matrix in A_i – that is the matrix with 1 in the j-th row and column, and zero elsewhere. The $\overline{g}_{ij}$ are mutually orthogonal primitive idempotents in A_i with $1_{A_i} = \sum_{j=1}^{n_i} \overline{g}_{ij}$. By the idempotent lifting theorem we can lift these idempotents to mutually orthogonal primitive idempotents $g_{ij} \in f_i A f_i$, $1 \leq j \leq n_i$, such that $1_{f_i A f_i} = f_i = \sum_{j=1}^{n_i} g_{ij}$. Define $\mathfrak{g}_i = g_{i1}$.

Theorem 2.4.3. *Suppose that A is a finite dimensional K-algebra generated by elements $\lambda_1, \ldots, \lambda_t$. Then $\mathrm{J}(A)$ is generated as a two sided ideal by elements of A of the form:*

(i) $\mathfrak{g}_i \tau_i^a \lambda_k \tau_j^c \mathfrak{g}_j$, $i \neq j$.

(ii) $\mathfrak{g}_i \tau_i^a (\lambda_k - \psi_i(\mu(\lambda_k))) \tau_i^b \mathfrak{g}_i$.

where $1 \leq k \leq t$, $1 \leq a, b \leq n_i$, and $1 \leq c \leq n_j$.

Proof. Let I' be the ideal in A generated by elements of the given form. Let I be as in 2.4.2. Since

$$f_i = \sum_{j=1}^{n_i} g_{ij} = \sum_{j=1}^{n_i} \tau_i^{n_i - j} \mathfrak{g}_i \tau_i^j$$

it follows that $I \subseteq I'$. The argument used in 2.4.2 to show that $f_i \lambda_k f_j \in \mathrm{J}(A)$, for $j \neq i$, also shows that elements of form (i) are in $\mathrm{J}(A)$. The argument used in 2.4.1 to show that $f_i(\lambda_k - \psi_i(\mu(\lambda_k)))f_i \in \mathrm{J}(A)$ also shows that elements of form (ii) are in $\mathrm{J}(A)$ (replace f_i on the left with $\mathfrak{g}_i \tau_i^a$, and f_i on the right with $\tau_i^b \mathfrak{g}_i$). Since $I = \mathrm{J}(A)$, we have that $I' = I$. □

Corollary 2.4.4. *A is generated as a K-algebra by elements, $\beta_1, \tau_1, \ldots, \beta_r, \tau_r$ generating the semisimple part of A as a K-algebra, and elements*

(i) $\mathfrak{g}_i \tau_i^a \lambda_k \tau_j^c \mathfrak{g}_j$, $i \neq j$

(ii) $\mathfrak{g}_i\tau_i^a(\lambda_k - \psi_i(\mu(\lambda_k)))\tau_i^b\mathfrak{g}_i$

$1 \leq k \leq t$, $1 \leq a, b \leq n_i$, $1 \leq c \leq n_j$, *which generate* $\mathrm{J}(A)$ *as a two-sided ideal.*

Proof. The result follows from 2.3.4, 2.4.3, and 2.3.3. □

2.5 The Basic Algebra Associated to A

We are now in a position to describe the basic algebra associated to A. Let $\mathfrak{g} = \sum_{i=1}^{r} \mathfrak{g}_i$ and $B = \mathfrak{g}A\mathfrak{g}$.

Theorem 2.5.1. *The basic algebra associated to A is the K-algebra B.*

Proof. By 1.6.9 the isomorphism classes of projective indecomposables have representatives $A\mathfrak{g}_i$, $1 \leq i \leq r$. Hence by 1.11.10 the basic algebra associated to A is

$$\begin{aligned}
\mathrm{End}_A(\bigoplus_{i=1}^{r} A\mathfrak{g}_i)^{\mathrm{op}} &= \mathrm{End}_A(A(\sum_{i=1}^{r} \mathfrak{g}_i))^{\mathrm{op}} \\
&\simeq (\sum_{i=1}^{r} \mathfrak{g}_i)A(\sum_{i=1}^{r} \mathfrak{g}_i) \\
&= \mathfrak{g}A\mathfrak{g} \\
&= B
\end{aligned}$$

□

By the Wedderburn Structure theorem

$$\overline{B} \simeq \bigoplus_{i=1}^{r} B_i$$

where $B_i = M_{n_i}(K_i)$, where K_i is a finite extension of K of degree d_i. Since B is basic, $n_i = 1$, so in fact $B_i \simeq K_i$. Let q_i denote the size of K_i.

Lemma 2.5.2. *Each B_i is generated as a K-algebra by an element β_i satisfying the relations:*

(i) $\beta_i^{q_i-1} = 1_{B_i}$.

(ii) $\beta_i h_i(\beta_i) = 0$.

where β_i corresponds to a primitive element $\varepsilon_i \in K_i$, and h_i is the minimal polynomial of ε_i over K.

Proof. Observe that

$$\mathfrak{g}\beta_i\mathfrak{g} = \mathfrak{g}_i\beta_i\mathfrak{g}_i = \beta_i$$

and

$$\mathfrak{g}\tau_i\mathfrak{g} = \mathfrak{g}_i\tau_i\mathfrak{g}_i = 0$$

The result now follows from 2.1.5. □

As before we can split each $\mathfrak{g}_i A\mathfrak{g}_i$, and hence split $\mathfrak{g}A\mathfrak{g}$. Specifically we have the following results.

Lemma 2.5.3. $\mathfrak{g}_i A\mathfrak{g}_i = B_i' \oplus \mathrm{J}(\mathfrak{g}_i A\mathfrak{g}_i)$, *where B_i' is isomorphic as a K-algebra to $\mathfrak{g}_i A\mathfrak{g}_i/\,\mathrm{J}(\mathfrak{g}_i A\mathfrak{g}_i)$, and $B_i' \simeq B_i$.*

Proof. The first claim follows from 2.2.1, and the second from 2.3.1. □

Theorem 2.5.4. *The exact sequence*

$$0 \to J(B) \to B \xrightarrow{\mu} \overline{B} \to 0$$

is right split by an algebra homomorphism. That is, there is an algebra homorphism $\psi : \overline{B} \to B$ such that $\mu \circ \psi = 1_{\overline{B}}$, and

$$B = B' \oplus \mathrm{J}(B)$$

where $B' = \operatorname{im}\psi = \sum_{i=1}^{r} B_i'$ is a subalgebra of B isomorphic to $\overline{B}$.

Proof. Apply 2.3.3 with $f_i = \mathfrak{g}_i$. □

Corollary 2.5.5. *$B' = \bigoplus_{i=1}^{r} B'_i$ is a subalgebra of B which is isomorphic to $\overline{B}$. Each component B'_i is a subalgebra of B' isomorphic to the corresponding Wedderburn component $B_i = K_i$. B' is generated as a K-algebra by elements $\{\beta_1, \ldots, \beta_r\}$, where each β_i generates B'_i and satisfies the relations of 2.5.2.*

Just as we can describe generators for B' we can also easily describe generators for $\mathrm{J}(B)$.

Lemma 2.5.6. *$\mathrm{J}(B)$ is generated as a two sided ideal by elements of the form:*

(i) $\mathfrak{g}_i \tau_i^a \lambda_k \tau_j^c \mathfrak{g}_j$, $i \neq j$

(ii) $\mathfrak{g}_i \tau_i^a (\lambda_k - \psi_i(\mu(\lambda_k))) \tau_i^b \mathfrak{g}_i$

where $1 \leq k \leq t$, $1 \leq a, b \leq n_i$, $1 \leq c \leq n_j$.

Proof. By 1.6.11

$$\mathrm{J}(B) = \mathrm{J}(\mathfrak{g}A\mathfrak{g}) = \mathfrak{g}A\mathfrak{g} \cap \mathrm{J}(A) = B \cap \mathrm{J}(A)$$

The result now follows from the fact that the generators for $\mathrm{J}(A)$ given in 2.4.3 are all elements of B. □

Corollary 2.5.7. *B is generated as a K-algebra by elements, $\beta_1, \ldots, \beta_r$, generating the semisimple part of B as a K-algebra, and elements of the form:*

(i) $\mathfrak{g}_i \tau_i^a \lambda_k \tau_j^c \mathfrak{g}_j$, $i \neq j$

(ii) $\mathfrak{g}_i \tau_i^a (\lambda_k - \psi_i(\mu(\lambda_k))) \tau_i^b \mathfrak{g}_i$

$1 \leq k \leq t$, $1 \leq a, b \leq n_i$, $1 \leq c \leq n_j$, which generate $\mathrm{J}(A)$ as a two-sided ideal.

Finally we can describe the split basic algebra associated to A.

Lemma 2.5.8. *Suppose $K = \mathbb{F}_{p^c}$, and that each K_i in the Wedderburn decomposition of B is a degree d_i extension field of K. Let $l = c \cdot \mathrm{LCM}(d_1, \ldots, d_r)$. Then $L = \mathbb{F}_{p^l}$ is the minimal splitting field for B, and $B^L = L \otimes_K B$ is the split basic algebra associated to A. Moreover B^L is generated as an L-algebra by elements, $1_L \otimes \beta_1, \ldots, 1_L \otimes \beta_r$, together with elements of the form:*

(i) $1_L \otimes \mathfrak{g}_i \tau_i^a \lambda_k \tau_j^c \mathfrak{g}_j$, $i \neq j$.

(ii) $1_L \otimes \mathfrak{g}_i \tau_i^a (\lambda_k - \psi_i(\mu(\lambda_k))) \tau_i^b \mathfrak{g}_i$.

$1 \leq k \leq t$, $1 \leq a, b \leq n_i$, $1 \leq c \leq n_j$, *which generate* $\mathrm{J}(B^L)$ *as a two-sided ideal.*

Proof. The result follows from the previous result, from 1.10.9, and from 1.10.7. □

Chapter 3

Core Algorithms

Abstract. We present the main algorithms developed for this dissertation, and prove their correctness. We make no attempt to present the most efficient algorithms possible, nor to present every detail of every algorithm. Rather, we concentrate on giving algorithms which exhibit the underlying theoretical idea. Efficient implementations can be derived from the given algorithms by using standard tricks from computational linear algebra, and by applying standard correctness preserving transformations from computer science. The programs given in the appendix were produced from these algorithms in this fashion.

3.1 Basic Definitions and Notation

Definition 3.1.1. An *algorithm* is a well-ordered collection of unambiguous and effectively computable operations that when executed produces a result and halts in a finite amount of time [18].

To prove an algorithm correct we must prove that:

(i) it terminates in a finite number of steps.

(ii) the result it produces is correct (according to some specification of what the algorithm is supposed to produce).

Definition 3.1.2. A *randomized algorithm* is a well-ordered collection of unambiguous operations, each operation being either effectively computable or a truly random choice, that when executed may or may not produce a result in any finite amount of time. Note that being a randomized algorithm is a transitive property, in the sense that if algorithm A calls algorithm B, and B is randomized, then A is also randomized. If a randomized algorithm has the property that any result produced is always correct, then the algorithm is called a *Las Vegas algorithm*. If it has the property that any result produced may be incorrect, but with bounded error probability, then the algorithm is called a *Monte Carlo* algorithm.

All our randomized algorithms are of the Las Vegas type. To prove a Las Vegas randomized algorithm correct we need only prove the correctness of any result produced by the algorithm. We give correctness proofs of all algorithms, randomized or otherwise, except those for which there is no obvious correctness specification, and those for which the proof is trivial.

All our algorithms are written in single assignment pseudo-code in which the symbol $\stackrel{\circ}{=}$ denotes binding of a value to a variable, and in which all 'for' loops compute lists (sequences) of values. For readers unfamiliar with single assignment pseudo-code we provide the following example of an algorithm which computes a list of the first n Fibonacci numbers (we start numbering Fibonacci numbers at 0, so the 0-th Fibonacci number is 1, the first Fibonacci number is 1, etc).

```
Fibonacci(n ≥ 0)
    if n = 0 then
        return 1
    else
        F_0 ≙ 1
        F_1 ≙ 1
        for i in [2...n] do
            F_i ≙ F_{i-2} + F_{i-1}

        return F
```

The key to understanding the notation used to describe this algorithm is to realize that the 'for' loop does **not** execute by continually reassigning a single variable F_i, rather the 'for' loop, and the two lines preceding it, implicitly define a list, $F = [F_0, F_1, F_2, \ldots, F_n]$, containing the values bound to F_i, for $0 \leq i \leq n$. Observe that for any $0 \leq i \leq n$, we define F_i exactly once, and we never redefine it – hence the moniker 'single assignment'. F_i therefore has an unambiguous meaning, namely the i-th entry of the list F.

We adopt one other pseudo-code convention, namely that if x is a function argument which represents a list, or x is a variable being assigned the result of a function which returns a list, then we write x inside square parenthesis. For example a call to the Fibonacci algorithm, for say $n = 5$, has the form

$$[F] \triangleq \textbf{Fibonacci}(5)$$

The square parenthesis have no meaning, but do serve to remind the reader that the result of the function is a list of values, and not a single value.

3.2 Computing Generators for The Semisimple Part of A

Let $\Lambda = \{\lambda_1, \ldots, \lambda_t\}$, where each λ_i is an $m \times m$ matrix over a finite field K of characteristic p. Let $A \subseteq M_m(K)$ be the algebra generated by Λ. We can express $1 \in \overline{A}$ as a direct sum of mutually orthogonal idempotents as

$$1_{\overline{A}} = \sum_{i=1}^{r} \overline{f}_i$$

By the idempotent lifting theory, there exists a lift of this sum to an expression

$$1_A = \sum_{i=1}^{r} f_i$$

with the f_i are mutually orthogonal idempotents in A. By the Wedderburn Structure theorem and by 2.3.3

$$\bigoplus_{i=1}^{r} A_i \simeq \overline{A} = \bigoplus_{i=1}^{r} A_i'$$

where r is the number of simple A-modules, and

$$M_{n_i}(K_i) = A_i \simeq f_i A f_i / \operatorname{J}(f_i A f_i) = A_i' \subseteq f_i A f_i$$

with K_i a finite extension of K_i of degree d_i and size q_i.

Computationally speaking it is relatively easy to find the A_i. The algebra A_i is the image of A in the endomorphism ring of the i-th simple A-module S_i. Since A is a matrix algebra, the representation $\rho : A \to \operatorname{End}_A(M)$, where M is the natural module for A (i.e., M is an m dimensional vector space over K), is a faithful representation, and hence by 1.12.3 every simple A-module occurs as a composition factor of M. We can therefore use the composition series programs in the MeatAxe to compute the A_i (and hence also r, and the n_i, d_i, and q_i). Specifically the MeatAxe programs compute, for each i, a finite set $\Lambda_i = \{\lambda_{i1}, \ldots, \lambda_{it}\}$, where λ_{ij} is an $n_i \times n_i$ matrix over K_i giving the action of λ_j on S_i with respect to some basis for S_i.

Note that the MeatAxe programs give us the A_i as standalone-algebras. The programs do not tell us how the A_i are embedded in A – i.e., they do not tell us the

isomorphism between A_i and A_i'. Our first main algorithm partially computes this isomorphism. It takes as input the finite sets Λ and Λ_i, the A_i, n_i, d_i, q_i, and r. For each i it computes a $\overline{\beta}_i$ and $\overline{\tau}_i$ in A_i which, by 2.1.5, generate A_i, and then 'lifts' these elements to β_i and τ_i in f_iAf_i which generate $A_i' \subseteq f_iAf_i \subseteq A$. The algorithm therefore 'reconstructs' each A_i as a subalgebra of f_iAf_i. Taking all the β_i and τ_i then allows us to reconstruct $\overline{A}$ as a subalgebra of A.

3.2.1 Two Fundamental Constructions

Before describing the remaining steps in the algorithm we describe two fundamental constructions which we make heavy use of.

Random Elements

Let μ denote the canonical map from A to $\overline{A}$, and μ_i denote the composition

$$A \xrightarrow{\mu} \overline{A} \xrightarrow{\pi_i} A_i$$

where π_i is the obvious projection map.

Some of our algorithms require us to generate a random element $a \in A$ (or more generally in eAe for some idempotent e) and then find the image of a in A_i, that is to compute $\mu_i(a)$. Computationally speaking we can compute $\mu_i(\lambda_j)$, since $\mu_i(\lambda_j) = \lambda_{ij}$. However, we have no way computing $\mu_i(a)$ for an arbitrary element $a \in A$, since computing $\mu_i(a)$ would require us to write a in terms of the λ_j generators (i.e., to solve the word problem for A). We resolve this problem by only generating random elements of A whose expression in terms of the λ_j generators we know a priori. The details of this approach are given below.

Let X be the free non-commutative algebra over K generated by $x_1, \ldots, x_t$. Define two set-theoretic maps,

$$\phi : \{x_1, \ldots, x_t\} \to \{\lambda_1, \ldots, \lambda_t\}$$

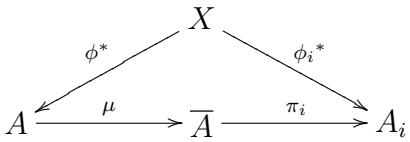
$$x_j \mapsto \lambda_j$$

and

$$\phi_i : \{x_1, \ldots, x_t\} \to \{\lambda_{i1}, \ldots, \lambda_{it}\}$$

$$x_j \mapsto \lambda_{ij}$$

Note that ϕ and the ϕ_i extend to algebra homomorphisms $\phi^* : X \to A$, and ${\phi_i}^* : X \to A_i$, such that, for each $1 \leq i \leq r$, the following diagram commutes



The point here is that computationally speaking the maps ϕ^* and ${\phi_i}^*$ are computable. A word $w \in X$ is stored as an expression, $w = \sum_j \kappa_j(x_{j1}x_{j2}\cdots x_{jm_j})$, with $\kappa_j \in K$. Hence

$$\begin{aligned}
\phi^*(w) &= \phi^*\Big(\sum_j \kappa_j \cdot (x_{j1}x_{j2}\cdots x_{jm_j})\Big) \\
&= \sum_j \kappa_j \cdot \phi^*(x_{j1})\phi^*(x_{j2})\cdots\phi^*(x_{jm_j}) \\
&= \sum_j \kappa_j \cdot \phi(x_{j1})\phi(x_{j2})\cdots\phi(x_{jm_j})
\end{aligned}$$

and likewise

$$\begin{aligned}
{\phi_i}^*(w) &= {\phi_i}^*\Big(\sum_j \kappa_j \cdot (x_{j1}x_{j2}\cdots x_{jm_j})\Big) \\
&= \sum_j \kappa_j \cdot {\phi_i}^*(x_{j1}){\phi_i}^*(x_{j2})\cdots{\phi_i}^*(x_{jm_j}) \\
&= \sum_j \kappa_j \cdot \phi_i(x_{j1})\phi_i(x_{j2})\cdots\phi_i(x_{jm_j})
\end{aligned}$$

We can therefore use ϕ^* and ${\phi_i}^*$ to create a random $a \in A$, and its image $\mu_i(a) \in A_i$, by generating a random word $w \in X$, and then taking $a = \phi^*(w)$, and $\mu_i(a) = \phi_i^*(w)$.

Henceforth we use ϕ to denote both ϕ and ϕ^*, and ϕ_i to denote both ϕ_i and ${\phi_i}^*$ – it will always be clear from the context which of the two is meant.

To create a random non-zero word in X we employ the following randomized algorithm.

RandomWord($X = K\langle x_1, \ldots, x_t \rangle$)

    repeat
        for j **in** $[1 \ldots$**Random**$(\mathbb{Z}^+)]$ **do**
            for k **in** $[1 \ldots$**Random**$(\mathbb{Z}^+)]$ **do**
                $u_{jk} \stackrel{\circ}{=} \text{Random}(\{x_1, \ldots, x_t\})$
            $w_j \stackrel{\circ}{=} \prod_k u_{jk}$
        $w \stackrel{\circ}{=} \sum w_j$
    until $w \neq 0$

    return w

The inner loop of the algorithm computes products of the generators, while the outer loop computes K-linear combinations of such products. Observe that while this algorithm involves randomized computation, it is a terminating algorithm. We use this algorithm to define another 'random element' algorithm.

Lemma 3.2.1. *Let e_1 and e_2 be idempotents in an algebra B, and let $f : X \to B$ be an algebra homomorphism. Then the following randomized algorithm computes a word $w \in X$ such that $e_1 f(w) e_2 \neq 0 \in e_1 B e_2$.*

RandomElement($f : X \to B$, $e_1 \in B$, $e_2 \in B$)

  repeat

    $w \stackrel{\circ}{=}$ **RandomWord**(X)

  until $e_1 \cdot f(w) \cdot e_2 \neq 0$

  return w

Idempotent Lifting

Our algorithms will require that we lift idempotents in $\overline{A}$ to idempotents in A. The following technical lemma, and algorithm, provide a constructive method for doing this.

Lemma 3.2.2. *Suppose $a \in A$ such that, for some $1 \leq i \leq r$, $\mu_i(a)$ is idempotent in A_i. Then there is a non negative integer j such that a^{p^j} is idempotent in A, and $\mu_i(a^{p^j}) = \mu_i(a)$. Moreover if $\mu_i(a)$ is primitive in A_i, then a^{p^j} is primitive in A.*

Proof. If $\mu_i(a)$ is idempotent in A_i, then

$$\mu_i(a^2 - a) = \mu_i(a)^2 - \mu_i(a) = 0$$

Thus $a^2 - a \in \mathrm{J}(A)$, and so $(a^2 - a)^{p^j} = 0$ for some j. Hence by the usual characteristic p binomial expansion

$$0 = (a^2 - a)^{p^j} = (a^2)^{p^j} - a^{p^j} = (a^{p^j})^2 - a^{p^j}$$

Therefore a^{p^j} is idempotent in A, and

$$\mu_i(a^{p^j}) = \mu_i(a)^{p^j} = \mu_i(a)$$

The primitivity of a^{p^j} follows from the Idempotent Lifting theorem. □

Theorem 3.2.3. *Suppose $a \in A$ such that, for some $1 \leq i \leq r$, $\mu_i(a)$ is idempotent in A_i. Let $x = p$ or $x = q_i$. Then the following algorithm computes a non negative integer j, such that a^{x^j} is idempotent in A, and $\mu_i(a^{x^j}) = \mu_i(a)$. Moreover if $\mu_i(a)$ is primitive in A_i, then a^{x^j} is primitive in A. In other words $a^{x^j} \in A$ is a lift over $\mathrm{J}(A)$ of $\mu_i(a) \in A_i$.*

Lift($a \in A$, $x \in \{p, q_i\}$)

$j \stackrel{\circ}{=} 0$
while $(a^{x^j})^2 \neq a^{x^j}$
$j \stackrel{\circ}{=} j + 1$

return j

Proof. Correctness follows from the previous lemma, and from the observation that if a^{p^j} is idempotent, then so is $a^{{q_i}^j}$. □

3.2.2 Step 1: Computing The Big Idempotents

The first step in the algorithm is to compute the idempotents $f_i \in A$. We call these idempotents the *big idempotents*.

