Calculating Ext for comodules over a finite field

Chris Vos

Supervised by
Gijs Heuts and Christian Carrick

Department of Mathematics
University Utrecht
Utrecht, Netherlands
14 June, 2024

1 Introduction

The goal of this thesis is to provide an algorithm which can compute $\mathrm{Ext}_A^{s,t}(k, N)$ for a given finite field k, where k is also a comodule, A is a k-coalgebra and N is an A-comodule. To do so we will first go over all the definitions we will need, where we will show the duality between modules, comodules, algebras and coalgebras. Then, we will prove that there is a simple way to find Ext. We will show that in our construction of the coresolution, the Ext^s group becomes equal to Hom at filtration index s and that the Hom vector space is isomorphic to the vector space V_s, which can be found in a cofree comodule at filtration index s in the cofree coresolution of N. Afterwards, we will show all routines (computer programs) with which we can instruct the computer to calculate Ext. Finally, we will show an optimization in our routines and discuss how this work can be improved upon.

There exists computer programs which can calculate resolutions and Ext for specific algebras. Namely, Robert R. Bruner's Ext solver[1] can find Ext over the specific algebra A and $A(2)$ for $\mathbb{F}_2$, the mod 2 steenrod (sub)algebras. The Sage CAS[6] also supports calculating Ext over any mod p steenrod (sub)algebra.

Our work expends on this by calculating Ext for an arbitrary connected coalgebra over any finite field. This means we can also find Ext for the dual mod p steenrod (sub)coalgebra.

To motivate why we work with coalgebras and comodules. In the context of Algebraic topology one usually works with hopf algebras, which are both algebras and coalgebras. In general, it is more natural to look at the dual coalgebra structure of hopf algebras, because it appears to be easier to define the dual comultiplication for hopf algebras than its non-dual multiplication.

Since no Ext solver exists to work over coalgebras and comodules, our work offers a new tool that other researchers can use to calculate Ext for their coalgebras and comodules.

As a disclaimer, during this thesis I have worked together with Jacco Hijmans. Together we have made the computer program and studied the required theory. However, we have written our theses independently, where Jacco's thesis will go into the motivations behind writing the computer program.

2 Preliminaries

Before we can explain how we calculate Ext, we should first lay out some basic groundwork. Where our goal is to define coalgebras, comodules and graded vector spaces. We expect the reader to be familiar with concepts such as linear algebra, modules (with its morphisms) and finite fields.

Commutative Diagrams

We will present a lot of objects and definitions in terms of their commutative diagram(s). A commutative diagram says something about the relation different morphisms have. Such a diagram also makes it explicit how the domain and codomain between different morphisms relate to each other.

Here is an example of a commutative diagram:

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "g"] & B \arrow[d, "h"] \\
C \arrow[r, "i"] & D
\end{tikzcd}$$

If we want to explicitly write down all information in this diagram we would do the following. First we define the four morphisms that we have, with their corresponding domains. So we have morphisms:

$$f : A \to B,\ g : A \to C,\ h : B \to D,\ i : C \to D,$$

It must hold that, $h \circ f = i \circ g$ as morphisms from $A \to D$. This requirement is also where a commutative diagram takes it name from.

What always has to hold in a commutative diagram is that, in whatever object one starts and ends, it should not matter how one gets to that end object. So in this example it should not matter if we go from $A \to B \to D$ using f and h or if we go from $A \to C \to D$ using g and i. We should end up with the same function.

This makes commutative diagrams way more effective at conveying information than defining everything explicitly. Another reason we will use these commutative diagrams is because it will make dualizing (in a categorical sense) a lot more clear.

k-(bi)linear maps

Definition 2.1. A k-linear is a map f from the k-vector space V to the k-vector space W. This map f has to respect the following property for each i.

$$f(k \cdot v_1 + v_2) = k \cdot f(v_1) + f(v_2)$$

Definition 2.2. A k-bilinear map is a map $f : V_1 \times V_2 \to W$ where V_1, V_2, W are k-vector spaces, for which the following holds,

$$f(k \cdot v_1 + v_2, v_3) = k \cdot f(v_1, v_3) + f(v_2, v_3)$$
$$f(v_1, k \cdot v_2 + v_3) = k \cdot f(v_1, v_2) + f(v_1, v_3)$$

In other words, it is linear in each of its arguments and respects the underlying field.

Graded vector spaces

Definition 2.3. A graded vector space V is a direct sum $\bigoplus_{i \in I} V^i$ where V^i is a vector space itself and I is any index set.

In this thesis we will only consider I to be $\mathbb{N}$[1].

Definition 2.4. Any homomorphism between vector spaces should respect this grading. Meaning that for $f : W \to V$ with W and V being $\mathbb{N}$-graded vector spaces, we have that, $f(W^i) \subset V^i$. We will refer to morphisms of graded vector spaces as graded linear maps.

[1] In general I could really be any index set, but once we define graded algebras we will require that I be a commutative monoid, which $\mathbb{N}$ clearly is

Tensors

A tensor product is an object which gets build from two k-modules M and N[2]. We will call $M \otimes_k N$ the tensor product of M and N over k. This tensor product is constructed by first considering the free $\mathbb{Z}$-module of $M \times N$. Now consider the ideal generated by the following elements,

$$(m_1 + m_2, n) - (m_1, n) - (m_2, n)$$

$$(m, n_1 + n_2) - (m, n_1) - (m, n_2)$$

$$k \cdot (m, n) - (k \cdot m, n)$$

$$k \cdot (m, n) - (m, k \cdot n)$$

Then we take the quotient of the free $\mathbb{Z}$-module of $M \times N$ with this ideal, and this quotient is what we define as the tensor product of M and N.

Definition 2.5. The tensor product of M and N over k is defined,

$$M \otimes_k N = {}^{M \times_{\mathbb{Z}} N}\big/_{I}$$

Where $M \times_{\mathbb{Z}} N$ is the free Z-module, and I is the ideal we described above.

