NTNU
Norwegian University of Science and Technology

Decomposition of Modules over finite-dimensional Algebras

Tormod Haugland

Master of Science in Physics and Mathematics
Submission date: August 2018
Supervisor: Øyvind Solberg, IMF

Norwegian University of Science and Technology
Department of Mathematical Sciences

Abstract

We investigate algorithms for decomposing a module M over a finite-dimensional path algebra Λ. The algorithms first have to construct the endomorphism ring

$$\mathrm{End}(M) = \mathrm{Hom}(M, M).$$

Consequently, we look at three different algorithms for constructing the set of homomorphisms $\mathrm{Hom}_\Lambda(M, N)$ between two modules M and N. By extension we get $\mathrm{End}_\Lambda(M) = \mathrm{Hom}_\Lambda(M, M)$. After calculating $\mathrm{End}_\Lambda(M)$ we investigate in detail a method for decomposing the module M, using a probabilistic approach by iteratively applying Fitting's Lemma.

Finally, we provide asymptotic bounds for the runtime of all the algorithms. We then categorise them into complexity classes. Constructing the set of homomorphisms $\mathrm{Hom}_\Lambda(M, N)$ is shown to be in the complexity class **P** of polynomial-time functions.

Sammendrag

Vi undersøker algoritmer for å dekomponere en modul M over en endelig-dimensonal veialgebra Λ. Alle algoritmene konstruerer først endomorfiringen

$$\mathrm{End}(M) = \mathrm{Hom}(M, M).$$

Vi ser dermed videre på tre forskjellige metoder for å konstruere mengden homomorfier mellom to moduler M og N. Dette gir oss $\mathrm{End}_\Lambda(M) = \mathrm{Hom}_\Lambda(M, M)$. Etter å ha regnet ut $\mathrm{End}_\Lambda(M)$ ser vi i detalj på en probabilistisk metode for å dekomponere en modul M, ved å anvende Fitting's Lemma flere ganger.

Til slutt analyserer vi alle algoritmene ved å gi asymptotiske grenser for deres kjøretid. Vi kategoriserer dem deretter inn i kompleksitetsklasser. Konstruksjonen av mengden homomorfier mellom to moduler M og N blir vist til å være i kompleksitetsklassen $\mathbf{P}$, alle funksjoner med polynomiell kjøretid.

Preface

This thesis represents my final work as a master's student at Norges Teknisk-Naturvitenskapelige Universitet (NTNU). The thesis is part of the course TMA4900.

I would like to thank my supervisor Øyvind Solberg for providing invaluable feedback and guidance, and for suggesting the topic of the thesis.

I would also like to thank Sondre, Simen and Therese for providing the necessary laughter, food, and motivation. Finally I would like to thank Erlend for providing criticism and feedback on more of the thesis than can be expected of a civilian.

Contents

Introduction

In representation theory, one of the primary tasks is describing modules over finite-dimensional algebras. The ultimate description of a module M would be a decomposition

$$M = M_1 \oplus M_2 \oplus \cdots \oplus M_n.$$

In this thesis we look at methods for decomposing such a module M. In this context, our bread and butter will be calculating the endomorphism ring $\mathrm{End}(M) = \mathrm{Hom}(M, M)$: Given the endomorphism ring $\mathrm{End}(M)$, we can find a decomposition of the identity Id_M into pairwise orthogonal idempotents

$$Id_M = \pi_1 + \pi_2 + \cdots + \pi_m.$$

We will show how this can be translated into a decomposition of M. Hence a majority of the thesis will be devoted to investigating different algorithms for calculating $\mathrm{Hom}_\Lambda(M, N)$, the set of homomorphisms between two modules M and N. By extension we can find $\mathrm{End}(M)$ by setting $N = M$. We look at three different algorithms for constructing $\mathrm{Hom}_\Lambda(M, N)$.

The first algorithm uses plain linear algebra to specify all the constraints required of any homomorphism $f : M \to N$, as a large system of linear equations.

The second is the algorithm introduced in [16], which uses the tensor-product as a functor on a projective presentation. This yields an exact sequence for which an isomorphism can be constructed to $\mathrm{Hom}_\Lambda(M, N)$, using adjoint associativity.

The third algorithm is a variation of the second, which uses the Hom-functor instead of the tensor-product. Again, we get an exact sequence for which an isomorphism can be constructed to $\mathrm{Hom}_\Lambda(M, N)$.

We will also investigate a method of decomposing a module M without decomposing the identity Id_M. Applying Fitting's Lemma, we can decompose M into two parts if we can find an endomorphism ϕ which is both non-invertible and non-nilpotent. Finding such an endomorphism deterministically can be challenging. Thus we take the probabilistic approach, and select random elements from $\mathrm{End}(M)$. The analysis provides a bound for the probability of finding a non-invertible non-nilpotent element ϕ.

Chapter 1 introduces basic theory of Quivers and Path Algebras, which provide the foundations upon which the remaining theory is based.

Chapter 2 introduces category theory. This gives us functors, and specifically adjoint functors. Chapter 3 recalls basic theory of endomorphisms and tensor products. The theory in Chapter 2 and 3 are both heavily employed in the second and third algorithm.

Chapter 4 introduces some key concepts in computational complexity theory. We use this theory to analyse all the algorithms introduced in the thesis, and categorise algorithms —and hence underlying problems —into complexity classes.

Chapter 5 introduces the algorithm using linear algebra, which we name *the Linear Algebra algorithm.*

Chapter 6 introduces the algorithm introduced in [16], which we name the *Green-Heath-Struble algorithm.*

Chapter 7 introduces the algorithm using the Hom-functor, which we name the *Hom-functor algorithm.*

Chapter 8 introduces the probabilistic decomposition of M.

Finally, Chapter 9 summarises the analysis of all the algorithms. It then attempts to compare the three first algorithms. Some experimental results of the probabilistic algorithm is also presented. Finally, a categorisation of the algorithms into complexity classes is performed.

Chapter 1

Quivers & Path Algebras

In this chapter we introduce the concepts of quivers and path algebras. We then introduce representations of a quiver, which we show are equivalent to modules over the path algebra over the quiver.

1.1 Quivers

Definition 1.1. A quiver Γ is a directed (multi)graph. We label the *vertices* of a quiver by $\Gamma_0 = \{v_1, v_2, v_3, \dots, n\}$; and its edges, or *arrows*, by some arbitrary identifiers $\Gamma_1 = \{\alpha_0, \alpha_1, \dots, \alpha_n\}$. We will exclusively work with quivers with a finite set of vertices and arrows. The vertex an arrow α starts in, is denoted $s(\alpha)$; and the vertex the arrow ends in, is denoted $e(\alpha)$.

Recall that a multigraph can have *multiple* arrows between each pair of vertices (or from a vertex to itself). Before proceeding, we look at some examples of quivers.

Example 1.2. Γ: $\begin{tikzcd} v_1 \arrow[loop right, "\alpha"] \end{tikzcd}$, $\Gamma_0 = \{1\}$ and $\Gamma_1 = \{\alpha\}$.

Example 1.3. Γ: $\begin{tikzcd} v_1 \arrow[r, bend left, "\alpha"] \arrow[r, bend right, "\beta"'] & v_2 \end{tikzcd}$, $\Gamma_0 = \{v_1, v_2\}$, and $\Gamma_1 = \{\alpha, \beta\}$

Example 1.4. Γ: $\begin{tikzcd} v_1 \arrow[r, "\alpha"] & v_2 \arrow[r, "\beta"] & v_3 \end{tikzcd}$, $\Gamma_0 = \{v_1, v_2, v_3\}$, and $\Gamma_1 = \{\alpha, \beta\}$

Starting in some vertex in a quiver, we can walk or traverse the quiver by following a sequence of arrows. A **path** p of a quiver Γ, is such an ordered sequence of arrows $p = a_0a_1a_2 \dots a_{n-1}a_n$, with the requirement that $e(a_{k-1}) = s(a_k)$, for all k. In other words, we write paths from left to right. In addition to the paths naturally arising from the arrows of a quiver, we have the *trivial paths*, e_i for each vertex i. In later chapters we will sometimes abuse notation, and write v_i in place of e_i. The start of a path $p = a_0a_1a_2 \dots a_{n-1}a_n$ is $s(a_0)$, and the end is $e(a_n)$. If $s(p) = e(p)$, and the path is not trivial, then we say that the path is an **oriented cycle**.

We see that the quivers in example 1.3 and 1.4 does not contain any oriented cycles, while all paths in example 1.2 are oriented cycles. To further illuminate oriented cycles, we construct another example.

Example 1.5. Consider the quiver Γ: $$\begin{tikzcd} v_1 \arrow[r, bend left, "\alpha"] & v_2 \arrow[l, bend left, "\beta"] \arrow[loop right, "\gamma"] \end{tikzcd}$$.

Then the set of all non-trivial paths are

$$\{\alpha, \alpha\gamma, \alpha\gamma^2, \dots, \beta, \beta\alpha, \beta\alpha\gamma, \dots\}$$

We see that both γ and $\alpha\beta$ are oriented cycles. This causes the set of paths to be infinite, as we can indefinitely traverse γ or $\alpha\beta$. In general; any quiver with oriented cycles have infinitely many paths. This will be investigated further in Proposition 1.8.

1.2 Path Algebras

In this section we continue by introducing some additional structure over a quiver, namely that of an algebra. This construction yields what we call a **path algebra**.

Definition 1.6. Let k be a field, and Γ a quiver. Then $k\Gamma$ is a **path algebra**, defined by the vector space over Γ with all paths in Γ as basis. An arbitrary element in $k\Gamma$ is then a linear combination of the paths of Γ, with elements of k as scalars,

$$x = a_0p_0 + a_1p_1 + \cdots + a_np_n, \quad a_i \in k,\ p_i \in \Gamma.$$

Addition in the algebra is performed in the obvious way,

$$\begin{aligned} x &= a_0p_0 + a_1p_1 + \cdots + a_np_n \\ y &= b_0p_0 + b_1p_1 + \cdots + b_np_n \\ x + y &= (a_0 + b_0)p_0 + (a_1 + b_1)p_1 + \cdots + (a_n + b_n)p_n. \end{aligned}$$

Multiplication of elements of the algebra distributes in the normal way. Let p, q be non-trivial paths and e_k be the trivial path in vertex k, then pointwise multiplication is defined by

$$p \cdot q = \begin{cases} pq & \text{if } p \text{ ends where q starts} \\ 0 & \text{otherwise} \end{cases}$$

$$p \cdot e_j = \begin{cases} p & \text{if } p \text{ ends in vertex } j \\ 0 & \text{otherwise} \end{cases}$$

$$e_i \cdot q = \begin{cases} q & \text{if } q \text{ starts in vertex } i \\ 0 & \text{otherwise} \end{cases}$$

$$e_i \cdot e_j = \begin{cases} e_i & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

The identity of the path algebra can easily be shown to be the linear combination of all trivial paths,

$$\mathrm{Id}_{k\Gamma} = e_1 + e_2 + \cdots + e_n.$$

To get our head around what a path algebra looks like, we continue with an example.

Example 1.7. $\Gamma\colon\ v_1 \xrightarrow{\alpha} v_2$
The paths of Γ are $\{e_1, e_2, \alpha\}$. Thus an arbitrary element of $k\Gamma$ will look like

$$x = a_1e_1 + a_2e_2 + a_3\alpha.$$

Addition of two elements in $k\Gamma$ gives

$$\begin{aligned}(x+y) &= (a_1e_1 + a_2e_2 + a_3\alpha) + (b_1e_1 + b_2e_2 + b_3\alpha)\\ &= (a_1+b_1)e_1 + (a_2+b_2)e_2 + (a_3+b_3)\alpha\end{aligned}$$

and multiplication of two elements gives

$$\begin{aligned}x\cdot y =&(a_1e_1 + a_2e_2 + a_3\alpha)\cdot(b_1e_1 + b_2e_2 + b_3\alpha)\\ =&a_1e_1\cdot b_1e_1 + a_1e_1\cdot b_2e_2 + a_1e_1\cdot b_3\alpha\\ &+ a_2e_2\cdot b_1e_1 + a_2e_2\cdot b_2e_2 + a_2e_2\cdot b_3\alpha\\ &+ a_3\alpha\cdot b_1e_1 + a_3\alpha\cdot b_2e_2 + a_3\alpha\cdot b_3\alpha\\ =&a_1b_1e_1 + a_2b_2e_2 + a_1b_3\alpha e_2 + a_3b_2e_1\alpha\\ =&a_1b_1e_1 + a_2b_2e_2 + (a_3b_2 + a_1b_3)\alpha\end{aligned}$$

Finally, we convince ourselves that $e_1 + e_2$ is in fact the identity,

$$\begin{aligned}x(e_1+e_2) &= (a_1e_1 + a_2e_2 + a_3\alpha)(e_1+e_2)\\ &= (a_1e_1e_1 + a_2e_2e_1 + a_3\alpha e_1) + (a_1e_1e_2 + a_2e_2e_2 + a_3\alpha e_2)\\ &= (a_1e_1 + 0 + 0) + (0 + a_2e_2 + a_3\alpha)\\ &= (a_1e_1 + a_2e_2 + a_3\alpha).\end{aligned}$$

A similar computation from the left shows that indeed $e_1 + e_2$ is the identity.

Noticeably, $k\Gamma$ is isomorphic to the lower triangular $2x2$ matrices $M_{2x2} = \begin{bmatrix} k & 0\\ k & k\end{bmatrix}$. This can be shown by considering multiplication of two elements in M_{2x2}, and comparing. The example gives a first indication that many familiar algebraic structures can be described using path algebras.

We continue by stating a very important proposition regarding the dimensionality of path algebras. The proposition gives us a very easy method to check whether or not a path algebra is finite-dimensional; namely that it has no oriented cycles.

Proposition 1.8. *Let $k\Gamma$ be a path algebra, with n vertices. Then $\dim_k k\Gamma < \infty$ if and only if Γ contains no oriented cycles.*

Proof. $\underline{\Rightarrow:}$ Given $\dim_k k\Gamma = m < \infty$, $k\Gamma$ has some finite basis $\mathcal{B} = \{p_0, p_1, \dots, p_m\}$. Suppose then that Γ has an oriented cycle, given by $x = p_{k_1}p_{k_2}\dots p_{k_n}$. But then either x^{m+1} is not in $\mathcal{B}$, or some $x^k, k \leq m$ is not in $\mathcal{B}$, since $\mathcal{B}$ contains m elements. This is a contradiction, as the basis $\mathcal{B}$ contains all paths of $k\Gamma$.

$\underline{\Leftarrow:}$ Suppose Γ has no oriented cycles. We wish to show that Γ has a finite number of paths. But since there are no oriented cycles, any path can at most have length $n-1$. Given a finite number of arrows, there can only be a finite amount of paths of maximum length $n-1$. Thus Γ contains only a finite amount of paths, and $\dim_k k\Gamma < \infty$. □

To wrap up the section, we introduce the *ideal of all non-trivial paths.* We use this construction later in Section 1.4.

Definition 1.9. Let Γ be some quiver. We denote J to be the ideal of $k\Gamma$ generated by all the arrows, i.e. the non-trivial paths of length 1.

1.3 Modules and representations

In this section we naturally extend the notion of a module over an algebra, to a module over a path algebra. We then introduce the concept of a *representation* over a quiver. We then show that these two notions are in fact equal, which gives us a handy way to construct modules over a path algebra.

Recall that a right module M over a ring R is an abelian group with an associated map $M \times R \to M$ such that

$$\begin{aligned}
&(i)\ (m_1 + m_2)r = m_1r + m_2r\\
&(ii)\ m(r_1 + r_2) = mr_1 + mr_2\\
&(iii)\ (mr_1)r_2 = m(r_1r_2)\\
&(iv)\ m1 = m, \quad 1 \in R
\end{aligned}$$

Replacing R with $k\Gamma$, we get the path algebra specific definition of a module. We would like to find out what a module over $k\Gamma$ looks like. The following proposition gives some very concrete ideas.

Proposition 1.10. *Let M be a right module over $k\Gamma$. Then M decomposes as vector spaces $M = Me_1 \oplus Me_2 \oplus \cdots \oplus Me_n$.*

Proof. Let $m \in M$. Then $m = m1_{k\Gamma} = m(e_1 + \cdots + e_n) \in Me_1 \oplus Me_2 \oplus \cdots \oplus Me_n$. Define $M_k = Me_k$. Then let $m \in M_i \cap \sum_{j \neq i} M_j$. So $m = m_i e_i = \sum_{j \neq i} m_j e_j$, for some m_i and m_j. But we also have the following relation: $me_i = (me_i)e_i = m_i(e_i e_i) = m_i e_i = m$. Combining the two expressions for m, we get

$$m = me_i = (\sum_{j \neq i} m_j e_j)e_i = \sum_{j \neq i} m_j(e_j e_i).$$

But this is 0, as $e_i e_j = 0$, $i \neq j$. Thus $m = 0$, $M_i \cap \sum_{j \neq i} M_j = 0$, and M decomposes as indicated. □

In other words, we know that any module M over a path algebra $k\Gamma$ with m vertices, decomposes into n vector spaces (due to the underlying structure of $k\Gamma$). This hints at the construction of what is called a *representation.*

Consider taking, for every vertex in a quiver Γ, a vector space $V(i)$, and putting it in the place of the vertex. Then for every arrow α from vertex i to vertex j, put a homomorphism $f_\alpha \colon V(i) \to V(j)$. We call such a construction a **representation** of Γ.

Before stating the main theorem of this section, namely that the representations and modules over a path algebra are the same, we give an example.

Example 1.11. Let

$$\begin{tikzcd} \Gamma\colon\ 1 \arrow[r, "\alpha"] & 2 \arrow[r, "\beta"] & 3 \end{tikzcd}$$

and let $k\Gamma$ be the path algebra over Γ. Then the following are representations of $k\Gamma$,

$$\begin{tikzcd} \Gamma\colon\ k \arrow[r, "1"] & k \arrow[r, "1"] & k \end{tikzcd}$$

$$\begin{tikzcd} \Gamma\colon\ k \arrow[r, "1"] & k \arrow[r, "0"] & 0 \end{tikzcd}$$

$$\begin{tikzcd} \Gamma\colon\ k^2 \arrow[r, "\begin{pmatrix}1 & 0\\ 1 & -1\\ 0 & 1\end{pmatrix}"] & k^3 \arrow[r, "\begin{pmatrix}1 & 1 & 0\\ 0 & -1 & 1\end{pmatrix}"] & k^2 \end{tikzcd}$$

Theorem 1.12. *The set of representations over a quiver $k\Gamma$ are precisely the set of right $k\Gamma$ modules.*

Proof. We first show how to go from a $k\Gamma$-module M to a representation: Suppose we are given a module M. For every vertex v_i define $V(i)$ to be Me_i. Then for every arrow α, define $f_\alpha : V(i) \to V(j)$ by

$$f_\alpha(me_i) = me_i\alpha$$

This yields a representation (V, f) of $k\Gamma$.

Suppose now that we are given a representation (V, f). We reverse the procedure above: Let $M = \oplus_{i=1}^{n} V(i)$, i.e. $m = (v_1, \dots, v_n)$. Define

$$me_i = (0, \dots, 0, v_i, 0, \dots, 0),$$

Where v_i is placed in the ith coordinate. For every f_α, let

$$m\alpha = (0, \dots, 0, f_\alpha(v_i), 0, \dots, 0).$$

Where $f_\alpha(v_i)$ is also placed in the ith coordinate. This yields a $k\Gamma$-module M.

□

1.4 Relations

In this section we introduce a class of ideals called the *relations* over $k\Gamma$. We take special notice to these as they occur prominently.

Definition 1.13. Let $\sigma = a_1p_1 + \cdots + a_kp_k$ be a linear combination of paths over a path algebra $k\Gamma$, with $e(p_i) = e(p_1)$, $s(p_i) = s(p_1)$ for all i, i.e. all paths start and end in the same vertex; and the length of each p_i greater than or equal to 2. Then σ is a **relation** over $k\Gamma$.

As hinted to, our primary use of a set of relations is to produce a quotient of $k\Gamma$. We illustrate with an example.

Example 1.14.

$$\begin{tikzcd}\Gamma\colon v_1 \arrow[r, "\alpha"] & v_2 \arrow[r, "\beta"] & v_3\end{tikzcd}$$

Let $\sigma = \alpha\beta$. Then $\Lambda = \frac{k\Gamma}{\langle\sigma\rangle}$ is the algebra over the quiver Γ with relations σ. So what does an element in Λ look like? We know that $k\Gamma$ has basis $\{e_1, e_2, e_3, \alpha, \beta, \alpha\beta\}$. But under our relation, the path $\alpha\beta$ is factored out, so our new basis becomes $\{e_1, e_2, e_3, \alpha, \beta\}$. As before, we can construct modules over Λ, albeit now with the restriction that $f_\beta \circ f_\alpha$ must compose to 0. E.g.

$$\begin{tikzcd}\Gamma\colon k \arrow[r, "1"] & k \arrow[r, "0"] & k\end{tikzcd}$$

$$\begin{tikzcd}\Gamma\colon k^2 \arrow[r, "\begin{pmatrix}0 & 0\\ 1 & 0\end{pmatrix}"] & k^2 \arrow[r, "\begin{pmatrix}0 & 0\\ 1 & 0\end{pmatrix}"] & k^2\end{tikzcd}$$

We finish the chapter with three definitions introducing notation and terminology which will come in handy.

Definition 1.15. Let $\rho = \{\sigma_t\}_t$ be a set of relations and $k\Gamma$ be a path algebra. Then $k\Gamma/\langle\rho\rangle$ is said to be **a quiver with relations**.

Definition 1.16. Recall that J is the ideal generated by all arrows of Γ. Let ρ be a set of relations over a path algebra $k\Gamma$. Then $\bar{J}$ is the ideal generated by all the arrows of Γ under the set of relations ρ, i.e. $\bar{J} = J/\langle\rho\rangle$.

Definition 1.17. Let ρ be a set of relations over a quiver Γ. Then ρ is an **admissible set of relations** if

$$\bar{J}^t \subseteq \langle \rho \rangle \subseteq \bar{J}^2.$$

In other words: ρ is an admissible set of relations if the ideal generated by ρ is a subset of all paths of length at least 2, and a superset of all paths of some minimum length t.

Chapter 2

Categories

In this chapter we introduce categories, functors, and a number of pertinent properties of these. Our goal is to lay the foundation required to prove Theorem 3.9, and consequently Corollary 3.10.

2.1 Definition

The notion of a category is created in an attempt to unify the different structures and objects worked on in mathematics, under one umbrella. Anyone having been exposed to different branchces of mathematics have undoubtedly been introduced to a variety of categories, e.g. rings, groups, fields, topological spaces, sets, measure spaces, and so on. Introducing something which generalises these concepts, brings us up one level of abstraction.

Definition 2.1. A **category** $\mathscr{C}$ is a collection of objects, denoted $\mathrm{Obj}(\mathscr{C})$, with an associated collection of morphisms for each pair of objects $A, B \in \mathscr{C}$: $\mathrm{Hom}_{\mathscr{C}}(A, B)$; such that the following holds:

(i) For every triplet of objects A, B, C, and for all maps $f \in \mathrm{Hom}_{\mathscr{C}}(A, B)$, $g \in \mathrm{Hom}_{\mathscr{C}}(B, C)$, there exists a composition $g \circ f \in \mathrm{Hom}_{\mathscr{C}}(A, C)$.

(ii) Associativity for composition of morphisms hold.

(iii) For each object $X \in \mathrm{Obj}(C)$, there exists an identity morphism, $1_X : X \to X$, such that for $f \in \mathrm{Hom}_{\mathscr{C}}(X, B)$, $g \in \mathrm{Hom}_{\mathscr{C}}(A, X)$, we have $f \circ 1_X = f$, $1_X \circ g = g$.

When no confusion can arise, we will omit the subscript of $\mathrm{Hom}_{\mathscr{C}}(A, B)$, instead writing $\mathrm{Hom}(A, B)$. And, in the absence of such confusion, for an object C belonging to $\mathrm{Obj}(\mathscr{C})$, we will abuse notation further and write $C \in \mathscr{C}$.