Lemma 3.2.4. *Let $w \in X$ be such that $\phi_i(w) \neq 0$ for some $1 \leq i \leq r$. Let $u = \prod_{j \neq i}^{r} g_j(w)$, where g_j is the minimal polynomial of $\phi_j(w)$. Then u is an element of X such that $\phi_k(u) = 0$ for all $k \neq i$ ($\phi_i(u)$ may also be zero).*

Proof. For $k \neq i$

$$\begin{aligned} \phi_k(u) &= \phi_k(\prod_{j \neq i} g_j(w)) \\ &= \prod_{j \neq i} g_j(\phi_k(w)) \end{aligned}$$

$$= \Big(\prod_{j \neq i,k} g_j(\phi_k(w)) \Big) \cdot g_k(\phi_k(w))$$
$$= \Big(\prod_{j \neq i,k} g_j(\phi_k(w)) \Big) \cdot 0$$
$$= 0 \qquad \square$$

Theorem 3.2.5. *The following randomized algorithm computes a word $u \in X$ such that $\phi_i(u) \neq 0$ and $\phi_k(u) = 0$ for $k \neq i$.*

Projection(i, $[\phi_k : X \to A_k]$)

  repeat
    $w \stackrel{\circ}{=}$ **RandomElement**(ϕ_i, 1_{A_i}, 1_{A_i})
    $u \stackrel{\circ}{=} \prod_{j \neq i}$ **MinimalPolynomial**($\phi_j(w)$)
  until $\phi_i(u) \neq 0$

  return u

Proof. We exit the loop with u a word in X such that $\phi_i(u) \neq 0$, and w a word in X such that $\phi_i(w) = 1_{A_i}\phi_i(w)1_{A_i} \neq 0$ by 3.2.1. Hence by the previous lemma, $\phi_k(u) = 0$ for $k \neq i$. $\square$

Definition 3.2.6. Let $a \in M_n(K)$ be an invertible matrix and $h \in K[x]$ be the minimal polynomial of a. Let $g = (c - h)/(c \cdot x)$, where c is the constant term of h. Then the inverse of a is given by $g(a)$. We call g the *inverting polynomial for* a. Clearly inverting polynomials are computable.

Lemma 3.2.7. *Suppose w is an element in X such that, for some $1 \leq i \leq r$, $\phi_i(w)$ is invertible and $\phi_k(w) = 0$ for $k \neq i$. Suppose also that $e \in A$ is an idempotent such that $\mu_i(e) = 1_{A_i}$. Let $a = e \cdot \phi(w) \cdot g(\phi(w)) \cdot e$, where g is the inverting polynomial for $\phi_i(w)$. Then there is a positive integer j such that $f = a^{p^j}$ has the following properties:*

(i) $f \neq 0 \in eAe$ *is idempotent.*

(ii) $\mu_i(f) = 1_{A_i}$.

(iii) $\mu_k(f) = 0$ *for* $k \neq i$.

In particular, $f \in eAe$ *is a lift over* $\mathrm{J}(A)$ *of* $\overline{f}_i \in \overline{A}$.

Proof. Observe that

$$\begin{aligned}
\mu_i(a) &= \mu_i(e \cdot \phi(w) \cdot g(\phi(w)) \cdot e) \\
&= \mu_i(e) \cdot \mu_i(\phi(w)) \cdot \mu_i(g(\phi(w))) \cdot \mu_i(e) \\
&= 1_{A_i} \cdot \mu_i(\phi(w)) \cdot g(\mu_i(\phi(w))) \cdot 1_{A_i} \\
&= \phi_i(w) \cdot g(\phi_i(w)) \\
&= 1_{A_i}
\end{aligned}$$

and for $k \neq i$

$$\begin{aligned}
\mu_k(a) &= \mu_k(e \cdot \phi(w) \cdot g(\phi(w)) \cdot e) \\
&= \mu_k(e) \cdot \mu_k(\phi(w)) \cdot \mu_k(g(\phi(w))) \cdot \mu_k(e) \\
&= \mu_k(e) \cdot \mu_k(\phi(w)) \cdot \mu_k(g(\phi(w))) \cdot \mu_k(e) \\
&= \mu_k(e) \cdot \phi_k(w) \cdot \mu_k(g(\phi(w))) \cdot \mu_k(e) \\
&= 0
\end{aligned}$$

Hence by 3.2.2 there is some value of j such that $f = a^{p^j}$ is idempotent, $\mu_i(f) = 1_{A_i}$ (so $f \neq 0$), and $\mu_k(f) = 0$ for $k \neq i$. □

Theorem 3.2.8. *Let* e *be an idempotent in* A *such that* $\mu_i(e) = 1_{A_i}$. *The following randomized algorithm computes* $f \in eAe$ *such that*

(i) f *is idempotent.*

(ii) $\mu_i(f) = 1_{A_i}$.

(iii) $\mu_k(f) = 0$ *for* $k \neq i$.

In particular, the algorithm computes a lift $f \in eAe$ *over* $\mathrm{J}(A)$ *of* $\overline{f}_i \in \overline{A}$.

Big($\phi : X \to A$, i, $[\phi_k : X \to A_k]$, $e \in A$)

  repeat
    $u \stackrel{\circ}{=}$ **Projection**(i, $[\phi_k]$)
  until IsInvertible($\phi_i(u)$)

  $a \stackrel{\circ}{=} e \cdot \phi(u) \cdot g(\phi(u)) \cdot e$ **where** $g \stackrel{\circ}{=}$ **InvertingPolynomial**($\phi_i(u)$)
  $f \stackrel{\circ}{=} a^{p^j}$ **where** $j \stackrel{\circ}{=}$ **Lift**(a, p)

  return f

Proof. We exit the loop with $u \in X$ such that $\phi_i(u) \in A_i$ is invertible, and, by 3.2.5, $\phi_k(u) = 0 \in A_j$ for $k \neq i$. The result now follows by the previous lemma. □

Theorem 3.2.9. *The following randomized algorithm computes a list* $f_1, \ldots, f_r$ *of mutually orthogonal idempotents in* A *such that* $1_A = \sum_{i=1}^{r} f_i$, *and each* $f_i \in A$ *is a lift over* $\mathrm{J}(A)$ *of* $\overline{f}_i \in \overline{A}$, $1 \leq i \leq r$.

AllBig($\phi : X \to A$, $[\phi_k : X \to A_k]$, r)

  for i **in** $[1 \ldots r-1]$ **do**
    $e_i \stackrel{\circ}{=} 1_A - \sum_{j=1}^{i-1} f_j$
    $f_i \stackrel{\circ}{=}$ **Big**(ϕ, i, $[\phi_k]$, e_i)

  $f_r \stackrel{\circ}{=} 1_A - \sum_{j=1}^{r-1} f_j$
  return f

Proof. We proceed by induction on i. Our induction hypothesis is that for $j < i$:

(i) $f_j \in e_j A e_j$ is a lift of $\overline{f}_j \in \overline{A}$ (so $\mu_i(f_j) = 1_{A_j}$ and $\mu_j(f_j) = 0$ for $k \neq j$).

(ii) $f_1, \ldots, f_j$ is a sequence of mutually orthogonal idempotents in A.

The base case for the induction is $j = 1$. Since $e_1 = 1_A$, we have that $e_1 A e_1 = A$, and $\mu_1(e_1) = \mu_1(1_A) = 1_{A_1}$. Thus, by 3.2.8, $f_1 \in A$ is a lift of $\overline{f}_1 \in \overline{A}$, and so (i) holds. (ii) holds trivially.

Assume the induction hypothesis holds for $j = l - 1$, and consider $j = l$ (i.e., we are assuming that our induction hypothesis holds at the end of the $(l-1)$-th loop iteration, and we are considering whether it still holds at the end of the l-th loop iteration). By the induction hypothesis we know that for $j < l$:

(i) $\mu_l(f_j) = 0$ (since $j < l$ implies $j \neq l$).

(ii) $f_1, \ldots, f_j$ is a sequence of mutually orthogonal idempotents in A.

Now

$$\begin{aligned}
\mu_l(e_l) &= \mu_k(1_A - \sum_{k=1}^{l-1} f_k) \\
&= \mu_k(1_A) - \sum_{k=1}^{l-1} \mu_l(f_k) \\
&= \mu_l(1_A) - 0 \\
&= 1_{A_l}
\end{aligned}$$

Thus, by 3.2.8, $f_l \in e_l A e_l$ is a lift of $\overline{f}_l \in \overline{A}$, and so (i) holds. For any $j < l$

$$\begin{aligned}
e_l \cdot f_j &= (1 - \sum_{k=1}^{l-1} f_k) f_j \\
&= f_j - \sum_{k=1}^{l-1} (f_k f_j) \\
&= f_j - {f_j}^2
\end{aligned}$$

$$= 0$$

Similarly $f_j \cdot e_l = 0$. Since $f_l \in e_l A e_l$, it follows that f_l is orthogonal to any f_j, $j < l$. Hence $f_1, \ldots, f_l$ is a sequence of mutually orthogonal idempotents, and so (ii) holds.

We excute the loop exactly $r - 1$ times. Arguing as above we then have that $\mu_r(f_r) = 1_{A_r}$, and $f_1, \ldots, f_r$ is a sequence of mutually orthogonal idempotents. Finally note that

$$\sum_{i=1}^{r} f_i = \sum_{i=1}^{r-1} f_i + f_r = \sum_{i=1}^{r-1} f_i + (1_A - \sum_{i=1}^{r-1} f_i) = 1_A \qquad \square$$

3.2.3 Step 2: Computing The Little Idempotents

Recall that

$$M_{n_i}(K_i) = A_i \simeq (f_i A f_i)/\operatorname{J}(f_i A f_i) \simeq A_i' \subseteq A$$

We can express 1_{A_i} as

$$1_{A_i} = \sum_{j=1}^{n_i} \overline{g}_{ij}$$

where the $\overline{g}_{ij}$ are mutually orthogonal primitive idempotents in A_i. By the idempotent lifting theorem we can lift the $\overline{g}_{ij}$ to mutually orthogonal primitve idempotents $g_{ij} \in f_i A f_i$, $1 \leq j \leq n_i$, such that

$$1_{f_i A f_i} = f_i = \sum_{j=1}^{n_i} g_{ij}$$

and $\mu_i(g_{ij}) = \overline{g}_{ij}$. The second step in the algorithm is therefore to compute, for each i, the g_{ij} idempotents in $f_i A f_i$. We call these idempotents the *little idempotents*. Our strategy in constructing the little idempotents will be to construct each $\overline{g}_{ij} \in A_i$, and then lift it to $g_{ij} \in f_i A f_i$.

At this point it becomes necessary to understand the connection between decompositions of the identity of 1_{A_i} into sums of mutually orthogonal primitive idempotents, and choices of basis for S_i. Given a decomposition

$$1_{A_i} = \sum_{j=1}^{n_i} \overline{g}_{ij}$$

we then have that

$$S_i = 1_{A_i} \cdot S_i = \Big(\sum_{j=1}^{n_i} \overline{g}_{ij} \Big) S_i = \bigoplus_{j=1}^{n_i} \overline{g}_{ij} S_i$$

where each $\overline{g}_{ij} S_i$ is one dimensional over K_i. For $1 \leq j \leq n_i$, pick any non-zero $w_{ij} \in \overline{g}_{ij} S_i$. Then $B = \{w_{i1}, \ldots, w_{in_i}\}$ is a basis for S_i. Hence decompositions of 1_{A_i} into sums of mutually orthogonal primitive idempotents correspond, up to choices of scalars (we can freely scale the w_{ij}), to choices of basis for S_i.

Lemma 3.2.10. *Suppose* $1_{A_i} = \sum_{j=1}^{n_i} \overline{g}_{ij}$, *and* $1_{A_i} = \sum_{j=1}^{n_i} \overline{h}_{ij}$, *are two decompositions of* 1_{A_i} *into sums of mutually orthogonal primitive idempotents in* A_i. *Then there is some invertible* $\omega \in A_i$, *such that* $\omega \overline{g}_{ij} \omega^{-1} = \overline{h}_{ij}$ *for all* $1 \leq j \leq n_i$.

The point here is that any two decompositions of the identity in 1_{A_i} into sums of mutually orthogonal primitive idempotents, are conjugate by some element of A_i.

Now let $\overline{e}_{ijk}$ denote the jk-th elementary matrix in A_i. Define $\overline{e}_{ij} = \overline{e}_{ijj}$. Note that the $\overline{e}_{ij}$ are primitive mutually orthogonal idempotents, and

$$1_{A_i} = \sum_{j=1}^{n_i} \overline{e}_{ij}$$

Recall that by 2.1.6 each A_i is generated by matrices $\overline{\beta}_i$ and $\overline{\tau}_i$, where

$$\overline{\beta}_i = \begin{pmatrix} \varepsilon_i & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \cdots & 0 \end{pmatrix}$$

where ε_i is any primitive element of K_i. Note that $\overline{\beta}_i = \varepsilon \cdot \overline{e}_{i1}$. Now we will not be able to construct our $\overline{g}_{ij}$ so that $\overline{g}_{ij} = \overline{e}_{ij}$. However as we shall see later this won't matter – with respect to some basis B_i for S_i, the $\overline{g}_{ij}$ that we construct will be equal to the $\overline{e}_{ij}$, and that will be sufficient. Note that we must be careful when constructing the first little idempotent g_{i1}, since we must ensure that with respect to B_i, $\overline{g}_{i1}$ is equal to a *primitive element* of K_i times $\overline{e}_{i1}$. The other little idempotents will be constructed via a generic process similar to that used to construct the big idempotents.

Lemma 3.2.11. *Suppose A is an L-algebra for some extension L of K, and M is an A-module with $\dim_L(M) = s$. Suppose $f \in \mathrm{End}_A(M)$ is not nilpotent, and $\dim_L(\mathrm{im}(f)) = 1$. Then $M = \mathrm{im}(f) \oplus \ker(f)$, and with respect to any basis for M the matrix α representing f is a scalar, $\varepsilon \neq 0 \in L$, times a primitive idempotent of A.*

Proof. We know by 1.9.1 that $M = \mathrm{im}(f^n) \oplus \ker(f^n)$ for some n. Now $\mathrm{im}(f^2) \subseteq \mathrm{im}(f)$, but $\mathrm{im}(f^2) \neq 0$, since f is not nilpotent. Since $\mathrm{im}(f)$ is one dimensional we must then have that $\mathrm{im}(f^2) = \mathrm{im}(f)$. Similarly $\ker(f^2) = \ker(f)$. It follows then that $n = 1$, i.e., $M = \mathrm{im}(f) \oplus \ker(f)$.

Now take $v_1 \neq 0 \in \mathrm{im}(f)$, and $v_2, \ldots, v_s$ a basis for $\ker(f)$. Then $B = \{v_1, v_2, \ldots, v_s\}$ is a basis for M such that $f(v_1) = \varepsilon v_1$, for some $\varepsilon \in L$, and $f(v_j) = 0$, for $j \neq i$. With respect to B

$$\alpha = \begin{pmatrix} \varepsilon & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \cdots & 0 \end{pmatrix} = \varepsilon \cdot \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \cdots & 0 \end{pmatrix}$$

Hence α is ε times a primitive idempotent of A. □

It is very important to understand a computational nuance at this point. While each A_i is theoretically a full $n_i \times n_i$ matrix algebra over K_i, the MeatAxe programs give each A_i as an $n_i d_i \times n_i d_i$ matrix algebra over the ground field K – specifically each λ_{ij} is given as an $n_i d_i \times n_i d_i$ matrix over K. Until now this distinction between A_i as a K-algebra and as a K_i-algebra has not been relevant – it now becomes important.

Lemma 3.2.12. *Every matrix in A_i has K-rank a multiple of d_i.*

Proof. For $a \in A_i$ let $\psi_a : S_i \to S_i$ be the linear transformation defined by a. Then

$$\operatorname{rank}_K(a) = \dim_K(\operatorname{im} \psi_a) = \dim_{K_i}(\operatorname{im} \psi_a)[K_i : K] = \dim_{K_i}(\operatorname{im} \psi_a) \cdot d_i \qquad \square$$

Corollary 3.2.13. *If $a \in A_i$ is idempotent with rank d_i, then a is a primitive idempotent in A_i.*

Theorem 3.2.14. *Let e be an idempotent in A_i. Then the following randomized algorithm returns a word $u \in X$ such that $e \cdot \phi_i(u) \cdot e$ is a scalar $\varepsilon \in K_i$ times a primitive idempotent in eAe.*

AlmostLittle($\phi_i : X \to A_i$, n_i, d_i, $e \in A_i$)
  if $n_i = 1$ **then**
    $u \stackrel{\circ}{=}$ **RandomElement**(ϕ_i, e, e)
    return u
  else
    repeat
      $u \stackrel{\circ}{=}$ **RandomElement**(ϕ_i, e, e)
      $a \stackrel{\circ}{=} e \cdot \phi_i(u) \cdot e$
    until $\operatorname{rank}_K(a) = d_i$ **and not Nilpotent**(a)
    return u

Proof. If $n_i = 1$ the theorem is trivial. If $n_i \neq 1$, we return from the algorithm when have found $u \in X$ such that $a = e \cdot \phi_i(u) \cdot e$ has rank d_i over K, and hence rank 1 over K_i, and a is not nilpotent. The result now follows by the previous lemma. □

We can now show how the first little idempotent is constructed.

Theorem 3.2.15. *The following randomized algorithm computes $\overline{g} \in A_i$ and $g \in f_i A f_i$ such that:*

(i) $\overline{g}$ *is a primitive idempotent in* A_i.

(ii) g *is a primitive idempotent in* $f_i A f_i$.

(iii) $\mu_i(g) = \overline{g}$.

In other words g_i is a lift over $\mathrm{J}(f_i A f_i)$ of $\overline{g}$. The algorithm also computes $\overline{\beta} \in A_i$ and $\beta \in f_i A f_i$ such that

(i) $\mu_i(\beta) = \overline{\beta}$.

(ii) $\overline{g} = \overline{\beta}^{q_i - 1}$

(iii) $g = \beta^{q_i - 1}$.

So β projects to $\overline{\beta}$ under quotienting of $f_i A f_i$ by its radical. Finally, there is a basis B for S_i over K_i such that with respect to B:

(i) $\overline{g} = \overline{e}_{i1}$.

(ii) $\overline{\beta} = \overline{\beta}_i$.

The algorithm also returns the polynomial $h_i(x)$, where $h_i(x)$ is the minimal polynomial of ε_i.

FirstLittle($\phi : X \to A$, $\phi_i : X \to A_i$, $f_i \in A$, n_i, d_i, q_i)

repeat

$u \mathrel{\mathring{=}}$ **AlmostLittle**(ϕ_i, n_i, d_i, 1_{A_i})

$\overline{\beta} \mathrel{\mathring{=}} \phi_i(u)$

until $\overline{\beta}$ **corresponds to a primitive element** $\varepsilon \in L$

$\overline{g} \mathrel{\mathring{=}} \overline{\beta}^{q_i - 1}$

$\gamma' \mathrel{\mathring{=}} f_i \cdot \phi(u) \cdot f_i$

$g' \mathrel{\mathring{=}} \gamma'^{q_i - 1}$

$g \mathrel{\mathring{=}} g'^{{q_i}^j}$

$\gamma \mathrel{\mathring{=}} \gamma'^{{q_i}^j}$ **where** $j \mathrel{\mathring{=}}$ **Lift**(g', q_i)

$$h \mathrel{\mathring{=}} \begin{cases} f & \textbf{if } n_i = 1 \\ f/x & \textbf{otherwise} \end{cases}$$

where $f \mathrel{\mathring{=}}$ **MinimalPolynomial**($\overline{\beta}$) $\in K[x]$

$\beta \mathrel{\mathring{=}} \gamma'^l$

where $k \mathrel{\mathring{=}}$ **smallest positive integer such that** $(\gamma h(\gamma'))^k = 0$

$l \mathrel{\mathring{=}}$ **smallest positive integer such that** ${q_i}^l \geq k$

return $(g, \overline{g}, \beta, h)$

Proof. By 3.2.14, and by the loop condition, we exit the loop with u a word in X such that $\overline{\beta} = \phi_i(u) \in A_i$, and with respect to the basis B (for S_i) defined in 3.2.14, we have that

$$\overline{\beta} = \begin{pmatrix} \varepsilon & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \cdots & 0 \end{pmatrix}$$

for some primitive element $\varepsilon \in K_i$. Hence with respect to B we have that $\overline{\beta} = \overline{\beta}_i$, and $g = \overline{\beta}^{q_i - 1} = \overline{e}_{i1}$.

Now observe that

$$\mu_i(\gamma') = \mu_i(f_i \cdot \phi(u) \cdot f_i) = \mu_i(f_i) \cdot \mu_i(\phi(u)) \cdot \mu_i(f_i) = 1_{A_i} \cdot \phi_i(u) \cdot 1_{A_i} = \overline{\beta}$$

and therefore

$$\mu_i(g') = \mu_i(\gamma'^{q_i - 1}) = \mu_i(\gamma)'^{q_i - 1} = \overline{\beta}^{q_i - 1} = \overline{g}$$

Since $\overline{g}$ is a primitive idempotent in A_i, it then follows by 3.2.3 that $g = (g')^{{q_i}^j}$ is a primitive idempotent in $f_i A f_i$ and $\mu_i(g) = \overline{g}$.

Since taking q_i-th powers in K_i is the identity map

$$\mu_i(\gamma) = \mu_i((\gamma')^{{q_i}^j}) = \overline{\beta}^{{q_i}^j} = \overline{\beta}$$

So γ is an inverse image of $\overline{\beta}$ under μ. Now let $h_i \in K[x]$ be the minimal polynomial for ε over K, and f be the minimal polynomial for $\overline{\beta}$. If $\deg A_i = 1$, then $f = h_i$, so we set $h = h_i$. If $\deg A_i \neq 1$, then $f = x \cdot h_i$, and we set $h = f/x = h_i$.