Note that the definition of a tensor product might seem somewhat complicated, but the important part to remember are its linearity properties, defined by the ideal with which we quotient. We see that we get the following equations which have to hold for the tensor,

$$m_1 + m_2 \otimes n = m_1 \otimes n + m_2 \otimes n$$

$$m \otimes n_1 + n_2 = m \otimes n_1 + m \otimes n_2$$

$$k \cdot (m \otimes n) = (k \cdot m \otimes n)$$

$$k \cdot (m \otimes n) = (m \otimes k \cdot n)$$

Note that an arbitrary element $x \in M \otimes_k N$ can be written as a finite sum, $\sum_i^n m_i \otimes n_i$. We have also not yet defined how this tensor product is a k-module. But by construction it is an additive group, and we can define the action as, $(k, \sum_i^n (m_i \otimes n_i) \mapsto \sum_i^n k_i \cdot (m_i \otimes n_i))$. It is easy to check that this satisfies the definition of a module. However, in general one should remember that the tensor product is a quotient, thus that any element could have multiple representatives.

If we have chosen a basis $\{v_i\}_{i \in I}$ for V and W with basis $\{w_j\}_{j \in J}$ then $V \otimes W$ has basis $\{v_i \otimes w_j\}_{i,j \in I,J}$.

There exists a canonical map from the direct product to the tensor product defined as follows, $(m, n) \mapsto m \otimes n$. It is clear that this map is bilinear by the properties of the tensor product.

Lemma 2.6. *There exists a bijective relation between k-bilinear maps from $M \times N$ to L and k-module homomorphism from $M \otimes_k N$ and L.*

$$\begin{tikzcd} M \times N \arrow[r, "i"] \arrow[dr, "\Phi"'] & M \otimes_k N \arrow[d, "\varphi"] \\ & L \end{tikzcd}$$

[2] Because k is a commutative ring, M and N are a bimodule over k in a canonical way. For brevity, we will ignore the subtleties that exist when talking about left and right comodules.

This means that we can express any bilinear map as a k-module homomorphism that originates from the tensor. We will not prove this lemma, the proof can be found in Dummit and Foote[2]. When the field is clear, we will not write the subscript k on the tensor anymore.

There is also a natural method to construct a tensor product between two graded vector spaces as follows.

Definition 2.7. If V and W and graded vector spaces, we define $V \otimes_k W$ in each degree i as,

$$(V \otimes W)^i = \bigoplus_{j+k=i} V^j \otimes W^k$$

(Associative) algebras

Because we only consider (associative) algebras over a field k, we can define an algebra as follows,

Definition 2.8. an algebra A is a vector space with a k-linear map (a k-module homomorphism) $* : A \otimes_k A \to A$ called the multiplication and a k-linear map $\eta : k \to A$ which we will call the unit map.

Note that we now have two multiplications. One from we got from the vector space which we will call the scalar multiplication, and the multiplication we have just defined. This multiplication also has to be associative, which means that the following diagram must commute,

$$\begin{tikzcd}
& (A \otimes A) \otimes A \arrow[dl, Rightarrow, shift left, "\cong"] \arrow[dr, "*\otimes 1"] & \\
A \otimes (A \otimes A) \arrow[ur, Rightarrow, shift left] \arrow[dr, "1\otimes *"'] & & A \otimes A \arrow[d, "*"] \\
& A \otimes A \arrow[r, "*"'] & A
\end{tikzcd}$$

To clarify the notation. $1 : A \to A$ is the identity function on A and $1 \otimes *$ means we will take the identity with respect to the left most A and multiply the two right A's. By this definition an algebra is also a ring, which might be a more familiar structure for the reader.

The unit function η defines how k "sits" inside our algebra. This function η should also make the following diagram commute.

$$\begin{tikzcd}
k \otimes_k A \arrow[rr, "\eta\otimes 1"] \arrow[dr, Rightarrow, shift left, "\cong"] & & A \otimes A \arrow[dl, "*"] \\
& A \arrow[ul, Rightarrow, shift left] \arrow[dl, Rightarrow, shift left, "\cong"] & \\
A \otimes_k k \arrow[rr, "1\otimes\eta"'] \arrow[ur, Rightarrow, shift left] & & A \otimes A \arrow[ul, "*"']
\end{tikzcd}$$

Where $A \cong A \otimes_k k$, which is easily deduced by the properties of the tensor product. We will assume throughout the rest of this thesis that any ring, thus also any algebra, should always have a multiplicative identity.

Example 2.9. We consider $M(k)^2$, the space of 2 by 2 matrices over the field k. The addition of matrices and scalar multiplication give it its vector space structure. The matrix multiplication (which we know to be associative) together with scalar multiplication of the identity matrix as the unit function completes its algebra structure. Note that matrix multiplication is not commutative in general!

This gives rise to a special class of algebras, namely commutative algebras. This requires the multiplication to be commutative as well. Thus,

$$\begin{tikzcd}
A \otimes A \arrow[r, "T"] \arrow[dr, "*"'] & A \otimes A \arrow[d, "*"] \\
 & A
\end{tikzcd}$$

Here, T is defined to be the twisting morphism, sending $a \otimes b \to b \otimes a$.

Example 2.10. $k[X]$ (the polynomials in one unknown over k) also has a natural algebra structure. Where we know that the polynomial multiplication is commutative.

Example 2.11. An example of a non-associative algebra (which we won't consider further in this thesis) is the vector space $\mathbb{R}^3$ where multiplication is defined to be the cross product.

Graded algebra

Up until now we defined an algebra A over a vector space, but we could also define it over a graded vector space.[3] What changes now is that the morphism of A are not linear maps anymore, but graded linear maps, thus have to respect the grading. We also define the unit function from $\eta : k \to A$ to be graded with respect to zero. Thus, that $\eta(k) \subset A^0$.

The multiplication of the algebra, which is defined as $* : A \otimes A \to A$, now also has to respect the grading. This means that in each degree i we have to have that $*(\bigoplus_{j+k=i} A^j \otimes A^k) \subset A^i$, or in other words that $A^j A^k \subset A^{j+k}$.