2.2 Functors

One of the motivations for introducing the concept of a category, is to facilitate comparisons between these, and in particular obtain a construction known as a duality. For this, the notion of a mapping between categories is needed. In this section we introduce functors, which are such mappings.

Definition 2.2. A **covariant functor** F is a mapping between two categories $F : \mathscr{C} \to \mathscr{D}$, such that:

(i) Each object $C \in \mathscr{C}$ gets sent to some $D = F(C) \in \mathscr{D}$.

(ii) Each morphism $\mathrm{Hom}_{\mathscr{C}}(A, B)$ gets sent to some morphism $\mathrm{Hom}_{\mathscr{D}}(F(A), F(B))$.

(iii) Compositions of morphisms are preserved through the functor: $F(g \circ f) = F(g) \circ F(f)$.

(iv) The identity is preserved uniquely through the functor: $F(1_X) = 1_{F(X)}$.

The perhaps canonical example of a functor is the one in *algebraic topology*, where one associates a topological space X with some group. Thus it is a functor from the category of topological spaces to the category of groups, allowing us to use the machinery of group theory, indirectly, on the category of topological spaces. Another important functor is the identity functor, acting in the obvious way.

There exist also another species of functor, namely a **contravariant functor**. The difference between these and their covariant siblings, lie in that a contravariant functor flips the direction of the morphisms:

(i) Each morphism $\mathrm{Hom}_{\mathscr{C}}(A, B)$ gets sent to some morphism $\mathrm{Hom}_{\mathscr{D}}(F(B), F(A))$.

(ii) Compositions of morphisms are flipped and preserved under the functor: $F(g \circ f) = F(f) \circ F(g)$.

Finally, one might consider ways of transforming one functor into another. This brings forth the idea of **morphisms of functors**, also called **natural transformations**.

Definition 2.3. Let F and H be two functors from $\mathscr{C}$ to $\mathscr{D}$. Then $\mathscr{F}$ is said to be a **natural transformation**, if for every object $C \in \mathscr{C}$, there exists a map $\mathscr{F}_C : F(C) \to G(C)$ in $\mathscr{D}$, such that for every map $f \in \mathrm{Hom}_{\mathscr{C}}(A, B)$, the following diagram commutes:

$$\begin{tikzcd}
F(A) \arrow[r, "F(f)"] \arrow[d, "\mathscr{F}_A"'] & F(B) \arrow[d, "\mathscr{F}_B"'] \\
G(A) \arrow[r, "G(f)"] & G(B)
\end{tikzcd}$$

If each $\mathscr{F}_C$ is an isomorphism, we say that $\mathscr{F}$ is a **natural isomorphism**. If the functors involved are contravariant, the direction of the arrows are flipped accordingly.

2.3 Equivalencies and Dualities

In this section we introduce equivalencies and dualities, which extend the idea of two mathematical objects being equal to categories. First however, we extend the notion of an isomorphism to a general category.

The question of whether two objects in a category are the same, or alike, is typically phrased as whether or not there exists an *isomorphism* between them. The way in which isomorphisms are (historically) defined vary from category to category — in the category of topological spaces it is not even called an isomorphism, but rather a *homeomorphism*. We now unify these definitions for a general category.

Definition 2.4. Let $\mathscr{C}$ be a category, and let $A, B \in \mathscr{C}$. Then $f : A \to B$ is an **isomorphism**, if there exists $g : B \to A$ in $\mathscr{C}$, such that

$$\begin{aligned} g \circ f =& 1_A \\ f \circ g =& 1_B \end{aligned}$$

Inevitably, one might ask oneself when two categories, for all intents and purposes are the same; extending the idea of isomorphisms to categories. The definition mirror the above definition of an isomorphism.

Definition 2.5. A functor $F : \mathscr{C} \to \mathscr{D}$ is said to be an **equivalence**, if there exists another functor $H : \mathscr{D} \to \mathscr{C}$, such that for $C \in \mathscr{C}$, $D \in \mathscr{D}$:

$$\begin{aligned} H \circ F \cong 1_{\mathscr{C}} \\ F \circ H \cong 1_{\mathscr{D}} \end{aligned}$$

In other words, an equivalence of categories is a functor for which there exists an inverse functor, and the composition of these is isomorphic to the identity functor. If the functor(s) in question are contravariant, the equivalence is called a **duality**.

2.4 Exact functors

We now introduce exact functors, which are functors preserving exact sequences.

Definition 2.6. Let $F : \mathscr{C} \to \mathscr{D}$ be a functor. Then let the following sequence be exact in $\mathscr{C}$:

$$0 \to A \to B \to C \to 0$$

Then F is said to be

- **Left-exact** if $0 \to F(A) \to F(B) \to F(C)$ is exact.
- **Right-exact** if $F(A) \to F(B) \to F(C) \to 0$ is exact.
- **Exact** if $0 \to F(A) \to F(B) \to F(C) \to 0$ is exact.

2.5 Adjoint functors

In this section we introduce adjoint functors, which facilitates the definition of Theorem 3.9.

Definition 2.7. Let $\mathscr{C}$ and $\mathscr{D}$ be two categories, and let $F : \mathscr{C} \to \mathscr{D}$ and $G : \mathscr{D} \to \mathscr{C}$ be two functors. (F, G) is said to be **adjoint functors**, or an **adjoint pair**, if for every $X \in \mathrm{Obj}(\mathscr{C})$ and $Y \in \mathrm{Obj}(\mathscr{D})$,

$$\mathrm{Hom}_{\mathscr{D}}(FX, Y) \cong \mathrm{Hom}_{\mathscr{C}}(X, GY),$$

and these bijections are natural in both variables.

Adjoint functors possess interesting properties. Suppose we have two adjoint functors F and G over categories $\mathscr{C}$ and $\mathscr{D}$. Select then two objects X and Y from $\mathscr{C}$ and $\mathscr{D}$ respectively, and transform them via F and G respectively. We now have two new objects, call them $Y_0 = FX$ and $X_0 = GY$. The fact that F and G are adjoint now means that the class of morphisms between Y_0 and Y; and X_0 and X admits a natural isomorphism between them, i.e. there is a family of bijections between any $f : Y_0 \to Y$ to some $g : X_0 \to X$.

Chapter 3

Homomorphisms and Tensor Products

In this chapter we recall and develop some basic theory for modules. We start by developing some elementary theory for endomorphism rings. We will then introduce projective presentations. Then we will briefly introduce and discuss tensor products, which allows us to introduce an important result for the Green-Heath-Struble algorithm; namely that of Adjoint Associativity.

In this chapter, Λ will unless otherwise specified be a ring, and M, N will unless otherwise specified be right Λ-modules.

3.1 Endomorphism rings

The endomorphism ring for a Λ-module M, denoted $\operatorname{End}_\Lambda(M)$, is the set of all morphisms from M to itself. If we endow this set with the operations of addition in the obvious way, and composition as multiplication, $\operatorname{End}_\Lambda(M)$ becomes a ring.

Proposition 3.1. *$\operatorname{End}_\Lambda(M)$ is a ring.*

Proof. Recall that M as a module is an additive abelian group. To show that $\operatorname{End}_\Lambda(M)$ is a ring, we go about showing the standard axioms. The reader is referred to [3, p. 150] for a list of these.

Let $f, g \in \operatorname{End}_\Lambda(M)$ and $x, y \in M$. We define addition and multiplication in $\operatorname{End}_\Lambda(M)$ in the following way:

$$(f+g)(m) = f(m) + g(m)$$
$$(fg)(m) = f(g(m))$$

This gives us the following:

$$\begin{aligned}(f+g)(x+y) &= f(x+y)+g(x+y)\\ &= f(x)+g(x)+f(y)+g(y)\\ &= (f+g)(x)+(f+g)(y)\\ (fg)(x+y) &= f(g(x)+g(y))\\ &= f(g(x))+f(g(y))\\ &= (fg)(x)+(fg)(y)\end{aligned}$$

Thus $f+g, fg \in \mathrm{End}_\Lambda(M)$. Scalar multiplication is defined by

$$f(x\lambda) = f(x)\lambda,$$

where $\lambda \in \Lambda$. In similar fashion, associativity of addition and multiplication in End_Λ is easily shown, as well as the distributive laws.

The identity of $\mathrm{End}_\Lambda(M)$ is given by $\mathrm{Id}_M \colon m \mapsto m$, and the zero element is given by $0_M \colon m \mapsto 0$. For any element $f \in \mathrm{End}_\Lambda(M)$, the additive inverse $-f$ is given by $-f \colon m \mapsto -f(m)$. Clearly, $(f-f)(m) = f(m) - f(m) = 0$.

Hence $\mathrm{End}_\Lambda(M)$ is a ring.

□

The fact that $\mathrm{End}_\Lambda(M)$ is a ring will become useful after Theorem 3.2. We will then be able to create a decomposition of M using $\mathrm{End}_\Lambda(M)$.

3.2 Decomposition by using the endomorphism ring

We now introduce a theorem which motivates the calculation of the endomorphism ring.

Theorem 3.2. *Let Id_M be the identity-endomorphism on M. If Id_M decomposes as $\mathrm{Id}_M = \pi_1 + \cdots + \pi_m$, where π_i are pairwise orthogonal idempotents, then M decomposes as $M = N_1 \oplus \cdots \oplus N_m$, where $N_i = \mathrm{Im}(\pi_i)$.*

Proof. We have $\mathrm{Id}_M = \pi_1 + \cdots + \pi_m$. Since each π_i are pairwise orthogonal idempotents, we have that $\pi_i \circ \pi_j = 0$. We claim that

$$\ker \pi_i = \sum_{j \neq i} \mathrm{Im}\, \pi_j.$$

The inclusion from right to left is obvious from the orthogonality of the π_i. So let $x \in \ker \pi_i$, i.e. $\pi_i(x) = 0$. If $x = 0$, then clearly $x \in \sum_{j\neq i} \mathrm{Im}\, \pi_j$, as $\pi_j(0) = 0$ for any j. So let $x \neq 0$, then $\mathrm{Id}_M(x) = x = (\pi_1 + \cdots + \pi_m)(x)$, which is not 0 by hypothesis. Thus at least one $\pi_j(x) \neq 0$, $j \neq i$; since $\pi_i(x) = 0$. Thus $x \in \sum_{j\neq i} \mathrm{Im}\, \pi_j$. But since $\ker \pi_i = \sum_{j\neq i} \mathrm{Im}\, \pi_j$, and the image and kernel of a map is disjoint, $\operatorname{Im}\pi_j \cap \sum_{j\neq i} \operatorname{Im}\pi_j = 0$. Thus M decomposes as $M = N_1 \oplus \cdots \oplus N_m$, where $N_i = \operatorname{Im}(\pi_i)$.

□

The utility of the previous theorem cannot be overstated. If we are able to construct the endomorphism ring of a module, which of course would give us the identity, then if we could find a decomposition of the identity into orthogonal piecewise idempotents, we would have the sought after decomposition for the module. A procedure for finding such a decomposition of the identity is shown in [6] . Consequently, we need now only describe a procedure or methodology for constructing the endomorphism ring of an arbitrary module over an arbitrary ring.

3.3 Projective presentations

In this section, we introduce the concept of projective presentations of a module over an artin algebra. We also quickly note one way a projective presentation can be constructed. Finally we look at a specific application of projective presentations to modules over a path algebra.

Recall that an artin algebra Λ is an R-algebra over a commutative artinian ring R, that is a finitely generated R-module.

Definition 3.3. Let Λ be an artin algebra, and let X be a Λ-module. Then a **projective presentation** of X is an exact sequence of projective modules

$$P_1 \xrightarrow{f_1} P_0 \xrightarrow{f_2} X \to 0.$$

As this is an exact sequence, we get the following isomorphism

$$X \cong \operatorname{coker}(f_1).$$

Given some right Λ-module X, there are ways to compute a projective presentation of the module. We present without detail or proof a method taken from QPA[20]. Recall that the Jacobsen radical $\underline{r}$ of Λ is the ideal consisting of all elements of Λ that annihilates all simple (right-)Λ-modules. The radical of a right Λ-module X is then $M\underline{r}$. Recall also that the *top* of M is $M/M\underline{r}$. To find a projective presentation, we construct the following diagram:

$$\begin{tikzcd}
 & & & P_1 \arrow[dl, "f_1"] \\
 & & P_0 \arrow[dl, "f_0"] \arrow[d] & \\
 & M \arrow[r, "\pi"] \arrow[dl] & M/M\underline{r} \arrow[r] \arrow[d] & 0 \\
0 & & 0 &
\end{tikzcd}$$

We first compute the top $M/M\underline{r}$, and note that we have an essential epimorphism from M to $M/M\underline{r}$. We then calculate the projective cover (P_0, f_0), implying that P_0 is projective. We can repeat this process by setting $M = \ker f_0$, which further yields P_1, and so on. This gives us a projective presentation of M.

One thing that will be of interest is the dimension of P_0 and P_1. We state some upper bounds here. Given the above algorithm, they can be found by considering what the worst cases are when iteratively creating the projective covers. They are

$$\begin{aligned}
\max_{v\in\Gamma_0}\{\dim_k v\Lambda\} &\overset{def}{=} \mathcal{M} \\
\dim_k P_0 &\leq \mathcal{M}\cdot\dim_k M \\
\dim_k \ker f_0 &\leq (\mathcal{M}\cdot\dim_k M) - \dim_k M \\
\dim_k(P_1) &\leq \mathcal{M}\cdot\dim_k \ker f_0 \\
&= \mathcal{M}\cdot((\mathcal{M}\cdot\dim_k M) - \dim_k M) \\
&= \dim_k M\left(\mathcal{M}^2 - \mathcal{M}\right)
\end{aligned}$$

Note that $\mathcal{M}$ is a constant for a given path algebra Λ.

3.4 Tensor Product

In this section we introduce the construction of the tensor product. The tensor product is an integral part of the Green-Heath-Struble algorithm. Specifically, the result of adjoint associativity shown in Theorem 3.9 is used in one of the core isomorphisms of the algorithm.

Definition 3.4. Let M and N be left and right R-modules, respectively. Then the **tensor product** of M and N is an abelian group $M \otimes_R N$ and a balanced map h; such that for every abelian group G and balanced map $f\colon M\times N \to G$, there exists a unique homomorphism $\bar{f}\colon M\otimes_R N \to G$ of groups, such that the following diagram commutes:

$$\begin{tikzcd}
M\times N \arrow[rr, "h"] \arrow[dr, "f"'] & & M\otimes_R N \arrow[dl, "\bar{f}"] \\
& G &
\end{tikzcd}$$

We say that the tensor product solves the *universal mapping problem* of the diagram above.

Let's take a step back and reason about what we are doing here. We are constructing an abelian group $M\otimes_R N$, which for any balanced map $f\colon M\times N \to G$ yields a map $\bar{f}\colon M\otimes_R N \to G$. In a sense, we can view elements $M\otimes_R N$ as corresponding to 'flattened' versions of the tuples in $A\times B$.

Another effect this has, is that it allows us to reason about balanced maps in the following way: Suppose we have a balanced map $A\times B \to G$. Does this map encode any structure that a linear map could not? If we construct the tensor product $(A \otimes_R B, h)$ (the existence of which will be proven shortly), we immediately get a homomorphism $\bar{f}$, which is additive by definition, corresponding to the balanced map f.

Theorem 3.5. *Let M and N be left and right R-modules, respectively. Then the tensor product of M and N exists.*

Proof. Let Z be the free abelian group with $M \times N$ as basis, i.e. the basis elements of Z are the ordered pairs (a, b). Then let F be the subgroup generated by the elements on the form:

$$\begin{aligned}
&(a, b + b') - (a, b) - (a, b') \\
&(a + a', b) - (a, b) - (a', b) \\
&(ar, b) - (a, rb).
\end{aligned}$$

Note that these describe the relations of a balanced map. Define $M \otimes_R N = Z/F$. We proceed to show that this group indeed is the tensor product of M and N.

We seek a balanced map $h\colon M \times N \to M \otimes_R N$. But clearly this is just the canonical homomorphism:

$$h\colon (a, b) \mapsto (a, b) + F$$

We label $(a, b) + F$ by $a \otimes b$. We have that h is balanced, as,

$$\begin{aligned}
h((a + a', b)) &= (a + a', b) + F = (a, b) + (a', b) + F = h((a, b) + (a', b)) \\
h((a, b + b')) &= (a, b + b') + F = (a, b) + (a, b') + F = h((a, b) + (a, b')) \\
h((ar, b)) &= (ar, b) + F = (a, rb) + F = h((a, rb)).
\end{aligned}$$

The last part in all equalities come from the definition of F. What remains now is finding $\bar{f}$. Define first $f' \colon Z \to G$ by $f'((a, b)) = f(a, b)$. As Z is free over $M \times N$ f' is clearly a homomorphism. Furthermore, as f is balanced, $F \subset \ker f'$, and f' induces a homomorphism $\bar{f}\colon Z/F \to G$, which is the desired map.

We note that this map is unique.

□

We continue by proving an important theorem, stating that any two tensor products over the same modules are isomorphic. While important in its own right, the technique involved in proving the theorem is also itself noteworthy, and is employed in the proof of Theorem 3.8.

Theorem 3.6. *Let M and N be left and right R-modules, respectively. Then any two tensor products of M and N are isomorphic.*

Proof. We echo the proof given in [17], with slightly different notation.

If one is familiar with the fact that any two solutions to an universal mapping problem are equal, then we are done. However, we solve this case specifically:

Let $M \otimes_R N$ be the tensor product of M and N. Then suppose there exists another tensor product X which also solves the universal mapping problem. I.e. for every abelian group G we have the following two diagrams:

$$\begin{tikzcd}
M \times N \arrow[rr, "h"] \arrow[dr, "f"] & & M \otimes_R N \arrow[dl, "\bar{f}"] \\
& G &
\end{tikzcd}$$

$$\begin{tikzcd}
M \times N \arrow[rr, "h'"] \arrow[dr, "f'"] & & X \arrow[dl, "\bar{f}'"] \\
& G &
\end{tikzcd}$$

We wish to show that the abelian groups $M \otimes_R N$ and X are equal. To do this, we construct two homomorphisms between $M \otimes_R N$ and X:

$$\begin{tikzcd}
M \times N \arrow[rr, "h"] \arrow[dr, "h'"] & & M \otimes_R N \arrow[dl, shift left, "k"] \\
& X \arrow[ur, shift left, "k'"] &
\end{tikzcd}$$

We have constructed k and k' such that $k'h' = h$ and $h' = kh$. These exist given the fact that both X and $M \otimes_R N$ solve the universal mapping problem, and are themselves abelian groups. Given the following diagram:

$$\begin{tikzcd}
M \times N \arrow[rr, "h"] \arrow[dr, "h"] & & M \otimes_R N \arrow[dl, dashed, "x"] \\
& M \otimes_R N &
\end{tikzcd}$$

we can set x equal to the identity to solve the diagram. However, we can also set $x = k'k$, since $k'kh = k'h' = h$. Hence $k'k = 1$, since the homomorphism x is unique. Similarly, we can construct a diagram with X in place of $M \otimes_R N$, and get that $1_X = kk'$. Hence $M \otimes_R N$ and X are isomorphic.

□

3.5 Of homomorphisms

Before continuing, we quickly introduce tensor products of homomorphisms. We will later use these when tensoring all parts of an exact sequence in Chapter 6.

Let $f\colon M \to M'$ be a homomorphism of left R-modules, and $g\colon N \to N'$ be a homomorphism of right R-modules. Then look at the map $M \times N \to M' \otimes_R N'$ defined by

$$(m,n) \to f(m) \otimes g(n)$$

We denote this mapping by $f \otimes g$, and call it the **tensor product of the homomorphisms f and g.** Define as before $Z(M,N)$ and $F(M,N)$. Then the induced map of $f \otimes g$ on $Z(M,N)$ induces a homomorphism from $M \otimes_R N \to M' \otimes_R N'$.

3.6 Associativity

One might wonder if the tensor product possesses desirable properties. We now proceed to show that tensor products are associative. First we need to introduce the concept of a *tri-additive* function.

Definition 3.7. Let R and S be rings (with unity). Then $f\colon A_R \times_R B_S \times_S C \to G$, G an abelian group, is a **tri-additive** function if f is linear in each variable, and

$$f(as,b,c) = f(a,sb,c)$$
$$f(a,bs,c) = f(a,b,sc)$$

Theorem 3.8. *Let M be a right R-module, N be a $R-S$-bimodule, and O a left S-module. Then both $M \otimes_R (N \otimes_S O)$ and $(M \otimes_R N) \otimes_S O$ are tensor products, and they are isomorphic.*

Proof. We give a sketch of the full proof. We wish to solve the following universal mapping problem for tri-additive functions:

$$\begin{tikzcd}
M \times N \times O \arrow[rr, "h"] \arrow[dr, "f"] & & T \arrow[dl, "\bar{f}"] \\
& G &
\end{tikzcd}$$

We claim that there are in fact two solutions to this problem: $T = M \otimes_R (N \otimes_S O)$ and $T = (M \otimes_R N) \otimes_S O$. To see this, proceed as in the bi-additive case, but from $M \times (N \otimes_S O)$ and $(M \otimes_R N) \times O$, respectively. As indicated in the proof of Theorem 3.6, any two solutions to a universal mapping problem are isomorphic. To show it in this scenario, proceed as in the proof of Theorem 3.6, by constructing the diagram:

$$\begin{tikzcd}
M \times N \times O \arrow[rr, "h"] \arrow[dr, "h'"] & & (M \otimes_R N) \otimes_S O \arrow[dl, shift left, "k"] \\
& M \otimes_R (N \otimes_S O) \arrow[ur, shift left, "k'"] &
\end{tikzcd}$$

Then show (as before) that k' and k are isomorphisms, taking the form $a \otimes (b \otimes c) \to (a \otimes b) \otimes c$.

□

3.7 Adjoint associativity

In this section we explore a satisfying relationship between the Hom functor and the tensor product $\otimes$: namely that they are adjoint. The theorem gives way to a specific application, introduced in Corollary 3.10, showing that the dual of a tensor product of a module M with a dual N^* of a module N, is isomorphic to $\mathrm{Hom}_\Lambda(M, N)$. This relationship will be the crux of the algorithm described in Chapter 6.

Theorem 3.9. *Let A_R, ${}_RB_S$ and ${}_SC$ be (bi-)modules of rings R and S. Then*

$$\mathrm{Hom}_S(A \otimes_R B, C) \cong \mathrm{Hom}_R(A, \mathrm{Hom}_S(B, C))$$

Or put in the terminology of the previous section, $A \otimes_R$ is adjoint to $\mathrm{Hom}_S(B, -)$.

Proof. We outline a proof. We wish to map some element $f\colon A \otimes B \to C$ to some element $g\colon A \to \mathrm{Hom}(B, C)$. We define the map by $g_f(a)(b) = f(a \otimes b)$. Conversely, we define the map the opposite way by $f_g(a \otimes b) = g(a)(b)$.

To show that this is in fact an isomorphism, one follows the typical steps involved in proving that two groups are isomorphic.

□

As hinted to in the introduction, the above theorem gives rise immediately to the following important corollary.

Corollary 3.10. *Let k be a field, Λ a k-algebra, and M and N be finite-dimensional right Λ-modules. Then*

$$\mathrm{Hom}_k(M \otimes_\Lambda N^*, k) \cong \mathrm{Hom}_\Lambda(M, N)$$

and by extension

$$\mathrm{Hom}_k(M \otimes_\Lambda M^*, k) \cong \mathrm{End}_\Lambda(M).$$

Proof. From Theorem 3.9 we know that

$$\mathrm{Hom}_k(M \otimes_R N^*, k) \cong \mathrm{Hom}_\Lambda(M, \mathrm{Hom}_k(N^*, k).$$

Using that $N = N^{**}$, and since $\mathrm{Hom}_k(N^*, k) = N^{**}$, we get

$$\mathrm{Hom}_k(M \otimes_R N^*, k) \cong \mathrm{Hom}_\Lambda(M, N).$$

The second claim follows immediately setting $N = M$.