Finally we note that the definition of β in the above algorithm agrees with the definition of β in 2.2.1. □

Theorem 3.2.16. *Suppose $\overline{e}$ is an idempotent in A_i, and e is an idempotent in $f_i A f_i$ such that $\mu_i(e) = \overline{e}$. Then the following algorithm computes a primitive idempotent $\overline{g} \in \overline{e} A_i \overline{e}$, and a primitive idempotent $g \in eAe$, such that $\mu_i(g) = \overline{g}$.*

NextLittle($\phi : X \to A$, $\phi_i : X \to A_i$, n_i, d_i, q_i, $e \in f_i A f_i$, $\overline{e} \in A_i$)

$u \triangleq$ **AlmostLittle**(ϕ_i, $\overline{e}$, n_i, d_i)
$\overline{g} \triangleq (\overline{e} \cdot \phi_i(u) \cdot \overline{e})^{q_i - 1}$
$g \triangleq ((e \cdot \phi(u) \cdot e)^{q_i - 1})^j$ **where** $j \triangleq$ **Lift**($(e \cdot \phi(u) \cdot e)^{q_i - 1}$, p)

return $(g, \overline{g})$

Proof. By 3.2.14, u is a word in X such that $\overline{e} \cdot \phi_i(u) \cdot \overline{e} = \varepsilon \cdot a$, where a is a primitive idempotent in $\overline{e}A\overline{e}$. Thus

$$\overline{g} = (\overline{e} \cdot \phi_i(u) \cdot \overline{e})^{q_i - 1} = (\varepsilon \cdot a)^{q_i - 1} = \varepsilon^{q_i - 1} \cdot a^{q_i - 1} = a$$

Now

$$\mu_i((e \cdot \phi(u) \cdot e)^{q_i - 1}) = (\mu_i(e) \cdot \mu_i(\phi(u)) \cdot \mu_i(e))^{q_i - 1} = (\overline{e} \cdot \phi_i(u) \cdot \overline{e})^{q_i - 1} = a$$

Hence by 3.2.3 g is a primitive idempotent in eAe projecting onto $\overline{g}$ under μ_i. □

The algorithm to compute all of the little idempotents for A_i is now straightforward.

Theorem 3.2.17. *The following randomized algorithm computes a list $\overline{g}_1, \ldots, \overline{g}_{n_i}$ of mutually orthogonal primitive idempotents in A_i, and a list $g_1, \ldots, g_{n_i}$ of mutually orthogonal primitive idempotents in $f_i A f_i$, such that:*

(i) $\mu_i(g_j) = \overline{g}_j$, $1 \leq j \leq n_i$.

(ii) $1_{A_i} = \sum_{j=1}^{n_i} \overline{g}_j$.

(iii) $f_i = \sum_{j=1}^{n_i} g_j$,

So each $g_j \in f_i A f_i$ is a lift over $\mathrm{J}(f_i A f_i)$ of $\overline{g}_j \in A_i$. The algorithm also computes $\overline{\beta} \in A_i$ and $\beta \in f_i A f_i$ such that:

(i) $\mu_i(\beta) = \overline{\beta}$.

(ii) $\overline{g}_1 = \overline{\beta}^{q_i - 1}$

(iii) $g_1 = \beta^{q_i - 1}$.

Finally, there is a basis B for A_i over K_i such that with respect to B:

(i) $\overline{g}_1 = \overline{e}_{i1}$.

(ii) $\overline{\beta} = \overline{\beta}_i$.

The algorithm also returns the polynomial $h_i(x)$, where $h_i(x)$ is the minimal polynomial of ε_i.

AllLittle($\phi : X \to A$, $\phi_i : X \to A_i$, f_i, n_i, d_i, q_i)

$(g_1, \overline{g}_1, \beta, h) \triangleq$ **FirstLittle**(ϕ, ϕ_i, f_i, n_i, d_i, q_i)

for j **in** $[2 \ldots n_i - 2]$ **do**

$e_j \triangleq f_i - \sum_{k=1}^{j-1} g_k$

$\overline{e}_j \triangleq 1_{A_i} - \sum_{k=1}^{j-1} \overline{g}_k$

$(g_j, \overline{g}_j) \triangleq$ **NextLittle**(ϕ, ϕ_i, n_i, d_i, e_j, $\overline{e}_j$)

$\overline{g}_r \triangleq 1_{A_i} - \sum_{j=1}^{r-1} \overline{g}_j$

$g_r \triangleq f_i - \sum_{j=1}^{r-1} g_j$

return $(g, \overline{g}, \beta, h)$

Proof. The first group of statements follow from 3.2.15, 3.2.16, and the same inductive argument used in 3.2.9. The second group of statements follow from 3.2.15. The third group of statements follow from 3.2.15, and the fact that $\overline{g}_1$ is orthogonal to $\overline{g}_j$, for $j \neq 1$. □

3.2.4 Step 3: Computing Generators for each A_i'

We previously showed how to construct $\overline{\beta}_i \in A_i$, and how to lift it to $\beta_i \in f_i A f_i$. We now show how to construct $\overline{\tau}_i \in A_i$, and how to lift it to $\tau_i \in f_i A f_i$.

As before let $\overline{e}_{ijk}$ denote the jk-th elementary matrix in A_i. Define $\overline{e}_{ij} = \overline{e}_{ijj}$ and $\overline{z}_{ij} = \overline{e}_{ij(j+1)}$ (all such indices should be taken modulo n_i, with the representatives of the n_i equivalence classes being 1 to n_i – in particular $\overline{z}_{in_i} = \overline{e}_{in_i(n_i+1)} = \overline{e}_{in_i1}$). Recall that

$$\overline{\tau}_i = \sum_{j=1}^{n_i} \overline{z}_{ij} = \begin{pmatrix} 0 & 1 & 0 & \cdots & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & \cdots & \cdots & \cdots & 0 & 1 \\ 1 & 0 & \cdots & \cdots & \cdots & 0 \end{pmatrix}$$

Our strategy for constructing τ_i will be to construct the $\overline{z}_{ij} \in A_i$, and then lift these to $z_{ij} \in f_i A f_i$. Then τ_i will be the sum of these lifts. As was the case with the $\overline{g}_{ij}$, we will not be able to construct the $\overline{z}_{ij}$ per se, rather we will construct a sequence of $\overline{a}_{ij}$, such that with respect to some basis B_i for S_i, each $\overline{a}_{ij}$ equals the corresponding $\overline{z}_{ij}$. As a consequence we will not in fact construct $\overline{\tau}_i$, but will construct a $\overline{\tau}$ such that, with respect to B_i, $\overline{\tau} = \overline{\tau}_i$.

The point here is that to generate A_i we only need elements $\overline{\beta}$ and $\overline{\tau}$ which satisfy the relations of 2.1.5. Since $\overline{\beta}_i$ and $\overline{\tau}_i$ satisfy these relations, any matrix similar to them will also satisfy these relations.

Theorem 3.2.18. *The following randomized algorithm computes $\overline{\beta}$, $\overline{\tau} \in A_i$, and β, $\tau \in f_i A f_i$ such that:*

(i) $\mu_i(\beta) = \overline{\beta}$.

(ii) $\mu_i(\tau) = \overline{\tau}$.

Moreover there is a basis B for A_i such that with respect to B:

(i) $\overline{\beta} = \overline{\beta}_i$.

(ii) $\overline{\tau} = \overline{\tau}_i$.

Hence $\overline{\beta}$ *and* $\overline{\tau}$ *generate* A_i*, and* β *and* τ *generate* $A_i' \subseteq f_i A f_i$*. The algorithm also returns the polynomial* $h_i(x)$*, where* $h_i(x)$ *is the minimal polynomial of* ε_i.

SemiSimpleGenerators$_i(\phi : X \to A,\ \phi_i : X \to A_i,\ f_i,\ n_i,\ d_i,\ q_i)$

$([g], [\overline{g}], \beta, h) \doteq$ **AllLittle**$(\phi : X \to A,\ \phi_i : X \to A_i,\ f_i,\ n_i,\ d_i,\ q_i)$

for j **in** $[1 \ldots n_i - 1]$ **do**

$w_j \doteq$ **RandomElement**$(\phi_i,\ \overline{g}_j,\ \overline{g}_{j+1})$

$\overline{a}_j \doteq \overline{g}_j \cdot \phi_i(w_j) \cdot \overline{g}_{j+1}$

$a_j \doteq g_j \cdot \phi(w_j) \cdot g_{j+1}$

$w_{n_i} \doteq$ **RandomElement**$(\phi_i,\ \overline{g}_{n_i},\ \overline{g}_1)$

$\overline{a}'_{n_i} \doteq \overline{g}_{n_i} \cdot \phi_i(w_{n_i}) \cdot \overline{g}_1$

$\overline{x} \doteq \Big(\prod_{j=1}^{n_i - 1} \overline{a}_j \Big)$

$\overline{a}_{n_i} \doteq \overline{a}'_{n_i} \cdot \Big(\sum_i \kappa_i \overline{\beta}_i \Big)^{-1}$ **where** $\overline{x a}'_{n_i} = \sum_i \kappa_i \overline{\beta}_i$

$\tau \doteq \sum_{j=1}^{n_i} \overline{a}_j$

$a'_{n_i} \doteq (g_{n_i} \cdot \phi(w_{n_i}) \cdot g_1) \Big(\sum_i \kappa_i \beta_i \Big)^{-1}$

$x \doteq \Big(\prod_{j=1}^{n_i - 1} a_j \Big)$

$a_{n_i} \doteq a'_{n_i} (x a'_{n_i})^{p^l - 1}$

where $l \doteq$ **Lift**$(x a'_{n_i} - g_1,\ p)$

$\tau \doteq \sum_{j=1}^{n_i} a_j$

return (β, τ, g, h)

Proof. First note that we can assume that we have changed basis for S_i so that the $[\overline{g}]$ little idempotents computed for A_i are in fact the $\overline{e}_{ij}$ elementary matrices.

We finish the loop with each w_j, $1 \leq j \leq n_i - 1$, a word in X such that

$$\overline{a}_j = \overline{g}_j \cdot \phi_i(w_j) \cdot \overline{g}_{j+1} \neq 0 \in \overline{g}_j A_i \overline{g}_{j+1}$$

We then set w_{n_i} to be a word in X such that

$$\overline{a}'_{n_i} = \overline{g}_{n_i} \cdot \phi_i(w_{n_i}) \cdot \overline{g}_1 \neq 0 \in \overline{g}_{n_i} A_i \overline{g}_1$$

Now $\overline{a}_1$ is a non-zero element of $\overline{g}_1 A_i \overline{g}_2$. With respect to the basis C_1 we have that

$$\overline{a}_1 = \begin{pmatrix} 0 & \gamma_1 & 0 & \cdots & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & \cdots & \cdots & \cdots & 0 & 0 \\ 0 & 0 & \cdots & \cdots & \cdots & 0 \end{pmatrix}$$

for some $\gamma_1 \neq 0 \in K_i$. Define $C_2 = \{w_1, \gamma_1^{-1} w_2, \ldots, w_{n_i}\}$. The change of basis matrix η_1 from C_1 to C_2 is

$$\eta_1 = \begin{pmatrix} 1 & 0 & 0 & \cdots & \cdots & 0 \\ 0 & \gamma_1 & 0 & 0 & \cdots & 0 \\ 0 & \cdots & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & \cdots & \cdots & \cdots & 1 & 0 \\ 0 & 0 & \cdots & \cdots & \cdots & 1 \end{pmatrix}$$

Hence with respect to C_2, we have that

$$\overline{a}_1 = \eta_1 \overline{a}_1 \eta_1^{-1} = \overline{z}_{i1}$$

and

$$\overline{g}_j = \eta_1 \overline{g}_j \eta_1^{-1} = \overline{g}_j$$

for all $1 \leq j \leq n_i$.

Now $\overline{a}_2$ is a non-zero element of $\overline{g}_2 A_i \overline{g}_3$. With respect to the basis C_2

$$\overline{a}_2 = \begin{pmatrix} 0 & 0 & 0 & \cdots & \cdots & 0 \\ 0 & 0 & \gamma_2 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & \cdots & \cdots & \cdots & 0 & 0 \\ 0 & 0 & \cdots & \cdots & \cdots & 0 \end{pmatrix}$$

for some $\gamma_2 \neq 0 \in K_i$. Define $C_3 = \{w_1, \gamma_1^{-1} w_2, \gamma_2^{-1} w_3, \ldots, w_{n_i}\}$. Let η_2 be the change of basis matrix from C_2 to C_3. Arguing as above we have that with respect to C_3:

(i) $\overline{a}_k = \eta_2 \overline{a}_k \eta_2^{-1} = \overline{z}_{ik}$, for $1 \leq k \leq 2$.

(ii) $\overline{g}_j = \eta_2 \overline{g}_j \eta_2^{-1} = \overline{g}_j$, for all $1 \leq j \leq n_i$.

Continuing inductively in this fashion define for $2 \leq k \leq n_i$ that

$$C_k = \{w_1, \gamma_1^{-1} w_2, \ldots, \gamma_{k-2}^{-1} w_{k-1}, \gamma_{k-1}^{-1} w_k, \ldots, w_{n_i}\}$$

Observe that with respect to C_k:

(i) $\overline{a}_l = \overline{z}_{il}$, for any $l \leq k$.

(ii) $\overline{g}_j$ is unchanged for all $1 \leq j \leq n_i$.

Now $\overline{a}'_{n_i}$ is a non-zero element of $\overline{g}_{n_i} A_i \overline{g}_1$. With respect to the basis C_{n_i} we have that

$$\overline{a}'_{n_i} = \begin{pmatrix} 0 & 0 & 0 & \cdots & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & \cdots & \cdots & \cdots & 0 & 0 \\ \gamma & 0 & \cdots & \cdots & \cdots & 0 \end{pmatrix}$$

for some $\gamma \neq 0 \in K_i$. At this point we cannot scale w_1 in the same fashion that we scaled w_2 through w_{n_i}. Instead observe that $\overline{xa}'_{n_i}$ is an element of $\overline{g}_1 A_i \overline{g}_1$, and hence with respect to the basis C_{n_i}, we have that

$$\overline{xa}'_{n_i} = \begin{pmatrix} \gamma & 0 & 0 & \cdots & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & \cdots & \cdots & \cdots & 0 & 0 \\ 0 & 0 & \cdots & \cdots & \cdots & 0 \end{pmatrix}$$

Now $\gamma = \sum_i \kappa_i \varepsilon_i$, with $\kappa_i \in K$. Hence with respect to C_{n_i} we have that $\overline{xa}'_{n_i} = \left(\sum_i \kappa_i \overline{\beta}_i \right)$, and therefore

$$\overline{a}_{n_i} = \begin{pmatrix} 0 & 0 & 0 & \cdots & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & \cdots & \cdots & \cdots & 0 & 0 \\ 1 & 0 & \cdots & \cdots & \cdots & 0 \end{pmatrix}$$

Taking $B = C_{n_i}$, we have that with respect to B, $\overline{a}_j = \overline{z}_{ij}$, for $1 \leq j \leq n_i$, and therefore $\overline{\tau} = \overline{\tau}_i$. Moreover by 3.2.17 we also have that $\mu_i(\beta) = \overline{\beta}$, and since w_1 is the first basis vector in B, that with respect to B, $\overline{\beta} = \overline{\beta}_i$.

It remains to show that $\mu_i(\tau) = \overline{\tau}$. Observe that by 3.2.17 we know that each w_j, $1 \leq j \leq n_i - 1$, is a word in X such that $\mu_i(a_j) = \overline{a}_j$. In particular, $a_j \neq 0$ for $1 \leq j \leq n_i - 1$. Likewise w_{n_i} is a word in X such that $\mu_i(a'_{n_i}) = \overline{a}'_{n_i}$, Finally we note that the definition of a_{n_i} and τ in the above algorithm, is precisely how z_{n_i} and τ were defined in 2.2.1 (identifying g_1 here with e_1 in 2.2.1, each $\overline{g}_j$ here with $\overline{E}_j$, and $\overline{a}_j$ here with $\overline{Z}_j$).

Finally since $\overline{\beta}$ and $\overline{\tau}$ are similar to $\overline{\beta}_i$ and $\overline{\tau}_i$, and the latter generate A_i, we must have that $\overline{\beta}$ and $\overline{\tau}$ also generate A_i. Hence β and τ generate A'_i by 2.2.1. □

Remark. Since $g = \beta^{q_i}$ it is not necessary to return g from the above algorithm. We do so for clarity of the presentation of later algorithms.

3.2.5 Computing the Semisimple Generators

We now present the complete algorithm for computing generators for the semisimple part of A.

Theorem 3.2.19. *The following randomized algorithm computes lists $\beta_1, \ldots, \beta_r$ and $\tau_1, \ldots, \tau_r$, such that:*

(i) β_i and τ_i generate A'_i, $1 \leq i \leq r$, as per 2.1.5.

(ii) the collection of β_i and τ_i generate the semisimple part of A as per 2.3.4.

The algorithm also computes the list $\mathfrak{g}_1, \ldots, \mathfrak{g}_r$ of the first little idempotents for each A'_i, and the list $h_i, \ldots, h_r$ of minimal polynomials associated to each β_i.

SemiSimpleGenerators($\phi : X \to A$, $[\phi_k : X \to A_k]$, $[n_k]$, $[d_k]$, $[q_k]$, r)
  $[f] \doteq$ **AllBig**(ϕ, $[\phi_k]$, r)
  for i **in** $[1 \ldots r]$ **do**
    $(\beta_i, \tau_i, \mathfrak{g}_i, h_i) \doteq$ **SemiSimpleGenerators**$_i$(ϕ, ϕ_i, f_i, n_i, d_i, q_i)

  return $(\beta, \tau, \mathfrak{g}, h)$

Proof. Correctness follows from 3.2.9, 3.2.18, and 2.3.4. □

3.3 Computing Generators for the Radical of A

Recall that if A is a finite dimensional K-algebra generated by elements $\Lambda = \{\lambda_1, \ldots \lambda_t\}$, then by 2.4.3, $\mathrm{J}(A)$ is generated as a two-sided ideal by elements of A of the form:

(i) $\mathfrak{g}_i \tau_i^a \lambda_k \tau_j^c \mathfrak{g}_j$, $i \neq j$.

(ii) $\mathfrak{g}_i \tau_i^a (\lambda_k - \psi_i(\mu(\lambda_k))) \tau_i^b \mathfrak{g}_i$.

where $1 \leq k \leq t$, $1 \leq a, b \leq d_i$, and $1 \leq c \leq d_j$, and $\overline{\mathfrak{g}}_i$ to be $\overline{g}_{i1}$ Our second main algorithm takes as inputs the set Λ, and the lists $[\phi_i : X \to A_i]$, $[\beta_i]$, $[\tau_i]$, $[\mathfrak{g}_i]$, $[\overline{\beta}_i]$, $[\overline{\tau}_i]$, $[\overline{\mathfrak{g}}_i]$. $[d_i]$, and $[n_i]$, and computes this set of generators for $\mathrm{J}(A)$.

Lemma 3.3.1. *The following algorithm constructs $y \in A_i'$ such that $y = \psi_i(\mu_i(\lambda_l))$.*

Psi$_i(\phi_i,\ \tau_i,\ \beta_i,\ \mathfrak{g}_i,\ \overline{\tau}_i,\ \overline{\beta}_i,\ \overline{\mathfrak{g}}_i,\ n_i,\ \lambda_l)$

for $j,\ k$ **in** $[1 \ldots n_i]$ **do**

$$\overline{a}_{jk} \stackrel{\circ}{=} \overline{\mathfrak{g}}_i \cdot {\overline{\tau}_i}^{j-1} \cdot \phi_i(\lambda_l) \cdot {\overline{\tau}_i}^{n_i-(k-1)} \cdot \overline{\mathfrak{g}}_i$$

$$a_{jk} \stackrel{\circ}{=} \begin{cases} 0 & \textbf{if } \overline{a}_{jk} = 0 \\ \tau_i^{n-(j-1)} \cdot {\beta_i}^{x_{jk}} \cdot \tau_i^{k-1} & \textbf{if } \overline{a}_{jk} = {\overline{\beta}_i}^{x_{jk}} \end{cases}$$

$$y \stackrel{\circ}{=} \sum_{j,k} a_{jk}$$

return y

Proof. Termination is immediate since n_i is finite. For correctness note that, since $\overline{\mathfrak{g}}_i = \overline{e}_{i1}$, we have

$$\begin{aligned} \mu_i(\lambda_l) &= \phi_i(\lambda_l) \\ &= \sum_{j,k}^{n_i} \overline{e}_{ij} \cdot \phi_i(\lambda_l) \cdot \overline{e}_{ik} \end{aligned}$$

$$= \sum_{j,k}^{n_i} \overline{\tau}_i{}^{n_i-(j-1)} \cdot \overline{e}_{i1} \cdot \overline{\tau}_i^{j-1} \cdot \phi_i(\lambda_l) \cdot \overline{\tau}_i{}^{n_i-(k-1)} \cdot \overline{e}_{i1} \cdot \overline{\tau}_i^{k-1}$$
$$= \sum_{j,k}^{n_i} \overline{\tau}_i{}^{n_i-(j-1)} \cdot \overline{\mathfrak{g}}_i \cdot \overline{\tau}_i^{j-1} \cdot \phi_i(\lambda_l) \cdot \overline{\tau}_i{}^{n_i-(k-1)} \cdot \overline{\mathfrak{g}}_i \cdot \overline{\tau}_i^{k-1}$$
$$= \sum_{j,k}^{n_i} \overline{\tau}_i{}^{n_i-(j-1)} \cdot \overline{a}_{jk} \cdot \overline{\tau}_i^{k-1}$$

Now observe that $\overline{a}_{jk} \in \overline{\mathfrak{g}}_i A_i \overline{\mathfrak{g}}_i$, and so either $\overline{a}_{jk} = 0$, or $\overline{a}_{jk} = \overline{\beta}_i^{x_{jk}}$, for some x_{jk}. If $\overline{a}_{jk} = 0$, then

$$\psi_i(\overline{\tau}_i{}^{n_i-(j-1)} \cdot \overline{a}_{jk} \cdot \overline{\tau}_i^{k-1}) = \psi_i(0) = 0 = a_{jk}$$

If $\overline{a}_{jk} = \overline{\beta}_i^{x_{jk}}$ then

$$\begin{aligned}
\psi_i(\overline{\tau}_i{}^{n_i-(j-1)} \cdot \overline{a}_{jk} \cdot \overline{\tau}_i^{k-1})) &= \psi_i(\overline{\tau}_i{}^{n_i-(j-1)} \cdot \overline{\beta}^{x_{jk}} \cdot \overline{\tau}_i^{k-1}) \\
&= \psi_i(\overline{\tau}_i{}^{n_i-(j-1)}) \cdot \psi_i(\overline{\beta}^{x_{jk}}) \cdot \psi_i(\overline{\tau}_i{}^{k-1}) \\
&= \tau_i{}^{n_i-(j-1)} \cdot \beta^{x_{jk}} \cdot \tau_i{}^{k-1} \\
&= a_{jk}
\end{aligned}$$

Hence

$$\psi_i(\mu_i(\lambda_l)) = \psi_i\Big(\sum_{j,k}^{n_i} \overline{\tau}_i{}^{n_i-(j-1)} \cdot \overline{a}_{jk} \cdot \overline{\tau}_i^{k-1}\Big) = \sum_{j,k}^{n_i} a_{jk} = y \qquad \square$$

We now present the complete algorithm for computing the generators for $\mathrm{J}(A)$ as a two-sided ideal.