Example 2.12. Again $k[X]$ gives us a perfect example. Here we define the grading to be equal to the degree of the homogenous elements of the polynomial. Thus, that $X^t \in (k[X])^t$. We also now that when we multiply two polynomials, the grades get summed together, thus our multiplication indeed respects the grading.

There is also a special class of algebras which we will see later when we are building the algorithm.

Definition 2.13. We call an algebra connected if η is an isomorphism with respect to A^{0}[4].

Note that in a connected algebra we don't need to explicitly give η as it can be naturally constructed from the isomorphism with A^0 and the fact that k always has degree zero.

[3] Previously we only required the index set to be a set, now we require that it is also a commutative monoid. However, as we only consider $\mathbb{N}$, which is a commutative monoid, we are fine.

[4] There is also a notion of connectiveness for algebras, which requires that there is no negative grading. But this automatically holds for $\mathbb{N}$.

(Coassociative) coalgebra

When we define a coalgebra the reader should keep special notice of the resemblance between the definition of an algebra and that of a coalgebra.

Definition 2.14. A coalgebra is a k-vector space A together with two k-linear maps.

$$\Delta : A \to A \otimes A \quad \text{and} \quad \varepsilon : A \to k$$

We will refer to Δ as the comultiplication and ε as the counit. For these two functions the following two diagrams should commute.

$$\begin{tikzcd}
A \arrow[r, "\Delta"'] \arrow[d, "\Delta"] & A \otimes A \arrow[d, "1\otimes\Delta"'] \\
A \otimes A \arrow[r, "\Delta\otimes 1"] & A \otimes A \otimes A
\end{tikzcd}$$

and,

$$\begin{tikzcd}
k \otimes_k A \arrow[dr, Leftrightarrow, "\cong"'] & & A \otimes A \arrow[ll, "\varepsilon\otimes 1"'] \\
& A \arrow[ur, "\Delta"'] \arrow[dr, "\Delta"] \arrow[dl, Leftrightarrow, "\cong"] & \\
A \otimes_k k & & A \otimes A \arrow[ll, "1\otimes\varepsilon"]
\end{tikzcd}$$

We say that Δ is coassociative and ε is a counit. We see that these diagrams and functions look like those for an algebra. What has changed is that the arrows in the diagrams have been flipped. This process is not something unique to algebras and coalgebras, but can be generalized to any such a construction, and is what we will be referring to as dualizing.

A coalgebra is cocommutative if the following diagram commutes, where T is the twisting morphism again,

$$\begin{tikzcd}
A \arrow[r, "\Delta"] \arrow[dr, "\Delta"] & A \otimes A \arrow[d, "T"] \\
& A \otimes A
\end{tikzcd}$$

Example 2.15. $k[X]$ can also be made into a coalgebra by letting,

$$\Delta(X^n) = \sum_{l=0}^{n} \binom{n}{l} X^l \otimes X^{n-l}$$

$$\varepsilon(X^n) = \begin{cases} 1 \text{ if n} = 0 \\ 0 \text{ otherwise} \end{cases}$$

This example is also cocommutative. Now we have an example which is both an algebra and a coalgebra, such a structure is a called a hopf algebra if the two structures are compatible with each other.

Graded coalgebra

Now for a coalgebra we take the exact same approach as we just did, where the object is now a graded vector space and every morphism is now a graded linear map.

Definition 2.16. We will call a coalgebra connected if $\varepsilon : A^0 \to k$ is an isomorphism.

Note that in a connected coalgebra we don't need to explicitly give ε as it can be naturally constructed from the isomorphism and the fact that k has degree zero.

Example 2.17. The comultiplication we had previously defined on $k[X]$,

$$\Delta(X^n) = \sum_{l=0}^{n} \binom{n}{l} X^l \otimes X^{n-l}$$

clearly respects the grading. Thus, we can extend $k[X]$ to a graded coalgebra.

Dualizing

Up until now we have mostly been dualizing structures/concepts. But the dualizing the reader is most familiar with is most probably the dualization of vector spaces.

This takes a vector space V, to the space of functionals on V namely $V^* := \{\varphi : V \to k\}$, where each φ is a linear map. We also know that for finite dimensional V, V is isomorphic to its dual V^*. For the infinite case V need not be isomorphic to V^*. It also holds that $(V \otimes W)^* \cong V^* \otimes W^*$ when V and W are finite.

Now when we look at any morphism between vector spaces $f : V \to W$, we see that if we dualize there exists an induced morphism f^* defined as follows. $f^* : W^* \to V^*$ by $\phi \mapsto \phi \circ f$. This is legal because both ϕ and f were linear maps and the resulting linear map goes from $V \to k$ lies in V^*.

Now if we look at this for our k-algebra A, we see that if this algebra is finite we can dualize it to make it into a coalgebra. Because the multiplication $* : A \otimes A \to A$ is a linear map we see that after dualizing this becomes a map $*^* : A^* \to A^* \otimes A^*$. The same happens for the unital function and if one dualizes all the diagrams we see that this dual is a coalgebra.

This means that (for a finite algebra) we can always consider its dual coalgebra, which might be easier to work with, and vice versa.

We will call a graded vector space V of finite type if each $V^i \in V$ is finite. For graded vector spaces of finite type one can dualize by taking the dual of each V^i, thus $V^* = \bigoplus_{i \in \mathbb{N}} (V^i)^*$.

Modules and comodules

Definition 2.18. An A-(left)module M is a vector space together with a linear map $* : A \otimes_k M \to M$, which we call the action of A, the algebra, on M. The action on M should also make the following diagrams commute,

$$\begin{tikzcd}
A \otimes A \otimes M \arrow[r, "1\otimes *_M"] \arrow[d, "*_A\otimes 1"'] & A \otimes M \arrow[d, "*_M"] \\
A \otimes M \arrow[r, "*_M"'] & M
\end{tikzcd}$$

$$\begin{tikzcd}
k \otimes_k M \arrow[rr, "\eta\otimes 1"] \arrow[dr, leftrightarrow, "\cong"] & & A \otimes M \arrow[dl, "*"] \\
& M &
\end{tikzcd}$$

Example 2.19. From the above definitions and diagrams we can see that any algebra is also a module. This can be seen by replacing M with A in the diagrams and noticing that the diagrams for a module are a subset of the diagrams we had for an algebra.