□

3.8 Exactness

In this final section of the chapter, we show that the tensor product is *right* exact.

Theorem 3.11. *Regarded as a functor, the tensor product is right exact.*

Proof. We outline a proof after [13]. We first need to prove the following claim: The $\mathrm{Hom}(X, -)$ functor is left exact. That is, if the following sequence of Λ-modules is exact

$$\begin{tikzcd} 0 \arrow[r] & A \arrow[r, "f"] & B \arrow[r, "g"] & C \arrow[r] & 0, \end{tikzcd}$$

then the sequence

$$\begin{tikzcd} 0 \arrow[r] & \mathrm{Hom}(X, A) \arrow[r, "{(f,-)}"] & \mathrm{Hom}(X, B) \arrow[r, "{(g,-)}"] & \mathrm{Hom}(X, C) \end{tikzcd}$$

is exact. Here $(x, -)$ means composition of the target function with x on the left. Suppose $x_b \in \mathrm{Hom}(X, B)$ such that $(g, x_b) = 0$. Then for every $y \in X$ and $x_b(y) \in B$, since f is mono, there exists an a_y such that $f(a_y) = x_b(y)$. Define then $x_a \in \mathrm{Hom}_\Lambda(X, A)$ by $y \mapsto a_y$. We need to show that x_a is a homomorphism: Let $y, z \in X$. Then $x_b(y+z) = x_b(y)+x_b(z)$, since x_b is a homomorphism. As f is mono, there is a_y and a_z such that $f(a_y+a_z) = f(a_y)+f(a_z) = x_b(y)+x_b(z) = x_b(y+z)$. Thus $x_a(y+z) = x(a_y)+x(a_z)$. This implies that $x_b = fx_a$, and since $(g, x_b) = 0$, then $(g,(f, x_a)) = 0$. Thus $\ker(g, -) \subseteq \mathrm{Im}(f, -)$. Or in other words: for any member x_b of $\ker(g, -)$, we can produce a unique element x_a in $\mathrm{Hom}(X, A)$ mapping to x_b. Importantly, this also shows that $(f, -)$ is a monomorphism. The diagram below illustrates the steps above:

$$\begin{tikzcd} & X \arrow[dl, "x_a"'] \arrow[d, "x_b"] & \\ A \arrow[r, "f"] & B \arrow[r, "g"] & C \end{tikzcd}$$

We easily get the other direction, $\mathrm{Im}(f, -) \subseteq \ker(g, -)$, since $(g, -)(f, -)(x_a) = (g, -)(f, x_a) = g(f(x_a) = (gf)(x_a) = 0$.

Getting back to our original question, we can now apply [13, Theorem 4.0.1] which states that two adjoint functors preserve (but reverse) exactness. Since the tensor product is the adjoint of the left-exact Hom-functor, as we know from Theorem 3.9, it is itself right-exact.

□

Chapter 4

Computational Complexity

In this chapter we will introduce the basic principles of analysing the runtime of algorithms. We will start with analysing what we define as the *growth of functions*. We will then introduce the concepts of *best-case*, *average-case* and *worst-case*. Then we will introduce asymptotic notation, which provides a toolset which greatly simplifies the analysis and reporting of an algorithm's complexity. As we will be working with a probabilistic algorithm in a later chapter, we will also briefly discuss the theory behind such algorithms. Finally we will look at complexity classes, in particular those that are pertinent to the algorithms discussed later.

We will generally avoid diving to deep into concepts in theoretical computer science, such as Turing machines. This will simplify our discussion significantly, at the cost of some rigor. For a more thorough introduction to the topics, the reader is referred to [7], [1] and [19].

Much of the theory presented here follows [7, Chapters 3 and 36].

4.1 Functions and their runtime

In the broadest of terms, our goal in this entire chapter is to be able to answer the following question:

> Given a function f, and input I of size n to the function, how long time do we expect the function f to run before returning an answer?

Note that we will continue using the words function and algorithm interchangeably throughout this chapter. To answer the question above, we need to define what we mean by input of size n, and what we mean by *time*. We will denote the time by $T(n)$. We illustrate with two examples.

Example 4.1. Let $f(n) = \prod_{i=1}^{n^2} i$. In this scenario, the size of the input is n. The number of calculations of the function is the number of multiplications in the product, which is n^2. Hence we say that the running time of the algorithm is proportional to n^2. In mathematical terms, we can say that $T(n) = c \cdot n^2$. Here c

is some constant, which in this particular case is proportional to the time it takes to carry out one multiplication.

Example 4.2. Let BubbleSort be the algorithm which sorts a list of numbers using the sorting algorithm bubble sort. A simple pseudocode implementation of the algorithm is

```
function BubbleSort($I$: List)
    $n \leftarrow I.Length$
    for $i = 0 \to n$ do
        for $j = 0 \to n$ do
            if $I(i+1) < I(i)$ then Swap($I, i, i+1$)
```

The size of the input here is the length of the list I. The algorithm consists of two nested loops, both of which run n iterations. Thus the total iteration count will be n^2. The function Swap simply swaps two elements in the list in-place, and runs in constant time (i.e. it is not dependent on n). Hence we again expected the running time of the algorithm to be $T(n) = c \cdot n^2$. The constant c depends here on both the Swap function, and the if-statement which compares the entries.

In the light of the above examples, we define the size n as any variable(s) in the input of I which directly affects the running time of the function f. We define $T(n)$ as the expected time it would take to run all the computations in f —under the assumption that f can be run on some computer which can perform basic mathematical operations, and store results, in constant time.

Going back to the example of BubbleSort, we see that the running time $T(n)$ depends on two factors: c and n^2. c is a constant, and does not change depending on the input. However it does depend on the implementation. For any given problem, and algorithm for that problem, there can be varying quality in the actual implementation encountered in practice. This will affect the constant c, but not the variable n. Hence, when we talk about the running time of an algorithm, what we are really interested in, is the *growth of the function f* depending on the input n. When inputs get large, which they tend to do, the variable n will at some point dominate the constant c.

4.2 Asymptotic notation

We now introduce three different asymptotic notations, which will simplify our expressions when finding $T(n)$.

Let $f(n)$ be a function of a variable n. We denote by $\Theta(f(n))$ all functions which grow at the same rate as $f(n)$. In other words, all functions which up to a constant grow no faster, and up to a constant no slower than $f(n)$. Formally, we write

$$\Theta(f(n)) = \{g(n) \mid \exists c_1, c_2, n_0 \text{ such that } c_1 f(n) \leq g(n) \leq c_2 f(n) \text{ when } n \geq n_0\}.$$

Example 4.3. Let $f(n) = 3n^2 + 2n + 1$. Then $f(n) = \Theta(n^2)$, since we can find constants c_1, c_2 and n_0 such that the above condition holds. For instance, let

$c_1 = 2$ and $c_2 = 4$. For $n_0 \approx 2.4$, we will find that $2n^2 \leq f(n) \leq 4n^2$. However, $f(n) \neq \Theta(n^3)$, as it would be impossible to find a c_1 and n_0 such that $f(n)$ is larger than $c_1 n^3$ for all $n \geq n_0$. Any term with n^3 simply grows too fast for n^2.

We denote by $\mathcal{O}(f(n))$ all functions which grow no faster than $f(n)$, up to a constant. Formally, we write

$$\mathcal{O}(f(n)) = \{g(n) \mid \exists c, n_0 \text{ such that } 0 \leq g(n) \leq cf(n) \text{ when } n \geq n_0\}.$$

Example 4.4. Let $f(n) = 3n^2 + 2n + 1$. Then $f(n) = \mathcal{O}(n^2)$. Selecting the constant $c = 4$, as in the example above we see that for $n \approx 2.4$, $4n^2$ will be an upper bound for $f(n)$. Here however, we also have that $f(n) = \mathcal{O}(n^3)$, as indeed any constant c will make cn^3 outgrow $f(n)$ and become an upper bound. Conversely, $f(n) \neq O(n)$, as the term $3n^2$ in $f(n)$ always will outgrow $c \cdot n$.

We denote by $\Omega(f(n))$ all functions which grow at least as fast as $f(n)$, up to a constant. Formally, we write

$$\Omega(f(n)) = \{g(n) \mid \exists c, n_0 \text{ such that } 0 \leq cf(n) \leq g(n) \text{ when } n \geq n_0\}.$$

Example 4.5. Let $f(n) = 3n^2 + 2n + 1$. Then $f(n) = \Omega(n^2)$. Selecting the constant $c = 2$, as in the first example above, we see that $f(n) \geq 2n^2$ for all n. Analogous to the second example, we have that $f(n) = \Omega(n)$ and $f(n) \neq \Omega(n^3)$.

We say that Ω provides an **asymptotically lower bound**, $\mathcal{O}$ provides an **asymptotically upper bound** and that Θ provides an **asymptotically tight bound**. Ideally, we would like a tight bound when we analyse algorithms. However, it is sometimes easier to provide just an upper bound, which will prove useful for us.

Example 4.6. Let $f(n) = g(n) + h(n)$, where $g(n) = \mathcal{O}(n^3)$ and $h(n) = \mathcal{O}(n^2)$. Then $f(n) = \mathcal{O}(n^3)$. This is clear, as an upper bound for $f(n)$ must simultaneously be an upper bound for both $g(n)$ and $h(n)$. But the greatest of the upper bounds for $g(n)$ and $h(n)$ is naturally an upper bound (up to a constant) for both, and hence (up to a constant) the sum of both.

The utility of the example above is that we can merge all terms in the runtime analysis of an algorithm into one upper bound, greatly simplifying our analysis. For complex algorithms consisting of multiple steps, this makes our life tremendously easier.

4.3 Best, average and worst cases

When analysing algorithms, it is sometimes worthwhile to look at the **best possible runtime**, **worst possible runtime** and the **average runtime**. For some algorithms, there may be a large discrepancy between the best possible input to the algorithm, and the worst possible input. We will primarily look at average and worst cases for the algorithms we introduce later. While best cases are interesting in their own right, we want to know whether or not the algorithm is **expected to** or **potentially can** complete within some bound time.

The perhaps canonical example of an algorithm with different average and best runtime is QuickSort, which has average time complexity $\mathcal{O}(n \log n)$, but worst time complexity $\mathcal{O}(n^2)$[17, p. 170-190].

One interesting example is LibrarySort, which has best time complexity $\mathcal{O}(n)$, average time complexity $\mathcal{O}(n \log n)$, and worst time complexity $\mathcal{O}(n^2)$[11].

4.4 Probabilistic algorithms

In this section we provide the building blocks for analysing probabilistic, or randomised, algorithms. This will be the basis for our analysis of the probabilistic decomposition algorithm in Chapter 8. The reader is expected to be familiar with basic probability theory.

Randomised algorithms are algorithms which not only depend on it's input, but also on some randomised procedure. Typically this randomised procedure is a random-number generator. We illustrate with an example.

Example 4.7. Let IsComposite be a randomised algorithm for checking whether or not an integer is a compound:

```
function IsComposite(n: Integer, k: Integer)
    for $i = 0 \to k$ do
        $x \leftarrow$ RandomNumberBetween$(2, n)$
        if Mod$(n, x) = 0$ then return True
    return Null
```

Note that if the algorithm does not find a divisor of n, it returns inconclusively. This is not atypical of randomised algorithms. The algorithm also asks for an input k which determines the number of iterations to run.

The worst case of this algorithm is $\Theta(n)$, and the best case is $\Theta(1)$. The probability of picking a divisor of any number x is proportional to the number of divisors the number has. Let $\sigma_0(n)$ be the number of divisors of the integer n. Then the probability of picking a random number between 0 and n which is a divisor of n is $\sigma_0(n)/n$. The probability of picking k wrong elements in a row is $\left(1 - \frac{\sigma_0(n)}{n}\right)^k$. Hence the probability of successfully returning true is $1 - \left(1 - \frac{\sigma_0(n)}{n}\right)^k$. The expected number of runs, were the loop to run indefinitely, is precisely the reciprocal of this value. The average case is consequently a bit tricky to analyse, and we will omit the full analysis. It is however clearly dependent on $\sigma_0(n)$.

In the light of the above analysis, we define a new property. We let **the chance of success** of a randomised algorithm to be the probability that it returns *conclusively.* This quantity *can* be 1 in certain scenarios, as some randomised algorithms are not intrinsically probabilistic. They instead use random-number generation for tasks like avoiding input-bias in scenarios where certain inputs can be significantly slower than the average. One example is the RandomizedQuicksort algorithm, which runs like a normal QuickSort after it has randomly permuted the input list. This is done to avoid scenarios where worst-case inputs occur frequently in the input data.

As with regular algorithms, we define $T(n)$ to be the expected time it takes the algorithm to complete. Sometimes, we may not be satisfied with the the chance of success $\mathfrak{p}$ of a given algorithm. However, we can simply increase the chance of success to a desired threshold (often $\frac{1}{2}$) by running the algorithm multiple times. This will be explored in Chapter 8.

4.5 Complexity classes

In this final section we briefly visit complexity classes. This allows us to categorise the algorithms introduced. We will only introduce the relevant classes.

4.5.1 The complexity class P

We define the complexity class **P** as all problems which admit a **polynomial-time** algorithm. In other words, any algorithm f in **P** has an algorithm which grows with some polynomial: $f = \Theta(n^k)$ for some constant k. Informally, we say that **P** is the class of all tractable problems. Problems in **P** include sorting, deciding whether or not a number is prime and the decision version of linear programming.

4.5.2 The complexity class NP

We define the complexity class **NP** as all problems for which solutions can be **verified** in polynomial time. Suppose we are given an algorithm f, some input I to f, and some alleged output O. If f is in **NP**, there is a procedure which allows us to verify, in polynomial-time, whether or not O is the correct, or valid, output of f given the input I.

Take for instance the problem to factor an integer n into it's divisors: $n = p_1 \cdot \ldots \cdot p_n$. Solving this problem is hard. The best current algorithms are not polynomial-time[4]. However, verifying a solution is very simple: Given an alleged factorisation $\{n_1, n_2, \ldots, n_k\}$ of n, we simply multiply the numbers together. If the result equals n, the factorisation is indeed valid.

4.5.3 The complexity class NP-complete

To introduce the class of **NP-complete** problems, we quickly have to discuss reducibility.

We say that a problem P is **reducible** to another problem Q, if there exists a procedure to translate any instance of the problem P into an instance of the problem Q, and any solution of the problem Q back to a solution of the problem

P. If the reduction procedure is polynomial-time, we say that P is **polynomial-time reducible** to Q.

We define the complexity class **NP-complete**, informally, as all problems which are at least as hard as all other problems in **NP**. By this we mean that for any **NP-complete** problem P, there exists a polynomial-time reduction from any problem in **NP** to P.

We note that we have the following relationship

$$\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{NP\text{-}complete}.$$

4.5.4 The complexity class PP

We define the complexity class **PP** as all decision problems which run in polynomial time with a probabilistic algorithm with an error probability of less than 1/2. In other words, all problems which admit some probabilistic algorithm which produces a definitive answer with at least 1/2 probability.

4.5.5 The complexity class ZPP

We define the complexity class **ZPP** as all problems which admits a probabilistic algorithm which:

(i) Always runs in polynomial time.

(ii) Always outputs YES, NO or UNSURE.

(iii) Always returns UNSURE or the correct answer.

(iv) Always returns UNSURE with probability at most 1/2.

Informally, we say that **ZPP** contains all algorithms for which there exists a tractable algorithm which with reasonable probability returns the correct, and never the wrong answer.

We finally note that we have the relationship

$$\mathbf{ZPP} \subseteq \mathbf{PP}.$$

4.5.6 The complexity classes PSPACE and EXPSPACE(-complete)

We define the complexity class **PSPACE** as all problems which can be solved using only a polynomial amount of space. Note that the runtime might still be exponentially slow, we are just not allowed to use more than polynomial space.

Similarly, we define the complexity class **EXPSPACE** as all problems which can be solved using an exponential amount of space. The **EXPSPACE**-complete problems are those which are in **EXPSPACE**, and to which there exists a polynomial-time reduction algorithm for every problem in **EXPSPACE**.

We have the relationships

$$\mathbf{P} \subseteq \mathbf{NP} \subseteq \textbf{NP-complete} \subset \mathbf{PSPACE} \subset \mathbf{EXPSPACE} \subseteq \textbf{EXPSPACE-complete}.$$

and

$$\mathbf{ZPP} \subseteq \mathbf{PP} \subset \mathbf{PSPACE}.$$

Chapter 5

Linear Algebra Algorithm

In this chapter we will be introducing the first algorithm for calculating the set of homomorphisms between two modules M, N over a finite-dimensional path algebra. We will be referring to this algorithm as the Linear Algebra Algorithm.

The algorithm has an implementation in QPA[20].

5.1 Overview

In this section we will provide an overview of how the algorithm works, arriving at a piece of pseudocode.

This algorithm is the most direct algorithm of the four discussed in this thesis. The input of the algorithm is two representations R_M, R_N (playing the role of M and N). We wish to find the set of homomorphisms $\mathrm{Hom}_\Lambda(M, N)$. For every pair of connected pair of vertices v_i, v_j, connected by an arrow α, we construct the following diagram:

$$\begin{tikzcd}
V(i) \arrow[r, "f_\alpha"] \arrow[d, "g_i"'] & V(j) \arrow[d, "g_j"'] \\
V'(i) \arrow[r, "h_\alpha"] & V'(j)
\end{tikzcd}$$

The maps $f_\alpha : V(i) \to V(j)$ and $h_\alpha : V'(i) \to V'(j)$ are given in the representations R_M, R_N as matrices. Any homomorphism between R_M and R_N must provide matrices g_i and g_j for every connected v_i and v_j such that the above diagram commutes. That is

$$h_\alpha g_i = g_j f_\alpha.$$

This equation gives us a number of constraints that needs to be satisfied for any v_i, v_j, α, which can be written as a matrix equation

$$g_\alpha X_\alpha = 0,$$

where g_α is a row vector containing the unknown elements of g_i and g_j, and X_α is a matrix containing known elements from f_α and h_α. The exact shape of this matrix will be shown in 5.2.

For every arrow α and pair of vertices v_i, v_j we can create a matrix X_α. This gives us a set of constraints all homomorphisms between R_M and R_N have to adhere to. Combining all X_α into one large matrix in a specific manner (also to be detailed in 5.2) gives us a matrix X whose linear equations encode the behaviour of all homomorphisms between R_M and R_N. Finally, these linear equations are used to create a basis for $\mathrm{Hom}_\Lambda(M, N)$.

Before we state the pseudocode of the algorithm, we note that we can reduce the number of calculations slightly, by only working with the *support* of R_M and R_N. Algorithm 1 contains the pseudocode of the algorithm.

Algorithm 1 Homomorphism Ring by Linear Algebra

```
 1: procedure HomomorphismRingOfRepresentations($R_M, R_N$)
 2:     $arrows \leftarrow$ Arrows($R_M$)
 3:     $Alg_M \leftarrow$ AlgebraOfRepresentation($R_M$)
 4:     $Alg_N \leftarrow$ AlgebraOfRepresentation($R_N$)
 5:     if $Alg_M \neq Alg_N$ then return fail
 6:     $Support_M \leftarrow$ FindSupport($R_M$)
 7:     $Support_N \leftarrow$ FindSupport($R_N$)
 8:     $numRows, numCols \leftarrow$ FindXDimensions($Support_M, Support_N$)
 9:     $X \leftarrow$ NullMat($numRows, numCols$)            ▷ Creates an empty matrix
10:     $X \leftarrow$ CreateLinearEquationsFromMats($X, Mats_M, Mats_N$)
11:     return CreateMapsFromLinearEquations($R_M, R_N, X$)
```

Some of the intermediate functions will be described in the following chapters.

5.2 Theory

In this section we dive deeper into the algorithm details. Throughout the section, we are working with representations over a finite-dimensional path algebra $\Lambda = k\Gamma / \langle \rho \rangle$.

5.2.1 Motivation

The motivation for this algorithm comes directly from the constraints put on any homomorphism between two representations R_M and R_N. As noted, a homomorphism between two such representations must for every pair of vertices $v_i, v_j \in \Gamma_0$ connected by an arrow $\alpha \in \Gamma_1$ make sure the following diagram commutes:

$$\begin{tikzcd}
V(i) \arrow[r, "f_\alpha"] \arrow[d, "g_i"'] & V(j) \arrow[d, "g_j"'] \\
V'(i) \arrow[r, "h_\alpha"] & V'(j)
\end{tikzcd}$$

Example 5.1. Let Γ be the quiver:

$$\begin{tikzcd}
\Gamma\colon\ v_1 \arrow[r, "\alpha"] & v_2 \arrow[r, "\beta"] & v_3
\end{tikzcd}$$

and let $\Lambda = k\Gamma$. A basis for Λ is thus $\{v_1, v_2, v_3, \alpha, \beta, \alpha\beta\}$. Suppose then we get the two representations

$$\begin{tikzcd}
R_M\colon\ k \arrow[r, "\begin{pmatrix}1 & 1\end{pmatrix}"] & k^2 \arrow[r, "\begin{pmatrix}1 & 0\\ 1 & 1\end{pmatrix}"] & k^2
\end{tikzcd}$$

$$\begin{tikzcd}
R_N\colon\ k \arrow[r, "1"] & k \arrow[r, "\begin{pmatrix}1 & 1\end{pmatrix}"] & k^2
\end{tikzcd}$$

We can find a homomorphism between the two representations by constructing the following diagram:

$$\begin{tikzcd}
k \arrow[r, "\begin{pmatrix}1 & 1\end{pmatrix}"] \arrow[d, "g_1"] & k^2 \arrow[r, "\begin{pmatrix}1 & 0\\ 1 & 1\end{pmatrix}"] \arrow[d, "g_2"] & k^2 \arrow[d, "g_3"] \\
k \arrow[r, "1"] & k \arrow[r, "\begin{pmatrix}1 & 1\end{pmatrix}"] & k^2
\end{tikzcd}$$

The homomorphism (g_1, g_2, g_3) is constrained by the commutative diagram containing the equations

$$g_1 \cdot 1 = \begin{pmatrix}1 & 1\end{pmatrix} \cdot g_2$$
$$g_2 \cdot \begin{pmatrix}1 & 1\end{pmatrix} = \begin{pmatrix}1 & 0\\ 1 & 1\end{pmatrix} \cdot g_3$$

We can easily give an example of a homomorphism satisfying the equations by inspection here: Let $g_1 = 1$ and let $g_2 = \begin{pmatrix}1 & 0\end{pmatrix}$. Then

$$\begin{pmatrix}1\\ 1\end{pmatrix} \cdot \begin{pmatrix}1 & 0\end{pmatrix} = \begin{pmatrix}1 & 0\\ 1 & 1\end{pmatrix} \cdot g_3 \tag{5.1}$$
$$\begin{pmatrix}1 & 0\\ 1 & 0\end{pmatrix} = \begin{pmatrix}1 & 0\\ 1 & 1\end{pmatrix} \cdot g_3 \tag{5.2}$$

For this equation to be commutative, we need g_3 to equal $\begin{pmatrix}1 & 0\\ 0 & 0\end{pmatrix}$ such that

$$\begin{pmatrix}1 & 0\\ 1 & 1\end{pmatrix} \cdot g_3 = \begin{pmatrix}1 & 0\\ 1 & 1\end{pmatrix} \cdot \begin{pmatrix}1 & 0\\ 0 & 0\end{pmatrix} = \begin{pmatrix}1 & 0\\ 1 & 0\end{pmatrix}.$$

The above example illustrates the procedure of directly finding the equations determining any homomorphism between two representations of Λ. To find the set of all such homomorphisms, we need to solve (generally) the equations 5.1 and 5.2. This is what the proposed algorithm does, which we will now detail.