Theorem 3.3.2. *The following algorithm computes a sequence (of sets), S_{ij}, $1 \leq i, j \leq r$, such that:*

(i) $\bigcup_{i,j} S_{ij}$ *is a generating set for* $\mathrm{J}(A)$ *as a two-sided ideal.*

(ii) $f_i A f_j$ *is spanned by* S_{ij} *as a* K*-vector space.*

RadicalGenerators(Λ, $[\phi_i]$, $[\beta_i]$, $[\tau_i]$, $[\mathfrak{g}_i]$, $[n_i]$, $[d_i]$, r)

for (i, j) **in** $\{1 \dots r\} \times \{1 \dots r\}$

if $i = j$

$S_{ii} \doteq \{\mathfrak{g}_i \cdot {\tau_i}^a \cdot \big(\lambda - \mathbf{Psi}_i(\phi_i, \beta_i, \tau_i, g_i, \overline{\beta}_i, \overline{\tau}_i, \overline{g}_i, n_i, \lambda)\big) \cdot {\tau_i}^b \cdot \mathfrak{g}_i :$
$a, b \in \{1 \dots d_i\}, \lambda \in \Lambda\}$

else

$S_{ij} \doteq \{\mathfrak{g}_i \cdot {\tau_i}^a \cdot \lambda \cdot {\tau_j}^c \cdot \mathfrak{g}_j : a \in \{1 \dots d_i\}, c \in \{1 \dots d_j\}, \lambda \in \Lambda\}$

return S

Proof. Termination is immediate since r, Λ, and the d_i are finite, and $\mathbf{Psi}_i$ terminates by the 3.3.1. Correctness follows from 3.3.1 and 2.4.3. □

3.4 Pruning The Generators for the Radical of A

Our algorithm to compute generators for $\mathrm{J}(A)$ as a two-sided ideal is correct, but produces a large number of redundant generators. The main (but not only) source of this redundancy is that generators in S_{ij} are often linear combinations of products xy, with x a generator in S_{ik} and y a generator in S_{kj}. Such linear combinations in S_{ij} are not necessary to generate $\mathrm{J}(A)$ as a two-sided ideal. The following algorithm removes (prunes) such redundant generators. Its correctness and termination are obvious.

Lemma 3.4.1. *Given a sequence (of sets), S_{ij}, $1 \leq i, j \leq r$, such that $\bigcup_{i,j} S_{ij}$ is a generating set for $\mathrm{J}(A)$ as a two-sided ideal, the following algorithm computes a sequence (of sets), T_{ij}, such that:*

(i) $\bigcup_{i,j} T_{ij}$ is also a generating set for $\mathrm{J}(A)$ as a two-sided ideal.

(ii) if $t \in T_{ij}$ then t is not a K-linear combination of products xy, with $x \in S_{ik}$ and $y \in S_{kj}$.

Prune($[S_{ij}]$ **with each** $S_{ij} \in M_m(K)$**,** r**)**

$V \triangleq$ **vector space of** $m \times m$ **matrices over** K

for (i, j) **in** $\{1 \dots r\} \times \{1 \dots r\}$

$U_{ij} \triangleq$ **subspace of** V **spanned by** S_{ij}

$L_{ij} \triangleq \{x \cdot y : x \in S_{ik}, y \in S_{kj}, k \in \{1 \dots r\}\}$

$W_{ij} \triangleq$ **subspace of** V **spanned by** L_{ij}

$T_{ij} \triangleq$ **a basis of the complement in** U_{ij} **of** $U_{ij} \cap W_{ij}$

return T

3.5 Computing Generators for A

We have algorithms to compute generators for the semisimple part of A, and for the radical of A. Composing them gives an algorithm to compute a new generating set for A.

Theorem 3.5.1. *Let $\Lambda = \{\lambda_1, \dots, \lambda_t\}$, where each λ_i is an $m \times m$ matrix over a finite field K of characteristic p. Let $A \subseteq M_m(K)$ be the algebra generated by Λ. The following randomized algorithm computes a generating set for A. Specifically the algorithm computes $\{\beta_1, \tau_1, \dots \beta_r, \tau_r\}$ and sets T_{ij} such that:*

(i) $\{\beta_1, \tau_1, \dots \beta_r, \tau_r\}$ generates the semisimple part of A as a K-algebra.

(ii) $\bigcup_{i,j} T_{ij}$ generates $\mathrm{J}(A)$ as a two-sided ideal.

The algorithm also computes the list $\mathfrak{g}_1, \dots, \mathfrak{g}_r$ of the first little idempotents for each A'_i, the list $n_1, \dots, n_r$ giving the dimensions of each S_i over K_i, the list $d_1, \dots, d_r$ giving the degree of each K_i over K, and the list $h_1, \dots, h_r$ of minimal polynomials associated to each β_i.

Generators($\Lambda = \{\lambda_1, \ldots, \lambda_t\}$ **with each** $\lambda_i \in M_m(K)$)

$[A_i]$, $[n_i]$, $[d_i]$, $[q_i] \mathrel{\mathring{=}}$ **MeatAxe**(K^m, Λ)

$X \mathrel{\mathring{=}} K\langle x_1, \ldots, x_t \rangle$
create $\phi : X \to \Lambda$
create $[\phi_j : X \to A_j]$

$([\beta_i], [\tau_i], [\mathfrak{g}_i], [h_i]) \mathrel{\mathring{=}}$
    SemiSimpleGenerators(ϕ, $[\phi_i]$, $[n_i]$, $[d_i]$, $[q_i]$, r)
$[S_{ij}] \mathrel{\mathring{=}}$ **RadicalGenerators**(Λ, $[\phi_i]$, $[\beta_i]$, $[\tau_i]$, $[\mathfrak{g}_i]$, $[d_i]$, $[n_i]$, r)
$[T_{ij}] \mathrel{\mathring{=}}$ **Prune**($[S_{ij}]$, r)

return $([\beta_i], [\tau_i], [\mathfrak{g}_i], [T_{ij}], [n_i], [d_i], [h_i])$

Proof. Correctness is immediate from 3.2.19, 3.4.1, and 2.4.4. □

Chapter 4

Derivative Algorithms

Abstract. We present several useful algorithms derived from the algorithms in the previous chapter. As usual we prove correctness of these algorithms, but make no attempt to present the most efficient algorithms possible, nor to present every detail of every algorithm. We employ the notation and conventions of the previous chapter.

4.1 Computing The Basic and Split Basic Algebra for A

In the previous chapter we described the basic algebra associated to A as the subalgebra $B = \mathfrak{g}A\mathfrak{g}$ of A. Since $A \subseteq \mathrm{End}_K(M) \simeq M_m(K)$, with M the natural m-dimensional module for A, the elements of B can be represented as $m \times m$ matrices over K. Note, however, that since A acts faithfully on M, $\mathfrak{g}A\mathfrak{g}$ must also act faithfully on $\mathfrak{g}M$. Hence we can embed $\mathfrak{g}A\mathfrak{g}$ in $\mathrm{End}_K(\mathfrak{g}M)$, and therefore represent the elements of B as $\hat{m} \times \hat{m}$ matrices over K, where $\hat{m} = \dim_K(\mathfrak{g}M)$. The point here is that $\hat{m}$ is usually much smaller than m. We can exploit this fact to reduce the degree of the generators for B.

The function **Condense** is a standard function which returns $\hat{m}$, together with the embedding $\pi : \mathfrak{g}A\mathfrak{g} \to \mathrm{End}_K(\mathfrak{g}M)$. Magma source code for the **Condense** function can be found in Appendix A. Using the **Condense** function we can define an algorithm to compute the basic algebra associated to A as a subalgebra of $M_{\hat{m}}(K)$. This algorithm then leads to an obvious algorithm to compute the split basic algebra

associated to A as a subalgebra of $M_{\hat{m}}(L)$, where L is the splitting field for A. Correctness of these algorithms is immediate from 2.5.7 and 2.5.8.

Theorem 4.1.1. *Let $\Lambda = \{\lambda_1, \ldots, \lambda_t\}$, where each λ_i is an $m \times m$ matrix over a finite field K of characteristic p. Let $A \subseteq M_m(K)$ be the algebra generated by Λ. The following randomized algorithm computes the basic algebra B associated to A as a subalgebra of $M_{\hat{m}}(K)$, where $\hat{m} = \dim_K(\mathfrak{g}M)$. The algorithm also returns the list $d_1, \ldots, d_r$, where d_i gives the degree of K_i over K.*

Basic($\Lambda = \{\lambda_1, \ldots, \lambda_t\}$ with each $\lambda_i \in M_m(K)$)
  $([\beta_i], \_, [\mathfrak{g}_i], [T_{ij}], \_, [d_i], \_) \stackrel{\circ}{=}$ **Generators(Λ)**
  $(\hat{m}, \pi) \stackrel{\circ}{=}$ **Condense($\mathfrak{g}$) where** $\mathfrak{g} \stackrel{\circ}{=} \sum_i \mathfrak{g}_i$
  $M_{\hat{m}}(K) \supseteq B \stackrel{\circ}{=}$ **algebra generated by** $[\pi(\beta_i)]$ **and** $[\pi(T_{ij})]$

  return $(B, [d_i])$

Theorem 4.1.2. *Let $\Lambda = \{\lambda_1, \ldots, \lambda_t\}$, where each λ_i is an $m \times m$ matrix over a finite field K of characteristic p. Let $A \subseteq M_m(K)$ be the algebra generated by Λ. The following randomized algorithm computes the splitting field L for A, and returns the split basic algebra B^L associated to A as a subalgebra of $M_{\hat{m}}(L)$, where $\hat{m} = \dim_K(\mathfrak{g}M) = \dim_L((\mathfrak{g}M)^L)$.*

SplitBasic($\Lambda = \{\lambda_1, \ldots, \lambda_t\}$ with each $\lambda_i \in M_m(\mathbb{F}_{p^c})$)

$(B \subseteq M_{\hat{m}}(K), [d_i]) \triangleq$ **Basic(Λ)**
$l \triangleq c \cdot \mathrm{LCM}([d_i])$
$L \triangleq \mathbb{F}_{p^l}$
$M_{\hat{m}}(L) \supseteq B^L \triangleq$ **algebra generated by** $[1_L \otimes \pi(\beta_i)] \cup [1_L \otimes \pi(T_{ij})]$
where B **is generated as a** K**-algebra by** $[\pi(\beta_i)]$ **and** $[\pi(T_{ij})]$

return B^L

4.2 Computing a Presentation for A

The final algorithm in this chapter is an algorithm to compute a presentation for an algebra A. The input to our algorithm is a set $\Lambda = \{\lambda_1, \ldots, \lambda_t\}$, where each λ_i is an $m \times m$ matrix over a finite field K of characteristic p. Let $A \subseteq M_m(K)$ be the algebra generated by Λ.

We can compute generators $\{\beta_1, \ldots, \beta_r, \tau_1, \ldots, \tau_r, \zeta_1, \ldots, \zeta_k\}$ for A, where the β_i and τ_i generate A' as a K-algebra, and the ζ_j generate $\mathrm{J}(A)$ as a two-sided ideal. Let P be the free non-commutative algebra $K\langle B_1, \ldots, B_r, T_1, \ldots T_r, Z_1, \ldots, Z_k\rangle$. We have an onto K-algebra homomorphism, $\psi : P \to A$, defined by $B_i \mapsto \beta_i$, $T_i \mapsto \tau_i$, and $Z_j \mapsto \zeta_j$. Let $\Omega = \ker \psi$. Then we have a presentation for A as P/Ω. We now consider how to compute generators for Ω.

Remark. If A is a basic algebra (so all the n_i are equal to 1), then we omit the T_i generators for P.

4.2.1 Computing the Semisimple Generators for Ω

There are generators for Ω coming from relations among the generators of A'. Recall that β_i and τ_i generate A'_i, and satisfy the relations implied by 2.1.5. Hence for $1 \leq i \leq r$ we have the following generators for Ω:

(i) $T_i^{n_i} B_i - B_i$.

(ii) $B_i T_i^{n_i} - B_i$.

(iii) $T_i^{n_i} T_i - T_i$.

(iv) $T_i T_i^{n_i} - T_i$.

(v) $B_i T_i^{\,j} B_i$, $1 \leq j \leq n_i$.

(vi) $\sum_{j=1}^{n_i} T_i^{\,n_i - j} B_i^{\,q_i - 1} T_i^{\,j} - T_i^{\,n_i}$.

(vii) $B_i h_i(B_i)$

We also have orthogonality relations between the A_i' components of A'. Specifically we have the following generators for Ω: $B_iB_j, T_iT_j, B_iT_j, T_iB_j$, for $1 \leq i, j \leq r$, $i \neq j$. We have a function **SemisimpleGenerators**$_\Omega$ which returns a list of the above generators. Source for this function can be found in Appendix A.

4.2.2 Computing The Radical Generators for Ω

There are generators for Ω coming from relations among the generators of $\mathrm{J}(A)$. Since A is finite dimensional, $J(A)^d = 0$ for some d. Let V be the K-vector space with basis the set of monomials in $Z_1, \ldots, Z_k$ of degree greater than or equal to 1 and less than or equal to d. Let $\theta' : V \to \mathrm{J}(A)$ be the vector space map defined by

$$\prod_j {Z_j}^{\alpha_j} \mapsto \prod_j {\zeta_j}^{\alpha_j}$$

One way to compute the relations among the $Z_1, \ldots, Z_k$ is to compute the kernel of θ'. Unfortunately such a computation could potentially be prohibitively expensive, as $\mathrm{J}(A)$ may have high dimension, and there may be a huge number of monomials in the basis for V. We can alleviate these problems via two 'tricks'.

First, recall that the ζ_j generators for $\mathrm{J}(A)$ are in fact all elements of the basic algebra $\mathfrak{g}A\mathfrak{g} \subseteq A$. Now let B be the basic algebra for A computed via the algorithm **Basic**. Then $\mathfrak{g}A\mathfrak{g} \stackrel{\pi}{\simeq} B$, and we can form the vector space map $\theta : V \to \mathrm{J}(B)$ defined by

$$\prod_j {Z_j}^{\alpha_j} \mapsto \prod_j \pi(\zeta_j)^{\alpha_j}$$

Observe that $x \in \ker \theta'$ if and only if $x \in \ker \theta$. In other words a relation among the $\zeta_j \in \mathfrak{g}A\mathfrak{g}$ holds if and only if the corresponding relation holds among the $\pi(\zeta_j) \in \mathrm{J}(B)$. The point here is that computing in $\mathrm{J}(B)$ will usually be much faster than computing in $\mathrm{J}(A)$, since the dimension of $\mathrm{J}(B)$ will usually be much smaller than the dimension of $\mathrm{J}(A)$.

Second, we can embed V into $K\langle Z_1, \ldots, Z_k\rangle$ (as a subspace), and use the ideal structure of $K\langle Z_1, \ldots, Z_k\rangle$ to control the number of monomials in $Z_1, \ldots, Z_k$ that we must consider. Some care is needed in this approach since computing with ideals in $K\langle Z_1, \ldots, Z_k\rangle$ will involve non-commutative Gröbner basis constructions, which may themselves be extremely expensive, or even non-terminating. In light of this we mainly use ideals in $K\langle Z_1, \ldots, Z_k\rangle$ generated by monomials (the calculation of Gröbner basis for such ideals almost always terminates, and almost always completes very rapidly).

Theorem 4.2.1. *Suppose that A is a K-algebra, and B is the basic algebra for A computed by the algorithm **Basic**. Suppose B is generated by $\{\beta_1, \ldots, \beta_r, \zeta_1, \ldots, \zeta_k\}$, where the ζ_j generate $\mathrm{J}(B)$ as a two-sided ideal. Let $Q = K\langle Z_1, \ldots, Z_k\rangle$, and let V and $\theta : V \to \mathrm{J}(B)$ be defined as before. Then the following algorithm either fails to terminate, or computes:*

(i) a set $\Delta_r \subseteq V$ such that $\ker \theta$ equals the ideal H generated by Δ_r, and hence $V/H \simeq \mathrm{J}(B)$ as vector spaces.

(ii) a basis C for V/H.

RadicalGenerators$_\Omega$(Q, $\theta : V \to \mathrm{J}(B)$)

$c \triangleq 1$
$I_1 \triangleq$ **the zero ideal in** Q
$L_1 \triangleq \{Z_1, \ldots, Z_k\}$
$\Delta_r \triangleq []$

repeat
$c \triangleq c + 1$

$M_c \triangleq$ **the set of monomials in** $Z_1, \ldots, Z_k$ **of degree** c
$L_c \triangleq L_{c-1} \cup (M_c \setminus I_{c-1})$
$V_c \triangleq$ **the subspace of** V **with basis** L_c
$N_c \triangleq$ **NullSpace**($\theta_{|_{V_c}}$)
$X_c \triangleq$ **a basis for** N_c
$\Delta_r \triangleq \Delta_r \cup X_c$
$I_c \triangleq$ **two-sided ideal of** Q **generated by** $I_{c-1} \cup (L_c \cap N_c)$

until $\theta(M_c) = \{0\}$

$H \triangleq$ **ideal in** Q **generated by** Δ_r
$S \triangleq Q/H$
$C \triangleq$ **MonomialBasis**(S)$\setminus\{1\}$

return (C, Δ_r)

Proof. The algorithm may not terminate due to the fact that the construction of ideals in Q involves Gröbner basis calcuations. If these calculations all terminate, however, then so does the algorithm, since $J(B)^d = 0$ for some d.

Assume the algorithm terminates. Inductively we have that L_c is a subset of the set of all monomials in $Z_1, \ldots, Z_k$ of degree greater than or equal to 1 and less than or equal to c, and hence V_c is a subspace of V and $\ker(\theta_{|_{V_c}}) \subseteq \ker\theta$. It follows that we finish the loop with L_d a subset of the set of all monomials in $Z_1, \ldots, Z_k$ of degree

greater than or equal to 1 and less than or equal to d, and $\Delta_r \subseteq \ker\theta_{|V}$. Hence $H \subseteq \ker\theta$.

Inductively we also have that $I_{c-1} \subseteq I_c$, and that I_c is contained in the ideal generated by Δ_{rc} (the value of Δ_r on the c-th loop iteration, where the first loop iteration corresponds to $c = 2$). It follows that $I_c \subseteq \ker\theta$. Now suppose that $v \in \ker\theta$. We can write

$$v = \sum_i \alpha_i m_i$$

where the $\alpha_i \in K$, and the m_i are monomials in $Z_1, \ldots, Z_k$. Let $a = \mathrm{Max}\{\deg m_i\}$. Then we can write v as

$$v = \sum_i \alpha_i m_i + \sum_j \alpha_j {m_j}'$$

where the ${m_j}' \in I_{a-1}$, and the ${m_i}' \notin I_{a-1}$. Since $I_{a-1} \subseteq \ker\theta$ we then have that

$$0 = \theta(v) = \theta(\sum_i \alpha_i m_i + \sum_j \alpha_j {m_j}') = \theta(\sum_j \alpha_j {m_j}')$$

Hence $\sum_j \alpha_j {m_j}' \in N_a$. Thus v is in the ideal generated $X_a \cup I_{a-1}$, and thus in the ideal generated by $\Delta_{ra} \cup I_{a-1}$. It follows that $v \in H$, and hence that $H = \ker\theta$.

To finish the proof we require one fact about Gröbner bases, namely that if I is an ideal in a free non-commutative (or commutative) algebra R over a field K, then R/I has a computable K-basis consisting of monomials. □

4.2.3 Computing the Cross Generators for Ω

There are generators for Ω coming from relations betwen the generators of A' and the generators of $\mathrm{J}(A)$. Recall that each ζ_j is an element of $f_i A f_l$, for some unique choice of i and l. Hence we have identity relations between ${B_i}^{q_i-1}$ and ${T_i}^{n_i}$, and any Z_j such that $\psi(Z_j) \in f_i A f_l$. Specifically for $1 \leq i, l \leq r$, $1 \leq j \leq k$, we have the following generators for Ω:

(i) ${B_i}^{q_i-1} Z_j - Z_j$, if $Z_j \in f_i A f_l$.

(ii) $Z_j {B_i}^{q_i-1} - Z_j$, if $Z_j \in f_l A f_i$.

(iii) ${T_i}^{n_i} Z_j - Z_j$, if $Z_j \in f_i A f_l$.

(iv) $Z_j {T_i}^{n_i} - Z_j$, if $Z_j \in f_l A f_i$.

We also have orthogonality relations between B_i and T_i, and any Z_j such that $\psi(Z_j) \notin f_i A f_l$. Specifically for $1 \leq i, l \leq r$, $1 \leq j \leq k$, we have the following generators for Ω:

(i) $B_i Z_j$, if $Z_j \notin f_i A f_l$.

(ii) $Z_j B_i$, if $Z_j \notin f_l A f_i$.

(iii) $T_i Z_j$, if $Z_j \notin f_i A f_l$.

(iv) $Z_j T_i$, if $Z_j \notin f_l A f_i$.

We have scaling relations between each B_i and any Z_j such that $\psi(Z_j) \in f_i A f_l$. Observe that since β_i is in the basic algebra B, we can compute these scaling relations using $\pi(\beta_i)$ and the $\pi(\zeta_j)$, rather than β_i and the ζ_j. Suppose $\{\tilde{Z}_1, \ldots, \tilde{Z}_t\}$ is the basis for $V/H \overset{\rho}{\simeq} \mathrm{J}(B)$ computed by the $\mathbf{RadicalGenerators}_\Omega$ algorithm. Then for $1 \leq i, l \leq r$, $1 \leq j \leq k$ we have the following generators for Ω:

(i) $B_i Z_j - \sum_{s=1}^{t} \alpha_s \tilde{Z}_s$, if $Z_j \in f_i A f_l$, and $\pi(\beta_i)\pi(\zeta_j) = \sum_{s=1}^{t} \alpha_s \rho(\tilde{Z}_s)$

(ii) $Z_j B_i - \sum_{s=1}^{t} \alpha_s \tilde{Z}_s$, if $Z_j \in f_l A f_i$, and $\psi(Z_j B_i) = \sum_{s=1}^{t} \alpha_s \rho(\tilde{Z}_s)$

Finally, recall that $\mathfrak{g} = \sum_{i=1}^{r} \mathfrak{g}_i$, where $\mathfrak{g}_i$ is the first little idempotent in $f_i A f_i$. For $1 \leq m \leq n_{i-1}$ we have

$$\mathfrak{g} {\tau_i}^m \mathfrak{g} = \mathfrak{g}_i {\tau_i}^m \mathfrak{g}_i = 0$$

Moreover for any ζ_j we have

$$\mathfrak{g} \zeta_j = \zeta_j \mathfrak{g} = \zeta_j$$

Hence

$$(\tau_i{}^m \zeta_j \tau_i{}^l) \cdot (\tau_i{}^{m'} \zeta_{j'} \tau_i{}^{l'}) = \tau_i{}^m \zeta_j \mathfrak{g} \tau_i{}^l \tau_i{}^{m'} \mathfrak{g} \zeta_{j'} \tau_i{}^{l'}$$
$$= \begin{cases} \tau_i{}^m \zeta_j \zeta_{j'} \tau_i{}^{l'} & \text{if } l + m' = 0 \bmod n_i \\ 0 & \text{otherwise} \end{cases}$$

The point here is that we already know how to compute $\zeta_j \zeta_{j'}$ using the radical generators for Ω. Hence any *obvious* scaling relations between T_i and any Z_j such that $\psi(Z_j) \in f_i A f_l$, are implicit in the relations we have already listed. We revisit the issue of non-obvious scaling relations at the end of the chapter.