A morphism of A-modules, which is also a linear map, should commute the following diagrams, where $f : M \to N$.

$$\begin{tikzcd}
A \otimes M \arrow[r, "*_M"] \arrow[d, "1\otimes f"'] & M \arrow[d, "f"] \\
A \otimes N \arrow[r, "*_N"'] & N
\end{tikzcd}$$

The definition of an A-(left)comodule is completely dual to that of an A-module, where A is now a coalgebra.

Definition 2.20. A comodule M is a vector space with a linear map, $\Delta : M \to A \otimes M$ which adheres to the dual of the above diagrams. Meaning,

$$\begin{tikzcd}
M \arrow[r, "\Delta_M"] \arrow[d, "\Delta_M"'] & A \otimes M \arrow[d, "\Delta_A\otimes 1"] \\
A \otimes M \arrow[r, "1\otimes\Delta_M"'] & A \otimes A \otimes M
\end{tikzcd}$$

and,

$$\begin{tikzcd}
k \otimes_k M & & A \otimes M \arrow[ll, "\varepsilon\otimes 1"'] \\
& M \arrow[ul, leftrightarrow, "\cong"] \arrow[ur, "\Delta"'] &
\end{tikzcd}$$

This can be intuitively understood to mean that the coaction on M is compatible with the comultiplication and counit of A.

Morphisms of comodules, which are also linear maps, should have that,

$$\begin{tikzcd}
M \arrow[r, "\Delta_M"'] \arrow[d, "f"] & A \otimes M \arrow[d, "1\otimes f"'] \\
N \arrow[r, "\Delta_N"] & A \otimes N
\end{tikzcd}$$

From now on, we will always consider (co)modules to be left A-(co)modules, where A is a (co)algebra.

Graded (co)modules

A graded (co)module is just a (co)module where the object is now a graded vector space and the maps are graded linear maps. In the graded setting we must have that $A^iM^j \subset M^{i+j}$ for modules and $\Delta(M^k) \subset \bigoplus_{i+j=k} A^i \otimes M^j$ for comodules, which follows from the fact that we have a graded linear map.

To avoid constantly writing (graded) everywhere, one can always assume that when it is not explicitly mentioned, the definitions and proofs work for both graded and non-graded objects and morphisms. When there is a special case for either the graded or non-graded case, it will be mentioned.

Example 2.21. When A is a connected coalgebra we can make k into a comodule. k is already clearly a vector space over k. The coaction on k is given by $\Delta_k : x \mapsto 1 \otimes x \in A^0$.

Subcomodules and quotients comodules

Definition 2.22. A **subcomodule** N of a comodule M is a subspace of the vector space of M for which there exist a lift,

$$\begin{tikzcd}
N \arrow[r, dashed] \arrow[d, hook, "i"] & A \otimes N \arrow[d, "1\otimes i"] \\
M \arrow[r, "\Delta_M"'] & A \otimes M
\end{tikzcd}$$

between N and $A \otimes N$. In other words, that the coaction stays within N, thus for $n \in N \subset M$ it holds that $\Delta(n) \in A \otimes N$.

Definition 2.23. For a comodule M, the **quotient comodule** Q is an object with a projection $\pi : M \twoheadrightarrow Q$. There should also be a natural lift from $Q \to A \otimes Q$ and the following diagram should commute,

$$\begin{tikzcd}
 & \ker\pi \arrow[dl, hook'] \arrow[dd, bend left, "0"] \\
M \arrow[r, "\Delta_M"] \arrow[d, two heads, "\pi"'] & A \otimes M \arrow[d, "1\otimes\pi"'] \\
Q \arrow[r, dashed] & A \otimes Q
\end{tikzcd}$$

This guarantees that for $m \in M$, if $\pi(m) = 0$ then $(1 \otimes \pi)(\Delta_M(m)) = 0$. In words, the coaction for elements in the $\ker\pi$ remain zero in $A \otimes Q$. This way we can also see that Q is a quotient space with

$Q = {}^{M}\!/_{\ker \pi}$. It is trivial to see that $\ker \pi$ is a subcomodule. In general, we can make a quotient comodule Q such that $Q = {}^{M}\!/_{N}$, where N is any subcomodule of M.

Hopf algebra

A hopf algebra[5] H is a k-vector space which is both an algebra and a coalgebra and for which the following diagrams commute. These diagrams define the way in which the two structures are "compatible" with each other. We won't require the properties of these diagrams, we only provide them for completeness.

$$\begin{tikzcd}
H \otimes H \arrow[r, "*"] \arrow[d, "\Delta\otimes\Delta"'] & H \arrow[r, "\Delta"] & H \otimes H \\
H \otimes H \otimes H \otimes H \arrow[rr, "1\otimes T\otimes 1"'] & & H \otimes H \otimes H \otimes H \arrow[u, "*\otimes*"']
\end{tikzcd}$$

$$\begin{tikzcd}
H \otimes H \arrow[r, "*"] \arrow[d, "\varepsilon\otimes\varepsilon"'] & H \arrow[d, "\varepsilon"] \\
k \otimes k \arrow[r, leftrightarrow, "\cong" description] & k
\end{tikzcd}$$

$$\begin{tikzcd}
H \arrow[r, "\Delta"] & H \otimes H \\
k \arrow[u, "\eta"] \arrow[r, leftrightarrow, "\cong" description] & k \otimes k \arrow[u, "\eta\otimes\eta"']
\end{tikzcd}$$

$$\begin{tikzcd}
k \arrow[dr, "\eta"] \arrow[dd, "1"'] & \\
& H \arrow[dl, "\varepsilon"] \\
k &
\end{tikzcd}$$

Now we are not really interested in the meaning or purpose of all these diagrams. What we are mostly interested in is the fact that a hopf algebra is self-dual[6]. This means that if there exist a dual of H, then that dual H^* is automatically also a hopf algebra. That means that when something is true about H^* then it is also true for H.