5.2.2 Creating the matrices M_α

We are given representations R_M and R_N of a path algebra Λ. We know that we for each pair of vertices v_i, v_j connected by an arrow α, have the equation

$$g_i h_\alpha = f_\alpha g_j \tag{5.3}$$

Since we are given the maps f_α and h_α as matrices, we can also deduce the necessary shape of the maps g_i and g_j: Suppose $V(i), V(j), V'(i), V'(j)$ are vector spaces of dimensions n_i, n_j, m_i, m_j. Then f_α is a $n_i \times n_j$-matrix, and h_α is a $m_i \times m_j$-matrix. For Equation 5.3 to commute, this means that g_i needs to be a $n_i \times m_i$-matrix, and that g_j needs to be a $n_j \times m_j$-matrix. Writing out the matrices in full form, we get

$$f_\alpha = \begin{pmatrix} f_{1,1} & \cdots & f_{1,n_j} \\ \vdots & \ddots & \vdots \\ f_{n_i,1} & \cdots & f_{n_i,n_j} \end{pmatrix}$$
$$h_\alpha = \begin{pmatrix} h_{1,1} & \cdots & h_{1,m_j} \\ \vdots & \ddots & \vdots \\ h_{m_i,1} & \cdots & h_{m_i,m_j} \end{pmatrix}$$
$$g_i = \begin{pmatrix} x_{1,1} & \cdots & x_{1,m_i)} \\ \vdots & \ddots & \vdots \\ x_{n_i,1} & \cdots & x_{n_i,m_i} \end{pmatrix}$$
$$g_j = \begin{pmatrix} y_{1,1} & \cdots & y_{1,m_j} \\ \vdots & \ddots & \vdots \\ y_{n_j,1} & \cdots & y_{n_j,m_j} \end{pmatrix},$$

which yields the equation

$$\begin{pmatrix} x_{1,1} & \dots & x_{1,m_i)} \\ \vdots & \ddots & \vdots \\ x_{n_i,1} & \dots & x_{n_i,m_i} \end{pmatrix} \begin{pmatrix} h_{1,1} & \dots & h_{1,m_j} \\ \vdots & \ddots & \vdots \\ h_{m_i,1} & \dots & h_{m_i,m_j} \end{pmatrix} =$$
$$\begin{pmatrix} f_{1,1} & \dots & f_{1,n_j} \\ \vdots & \ddots & \vdots \\ f_{n_i,1} & \dots & f_{n_i,n_j} \end{pmatrix} \begin{pmatrix} y_{1,1} & \dots & y_{1,m_j} \\ \vdots & \ddots & \vdots \\ y_{n_j,1} & \dots & y_{n_j,m_j} \end{pmatrix}.$$

Writing out the equations, we end up with

$$\begin{gathered}
x_{1,1}h_{1,1} + x_{1,2}h_{2,1} + \dots + x_{1,m_i}h_{m_i,1} = f_{1,1}y_{1,1} + f_{1,2}y_{2,1} + \dots + f_{1,n_j}y_{n_j,1} \\
x_{1,1}h_{1,2} + x_{1,2}h_{2,2} + \dots + x_{1,m_i}h_{m_i,2} = f_{1,1}y_{1,2} + f_{1,2}y_{2,2} + \dots + f_{1,n_j}y_{n_j,2} \\
\vdots \\
x_{1,1}h_{1,m_j} + x_{1,2}h_{2,m_j} + \dots + x_{1,m_i}h_{m_i,m_j} = f_{1,1}y_{1,m_j} + f_{1,2}y_{2,m_j} + \dots + f_{1,n_j}y_{m_j,m_j} \\
x_{2,1}h_{1,1} + x_{2,2}h_{2,1} + \dots + x_{2,m_j}h_{m_j,1} = f_{2,1}y_{1,1} + f_{2,2}y_{2,1} + \dots + f_{1,m_j}y_{m_j,1} \\
\vdots \\
x_{2,1}h_{1,m_j} + x_{2,2}h_{2,m_j} + \dots + x_{2,m_i}h_{m_i,m_j} = f_{2,1}y_{1,m_j} + f_{2,2}y_{2,m_j} + \dots + f_{2,n_j}y_{m_j,m_j} \\
\vdots\vdots\vdots \\
x_{n_i,1}h_{1,m_j} + x_{n_i,2}h_{2,m_j} + \dots + x_{n_i,m_i}h_{m_i,m_j} = f_{n_i,1}y_{1,m_j} + f_{n_i,2}y_{2,m_j} + \dots + f_{n_i,n_j}y_{n_j,m_j}.
\end{gathered}$$

We can write the entire system in matrix form, by letting

$$g_\alpha = \left(x_{1,1}, x_{1,2}, \dots, x_{n_i,m_i}, y_{1,1}, y_{2,1}, \dots, y_{n_j,m_j}\right)$$

be our vector of unknowns. Then our matrix M_α can be written

$$M_\alpha = \begin{pmatrix} h_\alpha & 0 & \dots & 0 \\ 0 & h_\alpha & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & h_\alpha \\ f^*_{1,1} & f^*_{1,2} & \dots & f^*_{1,n_j} \\ f^*_{2,1} & f^*_{2,2} & \dots & f^*_{2,n_j} \\ \vdots & \vdots & \ddots & \vdots \\ f^*_{n_i,1} & f^*_{n_i,2} & \dots & f^*_{n_i,n_j} \end{pmatrix}.$$

Here $f^*_{i,j}$ are matrices comprised of elements of f making up the right hand side of the above equations. They can be written

$$f^*_{i,j} = \begin{pmatrix} -a_{i,j} & 0 & \dots & 0 \\ 0 & -a_{i,j} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & -a_{i,j} \end{pmatrix}.$$

For simplicity, we will sometimes write M_α using the notation

$$M_\alpha = \begin{pmatrix} H_\alpha \\ F_\alpha \end{pmatrix},$$

with H_α being the block-upper-triangular matrix with h_α as entries, and F_α being the lower portion containing the $f^*_{i,j}$ elements.

Example 5.2. Suppose we have two representations wherein two vertices v_i, v_j have the commutative diagram between them:

$$\begin{tikzcd} k^2 \arrow[r, "f_\alpha"] \arrow[d, "g_i"'] & k^3 \arrow[d, "g_j"'] \\ k^2 \arrow[r, "h_\alpha"] & k^2 \end{tikzcd}$$

where the maps are given by

$$f_\alpha = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$
$$h_\alpha = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$
$$g_i = \begin{pmatrix} x_{1,1} & x_{1,2} \\ x_{2,1} & x_{2,2} \end{pmatrix}$$
$$g_j = \begin{pmatrix} y_{1,1} & y_{1,2} \\ y_{2,1} & y_{2,2} \\ y_{3,1} & y_{3,2} \end{pmatrix}$$

Given the above inference about the shape of M_α we expect it to equal

$$M_\alpha = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ -1 & 0 & -1 & 0 \\ 0 & -1 & 0 & -1 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{pmatrix}$$

Writing out Equation 5.3 we get

$$\begin{aligned} x_{1,1} &= y_{1,1} + y_{3,1} \\ x_{2,1} &= y_{1,1} + y_{2,1} \\ x_{1,1} + x_{1,2} &= y_{1,2} + y_{3,2} \\ x_{2,1} + x_{2,2} &= y_{1,2} + y_{2,2} \end{aligned}$$

which corresponds to

$$\begin{pmatrix} x_{1,1} & x_{1,2} & x_{2,1} & x_{2,2} & y_{1,1} & y_{1,2} & y_{2,1} & y_{2,2} & y_{3,1} & y_{3,2} \end{pmatrix} \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ -1 & -1 & 0 & 0 \\ 0 & 0 & -1 & -1 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \end{pmatrix} = 0.$$

That this matrix corresponds to the equations is readily checked by performing the matrix calculation. We see that the matrix M_α is as expected (up to permutation of the columns).

Finally we quickly address what happens when we have a vertex with an arrow to itself. In this case our earlier commutative diagram can be written

$$\begin{tikzcd} V(i) \arrow[r, "f_\alpha"] \arrow[d, "g_i"'] & V(i) \arrow[d, "g_i"'] \\ V'(i) \arrow[r, "h_\alpha"] & V'(i) \end{tikzcd}$$

and the relations become

$$g_i h_\alpha = f_\alpha g_i.$$

Writing out the equations as before we get

$$\begin{aligned}
x_{1,1}h_{1,1} + x_{1,2}h_{2,1} + \cdots + x_{1,m_i}h_{m_i,1} &= f_{1,1}x_{1,1} + f_{1,2}x_{2,1} + \cdots + f_{1,n_i}x_{n_i,1}\\
x_{1,1}h_{1,2} + x_{1,2}h_{2,2} + \cdots + x_{1,m_i}h_{m_i,2} &= f_{1,1}x_{1,2} + f_{1,2}x_{2,2} + \cdots + f_{1,n_i}x_{n_i,2}\\
&\vdots\\
x_{1,1}h_{1,m_i} + x_{1,2}h_{2,m_i} + \cdots + x_{1,m_i}h_{m_i,m_i} &= f_{1,1}x_{1,m_i} + f_{1,2}x_{2,m_i} + \cdots + f_{1,n_i}x_{m_i,m_i}\\
x_{2,1}h_{1,1} + x_{2,2}h_{2,1} + \cdots + x_{2,m_i}h_{m_i,1} &= f_{2,1}x_{1,1} + f_{2,2}x_{2,1} + \cdots + f_{1,m_i}x_{m_i,1}\\
&\vdots\\
x_{2,1}h_{1,m_i} + x_{2,2}h_{2,m_i} + \cdots + x_{2,m_i}h_{m_i,m_i} &= f_{2,1}x_{1,m_i} + f_{2,2}x_{2,m_i} + \cdots + f_{2,n_i}x_{m_i,m_i}\\
&\vdots\vdots\vdots\\
x_{n_i,1}h_{1,m_i} + x_{n_i,2}h_{2,m_i} + \cdots + x_{n_i,m_i}h_{m_i,m_i} &= f_{n_i,1}x_{1,m_i} + f_{n_i,2}x_{2,m_i} + \cdots + f_{n_i,n_i}x_{n_i,m_i}.
\end{aligned}$$

Note that we know only have the $x_{i,j}$ variables. Using the dense notation from before, the resulting matrix M_α can now be written

$$M_\alpha = H_\alpha + F_\alpha.$$

5.2.3 Creating the matrix X

Suppose now that we for every arrow in α have calculated M_α. We now wish to combine these matrices into a large matrix X which simultaneously encodes all the constraints imposed on any homomorphism between R_M and R_N. This procedure is named CreateLinearEquationsFromMats (which also finds the M_α) in Algorithm 1. Before we continue in the general case, we look at a simple example to get an idea of how to combine the individual matrices.

Example 5.3. Let Γ be the quiver

$$\begin{tikzcd} \Gamma\colon & v_1 \arrow[loop left, "\alpha"] \arrow[r, "\beta"] & v_2 \end{tikzcd}$$

and let R_M and R_N be the representations

$$\begin{tikzcd} R_M\colon & k \arrow[loop left, "d_1"] \arrow[r, "\begin{pmatrix} f_1 & f_2 \end{pmatrix}"] & k^2 \end{tikzcd}$$

$$\begin{tikzcd} R_N\colon & k^2 \arrow[loop left, "\begin{pmatrix} e_1 & e_2 \\ e_3 & e_4 \end{pmatrix}"] \arrow[r, "\begin{pmatrix} h_1 & h_2 \\ h_3 & h_4 \end{pmatrix}"] & k^2 \end{tikzcd}$$

Writing $g_1 = \begin{pmatrix} x_{1,1} & x_{1,2} \end{pmatrix}$ and $g_2 = \begin{pmatrix} y_{1,1} & y_{1,2} \\ y_{2,1} & y_{2,2} \end{pmatrix}$, we get the following equations

$$\begin{pmatrix} x_{1,1} & x_{1,2} \end{pmatrix} \begin{pmatrix} e_1 & e_2 \\ e_3 & e_4 \end{pmatrix} = \begin{pmatrix} d_1 \end{pmatrix} \begin{pmatrix} x_{1,1} & x_{1,2} \end{pmatrix}$$
$$\begin{pmatrix} x_{1,1} & x_{1,2} \end{pmatrix} \begin{pmatrix} h_1 & h_2 \\ h_3 & h_4 \end{pmatrix} = \begin{pmatrix} f_1 & f_2 \end{pmatrix} \begin{pmatrix} y_{1,1} & y_{1,2} \\ y_{2,1} & y_{2,2} \end{pmatrix}$$

resulting in the linear equations

$$\begin{aligned} x_{1,1}e_1 + x_{1,2}e_3 &= d_1 x_{1,1} \\ x_{1,1}e_2 + x_{1,2}e_4 &= d_1 x_{1,2} \\ x_{1,1}h_1 + x_{1,2}h_3 &= f_1 y_{1,1} + f_2 y_{2,1} \\ x_{1,1}h_2 + x_{1,2}h_4 &= f_1 y_{1,2} + f_2 y_{2,2} \end{aligned}$$

We can write this on matrix form as before, resulting in

$$\mathbf{x}M = (x_{1,1}, x_{1,2}, y_{1,1}, y_{1,2}, y_{2,1}, y_{2,2}) \begin{pmatrix} e_1 - d_1 & e_2 & h_1 & h_2 \\ e_3 & e_4 - d_1 & h_3 & h_4 \\ 0 & 0 & -f_1 & 0 \\ 0 & 0 & 0 & -f_1 \\ 0 & 0 & -f_2 & 0 \\ 0 & 0 & 0 & -f_2 \end{pmatrix} = 0$$

We note two things in particular about the above matrix. The first is that the matrix M_α adheres to the formula for arrows which start and end in the same vertex: $M_\alpha = H_\alpha + F_\alpha$. The second is that the shape of the last two columns are identical to M_β, as can be expected, and is located with it's top entry at the top row of X. Indeed the matrices M_γ for all arrows γ starting in the same vertex v_i will have the top entry located in the same row of X. If we employ the dense notation used earlier, we can write the above matrix in the following way:

$$\mathbf{x}X = \begin{pmatrix} H_\alpha + F_\alpha & F_\beta \\ 0 & H_\beta \end{pmatrix}$$

We can deduce two more facts about the shape of X. First, if an arrow γ were to start in vertex 2 in the example above, the "top" (herein meaning the part of the matrix containing F_γ) of M_γ would start at the third row. This is clear as the four bottom rows represent linear combinations of $y_{1,1}, y_{1,2}, y_{2,1}$ and $y_{2,2}$. Secondly, the matrix M_γ would be flipped when inserted into X. Suppose for instance that we append an arrow γ to the quiver Γ in the above example, going from 2 to 1. This would result in an additional matrix-equation being added:

$$\begin{pmatrix} f_1^* \\ f_2^* \end{pmatrix} \begin{pmatrix} x_{1,1} & x_{1,2} \end{pmatrix} = \begin{pmatrix} y_{1,1} & y_{1,2} \\ y_{2,1} & y_{2,2} \end{pmatrix} \begin{pmatrix} h_1^* & h_2^* \\ h_3^* & h_4^* \end{pmatrix}$$

resulting in the four new linear equations

$$\begin{aligned}
f_1^* x_{1,1} &= y_{1,1} h_1^* + y_{1,2} h_3^* \\
f_2^* x_{1,1} &= y_{2,1} h_1^* + y_{2,2} h_3^* \\
f_1^* x_{1,2} &= y_{1,1} h_2^* + y_{1,2} h_4^* \\
f_2^* x_{1,2} &= y_{2,1} h_2^* + y_{2,2} h_4^*
\end{aligned}$$

which gives us additional four columns in X:

$$M = \begin{pmatrix}
e_1 - d_1 & e_2 & h_1 & h_2 & -f_1^* & -f_2^* & 0 & 0 \\
e_3 & e_4 - d_1 & h_3 & h_4 & 0 & 0 & -f_1^* & -f_2^* \\
0 & 0 & -f_1 & 0 & h_1^* & 0 & h_2^* & 0 \\
0 & 0 & 0 & -f_1 & h_3^* & 0 & h_4^* & 0 \\
0 & 0 & -f_2 & 0 & 0 & h_1^* & 0 & h_2^* \\
0 & 0 & 0 & -f_2 & 0 & h_3^* & 0 & h_4^*
\end{pmatrix} = 0$$

Swapping column 6 and 7, we achieve the expected pattern. Using our convenient notation we can write

$$\begin{pmatrix} H_\alpha + F_\alpha & F_\beta & H_\gamma \\ 0 & H_\beta & F_\gamma \end{pmatrix}.$$

Temporarily using the notation v_i for vertices, and letting $\alpha_j^{v_i}$ be the jth arrow going out of v_i. Fix a total order $\prec$ on $\{v_1, v_2, \dots, e_k\}$ the vertices of Γ. We then end up with the general form for X:

$$\begin{pmatrix} \mathbf{M}_{e_1} & \mathbf{M}_{e_2} & \dots & \mathbf{M}_{e_k} \end{pmatrix}$$

where

$$\mathbf{M}_{v_i} = \begin{pmatrix} M^*_{\alpha_1^{v_i}} & M^*_{\alpha_2^{v_i}} & \dots & M^*_{\alpha_l^{v_i}} \end{pmatrix}$$

with

$$M^*_{\alpha_j^{v_i}} = \begin{cases}
\begin{pmatrix} \vdots \\ F_{\alpha_j} \\ \vdots \\ H_{\alpha_j} \end{pmatrix} & \text{If } s(\alpha_j) \prec e(\alpha_j) \\
\begin{pmatrix} \vdots \\ H_{\alpha_j} \\ \vdots \\ F_{\alpha_j} \end{pmatrix} & \text{If } e(\alpha_j) \prec s(\alpha_j) \\
\begin{pmatrix} \vdots \\ F_{\alpha_j} + H_{\alpha_j} \\ \vdots \end{pmatrix} & \text{If } s(\alpha_j) = e(\alpha_j)
\end{cases}$$

The row locations of H_{α_j} and F_{α_j} above depend on the start and end vertex of α_j and their position in the order $\prec$.

Example 5.4. Suppose we have the quiver

$$\begin{tikzcd}
\Gamma\colon & 1 \arrow[loop left, "\alpha"] \arrow[r, "\beta"] \arrow[dr, bend left, "\gamma"] & 2 \arrow[d, "\epsilon"] \\
& & 3 \arrow[ul, bend left, "\zeta"]
\end{tikzcd}$$

and that we are given two representations R_M and R_N. Let $\prec$ be the order $1 \prec 2 \prec 3$. Then independent of the representations, we can tell that X will have the following shape,

$$\begin{pmatrix} F_\alpha + H_\alpha & F_\beta & F_\gamma & H_\zeta & 0 \\ 0 & H_\beta & 0 & 0 & F_\epsilon \\ 0 & 0 & H_\gamma & F_\zeta & H_\epsilon \end{pmatrix}.$$

We are now ready to state the procedure CreateLinearEquationsFromMats.

Algorithm 2 Insert into X all F_α and H_α for all arrows α

```
function CreateLinearEquationsFromMats($X, \Gamma, Support_M, Support_N$)
    $Mats_M \leftarrow$ MatricesOfPathAlgebra($R_M$)                ▷ Get all $f_\alpha$
    $Mats_N \leftarrow$ MatricesOfPathAlgebra($R_N$)                ▷ Get all $h_\alpha$
    for vertex $i$ in $\Gamma_0$ do
        for outgoing arrow $\alpha_j$ of $i$ do
            if ArrowHasSupport($\alpha_j, Support_M, Support_N$) then
                continue
            InsertMatsFromArrow($X, \Gamma, \alpha_j, Mats_M, Mats_N$)
    return $X$
function ArrowHasSupport($alpha, Support_M, Support_N$)
    $start \leftarrow s(\alpha)$
    $end \leftarrow e(\alpha)$
    return $end \in Support_N$ and ($start \in Support_M$ or $end \in Support_N$)
```

The function InsertMatsFromArrow simply extracts F_α and H_α from $Mats_M$ and $Mats_N$ and inserts them into X at the right position as described above.

We wish to determine the total size of X. We know that we have as many rows as we have variables that needs to be determined. For every vertex $v \in \Gamma_0$, with $\dim_k V(i) = m_i$ and $\dim_k V'(i) = n_i$, we will acquire $m_i \cdot n_i$ variables. Hence, the total number of rows is

$$numRows = \sum_{i=1}^{d_V} m_i n_i$$

where d_V is the number of vertices.

For every arrow $\alpha : V(i) \to V(j)$, we get a new set of linear equations. The number of columns in the linear equation are $m_i \cdot n_j = m_j \cdot n_j$. Hence the number of columns is

$$numCols = \sum_{k=1}^{d_E} m_{s(\alpha_k)} \cdot n_{e(\alpha_k)}$$

where d_E is the number of arrows.

5.2.4 Creating $\mathrm{Hom}_\Lambda(M, N)$

The final step in the algorithm is determining a basis for the Homomorphism set $\mathrm{Hom}_\Lambda(M, N)$. This is what the function named CreateMapsFromLinearEquations does in Algorithm 1. We now proceed to outline how this function works.

We are given the matrix X above fully calculated. This matrix describes all the relations any homomorphism between R_M and R_N must obey. Consequently, finding the kernel of the matrix will give us the basis of the homomorphism set $\mathrm{Hom}_\Lambda(M, N)$ that we desire. Finally we convert the kernel into a set of (bases of) homomorphisms. First we take the row vectors of the kernel, and for each translate the row vector into a list of maps, one for each vertex $i \in G_0$. Then we pass this list of maps into a function named QuiverRepresentationHomomorphism. This function converts the list of maps into a homomorphism between R_M and R_N. It also performs some extra validation regarding the correctness of the maps. The function CreateMapsFromLinearEquations ends up being rather simple:

Algorithm 3 Creates a basis of $\mathrm{Hom}_\Lambda(M, N)$ from X

```
1: function CreateMapsFromLinearEquations($X, \Gamma_0, R_M, R_N$)
2:     $bases \leftarrow$ NullSpace($X$)
3:     $homBases \leftarrow$ EmptyList()
4:     for Each $basis$ in $bases$ do
5:         $maps \leftarrow$ EmptyList()
6:         for Each $vertex_i \in \Gamma_0$ do
7:             $maps[i] \leftarrow$ ExtractVertexMatFromBasis($basis, i, R_M, R_N$)
8:         $homomorphism \leftarrow$ QuiverRepresentationHomomorphism($R_M, R_N, maps$)
9:         Append($homBases, homomorphism$)
10:    return $homBases$
```

The functions NullSpace and EmptyList do exactly what you expect. The function ExtractVertexMatFromBasis performs the task of extracting matrices from the row vectors of the kernel. This concludes the algorithm.

5.3 Analysis

In this section we perform an analysis of the complexity of the above algorithm. We show that the majority of the runtime is due to the calculation of the null space of X. We define as before n_i to equal the dimension of the vector space $V(i)$ and m_i to equal the dimension of the vector space $V'(i)$. Let d_V equal the number of vertices in Γ and let d_E equal the number of arrows in Γ. We will also occasionally refer to *numRows* and *numCols* as the size of the matrix X.

The pseudo code of the algorithm is contained in Algorithm 1. To analyse the complexity of the algorithm, we need to analyse look at the individual functions called during the 10 lines of code.

The function Arrows is $\mathcal{O}(1)$, as it simply returns the set of arrows in Γ. So is the function AlgebraOfRepresentation for the same reason.

The function FindSupport is $\mathcal{O}(d_V)$, as it has to iterate through all the vertices to find the support of a representation.

The function FindXDimensions is $\mathcal{O}(d_V + d_E)$, as we have to iterate through all arrows of each vertex to calculate the dimension of X.

The function NullMat is a tad harder to analyse in terms of complexity, as it depends on the underlying operating systems ability to allocate memory. We can however assume that the worst case is at most $\mathcal{O}(numRows \cdot numCols)$.

The function CreateLinerEquationsFromMats is the function that creates the matrix X. There are multiple ways to go about analysing this function. The most obvious way would be to look at the pseudo code in Algorithm 2, and analyse line by line. However, we can reason about the algorithm in a different way: Essentially, the algorithm is really just a procedure for copying a number of smaller matrices —the matrices for every arrow α in R_M and R_N —into the larger matrix X in a specific order. Each entry of X receives at most two values, when $M_\alpha = F_\alpha + H_\alpha$. Thus the complexity is $\mathcal{O}(numRows \cdot numCols)$, the size of X. Note that this presupposes that we can implement the lookups of the function ArrowHasSupport in Algorithm 2, line 10, in $\mathcal{O}(1)$ —easily done with a hash map.