We have a function $\mathbf{CrossGenerators}_\Omega$ which returns a list of the above generators. Source for this function can be found in Appendix A.

4.2.4 Computing A Presentation for A

We conclude with the algorithm to compute a presentation for A.

Theorem 4.2.2. *Let $\Lambda = \{\lambda_1, \ldots, \lambda_t\}$, where each λ_i is an $m \times m$ matrix over a finite field K of characteristic p. Let $A \subseteq M_m(K)$ be the algebra generated by Λ. Then the following randomized algorithm either fails, or computes a presentation for A.*

Presentation($\Lambda = \{\lambda_1, \ldots, \lambda_t\}$ **with each** $\lambda_i \in M_m(K)$)

$([\beta_1, \ldots, \beta_r], [\tau_1, \ldots, \tau_r], \_, [\zeta_1, \ldots, \zeta_k], [n_i], \_, [h_i]) \triangleq$ **Generators**(Λ)
$P \triangleq K\langle B_1, \ldots, B_r, T_1, \ldots T_r, Z_1, \ldots, Z_k\rangle$

$\Delta_s \triangleq$ **SemisimpleGenerators**$_\Omega$($[B_i]$, $[T_i]$, $[n_i]$, $[q_i]$, $[h_i]$)

$B \triangleq$ **Basic**(Λ)
$Q \triangleq K\langle Z_1, \ldots, Z_k\rangle \subseteq P$
$\theta \triangleq$ **the map** $V \to B$ **defined by** $\prod_j Z_j{}^{\alpha_j} \mapsto \prod_j \pi(\zeta_j)^{\alpha_j}$
where $[\pi(\beta_i)] \cup [\pi(\zeta_j)]$ **generates** B
$(C, \Delta_r) \triangleq$ **RadicalGenerators**$_\Omega$(Q, θ)

$\Delta_c \triangleq$ **CrossGenerators**$_\Omega$(C, $[\pi(\beta_i)]$, $[B_i]$, $[T_i]$, $[Z_i]$, $[n_i]$, $[q_i]$)

$\Delta \triangleq$ **ideal in** P **generated by** $\Delta_s \cup \Delta_r \cup \Delta_c$

if $\dim_K(P/\Delta) \neq \dim_K(A)$
fail
else
return P/Δ

Proof. By 4.2.1 and our discussion of the semisimple generators for Ω and the cross generators for Ω, we know that $\Delta \subseteq \Omega$. The dimension check at the end of the algorithm ensures that $\Delta = \Omega$, and hence $A \simeq P/\Omega = P/\Delta$. □

We conclude by noting that the dimension check in the above algorithm can possibly cause the algorithm to fail. Such a failure happens if we do not have enough generators for Ω. It's clear that if A is a basic algebra, we always have enough generators for Ω. The question is whether such a failure can happen if A is not basic? Answering this questions amounts to proving that there are no non-obvious scaling relations between T_i and any Z_j such that $\psi(Z_j) \in f_i A f_l$, or more precisely

that the elements ${\tau_i}^m \zeta_j {\tau_i}^l$, $1 \leq i \leq r$, $1 \leq m, l \leq n_i$, $1 \leq j \leq k$, form a K-basis for $\mathrm{J}(A)$. While we strongly suspect that this is the case, we have not yet formally proven this claim.

Chapter 5

Examples

Abstract. We present several examples of computing presentations for basic algebras using the programs in the appendix.

5.1 The Structure of Each Example

In each example we define a K-algebra A, and a faithful representation ρ of A. We then compute the split basic algebra B associated to A. Run times are given for each of the main steps involved in computing B. These steps are:

(i) *MeatAxe* – computing the A_i, n_i, d_i, and q_i.

(ii) *Semisimple Generators* – computing generators for the semisimple part of A.

(iii) *Radical Generators* – computing generators for $\mathrm{J}(A)$.

(iv) *Pruning* – pruning the generators for $\mathrm{J}(A)$.

(v) *Split Basic* – computing the split basic algebra.

We also report the number of generators computed in steps (*ii*), (*iii*), and (*iv*).

For each example we also give information about the simple A-modules. Specifically, if S_i is the i-th simple A-module, then we give the K-dimension of S_i, and the degree of $K_i = \mathrm{End}_A(S_i)^{\mathrm{op}}$ over K in the Wedderburn decomposition of $\overline{A}$. Most of this information is either computed in Magma or read from the tables in [13]. Note that we list the simple modules in decreasing order by dimension. While this is

somewhat non-standard it remains faithful to our programs, which reverse the usual order for reasons of efficiency.

We give the splitting field L for A, and report the degree and dimension of B over L, together with the number of simple B-modules. We then compute a presentation for B as a free non-commutative algebra P, modulo an ideal Ω. Run times are given for each of the main steps involved in computing this presentation. These steps are:

(i) *Generators* – computing generators for B.

(ii) *Semisimple Generators for* Ω – computing generators for Ω coming from relations within B' (recall that $B = B' \oplus \mathrm{J}(B)$, where B' is the semisimple part of B).

(iii) *Radical Generators for* Ω – computing generators for Ω coming from relations within $\mathrm{J}(B)$, together with a basis for $\mathrm{J}(B)$.

(iv) *Cross Generators for* Ω – computing generators for Ω coming from relations between B' and $\mathrm{J}(B)$.

(v) *Gröbner Basis* – computing a non-commutative Gröbner basis for the ideal generated by the elements computed in steps (*ii*), (*iii*), and (*iv*).

We also report the number of relations computed in each of the above steps, and the dimensions over L of B, B', $\mathrm{J}(B)$, and $B' \cap \mathrm{J}(B)$. Finally we give the computed presentation for B. Note that the presentations we give contain a large number of redundant relations.

All reported run times are exact (to within the limits imposed by the operating system), while all reported memory usage figures are (reasonably accurate) estimates based on the memory usage information available in Magma. All performance figures were generated on a Sun Blade 1000 with 8 gigabytes of memory and two UltraSPARC-III 750 Mhz processors, running Solaris 5.9.

Note that some of the information we give is redundant, in the sense that we can trivially calculate it from other information (e.g. the dimension over L of B' must be equal to the number of simple B-modules). Reporting the actual calculated values for such information does, however, give some indication of the correctness of the implementations of our algorithms.

5.2 Background

To construct our examples we require some background results from the representation theory of finite groups.

Definition 5.2.1. Let G be a finite group, and $\rho : G \to \mathrm{GL}(V)$ be a representation of G. Then ρ is called a *faithful representation of* G if $\rho(g) = 1$ implies $g = 1$.

Note that being a faithful representation of a group G in the above sense is not the same as being a faithful representation of an algebra in the sense of 1.12.1. We use two facts about faithful representations of finite groups.

Lemma 5.2.2. *If G is a finite simple group, then every non-trivial representation of G is faithful.*

Theorem 5.2.3 (Burnside). *Suppose $\rho : G \to \mathrm{GL}(M)$ is a faithful representation of the finite group G. Let M^n denote the KG-module $M \otimes M \otimes \ldots \otimes M$ (n factors). Then every simple KG-module is a composition factor of at least one of the modules M^n, $n = 1, 2, \ldots$.*

Proof. Theorem 32.9 of [10]. □

We require several results about duality, projectivity and injectivity of KG-modules.

Lemma 5.2.4. *If U, V and W are KG-modules then*

$$\mathrm{Hom}_{KG}(U \otimes V, W) \simeq \mathrm{Hom}_{KG}(U, V^* \otimes W)$$

Proof. Lemma 7.3 of [1]. □

Lemma 5.2.5. *If P is a projective KG-module, and U is any KG-module, then $P \otimes U$ is a projective KG-module.*

Proof. Lemma 7.4 of [1]. □

Theorem 5.2.6. *Every finitely generated injective KG-module is projective, and every finitely generated projective KG-module is injective.*

Proof. Corollary 2.7 of [9]. □

We also require the following fact about the socle and head of projective indecomposable KG-modules.

Definition 5.2.7. The *socle* of a module M, denoted $\mathrm{Soc}(M)$, is the sum of its simple submodules. The *top* of a module M, denoted $\mathrm{Head}(M)$, is the module $M/\mathrm{Rad}(M)$.

Theorem 5.2.8. *Let P be a projective indecomposable KG-module. Then* $\mathrm{Soc}(P) \simeq \mathrm{Head}(P) \simeq P/\mathrm{Rad}(P) \simeq S$*, where S is the simple module corresponding to P.*

Proof. Theorem 3.6 of [9]. □

We conclude this section with some results from block theory.

Definition 5.2.9. A *central idempotent* in a finite dimensional algebra A is an idempotent in the center of A. A *primitive central idempotent* is a central idempotent not expressible as the sum of two orthogonal central idempotents.

Lemma 5.2.10. *In a finite dimensional algebra A we can write $1 = e_1 + \cdots + e_n$, with the e_i orthogonal central idempotents of A, and $A = B_1 \oplus \cdots \oplus B_n$, with each B_i a two-sided indecomposable ideal of A given by $B_i = e_i A$. Moreover, this decompositon of A is unique up to reordering of summands.*

Proof. Definition 1.8.1 and Lemma 1.8.2 of [4]. □

Definition 5.2.11. The indecomposable two-sided ideals in the above decomposition are called the *blocks* of A. Note that each block B_i is a subalgebra of A with identity e_i.

Now suppose that M is an indecomposable A-module. Then

$$M = 1 \cdot M = (e_1 + \cdots + e_n) \cdot M = e_1 M \oplus \cdots \oplus e_n M$$

and hence $M = e_i M$ for some i. In this case we say that M *belongs to* the block B_i. Thus the simple and projective indecomposable modules are classified into blocks. Note that if an indecomposable module is in a given block, then so are all of its composition factors.

Definition 5.2.12. The *principal block* of A, denoted $B_0(A)$, is the block containing the trivial A-module.

5.3 M_{11} in Characteristic 2

Let G be M_{11} – the smallest sporadic simple group. The order of G is 7920. G can be constructed as the subgroup of S_{11} generated by the two permutations (1 10)(2 8)(3 11)(5 7) and (1 4 7 6)(2 11 10 9).

Let K be $\mathbb{F}_2$, A be the group algebra of G over K, and M be a 7920 dimensional vector space over K. The regular representation $\rho : A \to \mathrm{End}_K(M)$ is faithful by 1.12.6, but it has degree 7920, which is quite large. We wish to find a smaller degree representation for A.

The simple A-modules are as follows

Simple	$\dim_K S_i$	$[K_i : K]$
S_4	44	1
S_3	32	2
S_2	10	1
S_1	1	1

S_3 is a projective A-module which is self dual. For $1 \leq i \leq 4$, let P_i be the projective indecomposable which corresponds by 1.6.9 to the simple module S_i.

Observe that as $G \subseteq S_{11}$, there is a natural permutation representation of G on an 11-dimensional vector space M. The two composition factors of M are the trivial module (S_1) and S_2. Let $\rho_2 : G \to \mathrm{End}_K(S_2)$ be the representation for G corresponding to S_2. Since G is simple, ρ_2 is faithful by 5.2.2. Hence by 5.2.3 we should be able to find all other simple A-modules by forming tensor powers of S_2. Let $T_{2,2} = S_2 \otimes S_2$. A composition series for $T_{2,2}$ has length 5, and one of the composition factors is S_4. We could form higher tensor powers of S_2 to find more simple A-modules, but instead we let $T_{2,4} = S_2 \otimes S_4$. A composition series for $T_{2,4}$ has length 20, and one of the composition factors is S_3. Let $\rho_3 : G \to \mathrm{End}_K(S_3)$ be the group representation for G corresponding to S_3. Since G is simple, ρ_3 is faithful by 5.2.2. Hence by 5.2.3 we should be able to find all simple A-modules by forming tensor powers of S_3. Let $T_{3,3} = S_3 \otimes S_3$. The K-dimension of $T_{3,3}$. is 1024.

In Magma we construct $T_{3,3}$ as follows

```
load m11;       //  G = M_11
F := GF(2);
M := PermutationModule(G, F);
C := isomorphism_classes(CompositionFactors(M));
S_1 := degree(C, 1);
S_2 := degree(C, 10);
```

```
T_2_2 := TensorProduct(S_2, S_2);
S_4 := degree_M(T_2_2, 44);

T_2_4 := TensorProduct(S_2, S_4);
S_3 := degree_M(T_2_4, 32);

T_3_3 := TensorProduct(S_3, S_3);
```

By 5.2.4 we observe that for any i

$$\begin{aligned}\mathrm{Hom}_A(S_i \otimes S_3, S_3) &\simeq \mathrm{Hom}_A(S_i, {S_3}^* \otimes S_3) \\ &= \mathrm{Hom}_A(S_i, S_3 \otimes S_3) \\ &= \mathrm{Hom}_A(S_i, T_{3,3})\end{aligned}$$

We can verify that $\mathrm{Hom}_A(S_i \otimes S_3, S_3) \neq 0$ for every i, by checking whether S_i occurs as a composition factor of $S_i \otimes S_3$. In Magma we do this as follows

```
error if #all_degree_M(TensorProduct(S_2, S_3), 10) eq 0, "";
error if #all_degree_M(TensorProduct(S_3, S_3), 32) eq 0, "";
error if #all_degree_M(TensorProduct(S_4, S_3), 44) eq 0, "";
```

Now fix a choice of i. Since $\mathrm{Hom}_A(S_i, T_{3,3}) \neq 0$, there is a nonzero map $\phi_i : S_i \to T_{3,3}$. Since $\mathrm{Soc}(P_i) = S_i$ by 5.2.8, we therefore have the following diagram

$$\begin{array}{ccccc} 0 & \longrightarrow & S_i & \xrightarrow{\iota_i} & P_i \\ & & \downarrow{\scriptstyle \phi_i} & & \\ & & T_{3,3} & & \end{array}$$

Since S_3 is projective, $T_{3,3}$ is projective by 5.2.5, and hence injective by 5.2.6. We therefore have a map $\psi_i : P_i \to T_{3,3}$ such that $\psi_i \circ \iota_i = \phi_i$. Since $\mathrm{Soc}(P_i) = S_i$ is mapped injectively by ψ_i, we must have that ψ_i is an injective map. We therefore have the exact sequence

$$0 \to P_i \xrightarrow{\psi_i} T_{3,3} \xrightarrow{\pi_i} T_{3,3}/P_i \to 0$$

Now P_i is projective, and therefore injective by 5.2.6. Hence the map π_i splits and so P_i is a direct summand of $T_{3,3}$. We have therefore proved that $T_{3,3}$ is an A-module containing every projective indecomposable A-module as a summand, and hence by 1.12.5 $T_{3,3}$ is a faithful A-module.

We construct the split basic algebra associated to A via the following Magma code

```
B := split_basic_M(T_3_3 : params := params_thesis);
```

Timings and memory usage for this computation are

Step	**Time**
MeatAxe	7.55 sec
Semisimple Generators	1102.11 sec $\approx$ 18.5 mins
Radical Generators	4531.08 sec $\approx$ 76 mins
Pruning	235.54 sec $\approx$ 4 mins
Split Basic	3.89 sec
Total Time	5880.17 sec $\approx$ 1 hr 38 mins
Total Memory	90.45 MB

The number of generators for A computed in the various phases is

Type	**Number**
Semisimple Generators	8
Radical Generators	34
Radical Generators After Pruning	6

The splitting field for A is $L = \mathbb{F}_4$. The computed split basic algebra B has degree 58 and dimension 24. There are 5 simple B-modules, all of which are of course 1-dimensional (over L). We compute a presentation for B via the following Magma code

```
Q := presentation_A(B : params := params_thesis);
```

Timings and memory usage for this computation are

Step	**Time**
Generators	1.3 sec
Semisimple Generators for Ω	0 sec
Radical Generators for Ω	1.92 sec
Cross Generators for Ω	0.06 sec
Gröbner Basis	0.03 sec
Total Time	3.31 sec
Total Memory	2.84 Mb

The number of generators for Ω computed in the various phases is

Type	**Number**
Semisimple Generators for Ω	30
Radical Generators for Ω	115
Cross Generators for Ω	72
Total	217

The dimensions for the various parts of B are as follows

Part	Dimension over L
B'	5
$J(B)$	19
$B \cap J(B)$	0
B	24

Theorem 5.3.1. *Let A be the group algebra of M_{11} over $\mathbb{F}_2$. Then the split basic algebra B associated to A has a presentation as the quotient of the free algebra $\mathbb{F}_4\langle B_1, B_2, B_3, B_4, B_5, Z_1, Z_2, Z_3, Z_4, Z_5, Z_6\rangle$ by the ideal generated by the following elements:*

$$Z_2Z_4Z_5Z_3 + Z_2Z_3, {Z_6}^4 + {Z_6}^3 + Z_5Z_4, Z_2Z_3Z_2, Z_2Z_4Z_6, Z_3Z_2Z_3, Z_3Z_2Z_4 + Z_4Z_6,$$

$$Z_4Z_5Z_4, Z_4{Z_6}^2, Z_5Z_3Z_2 + Z_6Z_5, Z_5Z_4Z_5, Z_5Z_4Z_6 + {Z_6}^3 + Z_5Z_4, Z_6Z_5Z_3,$$

$$Z_6Z_5Z_4 + {Z_6}^3 + Z_5Z_4, {Z_6}^2Z_5, {B_1}^2 + {B_1}^2B_1B_1B_2, B_1B_3, B_1B_4, B_1B_5, B_1Z_1,$$

$$B_1Z_2, B_1Z_3, B_1Z_4, B_1Z_5, B_1Z_6, B_2B_1, {B_2}^2 + B_1B_2, B_2B_3, B_2B_4, B_2B_5, B_2Z_1,$$

$$B_2Z_2, B_2Z_3, B_2Z_4, B_2Z_5, B_2Z_6, B_3B_1, B_3B_2, {B_3}^2 + B_1B_3, B_3B_4, B_3B_5,$$

$$B_3Z_1 + B_1Z_1, B_3Z_2 + B_1Z_2, B_3Z_3, B_3Z_4, B_3Z_5, B_3Z_6, B_4B_1, B_4B_2, B_4B_3,$$

$${B_4}^2 + {B_1}^2B_4, B_4B_5, B_4Z_1, B_4Z_2, B_4Z_3 + {B_1}^2Z_3, B_4Z_4 + {B_1}^2Z_4, B_4Z_5, B_4Z_6,$$

$$B_5B_1, B_5B_2, B_5B_3, B_5B_4, {B_5}^2 + {B_1}^2B_5, B_5Z_1,$$

$$B_5Z_2, B_5Z_3, B_5Z_4, B_5Z_5 + {B_1}^2Z_5, B_5Z_6 + {B_1}^2Z_6, Z_1B_1, Z_1B_2,$$

$$Z_1B_3 + B_1Z_1, Z_1B_4, Z_1B_5, {Z_1}^2 + Z_2Z_3, Z_1Z_2, Z_1Z_3, Z_1Z_4, Z_1Z_5, Z_1Z_6, Z_2B_1,$$

$$Z_2B_2, Z_2B_3, Z_2B_4 + {B_1}^2Z_2, Z_2B_5, Z_2Z_1, {Z_2}^2, Z_2Z_5, Z_2Z_6, Z_3B_1, Z_3B_2,$$

$$Z_3B_3 + B_1Z_3, Z_3B_4, Z_3B_5, Z_3Z_1, {Z_3}^2, Z_3Z_4, Z_3Z_5, Z_3Z_6, Z_4B_1,$$

$$Z_4B_2, Z_4B_3, Z_4B_4, Z_4B_5 + {B_1}^2Z_4, Z_4Z_1, Z_4Z_2, Z_4Z_3, {Z_4}^2,$$

$$Z_5B_1, Z_5B_2, Z_5B_3, Z_5B_4 + {B_1}^2Z_5, Z_5B_5, Z_5Z_1, Z_5Z_2, {Z_5}^2, Z_5Z_6,$$

$$Z_6B_1, Z_6B_2, Z_6B_3, Z_6B_4, Z_6B_5 + {B_1}^2 Z_6, Z_6Z_1, Z_6Z_2, Z_6Z_3, Z_6Z_4$$

Note that the B_i correspond to the 5 simple B-modules, while the Z_j correspond to the 6 generators for $\mathrm{J}(B)$ *(and for* $\mathrm{J}(A)$*) after pruning.*

5.4 A Split Extension of A_5

Conjecture. *Suppose G is a finite group and k is a field of characteristic p. Then the following are equivalent.*

(i) The centraliser of every element of order p in G is p-nilpotent (i.e., has a normal p-complement).

(ii) For every non-projective module M in the principal block $B_0(kG)$, $H^n(G, M) \neq 0$ for some (and hence for infinitely many) n.

This conjecture, Conjecture 1.2 in [5], is proved by the authors for p odd (the conjecture was proved for all p in [3]). At the end of Section 9 of [5] the authors comment that '*Gorenstein has classified the finite groups in which the centraliser of every involution is 2-nilpotent. It may be that the best way to prove Conjecture 1.2 in case $p = 2$ is to go through the cases in this theorem individually. We have already initiated this process in Section 4 by treating the case where the Sylow 2-subgroups are dihedral. An example which we have not been able to tackle by the methods presented here is the split extension $(\mathbb{Z}/2)^{2n} : SL_2(2n)$, where the normal subgroup is formed from the natural module by restricting the field of coefficients to $\mathbb{F}_2$.*'.

We construct a particular example of the split extension $(\mathbb{Z}/2)^{2n} : \mathrm{SL}_2(2n)$, for the case $n = 2$, and compute the basic algebra for the associated group algebra in characteristic 2. Let E be an elementary abelian group of order 2^4, $H = A_5$, and $G = E \ltimes H$. The order of G is 960. As $A_5 \simeq \mathrm{SL}(2, \mathbb{F}_4) \subseteq \mathrm{SL}(3, \mathbb{F}_4)$, we can construct G as a subgroup of $\mathrm{SL}(3, \mathbb{F}_4)$ generated by the following matrices

$$h_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix} \quad h_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & \omega \\ 0 & 0 & 1 \end{pmatrix} \quad h_3 = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

where ω is a primitive element of $\mathbb{F}_4$.

Let $K = \mathbb{F}_2$, A be the group algebra of G over K, and M be the natural 960 dimensional module for A. The regular representation $\rho : A \to \mathrm{End}_K(M)$ has degree 960, and is faithful by 1.12.6.