The reason we are interested in hopf algebras is that most coalgebras we get are actually hopf algebras, the more specific reason with regard to algebraic topology is given in Jacco Hijmans thesis. It is also more natural to work with coalgebras in context of Algebraic Topology.

Moreover, the multiplication $*$ is usually more complicated to express than its dually induced comultiplication Δ. One only has to describe what the coaction does on the ring generators for H.

[5] Milnor moore's[5] paper calls this definition a hopf algebra, but other sources will call this a bialgebra. And refer to it as a hopf algebra if it has an antipode. We are primarily interested in the self duality of this structure and won't bother with all the other details.

[6] This was already the case when we merely considered an object H which is both an algebra and a coalgebra

And because it is both an algebra and coalgebra, its coaction is also a ring homomorphism. Thus, we know what the coaction does for every element in H.

In the case that H is finite, a dual H^*. One should always remember that a finite algebra and its dual (finite) coalgebra are really two ways of looking at the same information.

3 Calculating Ext

We have seen some basic concepts we will need in order to define Ext. Now we will start of with some more specific concepts we will need in order to define Ext. Afterwards we will show how we can find Ext algorithmically.

Exactness

Consider the following sequence of A-comodules F_i with comodule morphisms f_i,

$$F_1 \xrightarrow{f_1} F_2 \xrightarrow{f_2} F_3 \xrightarrow{f_3} \cdots$$

Definition 3.1. We call this sequence exact if for each i we have that im $(f_i) = \ker(f_{i+1})$. Such a sequence can be finite or infinite.

Note that this definition works for different types of structures, for example, groups, rings and in our case A-comodules. In more generality one can speak about exactness when the category one looks at is an abelian category.

Example 3.2. In the category of k-vector spaces where morphisms are linear maps, we have that the following sequence $0 \to X \xrightarrow{f} Y$ is exact if and only if the function $f : X \to Y$ is injective. This follows immediately from the fact that for injective linear maps the kernel is equal to zero. And by linearity im $(0) = 0 \in X$. Thus $\ker f = 0 =$ im $(0) \subset X$

Example 3.3. In the same category we can also have the following sequence $X \xrightarrow{g} Y \to 0$ which is exact if and only if g is surjective.

Definition 3.4. An exact sequence with the following form is called a short exact sequence,

$$0 \to X \xrightarrow{f} Y \xrightarrow{g} Z \to 0$$

Such sequences are found all over homological algebra and algebra in general. We won't pay special attention to these sequences in this thesis, but note that these sequences are often seen.

(Co)free (Co)module

Definition 3.5. Let A be an algebra. A free A-module M is any module M which can be written as $A \otimes_k V$, with V a vector space, and for which the action is defined using the following diagram,

$$\begin{tikzcd}
A \otimes A \otimes V \arrow[r, "*_A \otimes 1"] \arrow[d, Leftrightarrow] \arrow[dr, phantom, "\Vert"] & A \otimes V \arrow[d, Leftrightarrow] \\
A \otimes M \arrow[r, "*_M"'] & M
\end{tikzcd}$$

Thus on elements it acts as $*_M(a \otimes m) = *_M(a \otimes (\sum a_i \otimes v_i)) = \sum(*_A(a \otimes a_i) \otimes v_i)$.

A more well known definition of a free module uses a basis set $S \subset M$ for which S is linearly independent and generates the whole of M. But one should notice that a basis of V, with the action we defined and the fact that we tensor with A, fulfills exactly this role!

Now a cofree comodule is the same, but dualized.

Definition 3.6. Let A be a coalgebra. An A-comodule M is called cofree if $M \cong A \otimes_k V$ and its coaction be defined as follows,

$$\begin{tikzcd}
A \otimes V \arrow[r, "\Delta_A \otimes 1"] \arrow[d, Leftrightarrow, "\cong"] \arrow[dr, phantom, "\|"] & A \otimes A \otimes V \arrow[d, Leftrightarrow, "\cong"] \\
M \arrow[r, "\Delta_M"'] & A \otimes M
\end{tikzcd}$$

Coresolution

A resolution[7] of M, where M is an A-module and A an algebra, is an exact sequence,

$$\cdots \xrightarrow{d_3} F_2 \xrightarrow{d_2} F_1 \xrightarrow{d_1} F_0 \xrightarrow{\epsilon} M \longrightarrow 0$$

where every F_i is a free module. This sequence need not be infinite, but in most cases will be.

Example 3.7. Consider A as an A-module. Note that $A \cong A \otimes e_1$ is trivially free, thus we can make the following resolution. $0 \to A \xrightarrow{1} A \to 0$. This sequence is clearly exact because the identity is both injective and surjective. Notice that 0 is also a free module, and we have a resolution of A.

Now a coresolution of N, where N is an A-comodule, is again dual to the definition we have just given.

Definition 3.8. A coresolution of N is an exact sequence where each F_i is a cofree comodule,

$$0 \longrightarrow N \xrightarrow{\epsilon} F_0 \xrightarrow{d_1} F_1 \xrightarrow{d_2} F_2 \xrightarrow{d_3} \cdots$$

Hom

To study the properties of a comodules N we can look at maps from another comodule to N. This can be useful when the comodule N self is too complex to understand. But there are a lot of reasons why one would want to do this. The way in which we express this is by using the Hom functor

A functor is like a function but instead of sending elements to elements, it sends an object to a new object in a possibly different category. We will consider the specific functor $\mathrm{Hom}_A(D, \_)$,[8] where A is a coalgebra and D is a comodule. When you apply this functor to an A-comodule M we get the object $\mathrm{Hom}_A(D, M)$, which is the collection of all A-comodule morphisms between D and M.

[7] One officially requires that F_i is projective, but any free module is also projective.

[8] One can describe the Hom functor in general for any category, but that would require making concepts from category theory more concrete than we want to, and we will try to avoid that in this thesis.