Finally we have the function CreateMapsFromLinearEquations. The pseudo code for this function can be found in Algorithm 3. The major cost of this function can be found at the very start, with the function NullSpace. Calculating the null space of a matrix of size $m \times n$ can be done in many different ways. Using QR-decomposition, we can perform the calculation in at most $\mathcal{O}(n^3 + mn^2)$[5], which is the complexity we will use going forward. The remainder of the algorithm takes the returned bases (at most *numCols* row vectors), and iterates over them, creating a homomorphism for every basis. The inner loop requires *numRows* operations, as it has to put every entry of the basis into it's corresponding map. The function QuiverRepresentationHomomorphism incurs a cost of $\mathcal{O}(d_V)$, as it has to iterate over each vertex in the quiver. Hence the cost of line 3-10 in the function is $\mathcal{O}(d_V numRows + numCols \cdot numRows)$.

Theorem 5.5. *The time complexity of Algorithm 1 is $\mathcal{O}(numCols^3 + numRows \cdot numCols^2 + d_V numRows)$. Inserting the expressions for numRows and numCols,* *we get the complexity*

$$\mathcal{O}\left(\left(\sum_{k=1}^{d_E} m_{s(\alpha_k)} \cdot n_{e(\alpha_k)}\right)^3 + \left(\sum_{i=1}^{d_V} m_i n_i\right) \cdot \left(\sum_{k=1}^{d_E} m_{s(\alpha_k)} \cdot n_{e(\alpha_k)}\right)^2 + d_V \left(\sum_{i=1}^{d_V} m_i n_i\right)\right)$$

Proof. Follows directly from the above argument: The most dominant term in the above analysis is the NullSpace function.

□

Chapter 6

Green-Heath-Struble Algorithm

In this chapter we introduce the Green-Heath-Struble algorithm for calculating $\mathrm{Hom}_\Lambda(M, N)$. The algorithm works by abusing the adjoint associativity relation introduced in Theorem 3.9: We can find $\mathrm{Hom}_\Lambda(M, N)$ by constructing an isomorphism to $(M \otimes N^*)^*$. Given a projective presentation of M, we can find $M \otimes N^*$ by tensoring the projective presentation with $- \otimes N^*$. Calculating (Gröbner) bases for M and N we can find $M \otimes N^*$as the cokernel of the map in the resulting tensored projective presentation, as it is an exact sequence. Hence we can find $(M \otimes N^*)^*$ by dualising.

Note that most of the examples will revolve around finding $\mathrm{Hom}(M, M) = \mathrm{End}(M)$. This is to make the examples slightly more tractable. Also our main use of the algorithm is to use $\mathrm{End}(M)$ to decompose a module.

Throughout the chapter, $k\Gamma$ will be a path algebra, I will be an admissible ideal, and Λ will be the factor $\Lambda = k\Gamma/I$.

6.1 Overview

The algorithm takes as input two modules M and N of a path algebra $\Lambda = k\Gamma/I$, given as (vertex) projective presentations

$$P_1 \xrightarrow{(\lambda_{ji})} P_0 \to M \to 0$$

$$Q_1 \xrightarrow{(\gamma_{ji})} Q_0 \to N \to 0.$$

We start out by applying Theorem 3.9 to M, which gives

$$\mathrm{Hom}_\Lambda(M, N) \cong \mathrm{Hom}_k(M \otimes_\Lambda N^*, k).$$

So if we can find $\mathrm{Hom}_k(M \otimes_\Lambda N^*, k)$, which is the dual of $M \otimes_\Lambda N^*$, then we can find $\mathrm{Hom}_\Lambda(M, N)$. Thus we have reduced our task to finding $M \otimes_\Lambda N^*$. Finding

$\mathrm{Hom}_k(M \otimes_\Lambda N^*, k)$ from $M \otimes_\Lambda N^*$ will be shown in Section 6.2.4 to be a straightforward task.

Given a vertex projective-presentation for M:

$$\oplus_{j=1}^{r} w(j)\Lambda \xrightarrow{(\lambda_{ji})} \oplus_{i=1}^{g} v(i)\Lambda \to M \to 0,$$

we tensor on the right with N^*, which gives us the new sequence

$$\oplus_{j=1}^{r} w(j)\Lambda \otimes_\Lambda N^* \xrightarrow{(\lambda_{ji})\otimes \mathrm{Id}_{N^*}} \oplus_{i=1}^{g} v(i)\Lambda \otimes_\Lambda N^* \to M \otimes_\Lambda N^* \to 0.$$

From Theorem 3.11 we know that the tensor product is right-exact, implying that the new sequence is exact. We now show that this sequence is in fact identical to the sequence

$$\oplus_{j=1}^{r} w(j)N^* \xrightarrow{(\lambda_{ji})} \oplus_{i=1}^{g} v(i)N^* \to M \otimes_\Lambda N^* \to 0.$$

Proposition 6.1. *Let ${}_\Lambda A$ be a right Λ-module, and let v be some idempotent of Λ. Then*

$$v\Lambda \otimes_\Lambda A \cong vA$$

Proof. Specifically, let $v = 1_\Lambda$. We wish to show $\Lambda \otimes_\Lambda A \cong A$. Let $\lambda \in \Lambda, a \in A$. We wish to map $\lambda \otimes a$ to some element of A. But $\lambda \otimes a = 1 \cdot \lambda \otimes a = 1 \otimes \lambda a$. Thus, the map sending $\lambda \otimes a \mapsto \lambda a$ is our desired isomorphism. By extension, this yields $v\Lambda \otimes_\Lambda A \cong vA$. □

While we definitely have complicated our matters a lot, what we now have achieved is to find a way to compute $M \otimes_\Lambda N^*$: The exact sequence is (right-)exact, so the cokernel of (λ_{ji}) is $M \otimes N^*$. And so we have further transformed our initial problem into describing (λ_{ji}). The map (λ_{ji}) is a matrix, where each row represents an element in $\oplus_{i=1}^{g} v(i)\Lambda$. So the entries of λ_{ji} has the form of some path in $v(i)\Lambda$.

The action of the map $\lambda_{ji}(n^*)$ can be described by $(\lambda_{i1}n^*, \lambda_{i2}n^*, \dots, \lambda_{in}n^*)$. Here n^* is an element of some (dual) basis of N^*. The construction of such a basis will be described in Section 6.2.1. To find the cokernel of this map, we construct a matrix D representing the map. We can then use basic linear algebra to compute the cokernel.

To construct D, we take the elements of (λ_{ji}) and calculate a matrix representation of the actions of each λ_{ji}. This involves finding the actions every arrow $p \in \Gamma_1$ exerts on the basis-elements of Λ. The construction of these matrices will be described below in Section 6.2.3. The solution space $\mathscr{D}$ of $x \cdot D^T = 0$ is by basic linear algebra isomorphic to the cokernel.

Having found the cokernel of (λ_{ji}), we now have a clear path to describing $\mathrm{End}_\Lambda(M)$, going through $M \otimes_\Lambda N^*$ and then $\mathrm{Hom}_k(M \otimes_\Lambda N^*, k)$. This will complete the construction of the endomorphism ring. The details of this procedure will be described in Section 6.2.4.

Algorithm 4 Homomorphism set by Green-Heath-Struble

```
procedure HomomorphismSetOfModules($M_{proj}, N_{proj}$)
    $(\lambda_{ji}) \leftarrow M_{proj}.maps[0]$
    $D \leftarrow$ FindDualLambda($\lambda_{ji}, N_{proj}$)
    $nullSpace \leftarrow$ NullSpace($D^T$)
    return DualizeAndConvertToHomBasis($nullSpace$)
function FindDualLambda($\lambda_{ji}, N$)
    $\Lambda \leftarrow$ Algebra($N$)
    $\mathcal{I} \leftarrow$ GröbnerBasis($\Lambda$)
    $F \leftarrow$ CreateFMatrix($N, \mathcal{I}$)
    $\mathcal{G} \leftarrow$ GröbnerBasis($F$)
    $L \leftarrow$ CreateLMatrices($\mathcal{G}, N$)
    return CreateDMatrix($L, \lambda_{ji}$)
```

Algorithm 4 contains the pseudocode of the algorithm. The function Algebra(N) returns the underlying algebra Λ of N (and M). The function GröbnerBasis calculates a Gröbner basis. The function NullSpace(D) calculates, as before, the null space of the matrix D.

The remaining intermediate functions will be described below.

6.2 Detailed algorithm

6.2.1 Calculating a basis of N^*

In this section we wish to show a procedure that calculates a basis for N^*. We start by introducing a theorem showing us how to lift a vertex projective presentation from Λ up to $k\Gamma$:

Theorem 6.2. *Let $k\Gamma$ be a path algebra, I an admissible ideal over $k\Gamma$ and $\Lambda = k\Gamma/I$. Then let X be a right Λ-module given by a vertex projective presentation. Then X is a right $k\Gamma$-module given by a vertex projective presentation.*

Proof. We mirror the proof given in [16]. We have the following projective presentation,

$$\begin{tikzcd} \oplus_{j\in\mathcal{J}} w(j)\Lambda \arrow[r, "(\lambda_{ji})"] & \oplus_{i\in\mathcal{I}} v(i)\Lambda \arrow[r] & X \arrow[r] & 0 \end{tikzcd}$$

for some index sets $\mathcal{J}$ and $\mathcal{I}$. We wish to lift the projective presentation up to a projective presentation given by the following diagram

$$\begin{tikzcd}
 & 0 & 0 & 0 & \\
0 \arrow[r] & \oplus_{j\in\mathcal{J}} w(j)\Lambda \arrow[r, "(\lambda_{ji})"] \arrow[u] & \oplus_{i\in\mathcal{I}} v(i)\Lambda \arrow[r] \arrow[u] & X \arrow[r] \arrow[u] & 0 \\
0 \arrow[r] & \oplus_{j'\in\mathcal{J}'} w(j')k\Gamma \arrow[r, "(f_{ji})"] \arrow[u] & \oplus_{i\in\mathcal{I}} v(i)k\Gamma \arrow[r] \arrow[u] & X \arrow[r] \arrow[u] & 0 \\
0 \arrow[r] & \oplus_{i\in\mathcal{I}} v(i)I \arrow[r, equal] \arrow[u] & \oplus_{i\in\mathcal{I}} v(i)I \arrow[r] \arrow[u] & 0 \arrow[u, equal] & \\
 & 0 \arrow[u] & 0 \arrow[u] & &
\end{tikzcd}$$

In other words, our task is to find the index set $\mathcal{J}'$, a map $w' : \mathcal{J}' \to \Gamma_0$, and the map (f_{ji}). Informally, we can say that we need to find out how to map the additional structure of the $\oplus_{i\in\mathcal{I}} v(i)I$, which is factored out of the initial projective presentation.

We wish to write $\oplus_{i\in\mathcal{I}} v(i)I$ as $\oplus_{l\in\mathcal{L}} h_l k\Gamma$. Where $\mathcal{L}$ is another index set over the vertices Γ_0. The h_l are themselves elements of $\oplus_{i\in\mathcal{I}} v(i)I$. To do this, we can create a right Gröbner basis $\mathcal{H}$ for I[15][16], such that

$$I = \oplus_{h\in\mathcal{H}} hk\Gamma.$$

To find the h_l, we do the following: For each component $v(i)I$, find the corresponding h_l by running over each entry h_i of $v(i)\mathcal{H}$, and create a new element with h_i in position i, and setting the rest to 0. Take then the matrix (λ_{ji}) and let the rows be a set of elements of $\oplus_{i\in\mathcal{I}} v(i)k\Gamma$. Append to this set the elements h_l, yielding a new set F. Take this set and calculate a tip-reduced right Gröbner basis $\mathcal{F}$. Interpret the elements of the set $\mathcal{F}$ as the rows (f_{ji}). The w' and j' are found by the terminating vertices of the rows of (f_{ji}).

□

Suppose now we are given a Λ-module M. We proceed to calculate a vertex projective presentation of M:

$$\oplus_{j\in\mathcal{J}} w(j)\Lambda \xrightarrow{(\lambda_{ji})} \oplus_{i\in\mathcal{I}} v(i)\Lambda \to M \to 0,$$

e.g. by the method outlined in Section 3.3. Using Theorem 6.2 we lift the projective presentation up to a projective presentation over $k\Gamma$:

$$\oplus_{j\in\mathcal{J}'} w(j)k\Gamma \xrightarrow{(f_{ji})} \oplus_{i\in\mathcal{I}} v(i)k\Gamma \to M \to 0.$$

This procedure gives us the matrix (f_{ji}). From this matrix we get a (tip-reduced uniform) right Gröbner basis $\mathcal{F}$ of $\oplus_{j\in\mathcal{J}'} w(j)k\Gamma$, by taking the rows of (f_{ji}) as the elements of $\mathcal{F}$. The elements in M can be viewed as the nontips of $\mathcal{F}$: Set $V = \oplus_{i\in\mathcal{I}} v(i)k\Gamma$ and $W = \oplus_{j\in\mathcal{J}'} w(j)k\Gamma$. Then given the exactness of the above sequence; $M = V/W$. But since $\mathcal{F}$ is a right Gröbner basis of W, we can write $V = \text{Nontips(W)} \oplus W$. Factoring out W, we get that Nontips(W) $\cong V/W \cong M$. Hence we can find a basis for M by finding a basis for the nontips of W through $\mathcal{F}$.

Viewing the elements in M as nontips, we get a basis $\mathcal{B}$ from $\mathcal{F}$. Dualising this basis, we end up with a basis B^* for M^*.

The above procedure corresponds to the function CreateFMatrix in Algorithm 4.

6.2.2 Calculating the actions of Λ

In this section we calculate the matrix D, describing the actions of (λ_{ji}).

As before, we are given a path algebra $k\Gamma$ over some quiver $\Gamma = (\Gamma_0, \Gamma_1)$, an (admissible) ideal I such that $\Lambda = k\Gamma/I$ is finite dimensional, and finally a (right) Λ-module M.

We first observe the following fact: Let v_i be a vertex of Γ_0, then Mv_i is a vector space. Furthermore, M can be written as a direct sum of these Mv_i: $M = \oplus_i Mv_i$. To observe this, note that under the path algebra $k\Gamma$, the v_i are orthogonal idempotents for all i, and $1_\Lambda = \sum_i v_i$. By extension we can write $M^* = \oplus_i v_i M^*$ where M^* is the dual of M.

Our goal now is to describe the actions of each path of Γ as matrices. This gives us a way to find a matrix description of (λ_{ji}). Let $M_i = Mv_i$ for some vertex v_i, with dimension d_i. Fix a k-basis $\{m^i_j\}_{j=1}^{d_i}$, and let σ from v_i to v_j be an arrow of Γ_1. Then define a k-linear map $L_M(\sigma) : M \to M$ defined by $L_M(\sigma)(m) = m\sigma$. Clearly the image of $L_M(\sigma)$ is contained in M_j. So $L_M(\sigma) : M_i \to M_j$. We can describe the actions of $L_M(\sigma)$ by where it sends a basis element m^i_s:

$$L_M(\sigma)(m^i_s) = \sum_{t=1}^{d_j} \beta_{st} m^j_t$$

Hence we can completely describe the actions of σ by the $d_i \times d_j$-matrix (β_{st}). By extension, for a path $x = \sigma_1\sigma_2 \ldots \sigma_n$ we can describe the actions of x by the matrix

$$L_M(\sigma_1)L_M(\sigma_2)\ldots L_M(\sigma_n).$$

We have now found a way to represent the actions of (λ_{ji}) by a matrix: For every entry λ_{ji}, replace path-elements x of λ_{ji} by the matrix $L_M(x)$. This gives the desired matrix.

Finally, we show the relationship between the action of an arrow σ over M and the action of the same arrow over M^*.

Theorem 6.3. *Let $p = \sigma_0\sigma_1 \ldots \sigma_q$ be a path in Γ. Given the action of p over M, defined by*

$$L_M(\sigma_1)L_M(\sigma_2)\ldots L_M(\sigma_q).$$

Then the action of p over M^* is given by the transpositions of the matrices:

$$L_M(\sigma_q)^T \dots L_M(\sigma_2)^T L_M(\sigma_1)^T$$

Proof. We are given as starting point a basis $\{m^i_j\}_{j=1}^{d_i}$ for every $M_i = Mv_i$. We dualise this basis, to obtain $\{(m^i_j)^*\}_{j=1}^{d_i}$ as a basis for $M_i^* = v_iM^*$. Recall that $(m^i_j)^*$ is the map acting on the basis elements of M_i by

$$(m^i_j)^* \left(\sum_{k=0}^{d_i} \beta_{ki} m^i_k \right) = \beta_{ji}.$$

We now wish to describe the **left-**action of an arrow $\sigma\colon v_i \to v_j$ on a basis element $(m^j_t)^*$. In other words, we wish to find the matrix $L_{M^*}(\sigma) : M_j^* \to M_i^*$. But the action of $\sigma \cdot (m^j_t)^*$ can be described by how it acts on a basis element m^i_s:

$$(\sigma \cdot (m^j_t)^*)(m^i_s) = (m^j_t)^*(m^i_s\sigma)$$

From before, we have that

$$m^i_s\sigma = \sum_{k=1}^{d_j} \beta_{sk} m^j_k$$

Which yields

$$\sigma \cdot (m^j_t)^* = \sum_{s=1}^{d_i} \beta_{ts}(m^i_s)^*$$

We see that the β_{ts} indexes act as the transposed of the β_{st}, thus $L_{M^*}(\sigma) = (L_M(\sigma))^T$.

□

6.2.3 Calculating $M \otimes_\Lambda N^*$

In this section we are going to perform the calculation of the tensor product $M \otimes_\Lambda N^*$. We do this by first finding the matrix D representing the actions of (λ_{ji}), and then finding it's cokernel which is isomorphic to $M \otimes_\Lambda N^*$.

Suppose that we have been given the exact sequence

$$\oplus_{j=1}^{r} w(j)N^* \xrightarrow{(\lambda_{ji})} \oplus_{i=1}^{g} v(i)N^* \to M \otimes_\Lambda N^* \to 0.$$

We wish to find $M \otimes_\Lambda N^*$, which we have previously shown is equivalent to finding the cokernel of $(\lambda_{ji} \otimes \mathrm{Id}_{N^*})$. We can find the cokernel by finding a matrix-representation of the actions of $(\lambda_{ji}) \otimes \mathrm{Id}_{N^*}$. Fix a basis $\mathcal{B}^* = \{n_1^*, n_2^*, \dots, n_{d^*}^*\}$ of N^*. We know we can find such a basis from the results of Section 6.2.3. For any given basis element n_s^*, our task is now to find a description of the actions of the map

$$(\lambda_{ji})(n_s^*)$$

Recall that an entry of (λ_{ji}) can be written

$$\lambda_{ji} = \sum_k \alpha_{jik} p_{jik0} p_{jik1} \dots p_{jikn}.$$

We can expand the map $(\lambda_{ji})(n_s^*)$ as

$$(\lambda_{i1} n_s^*, \lambda_{i2} n_s^*, \lambda_{i3} n_s^*, \dots, \lambda_{im} n_s^*)$$

We proceed by describing the action of each entry as we did in the previous section. Note that we have the dual L_{N^*} as our basis elements are n_s^*:

$$\begin{aligned} &L_{N^*}\left(\sum_k \alpha_{jik} p_{jik0} p_{jik1} \dots p_{jikn}\right) \\ &= \sum_k \alpha_{jik} L_{N^*}(p_{jik0}) L_{M^*}(p_{jik1}) \dots L_{N^*}(p_{jikn}) \\ &= \sum_k \alpha_{jik} L_N(p_{jikn})^T \dots L_N(p_{jik2})^T L_M(p_{jik1})^T, \end{aligned}$$

Where we in the final step have used the equivalence of the action of $L_{N^*}(x)$ on N^* with $L_N(x)^T$ on N.

The result is a matrix D which represent the actions of the map $(\lambda_{ji})(n_s^*)$. We can now use linear algebra to find $M \otimes_\Lambda N^*$: Take the linear system $x \cdot D^T = 0$. The solution space of this is the cokernel of D. But the cokernel of D is isomorphic to the cokernel of (λ_{ji}), which is isomorphic to $M \otimes_\Lambda N^*$.

The above procedure corresponds to the function CreateLMatrices in Algorithm 4. Combined with the previous chapter, we have now completed the function FindDualLambda.

6.2.4 Tying it all together

We started out by noting that if we can find $M \otimes_\Lambda N^*$ then we can find an isomorphism to $\mathrm{Hom}_\Lambda(M, N)$. We have now found $M \otimes_\Lambda N^*$, by constructing a set of isomorphisms from the exact sequence we got by tensoring the projective presentation of M by N^*. What then remains, is to explicitly write out the isomorphisms, taking us to $\mathrm{Hom}_\Lambda(M, N)$.

We start with the matrix D we constructed in the previous section. We first wish to map rows of the matrix onto elements $\sum_i v(i)N^*$. This is a fairly straight forward procedure: Every row of D has an associated n_s, and thus every column an associated pair $(n_s^*, v(i))$. Thus we can map every row in D to $\sum_i v(i)N^*$ by taking each non-zero entry, and sending it to its associated $v(i)n_s^*$.

We will now make a map from $\sum_i v(i)N^*$ to $M \otimes_\Lambda N^*$. This allows us to take a basis for the cokernel of D, and send it via $\sum_i v(i)N^*$ to $M \otimes_\Lambda N^*$. The map is constructed in the following way: We are given a basis element $v(i)n_s^* \in v(i)N^*$, for some i. We wish to map it to some element $m \otimes n^*$. Clearly we can send $n_s^* \to n_s^*$. We send $v(i)$ to b_i, the vector with $v(i)$ in the ith coordinate. Suppose that we have a basis $\{d_1, d_2, \dots, d_p\}$ of $\mathcal{D}$. We can now interpret this basis directly as a basis $\mathcal{C}$, by sending element d_i via the map defined above.

Finally, we construct an explicit map from $M \otimes_\Lambda N^*$ to $\mathrm{Hom}_\Lambda(M, N)$. By now, we have a basis for $M \otimes_\Lambda N^*$:

$$\sum_{s,t} \beta_{s,t} b_s \otimes n_t^*$$

By Theorem 3.9, we know that $(M \otimes_\Lambda N^*)^*$ is isomorphic to $\mathrm{Hom}_\Lambda(M, N)$, by the map $\Phi(c^*)(m) = \sum_i^d c^*(m \otimes n_i^*) m_i$. We wish to send an element $\left(\sum_{s,t} \beta_{s,t} b_s \otimes n_t^*\right)^*$ through this map:

$$\Phi\left(\left(\sum_{s,t} \beta_{s,t} b_s \otimes n_t^*\right)^*\right)(m) = \sum_i^d \left(\left(\sum_{s,t} \beta_{s,t} b_s \otimes n_t^*\right)^*\right)(m \otimes n_i^*) m_i.$$

We proceed to invoke a few tricks, to simplify this expression a bit. We know that the b_i generate M, since they are representations of the basis elements of $v(i)\Lambda$, and since the map from $\oplus_{i=1}^g v(i)\Lambda \to M$ is onto. So we can get a full description by substituting m with b_j:

$$\Phi\left(\left(\sum_{s,t} \beta_{s,t} b_s \otimes n_t^*\right)^*\right)(b_j) = \sum_i^d \left(\left(\sum_{s,t} \beta_{s,t} b_s \otimes n_t^*\right)^*\right)(b_j \otimes n_i^*) m_i$$

The final expression can be greatly simplified. Recall that for any basis element m_i, it's dual n_i^* is the map which preserves the ith coefficient: $n_j^*\left(\sum_j a_j m_j\right) = a_j$. Hence for the nested sum above, this means that the inner duals $\left(\sum_{s,t} \beta_{s,t} b_s \otimes n_t^*\right)^*$ only preserve the specific basis elements given by:

$$(\beta_{s,t} b_s \otimes n_t^*)^*(b_j \otimes n_i^*) = \begin{cases} \beta_{s,t}, & \text{if } s = j, i = t \\ 0 & \text{else} \end{cases}$$

This means that for a fixed basis element b_j, $s = j$ remains fixed, and we need only sum over t:

$$\sum_i^d \left(\left(\sum_{s,t} \beta_{s,t} b_s \otimes n_t^*\right)^*\right)(b_j \otimes n_i^*) m_i = \sum_t \beta_{j,t} m_t.$$

This tells us that for a basis element c^* of $(M \otimes_\Lambda N^*)^*$, we can describe the map $\Phi(c^*)$ of $\mathrm{Hom}_\Lambda(M, N)$ by the structure of c^* itself, i.e. where it sends the generators of M. This completes the construction of the endomorphism ring.