A composition series for M has length 384. There are 3 isomorphism classes of composition factors of M, and hence by 1.12.3 there are 3 simple A-modules. The simple A-modules are as follows

Simple	$\dim_K S_i$	$[K_i : K]$
S_1	4	1
S_2	4	2
S_3	1	1

The Magma code to construct M and to compute the split basic algebra associated to A is

```
F := GF(4);
G := SL(3, F);
H := sub< G |
        [1,0,0,0,1,1,0,1,0],
        [1,0,0,0,1,w,0,0,1],
        [1,1,0,0,1,0,0,0,1]>
      where w := PrimitiveElement(F);

_, R, _ := CosetAction(H, sub< H | 1 >);
M := PermutationModule(R, GF(2));

B := split_basic_M(M : params := params_thesis);
```

Timings and memory usage for this computation are

Step	**Time**
MeatAxe	2.76 sec
Semisimple Generators	37.72 sec
Radical Generators	68.4 sec
Pruning	1554.05 sec $\approx$ 26 mins
Split Basic	270.45 sec $\approx$ 4.5 mins
Total Time	1933.38 sec $\approx$ 32 mins
Total Memory	346.47 MB

The number of generators for A computed in the various phases is

Type	**Number**
Semisimple Generators	6
Radical Generators	90
Radical Generators After Pruning	12

The splitting field for A is $\mathbb{F}_4$. The computed split basic algebra B associated to A has degree 512 and dimension 275. There are 4 simple B-modules. Due to a problem with Magma we were unable to compute a presentation for B.

5.5 $\mathrm{SL}(2, \mathbb{F}_8)$

Let $G = \mathrm{SL}(2, \mathbb{F}_8)$, $K = \mathbb{F}_2$, and A be the group algebra of G over K. The simple A-modules are as follows

Simple	$\dim_K S_i$	$[K_i : K]$
S_4	12	3
S_3	8	1
S_2	6	3
S_1	1	1

Note that S_3 is a projective A-module. Let $T = (S_1 \oplus S_2 \oplus S_3 \oplus S_4) \otimes S_3$. The K-dimension of T is 216. An analysis similar to that done in the first example shows that T is an A-module containing every projective indecomposable A-module as a summand, and hence by 1.12.5, T is a faithful A-module.

We construct T, and the split basic algebra associated to A via the following Magma code

```
F := GF(8);
G := SL(2, F);
H := Sylow(G, 2);
_, R, _ := CosetAction(G, H);
M := PermutationModule(R, GF(2));
C := isomorphism_classes(CompositionFactors(M));

S_1 := degree(C, 12);
S_2 := degree(C, 8);
S_3 := degree(C, 6);
S_4 := degree(C, 1);

T := TensorProduct(DirectSum([S_1, S_2, S_3, S_4]), S_2);
B := split_basic_M(T : params := params_thesis);
```

Timings and memory usage for this computation are

Step	**Time**
MeatAxe	0.17 sec
Semisimple Generators	1.72 sec
Radical Generators	1.79 sec
Pruning	10.05 sec
Split Basic	1.13 sec
Total Time	14.86 sec
Total Memory	16.26 Mb

The number of generators for A computed in the various phases is

Type	**Number**
Semisimple Generators	8
Radical Generators	72
Radical Generators After Pruning	12

The splitting field for A is $\mathbb{F}_8$. The computed split basic algebra B has degree 94 and dimension 93. There are 8 simple B-modules. We compute a presentation for B via the following Magma code

```
Q := presentation_A(B : params := params_thesis);
```

Timings and memory usage for this computation are

Step	**Time**
Generators	18.36 sec
Semisimple Generators for Ω	0 sec
Radical Generators for Ω	42.95 sec
Cross Generators for Ω	0.9 sec
Gröbner Basis	0.67 sec
Total Time	62.88 sec
Total Memory	37.57 Mb

The number of generators for Ω computed in the various phases is

Type	**Number**
Semisimple Generators for Ω	72
Radical Generators for Ω	628
Cross Generators for Ω	216
Total	916

The dimensions for the various parts of B are as follows

Part	Dimension over L
B'	8
$J(B)$	85
$B \cap J(B)$	0
B	93

The computed presentation for B can be found in Appendix B.

5.6 A Split Extension of Q_8

The following example is taken from [6], in which the authors study non-principal blocks with one simple module.

Let $G = E \rtimes Q_8$, where E is an elementary abelian subgroup of order 9, and Q_8 is the quaternion group of order 8. The order of G is 72. We have a presentation for G with 4 generators, g, h, i and j, and relations:

(i) g^3, h^3, $gh = hg$.

(ii) $ij = j^3i$, $ij = ji^3$.

(iii) $igi^3 = g^{-1}$, $jgj^3 = g$, $ihi^3 = h$, $jhj^3 = h^{-1}$.

Let K be $\mathbb{F}_3$, and A be the group algebra of G over K. There are two blocks for A, the principal block $B_0(A)$, and a second block, C. There is a unique simple C-module S in the non principal block, and it has dimension 2 over K. Hence there is a single projective indecomposable C-module, P, and it has dimension 18 over K. In Magma we construct P as follows

```
F<g,h,i,j> := FreeGroup(4);
G<g,h,i,j> := quo< F |
  g^3, h^3, g*h = h*g,
  i*j = j^3*i, i*j = j*i^3,
  i*g*i^3=g^-1, j*g*j^3 = g, i*h*i^3 = h, j*h*j^3 = h^-1>;
K := GF(3);
_, R, _ := CosetAction(G, sub< G | 1>);
M := PermutationModule(R, K);
S := degree_M(M, 2);
N := PermutationModule(R, Sylow(R, 2), K);
P := TensorProduct(S, N);
```

Note that we have constructed P as an A-module, where $B_0(A)$ annihilates P, but C acts faithfully on P. We construct the split basic algebra B associated to the block C via the following Magma code

```
B := split_basic_M(P : params := params_thesis);
```

Timings and memory usage for this computation are

Step	**Time**
MeatAxe	0 sec
Semisimple Generators	0.01 sec
Radical Generators	0.01 sec
Pruning	0.04 sec
Split Basic	0 sec
Total Time	0.06 sec
Total Memory	17.64 Mb

The number of generators for A computed in the various phases is

Type	**Number**
Semisimple Generators	2
Radical Generators	16
Radical Generators After Pruning	2

The splitting field for C is $L = \mathbb{F}_3$. The computed split basic algebra B has degree 9 and dimension 9. There is one simple B-module. We compute a presentation for B via the following Magma code

```
Q := presentation_A(B : params := params_thesis);
```

Timings and memory usage for this computation are

Step	**Time**
Generators	0.01 sec
Semisimple Generators for Ω	0 sec
Radical Generators for Ω	0.04 sec
Cross Generators for Ω	0 sec
Gröbner Basis	0 sec
Total Time	0.05 sec
Total Memory	600 Kb

The number of generators for Ω computed in the various phases is

Type	**Number**
Semisimple Generators for Ω	2
Radical Generators for Ω	25
Cross Generators for Ω	8
Total	35

The dimensions for the various parts of B are as follows

Part	Dimension over L
B'	1
$J(B)$	8
$B \cap J(B)$	0
B	9

Theorem 5.6.1. *Let C be as above. Then the split basic algebra associated to A has a presentation as the quotient of the free algebra $\mathbb{F}_3\langle B_1, Z_1, Z_2\rangle$ by the ideal generated by the following elements:*

$$Z_2{}^2 Z_1 Z_2, Z_1{}^3 + Z_2 Z_1 Z_2 + Z_1 Z_2 + Z_2 Z_1 + 2Z_2{}^2,$$

$$Z_1{}^2 Z_2 + 2Z_2 Z_1 Z_2 + 2Z_2{}^2 Z_1 + 2Z_1 Z_2 + 2Z_2 Z_1 + Z_2{}^2,$$

$$Z_1 Z_2 Z_1 + 2Z_2 Z_1 Z_2 + Z_2{}^2 Z_1 + Z_1 Z_2 + Z_2 Z_1 + 2Z_2{}^2,$$

$$Z_1 Z_2{}^2 + 2Z_2{}^2 Z_1, Z_2 Z_1{}^2 + 2Z_2 Z_1 Z_2 + 2Z_2{}^2 Z_1 + 2Z_1 Z_2 + 2Z_2 Z_1 + Z_2{}^2,$$

$$Z_2{}^3, B_1{}^2 + B_1, B_1 Z_1 + Z_1, B_1 Z_2 + Z_2, Z_1 B_1 + Z_1, Z_2 B_1 + Z_2$$

A different presentation for B is given in [6], namely as the quotient of the free algebra $\mathbb{F}_3\langle X, Y\rangle$ by the ideal generated by X^3, Y^3, and $XY + YX$.

5.7 A Very Large Example

Let G be M_{11}, K be $\mathbb{F}_2$, A be the group algebra of G over K, and M be a 7920 dimensional vector space over K. The regular representation $\rho : A \to \mathrm{End}_K(M)$ is faithful by 1.12.6, and has degree 7920. We try to compute the split basic algebra B associated to A using the regular representation.

Mathematically this example is uninteresting, as we have already computed B in our first example. The point of the example is to show that the programs developed in this dissertation can handle representations of quite large degree. Given that algorithmic improvements, careful low level coding of algorithms, and detailed profiling of implementations, often yields performance gains of an order of magnitude, and memory savings of a factor of 4, this example suggests that with careful implementation the programs could reasonably cope with representations of degree 10000. This example also suggests a fundamental fact, namely that as representations become large, the sheer cost of multiplying two matrices becomes overwhelming.

The Magma code for this example is

```
load "top.m";
SetAssertions(false);
load m11;   //  G = M_11
F := GF(2);
_, R, _ := CosetAction(G, sub< G | 1 >);
M := PermutationModule(R, F);
B := split_basic_M(M : params := params_thesis);
```

Timings and memory usage for this computation are

Step	**Time**
MeatAxe	808.11 sec ≈ 13.5 mins
Semisimple Generators	247614.54 sec ≈ 2.87 days
Radical Generators	1695030.18 sec ≈ 19.62 days
Pruning	319267.24 sec ≈ 3.7 days **ran out of memory** **computation stopped**
Split Basic	
Total Time	2262720.07 sec ≈ 26.12 days
Total Memory	4.24 Gb

Remark. The Pruning step had pruned 11 of the 16 S_{ij} when the 4 Gigabyte virtual memory limit was reached, causing Magma to stop the computation. Despite the fact that we failed to compute the split basic algebra, these performance figures are quite encouraging. The 4 Gigabyte limit is an artificial one imposed by the use of 32 bit pointers, and the Pruning step and Split Basic step represent a very small part of the overall computation cost. Hence with only a little more time and memory the computation would have completed.

Bibliography

[1] J.L. Alperin. *Local Representation Theory*. Cambridge University Press, 1986.

[2] J.L. Alperin and R.B. Bell. *Groups and Representations*. Springer-Verlag, 1995.

[3] D.J. Benson. Cohomology of modules in the principal block of a finite group. *New York Journal of Mathematics*, 1:196–205, 1995.

[4] D.J. Benson. *Representations and Cohomology, I: Basic Representation Theory of Finite Groups and Associative Algebras*. Cambridge University Press, 2nd edition, 1997.

[5] D.J. Benson, J.F. Carlson, and G.R. Robinson. On the vanishing of group cohomology. *Journal of Algebra*, 131:40–73, 1990.

[6] D.J. Benson and E.L. Green. Non-principal blocks with one simple module. *Quarterly J. Math. (Oxford)*, 55:1–11, 2004.

[7] W. Bosma and J. Cannon. *Handbook of Magma Functions*. Sydney University, 1996.

[8] W. Bosma, J. Cannon, and C. Playoust. The Magma Algebra System I: The User Language. *Journal of Symbolic Computation*, 3/4(24):235–265, 1997.

[9] J.F. Carlson. *Modules and Group Algebras*. Birkhäuser, 1996.

[10] C.W. Curtis and I. Reiner. *Representation Theory of Finite Groups and Associative Algebras*. Wiley, 1962.

[11] C.W. Curtis and I. Reiner. *Methods of Representation Theory With Applications to Finite Groups and Orders*, volume 1. Wiley, 1981.

[12] L. Dornhoff. *Group Representation Theory, Part B Modular Representation Theory*. Marcel Dekker, 1972.

[13] C. Jansen, K. Lux, R. Parker, and R. Wilson. *An Atlas of Brauer Characters*. Oxford Science Publications, 1995.

[14] T.Y. Lam. *A First Course in Non-Commutative Rings*. Springer-Verlag, 1991.

[15] S. Lang. *Algebra*. Addison-Wesley, 1993.

[16] K. Lux. *Algorithmic Methods in Modular Representation Theory*. Habilitation thesis, Der Mathematisch-Naturwissenschaftlich Fakultät der Rheinisch-Westfälischen Technischen Hochschule Aachen, 1997.

[17] T. Mora. An introduction to commutative and noncommutative gröbner bases. *Theoretical Computer Science*, 134:134–173, 1994.

[18] M. Schneider and J. Gersting. *An Invitation to Computer Science*. West Publishing Company, 1995.

[19] M. Szöke. *Examining Green Correspondents of Weight Modules*. PhD thesis, Wissenschaftsverlag Mainz in Achen, 1998.

Appendix A

Magma Programs

Abstract. We present the Magma programs developed during the writing of this dissertation. Variable names in these programs follow as close as possible the notations and conventions of chapters 2 and 3.

A.1 Computing The Big Idempotents

```
function projection(Lambda, i, s_phi, randword)

  repeat
    w, m := random_element1(Lambda, s_phi[i], randword);
    F := [MinimalPolynomial(s_phi[j](w)) : j in [i+1..#s_phi]];
    a_i_bar := prod_pols(F, m);
  until a_i_bar ne 0;

  return w, F, a_i_bar;

end function;

function big(Lambda, phi, i, s_phi, e, randword)

  A_i := Codomain(s_phi[i]);
  rank := Degree(Codomain(s_phi[i]));
  sum_a_i_bar := A_i!0;
  wlist := [];
  plist := [];

  while true do

    w, F, a_i_bar := projection(Lambda, i, s_phi, randword);

    if Rank(a_i_bar) eq rank then
      g := inverting_polynomial(a_i_bar);
```

```
      m := prod_pols(F, phi(w));
      a := e * (m * Evaluate(g, m)) * e;
      f := lift_p(a);
      return f;
    else
      Append(~wlist, w);
      Append(~plist, F);
      sum_a_i_bar +:= a_i_bar;
      if Rank(sum_a_i_bar) eq rank then
        g := inverting_polynomial(sum_a_i_bar);
        sum_m := &+[prod_pols(plist[j], phi(wlist[j])) :
                  j in [1 .. #wlist]];
        a := e * sum_m * Evaluate(g, sum_m) * e;
        f := lift_p(a);
        return f;
      end if;
    end if;
  end while;

end function;
```

A.2 Computing The Little Idempotents

```
function compute_q(q, j)

  q_to_the_k := q;
  while q_to_the_k lt j do
    q_to_the_k *:= q;
  end while;
  return q_to_the_k;

end function;

function almost_little(Lambda, phi_i, n_i, d_i, e, randword)

  P<x> := PolynomialRing(CoefficientRing(Domain(phi_i)));

  if n_i eq 1 then
    w, a := random_element(Lambda, phi_i, e, e, randword);
    return w, x, a;
  else
```

```
    while true do

      w, m := random_element(Lambda, phi_i, e, e, randword);
      g := MinimalPolynomial(m);

      for fac in Factorization(g) do
        f := fac[1];
        if Degree(f) eq d_i then
          F := g div f;
          a := e * Evaluate(F, m) * e;
          if Rank(a) eq d_i and not
            IsDivisibleBy(P!MinimalPolynomial(a), x^2) then
              return w, F, a;
          end if;
        end if;
      end for;
    end while;

  end if;

end function;

function get_beta_bar(Lambda, phi, phi_i, n_i, d_i, q_i, randword)

  id := Codomain(phi_i)!1;

  repeat
    w, F, beta_bar :=
      almost_little(Lambda, phi_i, n_i, d_i, id, randword);
  until #Seqset(pows_of(beta_bar, q_i-1)) eq q_i-1;

  return w, F, beta_bar;

end function;

function first_little(Lambda, phi, phi_i, f_i, n_i,
                          d_i, q_i, randword)

  w, F, beta_bar :=
    get_beta_bar(Lambda, phi, phi_i, n_i, d_i, q_i, randword);
  g_bar := beta_bar^(q_i-1);

  gamma := f_i * Evaluate(F, phi(w)) * f_i;
```

```
  g := gamma^(q_i-1);
  while not is_non_zero_idempotent(g) do
    g ^:= q_i;
    gamma ^:= q_i;
  end while;

  f := MinimalPolynomial(beta_bar);
  h_prime := n_i eq 1 select Parent(f).1 * f else f;

  k := radical_power(Evaluate(h_prime, gamma));
  j := compute_q(q_i, k);
  beta := gamma^j;

  return g, g_bar, beta, beta_bar, h_prime;

end function;

function next_little(Lambda, phi, phi_i, n_i, d_i,
                        e_i, e_i_bar, randword)

  w, F, g_bar :=
    almost_little(Lambda, phi_i, n_i, d_i, e_i_bar, randword);
  g_prime := e_i * Evaluate(F, e_i * phi(w) * e_i) * e_i;
  g_bar, j := power_to_idempotent(g_bar);
  g := power_to_idempotent(g_prime^j);

  return g, g_bar;

end function;
```

A.3 Computing The SemiSimple Generators

```
function make_result(params, A, L, n_i, d_i, q_i, tau, tau_bar,
    beta, beta_bar, g, g_bar, f_i, id, h_prime, s_g, s_g_bar)

  res := rec< PerModInfo |
    L := L,
    tau := A!tau, tau_bar := tau_bar,
    beta := A!beta, beta_bar := beta_bar,
    g := A!g, g_bar := g_bar,
    h_prime := h_prime>;
```

```
  if params`verify_SS_generators then
    verify_SS_generators(f_i, id, n_i, q_i, s_g, s_g_bar, res);
  end if;

  return res;

end function;

function SS_generators_i(Lambda, phi, phi_i, f_i, n_i,
                         d_i, q_i, params)

  A := Codomain(phi);
  A_i := Codomain(phi_i);
  p := Characteristic(CoefficientRing(A));
  randword := params`random_elt;

  // compute g_1, g_1_bar, beta, beta_bar, and h_prime

  if params_printing(params) then
    print "Little 1";
  end if;

  g_1, g_1_bar, beta, beta_bar, h_prime :=
    first_little(Lambda, phi, phi_i, f_i, n_i, d_i, q_i, randword);

  L := cyclic_K_star(beta, beta_bar, d_i, q_i);

  //  initialise for the loop ahead

  s_g := [g_1];
  s_g_bar := [g_1_bar];

  // if n_i = 1 we can bail out now

  if n_i eq 1 then
    return make_result(params, A, L, n_i, d_i, q_i,
      g_1, g_1_bar, beta, beta_bar, g_1, g_1_bar, f_i,
      A_i!1, h_prime, s_g, s_g_bar);
  end if;

  e := f_i - g_1;
  e_bar := A_i!1 - g_1_bar;
```

```
prod := f_i;
prod_bar := A_i!1;
tau := A!0;
tau_bar := A_i!0;

pre := g_1;
pre_bar := g_1_bar;

//  compute g_2 through g_n, and g_2_bar through g_n_bar
//  compute all the z_j and z_j bar except z_n and z_n_bar

for j := 2 to n_i do
  if params_printing(params) then
    print "Little", j;
  end if;

  //  compute g_j and g_j_bar

  if j eq n_i then
    g_j := e;
    g_j_bar := e_bar;
  else
    g_j, g_j_bar := next_little(Lambda, phi, phi_i, n_i, d_i,
      e, e_bar, randword);
  end if;

  Append(~s_g, g_j);
  Append(~s_g_bar, g_j_bar);

  e -:= g_j;
  e_bar -:= g_j_bar;

  //  compute z_j_bar in (g_(j-1)_bar A_i g_j_bar) and lift to
  //  z_j in g_(j-1) A g_j

  w, z_j_bar :=
    random_element(Lambda, phi_i, pre_bar, g_j_bar, randword);
  z_j := pre * phi(w) * g_j;

  tau +:= z_j;
  prod *:= z_j;
  tau_bar +:= z_j_bar;
  prod_bar *:= z_j_bar;
```

```
    //  roll pre and pre_bar over to g_j and g_j_bar

    pre := g_j;
    pre_bar := g_j_bar;

  end for;

  // construct z_n_bar as a 1 in the (n,1)-th spot

  g_n_bar := pre_bar;
  w, z_n_bar :=
    random_element(Lambda, phi_i, g_n_bar, g_1_bar, randword);
  invpow := L`inverse_bar(prod_bar * z_n_bar);
  z_n_bar := z_n_bar * L`beta_bars[invpow];

  //  lift z_n_bar to z_n

  g_n := pre;
  z_n_prime := g_n * phi(w) * g_1;
  z_n_prime := z_n_prime * L`betas[invpow];
  prod_z_n_prime := prod * z_n_prime;
  l := radical_power(prod_z_n_prime - g_1);
  z_n := z_n_prime * (prod_z_n_prime)^(p^l-1);

  //  update tau and tau_bar

  tau +:= z_n;
  tau_bar +:= z_n_bar;

  return make_result(params, A, L, n_i, d_i, q_i, tau, tau_bar,
              beta, beta_bar, g_1, g_1_bar, f_i,
              A_i!1, h_prime, s_g, s_g_bar);

end function;

function SS_generators(phi, s_phi, n, d, q, params)

  X := Domain(phi);
  A := Codomain(phi);
  r := #s_phi;

  //print "Creating Identity";
```

```
  e := A!1;
  //print "Done Creating Identity";
  s_f_i := [];
  ss_I := [];
  Lambda := [X.j : j in [1..Rank(X)]];

  for i := 1 to r do

    if params_printing(params) then
      print "Big", i, "n_i, d_i, q_i =", n[i], d[i], q[i];
    end if;

    if i eq r then
      f_i := e;
    else
      f_i := big(Lambda, phi, i, s_phi, e, params`random_elt);
    end if;
    Append(~s_f_i, f_i);
    e -:= f_i;

    Append(~ss_I, SS_generators_i(Lambda, phi,
      s_phi[i], f_i, n[i], d[i], q[i], params));

  end for;

  assert is_mut_orth(s_f_i);
  assert &+(s_f_i) eq A!1;

  return ss_I;

end function;
```