Definition 3.9.

$$\mathrm{Hom}_A(D,M) = \{\textbf{\textit{A}-comodule morphisms } \mid D \to M\}$$

On first sight this is merely a set. We have lost the A-module structure we had on M and D. However, we can add the k-vector space structure again as follows. For two morphisms $f, g \in \mathrm{Hom}_A(D,M)$ let $(f+g) := x \mapsto f(x)+g(x)$ and $(k \cdot f) := x \mapsto k \cdot f(x)$.

Lemma 3.10. *For $f, g \in Hom_A(D,M)$ and k an element of our field, $(f + k \cdot g)$ is an A-comodule morphism. This makes $Hom_A(D,M)$ into a k-vector space.*

Proof. It is clear that this map is still a linear map. What we will verify is that it respects the coaction, thus checking that the following diagram commutes,

$$\begin{tikzcd}
D \arrow[r, "\Delta_D"] \arrow[d, "f+g"] & A \otimes D \arrow[d, "1\otimes(f+g)"] \\
M \arrow[r, "\Delta_M"'] & A \otimes M
\end{tikzcd}$$

First we see that for any $y \in A \otimes D$, y can be written as $\sum_i a_i \otimes d_i$. Now we see that

$$(1 \otimes f)(y) + k \cdot (1 \otimes g)(y) = \sum_i a_i \otimes f(d_i) + k \cdot \left(\sum_i a_i \otimes g(d_i) \right) =$$

$$\sum_i a_i \otimes f(d_i) + \sum_i a_i \otimes (k \cdot g(d_i)) = \sum_i a_i \otimes (f(d_i) + k \cdot g(d_i)) = (1 \otimes (f + k \cdot g))(y)$$

We recall that Δ_M and Δ_N are linear maps and see for every $d \in D$,

$$\Delta_M((f + k \cdot g)(d)) = \Delta_M(f(d)) + k \cdot \Delta_M(g(d)) =$$

$$(1 \otimes f)(\Delta_D(d)) + k \cdot (1 \otimes g)(\Delta_D(d)) = (1 \otimes (f + k \cdot g))(\Delta_D(d))$$

This shows that the diagram commutes and thus that $(f + k \cdot g) \in \mathrm{Hom}_A(D,M)$ □

Example 3.11. We have already seen an example of using the Hom functor in this thesis, namely dualizing vector spaces. V^* is defined as all k-linear maps from $V \to k$. But we could also express this as $\mathrm{Hom}_k(V,k)$. Note that we are now in the category of k-vector spaces and linear maps, not A-comodules and A-comodule morphisms.

Definition 3.12 (Induced morphisms)**.** Let f be a morphism between A-comodules M and N. Then there also exists an induced morphism $f_* : \mathrm{Hom}_A(D,M) \to \mathrm{Hom}_A(D,N)$ sending $\phi \to f \circ \phi$. Because f and ϕ are both morphisms of A-comodules, their composition is again an A-module morphism.

When we are considering graded comodules there also exists a natural grading on Hom as follows.

Definition 3.13.

$$\mathrm{Hom}_A(D,M) := \bigoplus_{t \in \mathbb{N}} \mathrm{Hom}_A^{graded}(D[t], M)$$

Where $\mathrm{Hom}_A^{graded}(D[t], M)$ are graded A-comodule morphisms and $D[t]$ is almost the same as D, but every element in D has had its grading shifted by t.

$\mathbf{Ext}_A(D,M)$

$\text{Ext}_A(D,M)$ is defined over a coalgebra A and two A-comodules D and M. We construct Ext by first making a cofree coresolution of M.

$$\begin{tikzcd} 0 \arrow[r] & M \arrow[r, "\epsilon"] & F_0 \arrow[r, "d_1"] & F_1 \arrow[r, "d_2"] & F_2 \arrow[r, "d_3"] & \cdots \end{tikzcd}$$

Then we remove M from this sequence, and we apply the $\text{Hom}_A(D, \_)$ functor to the remaining sequence to give us,

$$\begin{tikzcd} 0 \arrow[r] & \text{Hom}_A(D,F_0) \arrow[r, "d_1^*"] & \text{Hom}_A(D,F_1) \arrow[r, "d_2^*"] & \text{Hom}_A(D,F_2) \arrow[r, "d_3^*"] & \cdots \end{tikzcd}$$

Now we define,

Definition 3.14.

$$\text{Ext}_A^s(D,M) = {}^{\ker d_{s+1}^*}\!\big/_{\text{im } d_s^*}$$

For the case where $s = 0$ we define $Ext_A^0(D,M) = \ker d_1^*$.

It should be noted that Ext does not depend on the choice of coresolution. This is not trivial to see, a proof of this fact can be found in Dummit and Foote [2, Chapter 17, Theorem 6]. The action of taking the quotient of the kernel and the image, is something which occurs more often in math and especially algebraic topology. Such constructions are called cohomology groups.

When we consider graded modules we also gain an extra index t where Ext now becomes,

$$\text{Ext}_A^{s,t}(D,M) = \left({}^{\ker d_{s+1}^*}\!\big/_{\text{im } d_s^*}\right)^t$$

3.1 Algorithmically making a coresolution

To construct a coresolution we will need two ingredients.

- We need to be able to construct the cokernel.
- We need to inject from any comodule to a cofree comodule.

Constructing the cokernel

Consider any morphism $f : M \to N$, where M and N are A-comodules[9].

Definition 3.15. The cokernel of f is defined to be the object which makes the following sequence exact.

$$\begin{tikzcd} 0 \arrow[r] & \ker f \arrow[r, hook, "i"] & M \arrow[r, "f"] & N \arrow[r, two heads, "\pi"] & \text{coker } f \arrow[r] & 0 \end{tikzcd}$$

Note that $\ker f$ injects to M and that the image of that injection is by definition equal to the kernel of f. $\ker f$ is also clearly a subcomodule, and thus a comodule (this means it is a valid object to use in this sequence). This makes the first part of the sequence exact.