The above procedure corresponds to the function DualizeAndConvertToHomBasis in Algorithm 4.

6.3 Example

In this final section, we show a complete example for the calculation of an endomorphism ring $\mathrm{End}_\Lambda(M)$.

Example 6.4. We start with the following quiver Γ:

$$\begin{tikzcd} v_1 \arrow[loop left, "a"] \arrow[r, bend left, "b"] \arrow[r, bend right, "c"'] & v_2 \end{tikzcd}$$

Let then $\Lambda = k\Gamma/I$, where I is the ideal generated by the relations $\{a^2, ac\}$. This is in fact also a Gröbner basis $\mathcal{G}$ for I, using length lexicographic ordering. The set of nontips of Λ are

$$\mathrm{Nontips}(I) = \{v_1, v_2, a, b, c, ab\}$$

Thus $\dim_k \Lambda = 6$. Suppose now that we are given a Λ-module M, given by the projective presentation

$$v_1\Lambda \oplus v_2\Lambda \xrightarrow{(\lambda_{ji})} v_1\Lambda \oplus v_1\Lambda \to M \to 0,$$

with $(\lambda_{ji}) = \begin{pmatrix} a & 0 \\ -b & ab \end{pmatrix}$. A (tip-reduced uniform) right Gröbner basis for I can be found by calculating the nontips of I on the right by $\mathcal{G}$, yielding the set $\{a^2, a^3, ac, a^2c\}$, which we can reduce back to $\mathcal{G}$. Thus we see that $\mathcal{G}$ is a (tip-reduced uniform) right Gröbner basis for I.

We now proceed to lift (λ_{ji}) up to (f_{ji}) as described in Theorem 6.2. We get the set

$$F = \begin{pmatrix} a & 0 \\ -b & ab \\ a^2 & 0 \\ ac & 0 \\ 0 & a^2 \\ 0 & ac \end{pmatrix}$$

We order the vertices $v_1 \succ v_2$. We then tip-reduce this set by performing the following calculations:

$$\begin{aligned} (a^2, 0) - a(a, 0) &= \underline{(0, 0)} \\ (ac, 0) - c(a, 0) &= \underline{(0, 0)} \end{aligned}$$

All other elements stay put, and we get the right Gröbner basis $\mathcal{F}$ for M

$$\mathcal{F} = \begin{pmatrix} -b & ab \\ a & 0 \\ 0 & a^2 \\ 0 & ac \end{pmatrix}.$$

We can interpret the set directly as the matrix (f_{ji}). The tips of this set are $\{(-b,0),(a,0),(0,a^2),(0,ac)\}$. Viewing the elements of M as nontips, we get a basis for M by reducing via $\mathcal{F}$. In the first coordinate we have the tips $(-b,0)$ and $(a,0)$. Thus we are left with the nontips $(v_1,0)$ and $(c,0)$. In the second we have the tips $(0,a^2)$ and $(0,ac)$, and we are left with the nontips $(0,v1)$, $(0,a)$, $(0,b)$ and $(0,ab)$. Thus a vector space basis $\mathcal{B}$ for M is

$$\mathcal{B} = \{((v_1,0),(c,0),(0,v_1),(0,a),(0,b),(0,ab)\}.$$

This tells us that $\dim_k M = 6$.

We proceed by finding the actions of the arrows of G_1 in M. Or in other words, we wish to calculate the matrices $L_M(\sigma)$. To do this, we first need to split up $\mathcal{B}$ into bases for M_1 and M_2:

$$\begin{aligned} M_1 &= \{(v_1,0),(0,v_1),(0,a))\} \\ M_2 &= \{(c,0),(0,b),(0,ab)\} \end{aligned}$$

We then see how each of the elements a and b (the only ones present in (λ_{ji})) act on the basis-elements. We find $L_M(a)$ by calculating the following products:

$$\begin{aligned} (v_1,0)\cdot a &= (a,0) \overset{\mathcal{F}}{=} (0,0) \\ (0,v_1)\cdot a &= (0,a) \\ (0,a)\cdot a &= (0,a^2) \overset{\mathcal{F}}{=} (0,0) \end{aligned}$$

Hene $L_M(a) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$. Similarly, we find that $L_M(b) = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$.

We are now ready to describe the actions of λ_{ji}, by finding the matrix D:

$$D = \begin{pmatrix} L_M(a)^T & 0 \\ -L_M(b)^T & L_M(b)^T L_M(a)^T \end{pmatrix}$$
$$= \left(\begin{array}{ccc|ccc} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ -1 & 0 & -1 & 0 & 1 & 0 \end{array}\right)$$

A basis for the cokernel of D is the set

$$\mathcal{C} = \{(-1,0,1,0,0,0),(0,0,0,1,0,0),(1,0,0,0,1,0),(0,0,0,0,0,1)\}$$

The elements can be mapped to basis elements of $M \otimes_\Lambda M^*$, as described in Section 6.2.4. The correspondence between the columns of D and basis elements of $M \otimes_\Lambda M^*$ can be found in Table 6.1. The basis elements of $\mathcal{C}$ are interpreted directly as endomorphisms of $\operatorname{End}_\Lambda(M)$, describing where generators of M are being sent. For instance, the element $(-1,0,1,0,0,0)$ corresponds to

$$-(v_1,0) \otimes (v_1,0)^* + (v_1,0) \otimes (0,a)^*$$

which corresponds to the endomorphism mapping $(v_1,0) \mapsto (v_1 - a, 0)$. In particular, the third basis-element corresponds to

$$(v_1,0) \otimes (v_1,0)^* + (0,v_1) \otimes (0,v_1)^*$$

which corresponds to the identity map. This completes the construction of the endomorphism ring $\operatorname{End}_\Lambda(M)$.

Column	Element
1	$(v_1,0) \otimes (v_1,0)^*$
2	$(v_1,0) \otimes (0,v_1)^*$
3	$(v_1,0) \otimes (0,a)^*$
4	$(0,v_1) \otimes (v_1,0)^*$
5	$(0,v_1) \otimes (0,v_1)^*$
6	$(0,v_1) \otimes (0,a)^*$

Table 6.1: Correspondence between matrix D columns and elements of $M \otimes_\Lambda M^*$

6.4 Analysis

In this section we perform an analysis of the complexity of the above algorithm. We will give the complexity in terms of the size of the matrix (λ_{ji}). In Chapter 9 we will extend the complexity to be written in terms of when the input are representations R_M and R_N. As before, we let d_V denote the number of vertices, and d_E the number of arrows in the underlying quiver.

The input to the algorithm is two projective presentations M and N. The function FindDualLambda calculates the matrix D. Denote the size of the matrix (λ_{ji}) over Λ by $n' \times m'$, implying that the size of (λ^*_{ji}) is also $n' \times m'$. For every entry of the matrix (which contains elements of Λ), we have to insert an appropriate combination of L_{N^*} matrices as described above. The size of these matrices depend on the size of the underlying bases N_i. Writing $n_i = |N_i|$, the resulting size of the matrix D can be written

$$\sum_{j=1}^{r} n_j \times \sum_{i=1}^{g} n_i.$$

The function also have to find a Gröbner bases for Λ and then for the intermediate matrix F. The computational complexity for finding a Gröbner basis is hard to analyse, and will vary a lot depending on the input. We will denote this complexity by $\mathcal{G}(\Lambda)$, and leave it for further analysis in Chapter 9. The function also lifts (λ_{ji}) up to (f_{ji}) to calculate a basis for M. The primary cost of this calculation is constructing and reducing the matrix F as shown in Example 6.4. To create F, the matrix (λ_{ji}) is appended the element I in all coordinates i, as described in Section 6.2.1. The number of elements in I is at most n_i. Hence the size of this matrix is at most $n' \times m' + (\sum_{i=0}^{g} n_i) \times m$, which we denote by $n'' \times m''$. To reduce the matrix we have to calculate a Gröbner basis, and incur a cost of $\mathcal{G}(F)$.

Writing $n = \sum_{j=1}^{r} n_j$ and $m = \sum_{i=1}^{g} n_i$, we get that the complexity of FindDualLambda is $\mathcal{O}(n \cdot m + \mathcal{G}(\Lambda) + \mathcal{G}(F))$, as we have to insert $n \times m$ elements to the matrix D.

The following function NullSpace has has complexity $\mathcal{O}(m^3 + nm^2)$, as we know from Section 5.3. Note that m and n are swapped, as we transpose the matrix to find the cokernel. Finally we have the function DualizeAndConvertToHomBasis which performs the procedure outlined in Section 6.2.4. We can analyse this similar to the function CreateMapsFromLinearEquations in Section 5.3: The maximum number of bases in the kernel is m. If the function DualizeAndConvertToHomBasis checks the homomorphism for consistency, it has to iterate over each vertex and will incur a cost of $\mathcal{O}(d_V)$. Hence the complexity of CreateHomBasisFromLambdaNullSpace is $\mathcal{O}(d_V m)$. In the case where the function CreateHomBasisFromLambdaNullSpace does not check for consistency, we end up with complexity $\mathcal{O}(m)$.

Theorem 6.5. *The time complexity of Algorithm 4 is $\mathcal{O}(m^3 + nm^2 + d_V m + \mathcal{G}(\Lambda) + \mathcal{G}(F))$.*

Proof. Follows directly from the above argument. □

Chapter 7

Hom functor algorithm

In this chapter we will introduce the so-called *Hom-functor algorithm.* The algorithm works similarly to Green-Heath-Struble. The input to the algorithm is two modules M and N, given as projective presentations. The functor $\mathrm{Hom}(-, N)$ is applied to the projective presentation of M. This creates a new sequence, for which $\mathrm{Hom}_\Lambda(M, N)$ is isomorphic to the kernel of one of the maps of the sequence, which can be found.

This chapter relies on much of the material in Chapter 6, given the similarity of the algorithms.

7.1 Overview

In this section we will provide an overview of how the algorithm works, arriving at a piece of pseudocode.

The algorithm takes as input two modules M and N of a path algebra $\Lambda = k\Gamma/\langle\rho\rangle$, given as projective presentations

$$P_1 \xrightarrow{(\lambda_{ji})} P_0 \to M \to 0$$
$$Q_1 \xrightarrow{(\gamma_{ji})} Q_0 \to N \to 0.$$

We apply the $\mathrm{Hom}(-, N)$ functor to the projective presentation of M, to get the (left-)exact sequence

$$0 \to \mathrm{Hom}(M, N) \to \mathrm{Hom}(P_0, N) \xrightarrow{(\lambda^*_{ji})} \mathrm{Hom}(P_1, N).$$

This sequence gives us the relation $\ker(\lambda^*_{ji}) \cong \mathrm{Hom}_\Lambda(M, N)$. So if we can find (λ^*_{ji}) from (λ_{ji}), which is known, we can find $\mathrm{Hom}_\Lambda(M, N)$. The next step of the algorithm does exactly this, the details of which will be described below. Finally we calculate the kernel, and interpret the resulting null space in terms of homomorphisms from M to N.

Algorithm 5 Homomorphism set by Hom functor

```
1: procedure HomomorphismSetOfModules($M_{proj}, N_{proj}$)
2:     $(\lambda_{ji}) \leftarrow M_{proj}.maps[0]$
3:     $D \leftarrow$ FindDualLambda($\lambda_{ji}, N$)
4:     $nullSpace \leftarrow$ NullSpace($D$)
5:     return CreateHomBasisFromLambdaNullSpace($nullSpace$)
```

Algorithm 5 contains the pseudocode of the algorithm. Notably, we assume that the maps of the projective presentation is stored in a variable named *maps*, which is indexed from left to right. The intermediate functions will be detailed in the following sections.

7.2 Detailed algorithm

In this section we dive deeper into how the Hom functor algorithm works, and develop some necessary theory.

7.2.1 Applying Hom

We are given two modules M and N as (vertex) projective presentations:

$$P_1 \xrightarrow{(\lambda_{ji})} P_0 \to M \to 0.$$

$$Q_1 \xrightarrow{(\gamma_{ji})} Q_0 \to N \to 0.$$

We can write P_1 and P_0 on "vertex"-form, as in Chapter 6:

$$\oplus_{j=1}^{r} w(j)\Lambda \xrightarrow{(\lambda_{ji})} \oplus_{i=1}^{g} v(i)\Lambda \to M \to 0.$$

Everything so far is equivalent to the Green-Heath-Struble algorithm. Next we apply the functor $\mathrm{Hom}(-, N)$ to the projective presentation of M, to obtain the following exact sequence:

$$0 \to \mathrm{Hom}(M, N) \to \mathrm{Hom}(\oplus_{i=1}^{g} v(i)\Lambda, N) \xrightarrow{(\lambda_{ji}^{*})} \mathrm{Hom}(\oplus_{j=1}^{r} w(j)\Lambda, N).$$

We can slightly simplify the sequence, by applying a proposition similar to Proposition 6.1.

Proposition 7.1. *Let A_Λ be a right Λ-module, and let v be some idempotent of Λ. Then*

$$\mathrm{Hom}(v\Lambda, A) \cong Av$$

Proof. The proof is similar to Proposition 6.1. For an element $f \in \text{Hom}(v\Lambda, A)$, we have $f(v\lambda) = f(v)\lambda$. Clearly $f(v) \in Av$. Conversely, we can take any element $av \in Av$ and send it to the map $f_{av} : v\Lambda \to A$, defined by $f_{av}(v\lambda) = (av)v\lambda = (nv)\lambda \in A$. Noting finally that $f(v) = f(v \cdot v) = f(v) \cdot v \in Av$ we get that our map is an isomorphism. In particular, $\text{Hom}(w\Lambda, v\Lambda) \simeq v\Lambda w$ for two idempotents v and w in Λ. Furthermore, a homomorphism $f : w\Lambda \to v\Lambda$ is given as $f(m) = \lambda_f m$ for some λ_f in $v\Lambda w$. Given $w\Lambda \xrightarrow{\lambda \cdot -} v\Lambda$ for some $\lambda \in v\Lambda w$ with v, w in Λ, then $\text{Hom}_\Lambda(v\Lambda, N) \xrightarrow{\text{Hom}_\Lambda(\lambda \cdot -, N)} \text{Hom}_\Lambda(w\Lambda, N)$ is isomorphic to $Nv \xrightarrow{- \cdot \lambda} Nw$. □

Applying the above proposition, we can simplify the sequence to

$$0 \to \text{Hom}(M, N) \to \oplus_{i=1}^{g} Nv(i) \xrightarrow{(\lambda_{ji}^*)} \oplus_{j=1}^{r} Nw(j),$$

where each λ_{ji} is in $v_i \Lambda w_j$. The sequence is left-exact by Theorem 3.11. Consequently, from the basic attributes of exact sequences, we know that,

$$\text{Hom}_\Lambda(M, N) \cong \ker(\lambda_{ji}^*). \tag{7.1}$$

Note that the actions of the map (λ_{ji}^*) in the above sequence solely relies on N and $v(i), w(j)\, \forall i, j$.

7.2.2 Finding the actions of Λ

In the next step of the algorithm, the map (λ_{ji}^*) will be calculated. The way to do this is very similar to how the matrix D is found in Section 6.2.2. We are looking for a matrix, which we still will call D, describing the action of (λ_{ji}^*) on N. The first thing we note, is the shape of (λ_{ji}).

Proposition 7.2. *Suppose we are given the following exact sequences:*

$$P_1 \xrightarrow{(\lambda_{ji})} P_0 \to M \to 0$$

$$0 \to \text{Hom}(M, N) \to \text{Hom}(\oplus_{i=1}^{g} v(i)\Lambda, N) \xrightarrow{(\lambda_{ji}^*)} \text{Hom}(\oplus_{j=1}^{r} w(j)\Lambda, N)$$

$$0 \to \text{Hom}(M, N) \to \oplus_{i=1}^{g} Nv(i) \xrightarrow{(\lambda_{ji}^*)} \oplus_{j=1}^{r} Nw(j)$$

where the latter two are equivalent by Proposition 7.1. Then (λ_{ji}^*) is the transpose of (λ_{ji}). However, the actions of the paths $p_{i,j}$ differ in the application on elements of N instead of M.

Proof. We have the diagram

$$\begin{tikzcd}
0 \arrow[r] & \oplus_{j=1}^{r} w(j)\Lambda \arrow[r, "(\lambda_{ji})"] \arrow[d, "\text{Hom}(-,N)"'] & \oplus_{i=1}^{g} v(i)\Lambda \arrow[r] \arrow[d, "\text{Hom}(-,N)"'] & M \arrow[r] \arrow[d, "\text{Hom}(-,N)"'] & 0 \\
 & \oplus_{j=1}^{r} Nw(j) & \oplus_{i=1}^{g} Nv(i) \arrow[l, "(\lambda_{ji}^*)"'] & \text{Hom}_\Lambda(M, N) \arrow[l] & 0. \arrow[l]
\end{tikzcd}$$

The rows of (λ_{ji}) can be interpreted as elements of $\oplus_{i=1}^{g} v(i)\Lambda$, the columns as elements of $\oplus_{j=1}^{r} w(j)$. It should be clear that the desired form of (λ_{ji}^*) transposes the matrix size of (λ_{ji}): We would like the rows of (λ_{ji}^*) to be interpreted as elements of $\oplus_{j=1}^{r} Nw(j)$ and the columns as elements of $\oplus_{i=1}^{g} Nv(i)$. Defining (λ_{ji}^*) by

$$(\lambda_{ji}^*)((nv_1, \ldots, nv_g)) = n((v_1, \ldots, v_g))(\lambda_{ji}) \in \oplus_{j=1}^{r} Nw(j)$$

we get the desired map (λ_{ji}^*).

□

As in Section 6.2.3, we now wish to describe the action of the matrix D over k, i.e. as a k-linear map. The full details can be found in that section, so we simply restate the result: Let M be a module over Λ. Fix bases $\{m_j^i\}_{j=1}^{d_i}$, with each m_j^i being elements of Λ. For every arrrow $\sigma \in G_1$, we define $L_M(\sigma) : M \to M$ to be the map defined by $L_M(\sigma)(m) = m\sigma$. We can describe the actions of $L_M(\sigma)$ by where it sends a basis element m_s^i:

$$L_M(\sigma)(m_s^i) = \sum_{t=1}^{d_j} \beta_{st} m_t^j.$$

Hence, to create the action of the matrix D over k, we simply take the matrix (λ_{ji}) as defined above, and for each entry $p_{i,j}$ of (λ_{ji}) we calculate $L_M(p_{i,j})$.

Example 7.3. We follow [16, Example 4.1]. The quiver given is

$$\begin{tikzcd}
\Gamma: & v_1 \arrow[loop left, "\alpha"] \arrow[rr, "\beta"] \arrow[dr, bend right, "\zeta"'] & & v_2 \arrow[dl, bend left, "\gamma"] \\
& & v_3 &
\end{tikzcd}$$

with $\Lambda = k\Gamma/I$, where I is the ideal generated by the relations $\{\alpha^3, \alpha^2\zeta - \beta\gamma\}$. A Gröbner basis for I is $\{\alpha^3, \alpha^2\zeta - \beta\gamma, \alpha\beta\gamma\}$, using length left lexicographic ordering. The set of nontips of I is

$$\{v_1, v_2, v_3, \alpha, \beta, \gamma, \zeta, \alpha^2, \alpha\beta, \alpha\zeta, \beta\gamma, \alpha^2\beta\}.$$

Hence $\dim_k(\Lambda) = 12$. Let M be the module of Λ given by the projective presentation

$$v_1\Lambda \oplus v_2\Lambda \oplus v_3\Lambda \xrightarrow{(\lambda_{ji})} v_1\Lambda \oplus v_1\Lambda \to M \to 0,$$

with $\lambda_{ji} = \begin{pmatrix} \alpha^2 + \alpha & \alpha \\ \beta + \alpha\beta & \alpha^2\beta \\ \beta\gamma & \alpha\zeta \end{pmatrix}$. A tip-reduced uniform right Gröbner basis for I is $\{\alpha^3, \alpha^2\zeta - \beta\gamma, \alpha\beta\gamma, \alpha^2\beta\gamma\}$. We proceed as in Section 6.2.1, by lifting up the map to a projective presentation over $k\Gamma$. Doing this we end up with a basis for M:

$$\mathcal{B} = \{(0, v_1), (0, \alpha), (0, \beta), (0, \zeta), (0, \alpha^2), (0, \alpha\beta), (0, \alpha^2\beta), (v_1, 0), (\zeta, 0)\}.$$

The details can be found in [16, Example 4]. The example sets $M = N$, and hence we are looking for the endomorphism ring of M. As we now have a basis, we can calculate the matrix D. The procedure for doing this is identical to the calculation in Section 6.2.2. We note that bases for M_1, M_2 and M_3 respectively, where $M_i = Mv_i$, are

$$\begin{aligned}
\mathcal{B}_1 &= \{(0, v_1), (0, \alpha), (0, \alpha^2), (v_1, 0)\} \\
\mathcal{B}_2 &= \{(0, \beta), (0, \alpha\beta), (0, \alpha^2\beta)\} \\
\mathcal{B}_3 &= \{(0, \zeta), (\zeta, 0)\}
\end{aligned}$$

They are found simply by observing the end vertex of each basis element. The resulting matrices are

$$L_M(\alpha) = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 \end{pmatrix}$$

$$L_M(\beta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & -2 \end{pmatrix}$$

$$L_M(\gamma) = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}$$

$$L_M(\zeta) = \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix}$$

and hence we end up with the matrix D for (λ^*_{ji}):

$$D = \begin{pmatrix} L_M(\alpha)^2 + L_M(\alpha) & L_M(\beta) + L_M(\alpha)L_M(\beta) & L_M(\beta)L_M(\gamma) \\ L_M(\alpha) & L_M(\alpha)^2 L_M(\beta) & L_M(\alpha)L_M(\zeta) \end{pmatrix}$$

$$= \left(\begin{array}{cccc|ccc|cc} 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ \hline 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \end{array}\right)$$

Note that if we have the matrices defining the representation/module N (or M), the desired map induced by (λ_{ji}) can readily be obtained without calculating L_N.

7.2.3 Going from the kernel to $\mathrm{Hom}_\Lambda(M, N)$

The final step after having calculated (λ^*_{ji}), and having calculated the kernel of (λ^*_{ji}), is to associate the kernel back to $\mathrm{Hom}_\Lambda(M, N)$. In other words specify the isomorphism $\mathrm{Hom}_\Lambda(M, N) \cong \ker(\lambda^*_{ji})$.

Given the matrix D, we can quite literally read off the **rows** as basis elements of homomorphisms between M and N. This is the exact converse of what we do for Green-Heath-Struble in Section 6.2.4, where we associate **columns** with basis elements of $M \otimes M^*$. The argument becomes somewhat less complex however, given that we do not have to navigate through a number of dual transformations.

Every column of D has an associated basis element n_s, and every row represents an element in $\oplus_{i=0}^{g} Nv(i)$. We wish to make a map from $\oplus_{i=1}^{g} Nv(i)$ to $\mathrm{Hom}_\Lambda(M, N)$, similar to how we made a map from $\oplus_{i=1} v(i)N^*$ to $M \oplus N^*$ in Section 6.2.4. Define b_i by the vector with $v(i)$ in the ith component. For $i = \{1, \ldots, r\}$, b_i is the basis element of M_i which is also a generator of M_i. We send every element $n_s v(i)$ in a basis element of the kernel $\mathcal{C}$ of D to the homomorphism $b_i \mapsto n_s$. In other words, if we are given $n_s v(i)$, we create the map $f_{s,i}\colon M \to N$, defined by $f_{s,i}(b_i) = n_s$. These maps determine where to send the generators of M.