A.4 Computing Generators for the Radical

```
function psi(A, ss_I, n_i, taus, g_tau_bars, tau_g_bars, lambda_t)

  if IsZero(lambda_t) then
    return A!0;
  end if;

  betas := ss_I`L`betas;
  pow_bar := ss_I`L`pow_bar;
```

```
  res := A!0;
  for j := 1 to n_i do
    jm1 := j-1;
    left := g_tau_bars[jm1 eq 0 select n_i else jm1] * lambda_t;
    if IsZero(left) then continue; end if;
    for k := 1 to n_i do
      nkm1 := n_i-(k-1);
      m := left * tau_g_bars[nkm1 eq 0 select n_i else nkm1];
      if m ne 0 then
        l := pow_bar(m);
        njm1 := n_i-(j-1);
        km1 := k-1;
        res +:= taus[njm1 eq 0 select n_i else njm1] * betas[l] *
                taus[km1 eq 0 select n_i else km1];
      end if;
    end for;
  end for;

  return res;

end function;

function radical_generators_i(A, ss_I, n_i, taus,
                              g_taus, tau_gs, s_phi)

  X := Domain(s_phi);
  g := ss_I`g;
  g_bar := ss_I`g_bar;

  print "radical_generators_i =", n_i^2 * Ngens(A);

  //  build the powers of tau for each i

  tb_i := pows_of(ss_I`tau_bar, n_i);
  g_tau_bars := [g_bar * t : t in tb_i];
  tau_g_bars := [t * g_bar : t in tb_i];

  res := {};
  for t in [1..#Generators(A)] do
    lambda_rad := A.t -
      psi(A, ss_I, n_i, taus, g_tau_bars, tau_g_bars, s_phi(X.t));
    if IsZero(lambda_rad) then continue; end if;
```

```
    for x := 1 to n_i do
      left := g_taus[x] * lambda_rad;
      if IsZero(left) then continue; end if;
      for y := 1 to n_i do
        m := left * tau_gs[y];
        if not IsZero(m) then
          Include(~res, A!m);
        end if;
      end for;
    end for;
  end for;

  return res;

end function;

function radical_generators_ij(A, n_i, n_j, taus_i, taus_j,
                               g_taus_i, tau_gs_j)

  print "radical_generators_ij =", n_i*n_j * Ngens(A);

  res := {};
  for lambda in Generators(A) do
    for x := 1 to n_i do
      left := g_taus_i[x] * lambda;
      if IsZero(left) then continue; end if;
      for y := 1 to n_j do
        m := left * tau_gs_j[y];
        if not IsZero(m) then
          Include(~res, A!m);
        end if;
      end for;
    end for;
  end for;

  return res;

end function;

function radical_generators(A, s_phi, ss_I, beta,
                            tau, g, n, q, params)

  r := #n;
```

```
  S := Seqlist([Seqlist([{} : i in [1..r]]) : j in [1..r]]);

  //  build the powers of tau for each i

  taus := [pows_of(tau[i], n[i]) : i in [1..r]];

  //  build g * taus and taus * g for each i

  g_taus := [* *];
  tau_gs := [* *];
  for i := 1 to r do
    Append(~g_taus, [g[i] * t : t in taus[i]]);
    Append(~tau_gs, [t * g[i] : t in taus[i]]);
  end for;

  for i := 1 to r do
    for j := 1 to r do
      if params_printing(params) then
        print "J(e_i A e_j) with (i,j) =", i, j;
      end if;
      if i eq j then
        S[i][j] :=
          radical_generators_i(A, ss_I[i], n[i], taus[i],
                          g_taus[i], tau_gs[i], s_phi[i]);
      else
        S[i][j] :=
            radical_generators_ij(A, n[i], n[j], taus[i],
                            taus[j], g_taus[i], tau_gs[j]);
      end if;
    end for;
  end for;

  if params`verify_radical_generators then
    print "Verifying radical generators";
    verify_radical_generators(beta, tau, g, q, S, S);
  end if;

  return S;

end function;
```

A.5 Pruning The Generators for the Radical

```
function prune(S, A, beta, tau, g, q, params)

  r := #S;
  T := Seqlist([Seqlist([[] : i in [1..r]]) : j in [1..r]]);
  V := KMatrixSpace(CoefficientRing(A), Degree(A), Degree(A));

  for i := 1 to r do
    for j := 1 to r do
      print "<i,j> =", i, j;
      U := sub< V | S[i][j] >;
      L := {x*y : x in S[i][k], y in S[k][j],
                  k in [1..r] | x*y ne 0};
      W := sub< V | L >;
      T[i][j] := Basis(Complement(U, U meet W));
    end for;
  end for;

  if params`verify_radical_generators then
    print "Verifying radical generators after pruning";
    verify_radical_generators(beta, tau, g, q, T, S);
  end if;

  for i := 1 to r do
    for j := 1 to r do
      T[i][j] := [A | x : x in T[i][j]];
    end for;
  end for;

  return T;

end function;
```

A.6 Computing Generators for an Algebra

```
function generators(M : params := params_defaults)

  A, s_A, n, d, q := meataxe(M);
  r := #s_A;

  SetSeed(0);
  t := Cputime();
```

```
  X := FPA(CoefficientRing(A), Ngens(A));
  phi := rep(X, A);
  s_phi := [rep(X, s_A[i]) : i in [1..r]];
  ss_I := SS_generators(phi, s_phi, n, d, q, params);
  show_time("Generators", Cputime(t), params);

  if params`print_level eq params_THESIS then
    print "Number of generators =", r * 2;
  end if;

  beta := [x`beta : x in ss_I];
  tau := [x`tau : x in ss_I];
  g := [x`g : x in ss_I];
  h_prime := [x`h_prime : x in ss_I];

  t := Cputime();
  J_gens := radical_generators(A, s_phi, ss_I, beta,
                    tau, g, n, q, params);
  show_time("Radical", Cputime(t), params);

  if params`print_level eq params_THESIS then
    print "Number of generators for J(A) =",
      &+[#J_gens[i][j] : i, j in [1..#J_gens]];
  end if;

  t := Cputime();
  J_gens := prune(J_gens, A, beta, tau, g, q, params);
  show_time("Prune", Cputime(t), params);

  if params`print_level eq params_THESIS then
    print "Number of generators for J(A)/J(A)^2 =",
      &+[#J_gens[i][j] : i, j in [1..#J_gens]];
  end if;

  return beta, tau, g, J_gens, h_prime, n, d, q, ss_I;

end function;

function generators_M(M : params := params_defaults)
  return generators(M : params := params);
end function;

function generators_A(A : params := params_defaults)
```

```
  return generators(RModule(A) : params := params);
end function;
```

A.7 Computing The Basic and The Split Basic Algebra

```
function basic(beta, g, J)

  K := CoefficientRing(beta[1]);
  f, dim_c := condense(beta, g, J);
  beta_c := [f(x): x in beta];
  J_gens_c := flatten(J, f);
  B := MatrixAlgebra<K, dim_c | [x : x in beta_c cat J_gens_c]>;
  return B, [B!x : x in beta_c], map(J, func<x | B!f(x)>);

end function;

function split_basic(B, d)

  K := CoefficientRing(B);
  p := Characteristic(K);
  c := Degree(K) * LCM([d[i] : i in [1..#d]]);
  X := MatrixAlgebra(GF(p^c), Degree(B));
  A := MatrixAlgebra<GF(p^c), Degree(B) |
        [X!(B.i) : i in [1..#Generators(B)]]>;
  return A, c;

end function;

function basic_M(M : params := params_defaults)

  beta, _, g, J_gens, _, _, d, _ := generators(M : params := params);
  t := Cputime();
  B, beta_c, J_gens_c := basic(beta, g, J_gens);
  show_time("Basic", Cputime(t), params);

  return B, beta_c, J_gens_c, d;

end function;

function basic_A(A : params := params_defaults)
  return basic_M(RModule(A) : params := params);
end function;
```

```
function split_basic_M(M : params := params_defaults)

  B, _, _, d := basic_M(M : params := params);
  t := Cputime();
  S, c := split_basic(B, d);
  show_time("Split Basic", Cputime(t), params);

  return S;

end function;

function split_basic_A(A : params := params_defaults)
  return split_basic_M(RModule(A) : params := params);
end function;
```

A.8 Computing Generators for Ω

```
graph := func< f | [<x, f(x)> : x in Domain(f) ]>;

rel_seq := func< rels | [LHS(x) - RHS(x) : x in rels] >;

procedure R(params, ~rels, new)

  for r in new do
    if LHS(r) ne RHS(r) then
      if params`print_level ge params_VERBOSE then
        print "Adding relation", r;
      end if;
      Append(~rels, r);
    end if;
  end for;

end procedure;

procedure semisimple_relations(
          params, isB, r, h_prime, n, q, B, T, ~rels)

  //  relations for each matrix component in the semisimple part

  if not isB then
```

```
    R(params, ~rels, [T[i]^n[i] * B[i] = B[i] : i in [1..r]]);
    R(params, ~rels, [B[i] * T[i]^n[i] = B[i] : i in [1..r]]);
    R(params, ~rels, [T[i]^n[i] * T[i] = T[i] : i in [1..r]]);
    R(params, ~rels, [T[i] * T[i]^n[i] = T[i] : i in [1..r]]);
    R(params, ~rels,
        [B[i] * T[i]^k * B[i] = 0 : k in [1..n[i]-1], i in [1..r]]);
  end if;

  if isB then
    R(params, ~rels, [ B[i]^(q[i]-1) * B[i] = B[i] : i in [1..r]]);
  else
    R(params, ~rels,
      [
        &+[T[i]^(n[i]-j) * B[i]^(q[i]-1) * T[i]^j : j in [1..n[i]]]
              = T[i]^n[i] : i in [1..r]
      ]);
  end if;

  R(params, ~rels, [Evaluate(h_prime[i], B[i]) = 0 : i in [1..r]]);

  //  orthogonality relations in the semisimple part

  R(params, ~rels, [B[i] * B[j] = 0 : i, j in [1..r] | i ne j]);

  if not isB then
    R(params, ~rels, [T[i] * T[j] = 0 : i, j in [1..r] | i ne j]);
    R(params, ~rels, [B[i] * T[j] = 0 : i, j in [1..r] | i ne j]);
    R(params, ~rels, [T[i] * B[j] = 0 : i, j in [1..r] | i ne j]);
  end if;

end procedure;

procedure cross_relations(params, isB, r, G, n, q, B, T,
      J_gen, CV, CF, backmap, basis, ~rels)

  //  identity relations between the semisimple part and J(A)

  for i := 1 to r do
    for j := 1 to r do

      R(params, ~rels,
        [B[i]^(q[i]-1) * G(y) = G(y)  : y in J_gen[i][j]]);
      R(params, ~rels,
```

```
      [G(y) * B[j]^(q[j]-1) = G(y)  : y in J_gen[i][j]]);

    if not isB then
      R(params, ~rels,
        [T[i]^n[i] * G(y) = G(y)  : y in J_gen[i][j]]);
      R(params, ~rels,
        [G(y) * T[j]^n[j] = G(y)  : y in J_gen[i][j]]);
    end if;

  end for;
end for;

//  orthogonality relations between the semisimple part and J(A)

for i := 1 to r do
  for j := 1 to r do

    R(params, ~rels,
      [B[k]*G(y) = 0 : k in [1..r], y in J_gen[i][j] | k ne i]);
    R(params, ~rels,
      [G(y) * B[k] = 0 : k in [1..r], y in J_gen[i][j] | k ne j]);

    if not isB then
      R(params, ~rels,
        [T[k]*G(y) = 0 : k in [1..r], y in J_gen[i][j] | k ne i]);
      R(params, ~rels,
        [G(y)*T[k] = 0 : k in [1..r], y in J_gen[i][j] | k ne j]);
    end if;

  end for;
end for;

//  scaling relations from semisimple part on J(A)

P := Domain(CF);
J := KMatrixSpaceWithBasis(basis);

for i := 1 to r do
  for j := 1 to r do
    for x in J_gen[i][j] do

      lhs := P.i * G(x);
      C := Coordinates(J, CV!CF(lhs));
```

```
        w := &+[C[i] * backmap(basis[i]) : i in [1..#C]];
        R(params, ~rels, [lhs = w]);

        lhs := G(x) * P.j;
        C := Coordinates(J, CV!CF(lhs));
        w := &+[C[i] * backmap(basis[i]) : i in [1..#C]];
        R(params, ~rels, [lhs = w]);

      end for;
    end for;
  end for;

end procedure;

procedure radical_relations(params, A, offset, V, FF, gens_J,
                    ~rels, ~backmap, ~basis)

  P := Domain(FF);
  K := CoefficientRing(V);
  dimV := Dimension(V);

  rls :=[];
  Q := FPA(K, #gens_J);
  F := hom< Q -> A | gens_J>;
  L := Setseq(MonomialsOfDegree(Q, 1));
  I := ideal< Q | >;
  d := 2;
  dim_N := 0;
  P_embed := hom< Q -> P | [P.(i+offset) : i in [1..#gens_J]]>;

  repeat
    if params_printing(params) then
      print "Degree =", d;
      print "#L = ", #L;
    end if;

    new_L := L cat [x : x in MonomialsOfDegree(Q, d) | not x in I];
    if params_printing(params) then
      print "#new_L =", #new_L;
    end if;

    B := [V!F(w) : w in new_L];
    m := Matrix(K, #new_L, dimV,
```

```
            [Coordinates(V, B[i]) : i in [1..#new_L]]);
    D := Domain(m);

    N := NullSpace(m);
    new_dim_N := Dimension(N);
    for x in Basis(N) do
      C := Coordinates(D, D!x);
      lhs := &+[C[i] * new_L[i] : i in [1..#C]];
      Append(~rls, lhs);
      R(params, ~rels, [P_embed(lhs) = 0]);
    end for;

    U := [new_L[i] : i in [1..Dimension(D)] | Basis(D)[i] in N];

    done := new_dim_N eq dim_N + #new_L - #L;

    L := new_L;
    I := ideal< Q | I, U >;
print I;
    dim_N := new_dim_N;
    d +:= 1;

  until done;

  S := Q/ideal< Q | rls>;
  S_embed := hom< S -> P | [P.(i+offset) : i in [1..#gens_J]]>;
  mb := MonomialBasis(S) diff {1};
  emb := [S_embed(w) : w in mb];
  basis := [ V!FF(w) : w in emb];
  backmap := map< basis -> P | [<V!FF(w), w> : w in emb]>;

end procedure;

function relations(params, beta, tau, g, J_gen, h_prime, n, d, q)

  r := #n;
  A := Parent(beta[1]);
  K := CoefficientRing(A);
  V := KMatrixSpace(K, Degree(A), Degree(A));
  gens_J := flatten(J_gen, func<x | V!x>);
  isB := &and[n[i] eq 1 : i in [1..r]];

  if isB then
```

```
  P := FPA(K, r + #gens_J);
  AssignNames(~P,
          ["b_" cat IntegerToString(i) : i in [1..r]] cat
          ["z_" cat IntegerToString(i) : i in [1..#gens_J]]);
  B := [P.i : i in [1..r]];
  T := [];
  J := [P.i : i in [r+1..r+#gens_J]];
else
  P := FPA(K, 2*r + #gens_J);
  AssignNames(~P,
          ["b_" cat IntegerToString(i) : i in [1..r]] cat
          ["t_" cat IntegerToString(i) : i in [1..r]] cat
          ["z_" cat IntegerToString(i) : i in [1..#gens_J]]);
  B := [P.i : i in [1..r]];
  T := [P.i : i in [r+1..2*r]];
  J := [P.i : i in [2*r+1..2*r+#gens_J]];
end if;

F := hom< P -> A | [beta[i] : i in [1..#B]] cat
                       [tau[i] : i in [1..#T]] cat gens_J>;
G := map< Seqset(gens_J) -> P |
            [<gens_J[i], P.(#B+#T+i)> : i in [1..#gens_J]]>;

rels := [];

t := Cputime();
semisimple_relations(params, isB, r, h_prime, n, q, B, T, ~rels);
show_time("SemiSimple", Cputime(t), params);

if params_printing(params) then
  print "Number of semisimple relations", #rels;
end if;

l := #rels;
t := Cputime();
CA, cB, x := basic(beta, g, J_gen);
cJg := flatten(x, func<x|x>);
CV := KMatrixSpace(K, Degree(CA), Degree(CA));
CF := hom< P -> CA |
    [CA!x : x in cB] cat [0:x in T] cat [CA!x : x in cJg]>;
radical_relations(params, CA, #cB+#T, CV, CF, cJg,
      ~rels, ~backmap, ~basis);
show_time("K-basis", Cputime(t), params);
```

```
  if params_printing(params) then
    print "J(A) has dimension", #basis,
                Dimension(ideal< CA | basis>);
    print "Number of K-Basis relations", #rels - l;
  end if;

  if isB then
    A_prime := sub< CA | cB >;
    I := ideal< CA | cJg >;
    if params_printing(params) then
      print "Dim A", Dimension(CA);
      print "Dim A'", Dimension(A_prime);
      print "Dim J(A)", Dimension(I);
      print "Dim A' meet J(A)", Dimension(A_prime meet I);
    end if;
    error if Dimension(A_prime) + Dimension(I)
                ne Dimension(CA), "sum dim";
    error if Dimension(A_prime meet I) ne 0, "meet";
  end if;

  l := #rels;
  t := Cputime();
  cross_relations(params, isB, r, G, n, q, B, T, J_gen,
    CV, CF, backmap, basis, ~rels);
  show_time("Cross", Cputime(t), params);

  if params_printing(params) then
    print "Number of cross relations", #rels-l;
    print "Total number of relations", #rels;
  end if;

  if params`verify_relations then
    verify_relations(r, F, G, J_gen, rels);
  end if;

  return P, rel_seq(rels);

end function;
```

A.9 Computing A Presentation

```
function presentation_M(M : params := params_defaults)

  t := Cputime();
  beta, tau, g, J_gens, h_prime, n, d, q, _ :=
    generators(M : params := params);
  show_time("Generators", Cputime(t), params);
  t := Cputime();
  P, rels :=
    relations(params, beta, tau, g, J_gens, h_prime, n, d, q);
  show_time("Relations", Cputime(t), params);

  t := Cputime();
  I := ideal< P | rels >;
  GB := GroebnerBasis(I);
  show_time("GB", Cputime(t), params);

  if params`print_level eq params_THESIS then
    print "GB =";
    print GB;
    print "Number of GB relations =", #GB;
  end if;

  return quo< P | I >;

end function;

function presentation_A(A : params := params_defaults)
  return presentation_M(RModule(A) : params := params);
end function;
```

A.10 Condensing

```
function condense(beta, g, j_I)

  dim := Nrows(beta[1]);
  K := CoefficientRing(beta[1]);
  e := &+g;

  V := KSpace(K, dim);
  W := sub< V | [e[j] : j in [1..dim]]>;
  dim_c := Dimension(W);
```

```
  c_l := KMatrixSpace(K, dim_c, dim) ! 0;
  for j := 1 to dim_c do
    c_l[j] := Basis(W)[j];
  end for;

  WW := sub< V | [Basis(V)[j] - e[j]: j in [1..dim]]>;
  uu := Matrix(Basis(W) cat Basis(WW));
  uuu := uu^-1;
  c_r := Submatrix(uuu, 1, 1, dim, dim_c);
  assert c_l * e eq c_l;
  assert e * c_r eq c_r;

  return func< m | c_l * m * c_r >, dim_c, c_l, c_r;

end function;
```

A.11 Randomizing Functions

```
function random_element(gens, f, e_1, e_2, randword)

  repeat
    w := randword(gens);
    b := e_1 * f(w) * e_2;
  until b ne 0;

  return w, b;

end function;

function random_element1(gens, f, randword)

  repeat
    w := randword(gens);
    b := f(w);
  until b ne 0;

  return w, b;

end function;

function random_field_element(K)
  if #K eq 2 then
```

```
    return K!1;
  else
    repeat
      k := Random(K);
    until k ne 0;
    return k;
  end if;
end function;

random_MAX_SUM_LEN := 5;
random_MAX_PROD_LEN := 5;

function random_word(S)

  X := Parent(S[1]);
  K := CoefficientRing(X);

  w := X!0;
  repeat
    for i:= 1 to Random(1, random_MAX_SUM_LEN) do
      prod := X ! random_field_element(K);
      for j := 1 to Random(1, random_MAX_PROD_LEN) do
        prod *:= Random(S);
      end for;
      w := w + prod;
    end for;
  until w ne 0;

  return w;

end function;
```