[9] This works for any abelian category

Lemma 3.16. *For A-comodules the cokernel of f is isomorphic to the quotient ${}^{N}\!/_{im\ f}$*[10]

Proof. First we should quickly verify whether this quotient actually makes sense, and for that we will check if im f is a subcomodule. So we require that if $n \in \text{im } f$ then $\Delta_N(n) \in A \otimes \text{im } f$. Remember that for an A-comodule morphism the following diagram commutes,

$$\begin{tikzcd} M \arrow[r, "\Delta_M"] \arrow[d, "f"'] & A \otimes M \arrow[d, "1\otimes f"] \\ N \arrow[r, "\Delta_N"'] & A \otimes N \end{tikzcd}$$

Because $n \in \text{im } f$ we have that $\exists m \in M$ such that $f(m) = n$. Now by the diagram we have that

$$\Delta_N(n) = \Delta_N(f(m)) = (1 \otimes f)(\Delta_M(m)) = (1 \otimes f)\left(\sum_i^n a_i \otimes m_i\right) = \sum_i^n a_i \otimes f(m_i)$$

This shows that $\Delta_N(n) \in A \otimes \text{im } f$, and thus im f is a submodule with which we can take a quotient.

Call π the projection to the quotient ${}^{N}\!/_{\text{im } f}$. We now readily see that im $f = \ker \pi$ by definition of the quotient. We also know that the projection is always surjective. This proves that the whole sequence is indeed exact. □

Making an injection to a cofree comodule

Before we construct an injection from M to a cofree comodule we will show there exists a bijection between linear maps to V and A-comodule morphisms to $A \otimes V$.

Lemma 3.17. *There exists a bijective relation between $Hom_A(M, A \otimes V) \leftrightarrow Hom_k(M, V)$, A-comodule morphisms to cofree $A \otimes V$ and k-linear maps to V.*

The only thing we will need for now is the method to construct an A-comodule morphism from a k-linear map to V, later we will prove the full statement.

Proof. This map is constructed as follows, for $f : M \to V \in \text{Hom}_k(M, V)$, we define $\bar{f} := (1 \otimes f) \circ \Delta_M$.

$$M \xrightarrow{\Delta_M} A \otimes M \xrightarrow{1\otimes f} A \otimes V$$

This is clearly linear, but we verify if this is actually an A-comodule morphism. For this we will use the properties of the coaction for a cofree comodule. Consider the following diagram,

$$\begin{tikzcd} V & M \arrow[l, "f"'] \arrow[r, "\Delta_M"] \arrow[d, "\Delta_M"'] & A \otimes M \arrow[d, "1\otimes\Delta_M"] \\ & A \otimes M \arrow[r, "\Delta_A\otimes 1"] \arrow[d, "1\otimes f"'] & A \otimes A \otimes M \arrow[d, "1\otimes 1\otimes f"] \\ & A \otimes V \arrow[r, "\Delta_A\otimes 1"] & A \otimes A \otimes V \end{tikzcd}$$

[10] Such a quotient exists in general for abelian categories

Because M is a comodule, the top part of this diagram commutes. We need to verify that for $x = \sum(a_i \otimes m_i) \in A \otimes M$, $(1 \otimes 1 \otimes f)(\Delta_A \otimes 1)(x) = (\Delta_A \otimes 1)(1 \otimes f)(x)$, the commutativity of the bottom part.

$$(1 \otimes 1 \otimes f)(\Delta_A \otimes 1)(x) = (1 \otimes 1 \otimes f)\left(\sum \Delta_A(a_i) \otimes m_i\right) =$$

$$\left(\sum \Delta_A(a_i) \otimes f(m_i)\right) = (\Delta_A \otimes 1)\left(\sum a_i \otimes f(m_i)\right) =$$

$$(\Delta_A \otimes 1)(1 \otimes f)\left(\sum a_i \otimes m_i\right) = (\Delta_A \otimes 1)(1 \otimes f)(x)$$

Now to check if $\bar{f}$ is an A-comodule morphism, we check if the following diagram commutes,

$$\begin{tikzcd}
M \arrow[r, "\Delta_M"] \arrow[d, "\bar{f}"'] & A \otimes M \arrow[d, "1\otimes\bar{f}"] \\
A \otimes V \arrow[r, "\Delta_A\otimes 1"] & A \otimes A \otimes V
\end{tikzcd}$$

By using the previous diagram we see that the only thing which we should check is $1 \otimes \bar{f} = (1 \otimes 1 \otimes f)(1 \otimes \Delta_M)$.

We write $1 \otimes \bar{f}$ as $1 \otimes ((1 \otimes f) \circ \Delta_M)$. Now we do the same as we have just seen for the commutativity of the previous diagram to show the commutativity of the diagram.

With that we have shown that $\bar{f}$ is an A-comodule morphism. □

Now, for any comodule M, we can always create an injection to a cofree comodule. We construct this cofree comodule together with the morphism iteratively. Starting this process we create a morphism $i : M \to 0 \cong A \otimes 0$, a zero cofree comodule.

Now we describe a step in the iteration, assume $i : M \to A \otimes \{e_1, \cdots, e_n\}$. Here we abuse notation a little, with $\{e_1, \cdots, e_n\}$ we mean the span of the vector space by this basis.

- First we take the lowest graded element $q \neq 0 \in \ker i$, (if we are in the non-graded case, take any element). If such a q does not exist, $\ker i = 0$ and we are done.
- Make a morphism $f : M \to A \otimes e_{n+1}$ for which $f(q) \neq 0$. This guarantees that q won't be in the kernel.
- Now we define $\bar{i}$ to be $M \to A \otimes \{e_1, \cdots, e_n\} \cup \{e_{n+1}\}$, with $\bar{i}(m) := i(m) + f(m)$.
- We repeat the process with $i = \bar{i}$ until the $\ker i = 0$.

Here we should prove a few things, first that such a morphism f actually exists and that summing the i and f is legal. We should also check that the kernel is actually shrinking. To guarantee that our algorithm terminates we will have to assume that M is of finite type.