Hence a basis element D can be interpreted directly as a basis of $\mathrm{Hom}_\Lambda(M, N)$ on the form $\sum_{s,t} f_{s,i}$. Note that this is completely analogous to how we interpreted basis elements of the cokernel of D in Section 6.2.4. Sending all elements of the kernel $\mathcal{C}$ through this map, we get a basis for $\mathrm{Hom}_\Lambda(M, N)$.

7.3 Example

Continuing on Example 7.3, where we left off with the matrix

$$D = \begin{pmatrix} L_M(\alpha)^2 + L_M(\alpha) & L_M(\beta) + L_M(\alpha)L_M(\beta) & L_M(\beta)L_M(\gamma) \\ L_M(\alpha) & L_M(\alpha)^2 L_M(\beta) & L_M(\alpha)L_M(\zeta) \end{pmatrix}$$

$$= \left(\begin{array}{cccc|ccc|cc} 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ \hline 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \end{array}\right)$$

The correspondence between the rows of D and elements of $\mathrm{Hom}_\Lambda(M, N)$ is

Row	Element
1	$(v_1, 0) \to (v_1, 0)$
2	$(v_1, 0) \to (0, a)$
3	$(v_1, 0) \to (0, a^2)$
4	$(v_1, 0) \to (v_1, 0)$
5	$(0, v_1) \to (0, v_1)$
6	$(0, v_1) \to (0, a)$
7	$(0, v_1) \to (0, a^2)$
8	$(0, v_1) \to (v_1, 0)$

and a basis for the kernel of D is

$$\mathcal{C} = \{(0, 0, 0, 1, 1, 0, 0, 0), (0, 0, 0, 0, 0, 0, 1, 0), (0, 0, 0, -1, 0, -1, 0, 1)\}.$$

We read off the entries of each basis element as constituent parts of a homomorphism basis in $\mathrm{Hom}_\Lambda(M, N)$. Take for instance the basis element $(0, 0, 0, 1, 1, 0, 0, 0)$. It corresponds to the homomorphism which sends $(v_1, 0) \to (v_1, 0)$ and $(0, v_1) \to (0, v_1)$. In other words the identity map.

7.4 Analysis

In this section we perform an analysis of the complexity of the above algorithm. We will give the complexity in terms of the size of the matrix (λ_{ji}). In Chapter 9 we will extend the complexity to be written in terms of when the input are representations R_M and R_N. As before, we let d_V denote the number of vertices, and d_E the number of arrows in the underlying quiver. The analysis here is almost identical to the analysis in Section 6.4, up to some permutations of variables.

The input to the algorithm is two projective presentations M and N. The function FindDualLambda calculates the matrix D. Denote the size of the matrix (λ_{ji}) over Λ is by $m' \times n'$, implying that the size of (λ^*_{ji}) is $n' \times m'$. For every entry of the matrix (which contains elements of Λ), we have to insert an appropriate combination of L_M matrices as described above. The size of these matrices depend on the size of the underlying bases N_i. Writing $n_i = |N_i|$, the resulting size of the matrix D can be written

$$\sum_{i=1}^{g} n_i \times \sum_{j=1}^{r} n_j.$$

The function also have to find a Gröbner bases for Λ and then for the intermediate matrix F. The computational complexity for finding a Gröbner basis is hard to analyse, and will vary a lot depending on the input. As in Chapter 6 we will denote this complexity by $\mathcal{G}(\Lambda)$, and leave it for further analysis in Chapter 9. The function also lifts (λ_{ji}) up to (f_{ji}) to calculate a basis for M. The primary cost of this calculation is constructing and reducing the matrix F, as shown in Example 6.4. To create F, the matrix (λ_{ji}) is appended the element I in all coordinates i, as described in Section 6.2.1. The number of elements in I is at most n_i. Hence the size of this matrix is at most $m' \times n' + (\sum_{i=0}^{g} n_i) \times m$, which we denote by $n'' \times m''$. To reduce the matrix we have to calculate a Gröbner basis, and incur a cost of $\mathcal{G}(F)$.

Writing $n = \sum_{i=1}^{g} n_i$ and $m = \sum_{j=1}^{r} n_j$, we get that the complexity of FindDualLambda is $\mathcal{O}(n \cdot m + \mathcal{G}(\Lambda) + \mathcal{G}(F))$, as we have to insert $n \times m$ elements to the matrix D.

The following function NullSpace has has complexity $\mathcal{O}(n^3 + mn^2)$, as we know from Section 5.3. Finally we have the function CreateHomBasisFromLambdaNullSpace which performs the procedure outlined in Section 7.2.3. We can analyse this similar to the function CreateMapsFromLinearEquations in Section 5.3: The maximum number of bases in the kernel is n. If the function CreateHomBasisFromLambdaNullSpace checks the homomorphism for consistency, it has to iterate over each vertex and will incur a cost of $\mathcal{O}(d_V)$. Hence the complexity of CreateHomBasisFromLambdaNullSpace is $\mathcal{O}(d_V n)$. In the case where the function CreateHomBasisFromLambdaNullSpace does not check for consistency, we end up with complexity $\mathcal{O}(n)$.

Theorem 7.4. *The time complexity of Algorithm 5 is* $\mathcal{O}(n^3 + mn^2 + d_V n + \mathcal{G}(\Lambda) + \mathcal{G}(F))$.

Proof. Follows directly from the above argument. □

Chapter 8

Probabilistic Decomposition

In this chapter we introduce an algorithm which attempts to randomly select elements of a finite-dimensional algebra R over a **finite** field k. The elements we are trying to randomly select have to be both non-invertible and and non-nilpotent. This yields a decomposition by Fitting's Lemma.

The reason we are trying to do this originates from Theorem 3.2. Given a module M over R, the theorem states that finding a decomposition of the identity endomorphism on M into pairwise orthogonal idempotents, naturally yields a decomposition of M. The question then is how to find such a decomposition of the identity endomorphism. One answer can be found in [6]. In practice however, finding a set of pairwise orthogonal idempotents might be computationally expensive. This algorithm presents a probabilistic alternative.

The algorithm has an implementation in QPA[20, moduledecomp.gi, lines 492-556].

8.1 Overview

In this section we give an overview of how the algorithm works, arriving at a piece of pseudocode.

The input of the algorithm is a module M over a finite-dimensional algebra R. The first thing we have to do, is calculate the endomorphism ring $\mathrm{End}(M)$ of M. We can do this with either of the algorithms presented earlier. For brevity we will set $E = \mathrm{End}(M)$. We then wish to decompose M by successively applying Fitting's Lemma, which will be introduced below. To apply the lemma, we need to find members $\phi \in E$ which are non-invertible and non-nilpotent. We do this by random selection. Whenever we find an appropriate element, we can split the module M in two, and run the algorithm recursively on the two pieces. A condition for how many iterations we have to run will also be presented.

Algorithm 6 shows pseudocode for the procedure described above. The function IsInvertible and IsNilpotent can be implemented efficiently, see for instance [20, modulecomp.ci, line 516-517]. The function FindMaximumChainLength

Algorithm 6 Probabilistic decomposition of Id_M

```
function DECOMPOSE($M$, $E$)
    $n \leftarrow$ FINDMAXIMUMCHAINLENGTH($E$)
    $maxIterations \leftarrow$ DECOMPOSENUMBEROFITERATIONS($E$)
    $i \leftarrow 0$
    while $i \leq maxIterations$ do
        $i \leftarrow i + 1$
        $\phi \leftarrow$ RANDOMELEMENT($E$)
        if not ISINVERTIBLE($\phi$) and not ISNILPOTENT($\phi$) then
            $\phi_n \leftarrow \phi^n$
            ▷ $E_{\ker_n}$ is the endomorphism ring restricted to the elements of $\ker_n$
            $E_{\ker_n}, \ker_n \leftarrow$ CALCULATEKERNEL($\phi_n, E$)
            ▷ $E_{\mathrm{Im}_n}$ is the endomorphism ring restricted to the elements of $\mathrm{Im}_n$
            $E_{\mathrm{Im}_n}, \mathrm{Im}_n \leftarrow$ CALCULATEIMAGE($\phi_n, E$)
            return DECOMPOSE($\mathrm{Im}_n, E_{\mathrm{Im}_n}$) $\oplus$ DECOMPOSE($\ker_n, E_{\ker_n}$)
    return $M$
```

[20, modulecomp.ci, line 506-507] can also be implemented efficiently. The function RANDOMELEMENT simply selects a random element from E. The functions CALCULATEKERNEL and CALCULATEIMAGE are the crux of the procedure, and calculates our sought after $\mathrm{Im}(\phi^n)$ and $\ker(\phi^n)$. They also have the additional task of restricting the endomorphisms in E to elements of $\mathrm{Im}(\phi^n)$ and $\ker(\phi^n)$ respectively. Finally, the function DECOMPOSENUMBEROFITERATIONS returns the max number of iterations to return. Finally, we note that the algorithm returns inconclusively if it is unable to find a decomposition. One might still exist, but the algorithm has been unable to find it.

The intermediate functions will be described in detail in the following section.

8.2 Theory and detailed algorithm

In this section we describe in detail how the algorithm works. In particular, we will expand on the theory behind Fitting's lemma, and how it can be used to decompose a module M.

The reader is assumed to be familiar with basic concepts and results in ring theory.

8.2.1 Fitting's Lemma

We start of by stating Fitting's lemma. The utility this has in providing a decomposition of M should be immediately clear.

Lemma 8.1 (Fitting's Lemma)**.** *Let R be a ring, and M be a module over R of finite length. Let then ϕ be any element of the endomorphism ring of M,* $\mathrm{End}(M)$*. Then there exists an integer n such that*

$$M = \mathrm{Im}(\phi^n) \oplus \ker(\phi^n).$$

Proof. Let $\phi \in \mathrm{End}(M)$. First note that $\mathrm{Im}(\phi) \subseteq M$ and $\ker(\phi) \subseteq M$. It should be clear that we have $\mathrm{Im}(\phi^n) \supseteq \mathrm{Im}(\phi^{n+1})$, and $\ker(\phi^n) \subseteq \ker(\phi^{n+1})$ for any $n \geq 0$. Thus we have the following chains

$$\begin{aligned}&\mathrm{Im}(\phi) \supseteq \mathrm{Im}(\phi^2) \supseteq \dots \supseteq \mathrm{Im}(\phi^{n-1}) \supseteq \mathrm{Im}(\phi^n) \subseteq \dots\\&\ker(\phi) \subseteq \ker(\phi^2) \subseteq \dots \subseteq \ker(\phi^{m-1}) \subseteq \ker(\phi^m) \subseteq \dots\end{aligned}$$

Since R is of finite length, it is both *noetherian* and *artinian*. Consequently we know that both of these sequences at some point, say k and m respectively, will stop:

$$\begin{aligned}&\mathrm{Im}(\phi^k) = \mathrm{Im}(\phi^{k+1}) = \dots\\&\ker(\phi^m) = \ker(\phi^{m+1}) = \dots\end{aligned}$$

Select $n = \max(k, m)$. At n, both chains have terminated. Now let $u \in M$, and note that $\phi^n(u) \in \mathrm{Im}(\phi^n) = \mathrm{Im}(\phi^{2n})$. This means that for some $v \in M$, $\phi^n(u) = \phi^{2n}(v) = \phi^n(\phi^n(v))$. Now write u as

$$u = (u - \phi^n(v)) + \phi^n(v).$$

We wish to prove that of these two terms, one of them is in $\mathrm{Im}(\phi^n)$ and one is in $\ker(\phi^n)$, so that $M = \mathrm{Im}(\phi^n) + \ker(\phi^n)$. Applying ϕ^n to both the terms, we get

$$\begin{aligned}&\phi^n(u) - \phi^{2n}(v) = 0 \implies \text{So } (u - \phi^n(v)) \in \ker(\phi^n)\\&\phi^{2n}(v) = \phi^n(u) \implies \text{So } \phi^n(v) \in \mathrm{Im}(\phi^n)\end{aligned}$$

Hence $M = \mathrm{Im}(\phi^n) + \ker(\phi^n)$ as stated.

We now wish to show that $\mathrm{Im}(\phi^n) \cap \ker(\phi^n) = 0$. So let $u \in \mathrm{Im}(\phi^n) \cap \ker(\phi^n)$. Then, as before, $u = \phi^n(v)$ for some $v \in M$. Applying ϕ^n on both sides, we get that $\phi^n(u) = \phi^{2n}(v)$. However, since $u \in \ker(\phi^n)$, then $\phi^{2n}(v) = 0 \implies v \in \ker(\phi^{2n}) = \ker(\phi^n)$. Thus $u = \phi^n(v) = 0$. So any $u \in \mathrm{Im}(\phi^n) \cap \ker(\phi^n)$ must be 0, giving that $M = \mathrm{Im}(\phi^n) \oplus \ker(\phi^n)$. □

We can use the above lemma to decompose M, by finding an appropriate ϕ. However, to get a real decomposition of M we need to make sure that the ϕ we find is both non-invertible and non-nilpotent: Suppose ϕ is nilpotent. Then $\phi^n = 0$, which means that the decomposition arising from Fitting's lemma will be trivial, as $\ker(\phi^n) = M$. Suppose instead that ϕ is invertible. Then it is necessarily an **automorphism**, implying that $\ker(\phi^n) = 0$. Consequently $M = \mathrm{Im}(\phi^n)$.

8.2.2 Performing the decomposition

The procedure for decomposing M should now be obvious: Starting with M we calculate $\text{End}(M)$. We then chose a random element of $\text{End}(M)$, and check if it is non-invertible and non-nilpotent. If we have found such an element, we can decompose M into $\text{Im}(\phi^n)$ and $\ker(\phi^n)$, for some n. This might not be the complete decomposition, as both $\text{Im}(\phi^n)$ and $\ker(\phi^n)$ may be further decomposable. Thus we have to recursively call our procedure with $\text{Im}(\phi^n)$ and $\ker(\phi^n)$ in the place of M. However, we need to make sure we alter $\text{End}(M)$ by letting all $\phi \in \text{End}(M)$ be restricted to $\text{Im}(\phi_n)$ and $\ker(\phi_n)$ respectively. This is a fairly straight forward calculation, relying on the following diagram:

$$\begin{tikzcd}
\text{Im}(\phi^n) \arrow[rrr, "g"] \arrow[dr, shift right, "\nu"'] & & & \text{Im}(\phi^n) \arrow[dl, shift right, "\nu"'] \\
& M \arrow[ul, shift right, "\pi"'] \arrow[r, "f"] & M \arrow[ur, shift right, "\pi"'] &
\end{tikzcd}$$

Observing that $\nu\pi = Id_{\text{Im}_{\phi^n}}$, we can bring f up to g by applying this isomorphism before and after f. This can be done similarly for $\ker \phi^n$ with minor adjustments.

We can now state the pseudocode for the functions CalculateImage and CalculateKernel, which can be found in Algorithm 7.

Algorithm 7 Functions for calculating the image and kernel of a map ϕ^n

```
function CalculateImage($\phi^n, E$)
    $\nu \leftarrow$ GetImageInclusion($\phi^n$)
    $\pi \leftarrow$ GetImageProjection($\phi^n$)
    $\nu\pi \leftarrow \nu \cdot \pi$
    return Range($\pi$), RestrictEndomorprhismToImage($\phi^n, \nu, \pi, E$))
function CalculateKernel($\phi^n, E$)
    $\nu \leftarrow$ GetKernelInclusion($\phi^n$)
    $\pi \leftarrow$ GetCoKernelProjection($\phi^n$)
    $\nu\pi \leftarrow \nu \cdot \pi$
    return Range($\pi$), RestrictEndomorphismToKernel($\phi^n, \nu, \pi, E$))
```

The functions GetImageInclusion, GetImageProjection, GetKernelInclusion, and GetCoKernelProjection does exactly what one would expect. Reference implementations can be found in [20, modulehom.gi, line 723 and onward]. The function RestrictEndomorphismRingToKernel and RestrictEndomorphismRingToImage performs the restriction of E to the kernel and image of ϕ^n respectively. The relevant lines in QPA can be found in [20, moduledecomp.gi, lines 541-547 and 532-538 respectively].

8.2.3 Number of runs

The only remaining unexplained portion of Algorithm 7 is the function DecomposeNumberOfIterations. The algorithm is of importance for two reasons: It gives an indication of how long one should expect the algorithm to run before it yields a likely definitive answer. Secondly, it will be of importance when deciding which complexity class the algorithm belongs to.

Given that the probability for finding an appropriate element is $\mathfrak{p}$, we have two cases: $\mathfrak{p}$ is less than $1/2$, and $\mathfrak{p}$ is greater than or equal to $1/2$. In the former scenario, we wish to run the algorithm k times, i.e. select a maximum of k elements, such that the overall probability of success is greater than or equal to $1/2$. We can find k directly:

$$\begin{aligned} 1-(1-\mathfrak{p})^k &= 0.5 \\ (1-\mathfrak{p})^k &= 0.5 \\ k\log(1-\mathfrak{p}) &= \log(0.5) \end{aligned}$$

and so we end up with the formula for k

$$k = \frac{\log(0.5)}{\log(1-\mathfrak{p})}. \tag{8.1}$$

Example 8.2. Let $\mathfrak{p} = 0.25$. Then we have to run the algorithm $\frac{\log(0.5)}{\log(1-0.25)} = 2.4$ times.

Algorithm 8 shows pseudocode for the function DecomposeNumberOfIterations. The intermediate function ProbabilityOfFindingValidElement returns the probability $\mathfrak{p}$ of finding a non-nilpotent non-invertible element ϕ. This probability will be calculated in Section 8.3.

Algorithm 8 Calculate the number of iterations

```
function DecomposeNumberOfIterations($E$)
    $\mathfrak{p} \leftarrow$ ProbabilityOfFindingValidElement($E$)
    if $\mathfrak{p} \geq 0.5$ then
        return 1
    else
        return $\log(0.5)/\log(1-\mathfrak{p})$
```

8.3 Analysis

In this section we analyse the probabilistic algorithm described in the chapter so far. Specifically, we wish to find the probability of by chance selecting a non-nilpotent non-invertible element in the endomorphism ring $\mathrm{End}(M)$. However, in this section we slack the conditions slightly, and work with a finite-dimensional algebra R over a field k.

8.3.1 Preliminary results

We start by proving some preliminary results which will aid us in finding a bound for $\mathfrak{p}$.

Lemma 8.3. *Let R be a finite-dimensional algebra over a field k with Jacobsen radical $\mathfrak{r}$. Let x be an element in R, and $\bar{x}$ the projection of x into $R/\mathfrak{r}$. Then the following assertions hold*

(i) x is nilpotent in R if and only if $\bar{x}$ is nilpotent in $R/\mathfrak{r}$.

(ii) x is invertible in R if and only if $\bar{x}$ is invertible in $R/\mathfrak{r}$.

Proof. (i) Clearly $\bar{x}$ is nilpotent in $R/\mathfrak{r}$ if x is nilpotent.

Conversely suppose $\bar{x}$ is nilpotent in $R/\mathfrak{r}$. Then there exists $n \geq 1$ such that $\bar{x}^n \in \mathfrak{r}$. Since $\mathfrak{r}$ is a nilpotent ideal, there exists $m \geq 1$ such that $(\bar{x}^n)^m = 0$. Hence $\bar{x}$ is nilpotent in R.

(ii) Clearly $\bar{x}$ is invertible in $R/\mathfrak{r}$ if x is invertible.

Conversely suppose $\bar{x}$ is invertible in $R/\mathfrak{r}$. Then $\bar{x}\bar{y} = \bar{y}\bar{x} = 1 + \mathfrak{r}$ for some $y \in R$. Then both $(1 - xy)$ and $(1 - yx)$ is in $\mathfrak{r}$. Hence there exists an m such that $(1 - xy)^m = (1 - yx)^m = 0$. If $m = 1$ we are done, as y is the inverse of x. So suppose $m \geq 2$. Then

$$\begin{aligned}
0 = (1 - xy)^m &= \sum_{i=0}^{m} \binom{m}{i} (-xy)^{m-i} \\
&= \sum_{i=0}^{m} \binom{m}{i} (-1)^{m-i} (xy)^{m-i} \\
&= 1 + \sum_{i=0}^{m-1} \binom{m}{i} (-1)^{m-i} (xy)^{m-i} \\
&= 1 - \sum_{i=0}^{m-1} \binom{m}{i} (-1)^{m-i+1} (xy)^{m-i} \\
&= 1 - x \cdot \left(y \sum_{i=0}^{m-1} \binom{m}{i} (-1)^{m-i+1} (xy)^{m-i-1} \right)
\end{aligned}$$

and hence x has a right inverse. Similarly we can show that x has a left inverse. □

We use the above lemma to prove some results about the density of non-invertible and non-nilpotent elements in R. We can now solely restrict our attention to $R/\mathfrak{r}$, where it will be easier for us to determine how frequent such elements are. First we introduce some short-hand notation for the count of non-invertible and non-nilpotent elements in a ring R. We let $|X|$ denote the number of elements in a set X.

Definition 8.4. Let R be a finite ring. We define the following:

(i) Denote by $\mathcal{N}(R)$ the number of nilpotent elements in R.

(ii) Denote by $\mathcal{I}(R)$ the number of invertible elements in R.

(iii) Denote by $\mathrm{n}\mathcal{N}\mathrm{n}\mathcal{I}(R)$ the number of non-invertible non-nilpotent elements in R.

The quantity $|\mathrm{n}\mathcal{N}\mathrm{n}\mathcal{I}(R)|$ is what we are interested in calculating. In particular in relation to the quantity $|R|$. The following lemma again helps us to reduce the problem to $R/\mathfrak{r}$.

Lemma 8.5. *Let R be a finite-dimensional algebra over a finite field k with Jacobsen radical $\mathfrak{r}$. Then*

$$\frac{|\mathrm{n}\mathcal{N}\mathrm{n}\mathcal{I}(R)|}{|R|} = \frac{|\mathrm{n}\mathcal{N}\mathrm{n}\mathcal{I}(R/\mathfrak{r})|}{|R/\mathfrak{r}|}$$

Proof. By direct counting, $|R| = |R/\mathfrak{r}|\cdot|\mathfrak{r}|$. By Lemma 8.3 we know that $|\mathrm{n}\mathcal{N}\mathrm{n}\mathcal{I}(R)| = |\mathrm{n}\mathcal{N}\mathrm{n}\mathcal{I}(R/\mathfrak{r})| \cdot |\mathfrak{r}|$. The result follows immediately by combining the equations. □

Lemma 8.6. *Let $R = R_1 \times R_2 \times \cdots \times R_t$ be a finite direct product of rings R_i. Then the likelihood of any random element $x \in R$ to be non-invertible non-nilpotent is*

$$\frac{|\mathrm{n}\mathcal{N}\mathrm{n}\mathcal{I}(R)|}{|R|} = 1 - \frac{\prod_{i=1}^t |\mathcal{N}(R_i)| + \prod_{i=1}^t |\mathcal{I}(R_i)|}{|R|}$$

Proof. The set of elements in R which are nilpotent is completely disjoint from the set of elements which are invertible. Thus the number of elements in R which are non-nilpotent non-invertible is precisely

$$|\mathrm{n}\mathcal{N}\mathrm{n}\mathcal{I}(R)| = |R| - |\mathcal{N}(R)| - |\mathcal{I}(R)|.$$

So immediately we have that

$$\frac{|\mathrm{n}\mathcal{N}\mathrm{n}\mathcal{I}(R)|}{|R|} = 1 - \frac{|\mathcal{N}(R)| + |\mathcal{I}(R)|}{|R|}.$$

Any element in R can be written $x = (r_1, r_2, \ldots, r_t)$. For this element to nilpotent (invertible), all the elements r_i need to be nilpotent (invertible). Hence

$$|\mathcal{N}(R)| = \prod_{i=1}^{t} |\mathcal{N}(R_i)|$$

and

$$|\mathcal{I}(R)| = \prod_{i=1}^{t} |\mathcal{I}(R_i)|$$

The claim follows.