A.12 Functions to Verify Computation Results

```
procedure verify_SS_generators(f, id, n_i, q_i, s_g, s_g_bar, ss_I)

  L := ss_I`L;
  beta := ss_I`beta;
  tau := ss_I`tau;
  g := ss_I`g;
  beta_bar := ss_I`beta_bar;
  tau_bar := ss_I`tau_bar;
```

```
g_bar := ss_I`g_bar;

print "Checking big idempotent";

error if not &and[f*x eq x and x*f eq x : x in s_g], "";
error if not f * beta eq beta, "";
error if not beta * f eq beta, "";
error if not f * tau eq tau, "";
error if not tau * f eq tau, "";

error if not &and[id*x eq x and x*id eq x : x in s_g_bar], "";
error if not id * beta_bar eq beta_bar, "";
error if not beta_bar * id eq beta_bar, "";
error if not id * tau_bar eq tau_bar, "";
error if not tau_bar * id eq tau_bar, "";

print "Checking little idempotents";

error if not s_g[1] eq g, "";
error if not s_g_bar[1] eq g_bar, "";

error if not #s_g eq n_i, "";
error if not is_mut_orth(s_g), "";
error if not &+s_g eq f, "";

error if not #s_g_bar eq #s_g, "";
error if not is_mut_orth(s_g_bar), "";
error if not &+s_g_bar eq id, "";

print "Checking the fields";

error if not #L`betas eq q_i-1, "";
error if not #L`beta_bars eq #L`betas, "";

print "Checking matrix algebra relations";

error if not tau^n_i eq f, "";
error if not tau^n_i * beta eq beta, "";
error if not beta * tau^n_i eq beta, "";
error if not tau^n_i * tau eq tau, "";
error if not tau * tau^n_i eq tau, "";

error if not tau_bar^n_i eq id, "";
```

```
error if not tau_barˆn_i * beta_bar eq beta_bar, "";
error if not beta_bar * tau_barˆn_i eq beta_bar, "";
error if not tau_barˆn_i * tau_bar eq tau_bar, "";
error if not tau_bar * tau_barˆn_i eq tau_bar, "";

x := &+[tauˆ(n_i-j) * L‘betas[q_i-1] * tauˆj : j in [1..n_i]];
error if not x eq f, "";
error if not x * beta eq beta, "";
error if not beta * x eq beta, "";
error if not x * tau eq tau, "";
error if not tau * x eq tau, "";

x_bar := &+[tau_barˆ(n_i-j) *
          L‘beta_bars[q_i-1] * tau_barˆj : j in [1..n_i]];
error if not x_bar eq id, "";
error if not x_bar * beta_bar eq beta_bar, "";
error if not beta_bar * x_bar eq beta_bar, "";
error if not x_bar * tau_bar eq tau_bar, "";
error if not tau_bar * x_bar eq tau_bar, "";

error if not Evaluate(ss_I‘h_prime, beta) eq 0, "";
error if not Evaluate(ss_I‘h_prime, beta_bar) eq 0, "";

error if not &and [beta * tauˆk * beta eq 0 : k in [1..n_i-1]], "";
error if not &and
    [beta_bar * tau_barˆk * beta_bar eq 0 : k in [1..n_i-1]], "";

print "Checking other relations";

error if not g * beta eq beta, "";
error if not beta * g eq beta, "";
error if not g_bar * beta_bar eq beta_bar, "";
error if not beta_bar * g_bar eq beta_bar, "";

error if not betaˆq_i eq beta, "";
error if not betaˆ(q_i-1) eq g, "";
error if not is_non_zero_idempotent(L‘betas[q_i-1]), "";
error if not #{tauˆj : j in [1..n_i]} eq n_i, "";

error if not beta_barˆq_i eq beta_bar, "";
error if not beta_barˆ(q_i-1) eq g_bar, "";
error if not is_non_zero_idempotent(L‘beta_bars[q_i-1]), "";
error if not #{tau_barˆj : j in [1..n_i]} eq n_i, "";
```

```
end procedure;

procedure verify_radical_generators(beta, tau, g, q, S, U)

  r := #q;

  error if not
    &and[IsNilpotent(Universe(U[i][j])!x) : x in S[i][j],
          i,j in [1..r]], "Nilpotent";

  error if not &and[g[i] * x eq x : x in S[i][j],
          i,j in [1..r]], "left g";
  error if not &and[x * g[j] eq x : x in S[i][j],
          i,j in [1..r]], "right g";

  error if not
    &and[beta[i]^(q[i]-1) * x eq x : x in S[i][j], i, j in [1..r]],
    "left beta^q";

  error if not
    &and[x * beta[j]^(q[j]-1) eq x : x in S[i][j], i, j in [1..r]],
    "right beta^q";

  error if not
    &and[beta[k] * x eq 0 : x in S[i][j],
          i, j, k in [1..r] | k ne i], "left beta = 0";

  error if not
    &and[x * beta[k] eq 0 : x in S[i][j],
          i, j, k in [1..r] | k ne j], "right beta = 0";

  error if not
    &and[tau[k] * x eq 0 : x in S[i][j], i, j, k in [1..r] | k ne i],
    "left tau = 0";

  error if not
    &and[x * tau[k] eq 0 : x in S[i][j], i, j, k in [1..r] | k ne j],
    "right tau = 0";

end procedure;

procedure verify_relations(r, F, G, J_gen, rels)
```

```
  print "Checking presentation relations";
  error if not
    &and[F(LHS(x)) eq F(RHS(x)) : x in rels ], "LHS <> RHS";

  //  check that the orthogonality relations in the radical were
  //  picked up in the basis phase

  for i := 1 to r do
    for j := 1 to r do
      for y in J_gen[i][j] do
        error if not &and [(G(y) * G(x) = 0) in rels :
                  x in J_gen[k][l], k, l in [1..r] | k ne j], "";
        error if not &and [(G(x) * G(y) = 0) in rels :
                  x in J_gen[k][l], k, l in [1..r] | l ne i], "";
        error if not &and[y * x eq 0 :
                  x in J_gen[k][l], k, l in [1..r] | k ne j], "";
        error if not &and[x * y eq 0 :
                  x in J_gen[k][l], k, l in [1..r] | l ne i], "";
      end for;
    end for;
  end for;

end procedure;
```

A.13 Top Level Parameters

```
Params := recformat
  <
    print_level : Integers(),
    verify_SS_generators : Booleans(),
    verify_radical_generators : Booleans(),
    verify_relations : Booleans(),
    random_elt
  >;

params_THESIS := -1;
params_SILENT := 0;
params_VERBOSE := 1;
params_SUPER_VERBOSE := 2;

params_defaults := rec< Params |
```

```
  //print_level := params_VERBOSE,
  print_level := params_THESIS,
  verify_SS_generators := true,
  verify_radical_generators := true,
  verify_relations := true,
  random_elt := random_word>;

params_thesis := rec< Params |
  //print_level := params_VERBOSE,
  print_level := params_THESIS,
  verify_SS_generators := false,
  verify_radical_generators := false,
  verify_relations := false,
  random_elt := random_word>;

function params_printing(params)
  return params`print_level ge params_VERBOSE or
    params`print_level eq params_THESIS;
end function;
```

A.14 Record Formats

```
K_star := recformat
  <
    betas, beta_bars,
    pow_bar, inverse_bar
  >;

PerModInfo := recformat
  <
    L,
    tau, tau_bar,
    beta, beta_bar,
    g, g_bar,
    h_prime
  >;
```

A.15 Functions for the Cylic Group of Units of a Field

```
function cyclic_K_star(beta, beta_bar, d, q)
```

```
  beta_bars := pows_of(beta_bar, q-1);
  map_bar := {@ beta_bar : beta_bar in beta_bars @};
  pow_bar := func< m | Index(map_bar, m) >;
  inverse_bar := func< m |
    i eq q-1 select i else (q-1)-i where i := pow_bar(m) >;

  return rec< K_star |
    betas := pows_of(beta, q-1),
    beta_bars := beta_bars,
    pow_bar := pow_bar,
    inverse_bar := inverse_bar>;

end function;
```

A.16 Matrix Functions

```
function isomorphism_classes(mods)

  isos := [mods[1]];
  for i := 2 to #mods do
    for j := 1 to #isos do
      if IsIsomorphic(mods[i], isos[j]) then
        continue i;
      end if;
    end for;
    Append(~isos, mods[i]);
  end for;

  return isos;

end function;

rep := func< X, A | hom< X -> A | [A.i : i in [1 .. Ngens(A)]] >>;

function inverting_polynomial(m)

  P<x> := PolynomialRing(CoefficientRing(m));
  h := P ! MinimalPolynomial(m);
  c := Coefficient(h, 0);
  return (c-h) div (c*x);

end function;
```

```
prod_pols := func< pols, m | Evaluate(&*pols, m) >;

function power_to_idempotent(m)

  j := 1;
  res := m;
  while not IsIdempotent(res) do
    res *:= m;
    j +:= 1;
  end while;

  return res, j;

end function;

function lift_p(m)

  p := Characteristic(CoefficientRing(m));
  j := 1;
  while not IsIdempotent(m) do
    m ^:= p;
    j +:= 1;
  end while;

  return m, j;

end function;

function radical_power(m)

  isNP, c := IsNilpotent(m);
  assert isNP;
  return c;

end function;

function pows_of(m, n)

  res := [m];
  temp := m;
  for i := 2 to n do
    temp *:= m;
    Append(~res, temp);
```

```
  end for;
  return res;

end function;

all_degree := func< C, n | [ x : x in C | Dimension(x) eq n]>;

degree := func< C, n | all_degree(C, n)[1] >;

function all_degree_M(M, n)
  CF := CompositionFactors(M);
  C := isomorphism_classes(CF);
  return all_degree(C, n);
end function;

function degree_M(M, n)
  CF := CompositionFactors(M);
  C := isomorphism_classes(CF);
  return degree(C, n);
end function;

flatten := func<J, f | [f(x) : x in J[i][j], i, j in [1 ..#J]]>;

function map(J, f)

  r := #J;
  T := Seqlist([Seqlist([[] : i in [1..r]]) : j in [1..r]]);
  for i := 1 to r do
    for j := 1 to r do
      T[i][j] := [f(x) : x in J[i][j]];
    end for;
  end for;

  return T;
end function;
```

A.17 Checking Functions

```
is_non_zero_idempotent := func< x | x ne 0 and IsIdempotent(x) >;

is_mut_orth := func< s |
    &and[is_non_zero_idempotent(x) : x in s] and
```

```
&and[s[i]*s[j] eq 0 and s[j]*s[i] eq 0 :
        i, j in [1..#s] | i ne j]>;
```

Appendix B

Presentation for a Split Extension of A_5

Theorem B.0.1. *Let $G = SL(2, \mathbb{F}_8)$, $K = \mathbb{F}_2$, and A be the group algebra of G over K. Then the split basic algebra associated to A has a presentation as the quotient of the free algebra $\mathbb{F}_4\langle B_1, \ldots, B_8, Z_1, \ldots, Z_{12}\rangle$ by the ideal generated by the following elements:*

$$Z_1Z_5Z_{10}Z_{11}Z_9 + B_1Z_1Z_5Z_9 + {B_1}^6Z_2Z_8Z_1, Z_2Z_8Z_1Z_3Z_2,$$
$$Z_2Z_8Z_1Z_4Z_6 + {B_1}^2Z_1Z_5Z_9 + Z_2Z_8Z_1, Z_3Z_1Z_5Z_{10}Z_{11} + B_1Z_4Z_6Z_5 + Z_5Z_{10}Z_{11},$$
$$Z_4Z_7Z_{12}Z_6Z_4, Z_4Z_7Z_{12}Z_6Z_5 + {B_1}^4Z_4Z_6Z_5 + {B_1}^3Z_5Z_{10}Z_{11},$$
$$Z_5Z_9Z_4Z_7Z_{12} + {B_1}^4Z_3Z_1Z_4 + Z_4Z_7Z_{12},$$
$$Z_5Z_{10}Z_{11}Z_9Z_3 + {B_1}^6Z_3Z_2Z_8 + B_1Z_5Z_9Z_3,$$
$$Z_5Z_{10}Z_{11}Z_9Z_5, Z_6Z_5Z_9Z_3Z_1 + Z_6Z_3Z_1 + {B_1}^3Z_7Z_{12}Z_6,$$
$$Z_7Z_{12}Z_6Z_4Z_7, Z_7Z_{12}Z_6Z_5Z_9 + {B_1}^4Z_6Z_3Z_1 + Z_7Z_{12}Z_6,$$
$$Z_9Z_4Z_7Z_{12}Z_6 + {B_1}^4Z_9Z_4Z_6 + {B_1}^3Z_{10}Z_{11}Z_9,$$
$$Z_{10}Z_{11}Z_9Z_3Z_1 + B_1Z_9Z_4Z_6 + Z_{10}Z_{11}Z_9,$$
$$Z_{10}Z_{11}Z_9Z_5Z_{10}, Z_{12}Z_6Z_5Z_9Z_3, Z_1Z_3Z_2Z_8 + Z_2Z_8Z_1Z_3,$$
$$Z_1Z_4Z_6Z_3 + {B_1}^4Z_2Z_8Z_1Z_3 + {B_1}^4Z_1Z_3,$$
$$Z_1Z_4Z_6Z_5 + {B_1}^6Z_1Z_5Z_{10}Z_{11},$$
$$Z_1Z_5Z_9Z_3 + {B_1}^3Z_2Z_8Z_1Z_3, Z_1Z_5Z_9Z_4 + {B_1}^5Z_2Z_8Z_1Z_4,$$
$$Z_3Z_1Z_4Z_6 + {B_1}^3Z_4Z_7Z_{12}Z_6 + Z_5Z_9Z_4Z_6 + {B_1}^6Z_5Z_{10}Z_{11}Z_9,$$
$$Z_3Z_1Z_5Z_9 + Z_5Z_9Z_3Z_1,$$
$$Z_3Z_2Z_8Z_1 + {B_1}^2Z_5Z_9Z_3Z_1 + {B_1}^2Z_5Z_9Z_4Z_6 + B_1Z_5Z_{10}Z_{11}Z_9,$$
$$Z_4Z_6Z_3Z_1 + {B_1}^3Z_4Z_7Z_{12}Z_6 + Z_5Z_9Z_4Z_6 + {B_1}^6Z_5Z_{10}Z_{11}Z_9,$$
$$Z_4Z_6Z_5Z_9 + Z_5Z_9Z_4Z_6, Z_6Z_3Z_1Z_4 + {B_1}^6Z_7Z_{12}Z_6Z_4,$$
$$Z_6Z_3Z_1Z_5 + {B_1}^3Z_7Z_{12}Z_6Z_5, Z_6Z_3Z_2Z_8 + {B_1}^2Z_6Z_5Z_9Z_3,$$
$$Z_6Z_4Z_7Z_{12} + Z_7Z_{12}Z_6Z_4,$$
$$Z_6Z_5Z_9Z_4 + B_1Z_7Z_{12}Z_6Z_4 + B_1Z_6Z_4,$$
$$Z_9Z_3Z_1Z_4 + {B_1}^3Z_9Z_4Z_7Z_{12},$$
$$Z_9Z_3Z_1Z_5 + {B_1}^2Z_{10}Z_{11}Z_9Z_5 + {B_1}^2Z_9Z_5,$$
$$Z_9Z_4Z_6Z_3 + {B_1}^6Z_{10}Z_{11}Z_9Z_3, Z_9Z_4Z_6Z_5 + {B_1}^5Z_{10}Z_{11}Z_9Z_5,$$
$$Z_9Z_5Z_{10}Z_{11} + Z_{10}Z_{11}Z_9Z_5, Z_1Z_3Z_1, Z_1Z_4Z_7, Z_3Z_1Z_3, Z_4Z_6Z_4,$$

$$Z_5Z_9Z_5, Z_6Z_4Z_6, Z_6Z_5Z_{10}, Z_8Z_1Z_5, Z_9Z_3Z_2, Z_9Z_5Z_9, Z_{11}Z_9Z_4, Z_{12}Z_6Z_3,$$
$$B_1{}^2 + B_1B_1, B_1B_2, B_1B_3, B_1B_4, B_1B_5, B_1B_6, B_1B_7, B_1B_8,$$
$$B_1Z_1 + B_1Z_1, B_1Z_2 + B_1Z_2, B_1Z_3, B_1Z_4, B_1Z_5, B_1Z_6, B_1Z_7, B_1Z_8,$$
$$B_1Z_9, B_1Z_{10}, B_1Z_{11}, B_1Z_{12}, B_2B_1, B_2{}^2 + B_1{}^6B_2, B_2B_3, B_2B_4, B_2B_5,$$
$$B_2B_6, B_2B_7, B_2B_8, B_2Z_1, B_2Z_2, B_2Z_3 + B_1{}^6Z_3, B_2Z_4 + B_1{}^6Z_4,$$
$$B_2Z_5 + B_1{}^6Z_5, B_2Z_6, B_2Z_7, B_2Z_8, B_2Z_9, B_2Z_{10}, B_2Z_{11}, B_2Z_{12}, B_3B_1,$$
$$B_3B_2, B_3{}^2 + B_1{}^4B_3, B_3B_4, B_3B_5, B_3B_6, B_3B_7, B_3B_8, B_3Z_1, B_3Z_2,$$
$$B_3Z_3, B_3Z_4, B_3Z_5, B_3Z_6 + B_1{}^4Z_6, B_3Z_7 + B_1{}^4Z_7, B_3Z_8, B_3Z_9,$$
$$B_3Z_{10}, B_3Z_{11}, B_3Z_{12}, B_4B_1, B_4B_2, B_4B_3, B_4{}^2 + B_1B_4, B_4B_5, B_4B_6,$$
$$B_4B_7, B_4B_8, B_4Z_1, B_4Z_2, B_4Z_3, B_4Z_4, B_4Z_5, B_4Z_6, B_4Z_7,$$
$$B_4Z_8 + B_1Z_8, B_4Z_9, B_4Z_{10}, B_4Z_{11}, B_4Z_{12}, B_5B_1, B_5B_2, B_5B_3,$$
$$B_5B_4, B_5{}^2 + B_1B_5, B_5B_6, B_5B_7, B_5B_8, B_5Z_1, B_5Z_2, B_5Z_3,$$
$$B_5Z_4, B_5Z_5, B_5Z_6, B_5Z_7, B_5Z_8, B_5Z_9 + B_1Z_9, B_5Z_{10} + B_1Z_{10},$$
$$B_5Z_{11}, B_5Z_{12}, B_6B_1, B_6B_2, B_6B_3, B_6B_4, B_6B_5, B_6{}^2 + B_1{}^5B_6,$$
$$B_6B_7, B_6B_8, B_6Z_1, B_6Z_2, B_6Z_3, B_6Z_4, B_6Z_5, B_6Z_6, B_6Z_7, B_6Z_8,$$
$$B_6Z_9, B_6Z_{10}, B_6Z_{11} + B_1{}^5Z_{11}, B_6Z_{12}, B_7B_1, B_7B_2, B_7B_3,$$
$$B_7B_4, B_7B_5, B_7B_6, B_7{}^2 + B_1{}^2B_7, B_7B_8, B_7Z_1, B_7Z_2, B_7Z_3,$$
$$B_7Z_4, B_7Z_5, B_7Z_6, B_7Z_7, B_7Z_8, B_7Z_9, B_7Z_{10}, B_7Z_{11},$$
$$B_7Z_{12} + B_1{}^2Z_{12}, B_8B_1, B_8B_2, B_8B_3, B_8B_4, B_8B_5, B_8B_6, B_8B_7,$$
$$B_8{}^2 + B_1{}^2B_8, B_8Z_1, B_8Z_2, B_8Z_3, B_8Z_4, B_8Z_5, B_8Z_6, B_8Z_7, B_8Z_8,$$
$$B_8Z_9, B_8Z_{10}, B_8Z_{11}, B_8Z_{12}, Z_1B_1, Z_1B_2 + B_1{}^6Z_1, Z_1B_3, Z_1B_4,$$
$$Z_1B_5, Z_1B_6, Z_1B_7, Z_1B_8, Z_1{}^2, Z_1Z_2, Z_1Z_6, Z_1Z_7, Z_1Z_8, Z_1Z_9,$$
$$Z_1Z_{10}, Z_1Z_{11}, Z_1Z_{12}, Z_2B_1, Z_2B_2, Z_2B_3, Z_2B_4 + B_1Z_2, Z_2B_5, Z_2B_6,$$
$$Z_2B_7, Z_2B_8, Z_2Z_1, Z_2{}^2, Z_2Z_3, Z_2Z_4, Z_2Z_5, Z_2Z_6, Z_2Z_7, Z_2Z_9,$$
$$Z_2Z_{10}, Z_2Z_{11}, Z_2Z_{12}, Z_3B_1 + B_1Z_3, Z_3B_2, Z_3B_3, Z_3B_4, Z_3B_5, Z_3B_6,$$
$$Z_3B_7, Z_3B_8, Z_3{}^2, Z_3Z_4, Z_3Z_5, Z_3Z_6, Z_3Z_7, Z_3Z_8, Z_3Z_9, Z_3Z_{10},$$
$$Z_3Z_{11}, Z_3Z_{12}, Z_4B_1, Z_4B_2, Z_4B_3 + B_1{}^4Z_4, Z_4B_4, Z_4B_5, Z_4B_6,$$
$$Z_4B_7, Z_4B_8, Z_4Z_1, Z_4Z_2, Z_4Z_3, Z_4{}^2, Z_4Z_5, Z_4Z_8, Z_4Z_9, Z_4Z_{10},$$
$$Z_4Z_{11}, Z_4Z_{12}, Z_5B_1, Z_5B_2, Z_5B_3, Z_5B_4, Z_5B_5 + B_1Z_5, Z_5B_6,$$
$$Z_5B_7, Z_5B_8, Z_5Z_1, Z_5Z_2, Z_5Z_3, Z_5Z_4, Z_5{}^2, Z_5Z_6, Z_5Z_7, Z_5Z_8,$$
$$Z_5Z_{11}, Z_5Z_{12}, Z_6B_1, Z_6B_2 + B_1{}^6Z_6, Z_6B_3, Z_6B_4, Z_6B_5, Z_6B_6,$$
$$Z_6B_7, Z_6B_8, Z_6Z_1, Z_6Z_2, Z_6{}^2, Z_6Z_7, Z_6Z_8, Z_6Z_9, Z_6Z_{10}, Z_6Z_{11}, Z_6Z_{12},$$
$$Z_7B_1, Z_7B_2, Z_7B_3, Z_7B_4, Z_7B_5, Z_7B_6, Z_7B_7 + B_1{}^2Z_7, Z_7B_8, Z_7Z_1, Z_7Z_2,$$
$$Z_7Z_3, Z_7Z_4, Z_7Z_5, Z_7Z_6, Z_7{}^2, Z_7Z_8, Z_7Z_9, Z_7Z_{10}, Z_7Z_{11},$$
$$Z_8B_1 + B_1Z_8, Z_8B_2, Z_8B_3, Z_8B_4, Z_8B_5, Z_8B_6, Z_8B_7, Z_8B_8, Z_8Z_2,$$
$$Z_8Z_3, Z_8Z_4, Z_8Z_5, Z_8Z_6, Z_8Z_7, Z_8{}^2, Z_8Z_9, Z_8Z_{10}, Z_8Z_{11}, Z_8Z_{12},$$

$$Z_9B_1, Z_9B_2 + {B_1}^6Z_9, Z_9B_3, Z_9B_4, Z_9B_5, Z_9B_6, Z_9B_7, Z_9B_8, Z_9Z_1,$$
$$Z_9Z_2, Z_9Z_6, Z_9Z_7, Z_9Z_8, {Z_9}^2, Z_9Z_{10}, Z_9Z_{11}, Z_9Z_{12}, Z_{10}B_1, Z_{10}B_2, Z_{10}B_3,$$
$$Z_{10}B_4, Z_{10}B_5, Z_{10}B_6 + {B_1}^5Z_{10}, Z_{10}B_7, Z_{10}B_8, Z_{10}Z_1, Z_{10}Z_2,$$
$$Z_{10}Z_3, Z_{10}Z_4, Z_{10}Z_5, Z_{10}Z_6, Z_{10}Z_7, Z_{10}Z_8, Z_{10}Z_9, {Z_{10}}^2, Z_{10}Z_{12}, Z_{11}B_1,$$
$$Z_{11}B_2, Z_{11}B_3, Z_{11}B_4, Z_{11}B_5 + B_1Z_{11}, Z_{11}B_6, Z_{11}B_7, Z_{11}B_8,$$
$$Z_{11}Z_1, Z_{11}Z_2, Z_{11}Z_3, Z_{11}Z_4, Z_{11}Z_5, Z_{11}Z_6, Z_{11}Z_7, Z_{11}Z_8, Z_{11}Z_{10}, {Z_{11}}^2,$$
$$Z_{11}Z_{12}, Z_{12}B_1, Z_{12}B_2, Z_{12}B_3 + {B_1}^4Z_{12}, Z_{12}B_4, Z_{12}B_5, Z_{12}B_6, Z_{12}B_7,$$
$$Z_{12}B_8, Z_{12}Z_1, Z_{12}Z_2, Z_{12}Z_3, Z_{12}Z_4, Z_{12}Z_5, Z_{12}Z_7, Z_{12}Z_8,$$
$$Z_{12}Z_9, Z_{12}Z_{10}, Z_{12}Z_{11}, {Z_{12}}^2$$