□

Finally we state a lemma which helps us reduce $R/\mathfrak{r}$ to a form akin to the ring R in Lemma 8.6.

Lemma 8.7. *Let R be a finite-dimensional algebra over a finite field k with Jacobsen radical $\mathfrak{r}$. Then $R/\mathfrak{r}$ is isomorphic to*

$$M_{n_1}(F_1) \times M_{n_2}(F_1) \times \cdots \times M_{n_t}(F_t)$$

for some integers $n_i \geq 1$ and finite field extensions F_i of k.

Proof. The Artin-Wedderburn theorem states that $R/\mathfrak{r}$ is isomorphic to a finite direct product of full matrix rings over division rings D_i which contains the finite field k

$$M_{n_1}(D_1) \times M_{n_2}(D_1) \times \cdots \times M_{n_t}(D_t).$$

for some integers $n_i \geq 1$[3, Theorem 3.2, p. 382]. Since R is finite-dimensional over k, so is each of the division rings D_i. Hence each D_i is a finite field extension of k. □

8.3.2 Finding a bound for p

Combining Lemma 8.6 and Lemma 8.7, we can now provide a bound for the probability

$$\mathfrak{p} = \frac{|\text{n}\mathcal{N}\text{n}\mathcal{I}(R)|}{|R|}.$$

Let R be a finite-dimensional algebra over a finite field k, with Jacobsen radical $\mathfrak{r}$. Without loss of generality, let $k = \mathbb{F}_{p^d}$ for a prime p and $d \geq 1$. Then in the decomposition of $R/\mathfrak{r}$ each $F_i = \mathbb{F}_{p^{d+d_i}}$, with $d_i \geq 0$. Hence we can write

$$R/\mathfrak{r} = M_{n_1}(\mathbb{F}_{p^{d+d_1}}) \times M_{n_1}(\mathbb{F}_{p^{d+d_2}}) \times \cdots \times M_{n_t}(\mathbb{F}_{p^{d+d_t}})$$

Let $q = p^d$. For a general matrix rings $M_n(\mathbb{F}_q)$ we have the following results:

$$|\mathcal{N}(M_n(\mathbb{F}_q))| = q^{n^2-n}$$
$$|\mathcal{I}(M_n(\mathbb{F}_q))| = \prod_{i=0}^{n-1}(q^n - q^i).$$

The first result can be found in [12][14]. The second can be found in [9, Example 1, p. 412].

We also have that

$$|R| = \prod_{n=1}^{t} q^{n_1^2(d+d_i)}$$

Hence we can write the expression in Lemma 8.3 as

$$\begin{aligned}
\mathfrak{p} = \frac{|\mathrm{n}\mathcal{N}\mathrm{n}\mathcal{I}(R)|}{|R|} &= 1 - \frac{\prod_{i=1}^{t} q^{(d+d_i)(n_i^2-n_i)} + \prod_{i=1}^{t}\prod_{j=0}^{n_i-1}\left(p^{(d+d_i)n_i} - p^{(d+d_i)j}\right)}{\prod_{i=1}^{t} p^{n_1^2(d+d_i)}} \\
&= 1 - \frac{\prod_{i=1}^{t} p^{(d+d_i)(n_i^2-n_i)}}{\prod_{n=1}^{t} p^{n_1^2(d+d_i)}} - \frac{\prod_{i=1}^{t}\prod_{j=0}^{n_i-1}\left(p^{(d+d_i)n_i} - p^{(d+d_i)j}\right)}{\prod_{i=1}^{t} p^{n_1^2(d+d_i)}} \\
&= 1 - \frac{\prod_{i=1}^{t} p^{(d+d_i)(n_i^2-n_i)}}{\prod_{n=1}^{t} p^{n_i^2(d+d_i)}} - \frac{\prod_{i=1}^{t}\prod_{j=0}^{n_i-1}\left(p^{(d+d_i)n_i} - p^{(d+d_i)j}\right)}{\prod_{i=1}^{t} p^{n_1^2(d+d_i)}} \\
&= 1 - \prod_{i=1}^{t}\left(p^{-(d+d_i)n_i}\right) - \prod_{i=1}^{t}\prod_{j=0}^{n_i-1}\left(1 - p^{(d+d_i)(j-n_i)}\right).
\end{aligned}$$

We wish to find a lower bound for this expression, which by necessity has to be greater than 0. We have that

$$d \leq (d+d_i) \leq (d+d_i)n_i.$$

Hence we now have an upper (and lower) bound on all the variables in the final expression for $\mathfrak{p}$ above. Which in term implies that $\mathfrak{p}$ is a bound constant. Further simplifying, we end up with

$$\begin{aligned}
\mathfrak{p} &= 1 - \prod_{i=1}^{t}\left(p^{-(d+d_i)n_i}\right) - \prod_{i=1}^{t}\prod_{j=0}^{n_i-1}\left(1 - p^{(d+d_i)(j-n_i)}\right) \\
&= 1 - p^{\sum_{i=1}^{t} -(d+d_i)n_i} - \prod_{i=1}^{t}\prod_{j=0}^{n_i-1}\left(1 - p^{(d+d_i)(j-n_i)}\right) \\
&\geq 1 - p^{-dt} - \prod_{i=1}^{t}\left(1 - p^{-n_i(d+d_i)}\right)^{n_i} \\
&\geq 1 - p^{-dt} - (1 - p^{-\max(n_i)(d+\max(d_i))})^{t\min(n_i)}
\end{aligned}$$

Letting $\max(n_i) = x, \min(n_i) = y, \min(d_i) = z$, we get the following expression:

$$\mathfrak{p} \geq 1 - p^{-dt} - (1 - p^{-x\cdot(d+z)})^{t\cdot y}$$

Inserting this into Equation 8.1, assuming the worst (lowest) value for $\mathfrak{p}$, we get

$$\begin{aligned}
k &= \frac{\log(0.5)}{\log(1-\mathfrak{p})} \\
&= \frac{\log(0.5)}{\log(1-\big(1-p^{-dt}-(1-p^{-x\cdot(d+z)})^{t\cdot y}\big))} \\
&= \frac{\log(0.5)}{\log(p^{-dt}-(1-p^{-x\cdot(d+z)})^{t\cdot y})} \\
&= \frac{\log(0.5)}{\log(p^{-dt}+(1-p^{-x\cdot(d+z)})^{t\cdot y})}
\end{aligned}$$

Chapter 9

Results

In this chapter we summarise the results of the analysis. We will compare the different algorithms in terms of runtime, and categorise the problem of constructing $\mathrm{Hom}_\Lambda(M, N)$ in terms of the complexity classes introduced in Chapter 4.

9.1 Comparing the algorithms for creating $\mathrm{Hom}_\Lambda(M, N)$

In this section we directly compare the three algorithms presented that constructs the set $\mathrm{Hom}_\Lambda(M, N)$.

Proposition 9.1. *The matrix D in Green-Heath-Struble, Algorithm 4, is the transpose of the matrix D in the Hom-functor algorithm, Algorithm 5.*

Proof. Clearly the input matrix (λ_{ji}) is identical in both algorithms. Suppose the matrix is written

$$(\lambda_{ji}) = \begin{pmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,g} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,g} \\ \vdots & \vdots & \ddots & \vdots \\ p_{r,1} & p_{2,r} & \cdots & p_{r,g} \end{pmatrix}$$

then the matrix (λ^*_{ji}) in Green-Heath-Struble will have the same shape, while in Hom-functor it will be transposed:

$$(\lambda_{ji}) = \begin{pmatrix} p_{1,1} & p_{2,1} & \cdots & p_{g,1} \\ p_{1,2} & p_{2,2} & \cdots & p_{g,2} \\ \vdots & \vdots & \ddots & \vdots \\ p_{1,r} & p_{2,r} & \cdots & p_{g,r} \end{pmatrix}$$

The matrix D is crafted in both scenarios by adding appropriate matrices $L_N(p_{i,j})$. In Green-Heath-Struble each matrix L_N is transposed, as specified in Theorem 6.3. Hence the entire resulting matrix D in Green-Heath-Struble is the transposed of the matrix D in Hom-functor.

□

Theorem 9.2. *The runtime complexity of the algorithm Green-Heath-Struble, Algorithm 4, and the Hom-functor algorithm, Algorithm 5, are identical up to asymptotical growth.*

Proof. The runtime complexity of Green-Heath-Struble, given the $m \times n$ matrix D, is

$$\mathcal{O}(n^3 + mn^2 + d_V m + \mathcal{G}(\Lambda) + \mathcal{G}(F)).$$

The runtime complexity of Hom-functor, given the $m' \times n'$ matrix D', is

$$\mathcal{O}(m'^3 + n'm'^2 + d_V n' + \mathcal{G}(\Lambda) + \mathcal{G}(F)).$$

From Proposition 9.1 we know that the resulting matrices are in fact transpositions of each other, so $m' = n$ and $n' = m$. The claim follows immediately. □

Before we are able to compare Green-Heath-Struble and Hom-functor with the linear algebra algorithm, we need to enter two missing pieces. First we need to provide bounds for the cost of calculating Gröbner bases, $\mathcal{G}(\Lambda)$ and $\mathcal{G}(F)$. Secondly, the input to the linear algebra algorithm is different than to the others; it takes two representations. Hence we need to analyse Green-Heath-Struble and Hom-functor in the scenario where there they take two representations, and constructs a projective presentation from those.

As mentioned in previous chapters, the analysis for calculating Gröbner basis is complex. The classical Buchberger's algorithm generally does not perform well, with runtimes and memory usage in the realm of **EXPSPACE-complete**[18][8]. A bound for the degrees of the elements input to the Buchberger's algorithm is[8]

$$2\left(\frac{d^2}{2} + d\right)^{2^{n-1}}.$$

A better alternative can be found in the F4/F5 algorithms[10]. However, these are still exponential in space usage[2], and consequently also in runtime. Our only chance is that the inputs Λ and F to the calculation of the bases not necessarily grow with the size of the modules. Unfortunately, while Λ does not grow with the size of the modules M and N, F does.

Given the complexity of the analysis of the F4/F5 algorithm, we will not reiterate it here. There reader is referred to [10], [2] for both the algorithm details and the analysis. We can now prove the following:

Proposition 9.3. *The runtime complexity of the algorithm Green-Heath-Struble, Algorithm 4, and the Hom-functor algorithm, Algorithm 5, are*

$$\mathcal{O}(\mathcal{G}(\Lambda) + \mathcal{G}(F))$$

Proof. Follows from the analysis of the algorithms F4/F5 for calculating Gröbner bases[2]. The exponential terms for calculating such a basis dominates the other terms. □

We now look at changing the input to the algorithms Green-Heath-Struble and Hom-functor to representations R_M and R_N, instead of projective presentations. In this scenario, a few things change with both algorithms. First, we have to calculate a (vertex) projective presentation of M. We can do this by iteratively calculating projective covers. A routine for doing this is in [20, modulehomalg.gi, line 549]. This routine can be seen to run in $\mathcal{O}(d_V \cdot d_E)$

Recall that the variables n and m in the Hom-functor analysis was defined by

$$n \times m = \sum_{i=1}^{g} n_i \times \sum_{j=1}^{r} n_j$$

To be able to compare with the Linear Algebra algorithm, we need to determine r and g in terms of properties of the input representations R_M and R_N. From the analysis and bounds given in Section 3.3, we know that in the projective presentation

$$P_1 \to P_0 \to M \to 0$$

we have $\dim_k P_1 \leq \dim_k M \cdot (\mathcal{M}^2 - \mathcal{M})$, $\dim_k P_0 \leq \dim_k M \cdot \mathcal{M}$. Here $\mathcal{M} \overset{def}{=} \max_{v\in\Gamma_0}\{\dim_k v\Lambda\}$. Expanding on this analysis, we have that $g = \dim_k P_0/\operatorname{rad} P_0$ and $r = \dim_k \ker f_0/\operatorname{rad}\ker f_0$. Hence we get

$$\begin{aligned} g &\leq \dim_k M \\ r &\leq \dim_k \ker f_0 = \dim_k P_0 - \dim_k M \\ &\leq \mathcal{M}\dim_k M - \dim_k M \\ &= \dim_k M(\mathcal{M} - 1) \end{aligned}$$

These represent the worst-case scenarios, where P_1 and P_0 are the greatest. We can rewrite the complexity of Hom-functor (Green-Heath-Struble) in terms of these new variables, setting $r = \dim_k M \cdot (\mathcal{M} - 1)$, $g = \dim_k M$.

$$\mathcal{O}\left(\left(\sum_{i=1}^{\dim_k M\cdot(\mathcal{M}-1)} n_i\right)^3 + \left(\sum_{j=1}^{\dim_k M} n_j\right)\cdot\left(\sum_{i=1}^{\dim_k M\cdot(\mathcal{M}-1)} n_i\right)^2 \right.$$
$$\left. + d_V\left(\sum_{i=1}^{\dim_k M\cdot(\mathcal{M}-1)} n_i\right) + \mathcal{G}(\Lambda) + \mathcal{G}(F)\right)$$

Note that the n_i and n_j are not necessarily all unique. We also note that when given representations as input to the Hom-functor (Green-Heath-Struble) algorithm, we no longer have to calculate the Gröbner bases, nor the matrices L_N. This is because the representations intrinsically contain both the bases and the maps for R_M and R_N. Hence the final complexity for the representation case of the algorithms is

$$\mathcal{O}\left(\left(\sum_{i=1}^{\dim_k M\cdot(\mathcal{M}-1)} n_i\right)^3+\left(\sum_{j=1}^{\dim_k M} n_j\right)\cdot\left(\sum_{i=1}^{\dim_k M\cdot(\mathcal{M}-1)} n_i\right)^2+d_V\left(\sum_{i=1}^{\dim_k M\cdot(\mathcal{M}-1)} n_i\right)\right)$$

Recall that the complexity of the Linear Algebra algorithm is

$$\mathcal{O}\left(\left(\sum_{k=1}^{d_E} m_{s(\alpha_k)}\cdot n_{e(\alpha_k)}\right)^3+\left(\sum_{i=1}^{d_V} m_i n_i\right)\cdot\left(\sum_{k=1}^{d_E} m_{s(\alpha_k)}\cdot n_{e(\alpha_k)}\right)^2+d_V\left(\sum_{i=1}^{d_V} m_i n_i\right)\right)$$

The two complexities presented here are not directly comparable, as they depend on distinct quantities. We do note that they in some respects scale similarly, as $\dim_k M = \sum_i^{d_V} m_i$. We also note that the complexity of Hom-functor (Green-Heath-Struble) is more sensitive to the underlying path algebra than the Linear Algebra algorithm. Finally we note that both algorithms are polynomial in their runtime.

9.2 On probabilistic decomposition

In this section we summarise the analysis of the probabilistic algorithm in Chapter 8.

Recall the the final analysis of the algorithm showed that the number of runs required for the algorithm to possess a 1/2 chance of succeeding, is

$$k=\frac{\log(0.5)}{\log(p^{-dt}+(1-p^{-x\cdot(d+z)})^{t\cdot y})}.$$

By succeeding we mean to find an element which is both non-invertible and non-nilpotent. The decomposition process itself runs in polynomial time. Hence the question of whether the algorithm in its entirety runs in polynomial time depends on k. The function k depends on 6 variables: x, y, z, d, t and p. All of these are defined in Chapter 8. Note that $x \geq y$.

Unfortunately, we see that the expression for k grows exponentially in the variables x, y, z, d, t. The bound is not tight enough to give us good predictions about what happens for fixed x, y, z, d and t, with variable p. However, experimental results, shown in Figure 9.1, seem to indicate that the function grows at most polynomially. The results are found by tabulating values of k using the formula

$$\mathfrak{p}=1-\prod_{i=1}^{t}\left(p^{-(d+d_i)n_i}\right)-\prod_{i=1}^{t}\prod_{j=0}^{n_i-1}\left(1-p^{(d+d_i)(j-n_i)}\right)$$

for $\mathfrak{p}$. The value t is fixed at 4, d_i fixed at $\{1,2,1,1\}$, n_i at $\{1,3,4,5\}$ and $d=1$.

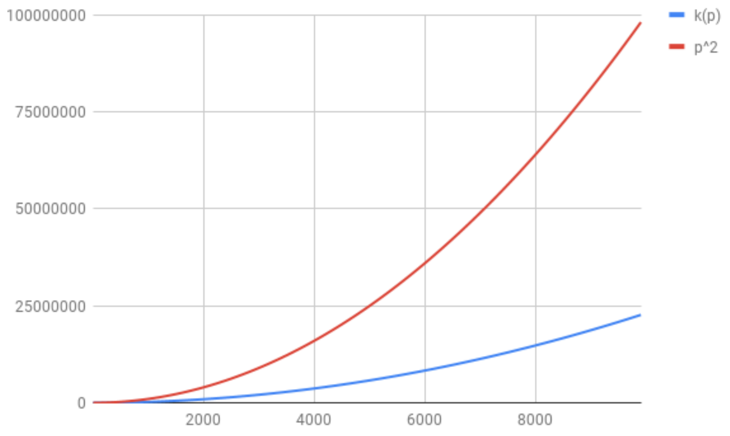

Figure 9.1: Experimental results for the variable k, with only p variable.

9.3 Complexity classes

In this final section we categorise the different algorithms into the complexity classes introduced in Chapter 4.

Theorem 9.4. *The algorithms Green-Heath-Struble and Hom-functor for finding* $\mathrm{Hom}_\Lambda(M, N)$*, are* ***EXPSPACE****, in the case of the input being projective presentations.*

Proof. Follows directly from the above arguments. □

Theorem 9.5. *The algorithms Green-Heath-Struble and Hom-functor are* ***P****, in the case of the input being representations.*

Proof. Follows directly from the above argument. □

Theorem 9.6. *The algorithm for finding* $\mathrm{Hom}_\Lambda(M, N)$ *with the Linear Algebra algorithm is* ***P****.*

Proof. Follows directly from the above argument. □

Theorem 9.7. *The problem of determining the set of homomorphisms between two modules M and N over a finite-dimensional path algebra Λ, is in* ***P****.*

Proof. Follows directly from both Theorem 9.5 and Theorem 9.6. □

Conjecture 9.8. The problem of decomposing the module M probabilistically using Algorithm 6 has exponential runtime in all variables but p.

Proof. Proof of the exponential runtimes follows from the analysis in Section 9.2. We note that while not proven, experimental results seem to suggest that the algorithm is polynomial in the variable p. $\square$

Given the analysis in Section 9.2, we have reason to believe that decomposing probabilistically using Algorithm 6 in general is **PSPACE**, and neither **ZPP** nor **PP**. We note that experimentation seem to suggest that there exists certain edge cases where the probability becomes sufficiently large to warrant usage of the algorithm.

Bibliography

[1] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach.* 1st. New York, NY, USA: Cambridge University Press, 2009. ISBN: 0521424267, 9780521424264.

[2] Magali Bardet, Jean-Charles Faugère, and Bruno Salvy. "On the complexity of the F5 Gröbner basis algorithm". In: *Journal of Symbolic Computation* 70 (2015), pp. 49–70. ISSN: 0747-7171. DOI: https://doi.org/10.1016/j.jsc.2014.09.025. URL: http://www.sciencedirect.com/science/article/pii/S0747717114000935.

[3] P.B. Bhattacharya, S.K. Jain, and S.R. Nagpaul. *Basic Abstract Algebra.* Cambridge University Press, 1994. ISBN: 9780521466295. URL: https://books.google.no/books?id=hiQ8e0b48swC.

[4] J. Buchmann et al. "An Implementation of the General Number Field Sieve". In: *In Proceedings of Crypto'93.* Springer-Verlag, 1993, pp. 159–165.

[5] Tony F. Chan. "Rank revealing QR factorizations". In: *Linear Algebra and its Applications* 88-89 (1987), pp. 67–82. ISSN: 0024-3795. DOI: https://doi.org/10.1016/0024-3795(87)90103-0. URL: http://www.sciencedirect.com/science/article/pii/0024379587901030.

[6] Alexander Chistov, Gábor Ivanyos, and Marek Karpinski. "Polynomial Time Algorithms for Modules over Finite Dimensional Algebras". In: *Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation.* ISSAC '97. Kihei, Maui, Hawaii, USA: ACM, 1997, pp. 68–74. ISBN: 0-89791-875-4. DOI: 10.1145/258726.258751. URL: http://doi.acm.org/10.1145/258726.258751.

[7] Thomas H. Cormen et al. *Introduction to Algorithms, Third Edition.* 3rd. The MIT Press, 2009. ISBN: 0262033844, 9780262033848.

[8] T. Dubé. "The Structure of Polynomial Ideals and Gröbner Bases". In: *SIAM Journal on Computing* 19.4 (1990), pp. 750–773. DOI: 10.1137/0219053. eprint: https://doi.org/10.1137/0219053. URL: https://doi.org/10.1137/0219053.

[9] D.S. Dummit and R.M. Foote. *Abstract Algebra.* Wiley, 2004. ISBN: 9780471433347. URL: https://books.google.no/books?id=KJDBQgAACAAJ.

[10] Jean-Charles Faugére. “A new efficient algorithm for computing Gröbner bases (F4)”. In: *Journal of Pure and Applied Algebra* 139.1 (1999), pp. 61–88. ISSN: 0022-4049. DOI: https://doi.org/10.1016/S0022-4049(99)00005-5. URL: http://www.sciencedirect.com/science/article/pii/S0022404999000055.

[11] N. Faujdar and S. P. Ghrera. “A detailed experimental analysis of library sort algorithm”. In: *2015 Annual IEEE India Conference (INDICON)*. Dec. 2015, pp. 1–6. DOI: 10.1109/INDICON.2015.7443165.

[12] N. J. Fine and I. N. Herstein. “The probability that a matrix be nilpotent”. In: *Illinois J. Math.* 2.4A (Nov. 1958), pp. 499–504. URL: https://projecteuclid.org:443/euclid.ijm/1255454112.

[13] Paul Garrett. *Half-exactness of adjoiunt functors, Yoneda Lemma*. Dec. 11, 2017. URL: http://www-users.math.umn.edu/~garrett/m/algebra/yoneda_and_tensors.pdf.

[14] Murray Gerstenhaber. “On the number of nilpotent matrices with coefficients in a finite field”. In: *Illinois J. Math.* 5.2 (June 1961), pp. 330–333. URL: https://projecteuclid.org:443/euclid.ijm/1255629831.

[15] Edward L. Green. “Multiplicative Bases, Gröbner Bases, and Right Gröbner Bases”. In: *Journal of Symbolic Computation* 29.4 (2000), pp. 601–623. ISSN: 0747-7171. DOI: https://doi.org/10.1006/jsco.1999.0324. URL: http://www.sciencedirect.com/science/article/pii/S0747717199903243.

[16] Edward L. Green, Lenwood S. Heath, and Craig A. Struble. “Constructing Homomorphism Spaces and Endomorphism Rings”. In: *Journal of Symbolic Computation* 32.1 (2001), pp. 101–117. ISSN: 0747-7171. DOI: http://dx.doi.org/10.1006/jsco.2001.0454. URL: http://www.sciencedirect.com/science/article/pii/S0747717101904547.

[17] A. Hatcher. *Algebraic Topology*. Algebraic Topology. Cambridge University Press, 2002. ISBN: 9780521795401. URL: https://books.google.com/books?id=BjKs86kosqgC.

[18] Ernst W Mayr and Albert R Meyer. “The complexity of the word problems for commutative semigroups and polynomial ideals”. In: *Advances in Mathematics* 46.3 (1982), pp. 305–329. ISSN: 0001-8708. DOI: https://doi.org/10.1016/0001-8708(82)90048-2. URL: http://www.sciencedirect.com/science/article/pii/0001870882900482.

[19] Michael Sipser. “Introduction to the Theory of Computation”. In: *SIGACT News* 27.1 (Mar. 1996), pp. 27–29. ISSN: 0163-5700. DOI: 10.1145/230514.571645. URL: http://doi.acm.org/10.1145/230514.571645.

[20] Øyvind Solberg. *Quivers and path algebras*. Dec. 7, 2017. URL: https://folk.ntnu.no/oyvinso/QPA/.