First that summing is legal,

Proof. To show this we will extend i and f to be maps from $M \to A \otimes \{e_1, \cdots, e_n, e_{n+1}\}$. Call these maps $\tilde{i}$ and $\tilde{f}$. Now these are clearly still linear maps, because we just ignore all the added basis elements. What we will verify is that they are still A-comodule morphisms. Thus, that the following diagram commutes,

$$\begin{tikzcd}
M \arrow[r, "\Delta_M"] \arrow[d, "\tilde{i}"'] & A \otimes M \arrow[d, "1\otimes\tilde{i}"] \\
A \otimes \{e_1, \cdots, e_n, e_{n+1}\} \arrow[r, "\Delta_A \otimes 1"'] & A \otimes A \otimes \{e_1, \cdots, e_n, e_{n+1}\}
\end{tikzcd}$$

We know that this commutes for our normal i. This implies that $\tilde{i}(M)$ doesn't go outside $A \otimes \{e_1, \cdots, e_n\}$, meaning $\tilde{i}(M) \subset A \otimes \{e_1, \cdots, e_n\}$. So now we can immediately conclude that because i commutes, $\tilde{i}$ must also commute. The same works for f.

When we were proving Lemma 3.10 we have already seen that the sum of two A-comodule morphisms is still an A-comodule morphism. This proves that the sum we are taking is legal.

□

We show that the kernel is shrinking,

Proof. First we note that the element q we chose to construct our f, won't appear in the kernel in the next iteration. This follows from the fact that $\tilde{i}(q) = i(q) + f(q) = 0 + f(q) \neq 0$. Now the question remains, are there elements in $\ker \tilde{i}$ which were not in the $\ker i$. This would mean that there exists an $m \in M$ such that $\tilde{i}(m) = 0$ but $i(m) \neq 0$. Let $i(m) = \sum_j^n a_j \otimes e_j$ and $f(m) = a_{n+1} \otimes e_{n+1}$. Now,

$$0 = \tilde{i}(m) = i(m) + f(m) = \sum_i^n a_i \otimes e_i + a_{n+1} \otimes e_{n+1}$$

This would imply that $\sum_i^n a_i \otimes e_i = -a_{n+1} \otimes e_{n+1}$ but because e_i is a basis, this can only be true if all a_i's are zero. But this contradicts our assumption that $i(m) \neq 0$.

This concludes that our kernel is shrinking at each step. Because M is of finite type, the algorithm eventually terminates. □

Now to show that such an f actually exists,

Proof. We will start by constructing a linear map from $M \to k$. The map we want has to at least send q to $1 \in k$. In the graded setting we have to be a bit more careful. Here we will define the map to be from $M \to \{e_{n+1}\}$, a 1-dimensional graded k-vector space where e_{n+1} has the same degree as q.

The element q can be written as $\sum_i k_i \cdot m_i$, where m_i is a basis element of M. Without loss of generality we will assume that $k_1 = 1$, if not we can always look at $k_1^{-1} \cdot q$ as our element from $\ker i$.

For a linear map it is enough to define what the map does on basis elements of M. We define $\bar{f}(m_1) = e_{n+1}$ and $\bar{f}(m_{i\neq 1}) = 0$. This gives us that $\bar{f}(q) = e_{n+1}$.

We now use Lemma 3.17 to get a map $f : M \to A \otimes \{e_{n+1}\}$. All we need to verify now is that $f(q) \neq 0$. Because M is a comodule we know that,

$$\begin{tikzcd}
k \otimes_k M & & A \otimes M \arrow[ll, "\varepsilon\otimes 1"'] \\
& M \arrow[ul, leftrightarrow, "\cong"] \arrow[ur, "\Delta"'] &
\end{tikzcd}$$

This diagram says that $q \cong 1 \otimes q = (\varepsilon \otimes 1)(\Delta_M(q))$. Consider the following diagram,

$$\begin{tikzcd}
M \arrow[rr, "\Delta_M"] \arrow[dd, "\bar{f}"'] \arrow[dr, Leftrightarrow, "\cong"] & & A \otimes M \arrow[dl, "\varepsilon\otimes 1"'] \arrow[dd, "1\otimes\bar{f}"] \\
& k \otimes M \arrow[d, "1\otimes\bar{f}"'] & \\
V \arrow[r, Leftrightarrow] & k \otimes V & A \otimes V \arrow[l, "\varepsilon\otimes 1"']
\end{tikzcd}$$

The top triangle commutes because M is a comodule and the left square commutes trivially. The right square commutes because $(\varepsilon \otimes 1) \circ (1 \otimes f) = \varepsilon \otimes f = (1 \otimes f) \circ (\varepsilon \otimes 1)$. This gives us that $\bar{f} \cong (\varepsilon \otimes 1) \circ (1 \otimes \bar{f}) \circ \Delta_M = (\varepsilon \otimes 1) \circ f$. Because $\bar{f}(q) \neq 0$ we must have that $f(q) \neq 0$. □

This concludes everything we had to check. Now we can construct an injection from a comodule M to a cofree comodule $A \otimes V$.

Combining both

So if we combine these two techniques we see that if we have a morphism, $f : M \to N$. We can iteratively, extend this to a sequence with cofree comodules as follows,

$$\begin{tikzcd}
M \arrow[r, "f"] & N \arrow[rr, dashed, "i\circ\pi"] \arrow[dr, "\pi"'] & & F \\
& & Q \cong {}^{N}/_{\text{im } f} \arrow[ur, "i"'] &
\end{tikzcd}$$

Now because i is injective we know that $\ker i = 0$ and thus that $\ker i \circ \pi = \ker \pi = \text{im } f$. And we can conclude that the sequence,

$$M \xrightarrow{f} N \xrightarrow{i\circ\pi} F$$

is exact.

Now we can also do this for the special case where $M = 0$ and N is the comodule for which we make a coresolution. We can do it in the following way, and this gives us the following coresolution,

$$\begin{tikzcd}
0 \arrow[r, "0"] & N \arrow[rr, "\varepsilon"] \arrow[dr, dashed, "\pi"'] & & F_0 \arrow[rr, "d_1"] \arrow[dr, dashed, "\pi"'] & & F_1 \arrow[rr, "d_2"] \arrow[dr, dashed, "\pi"'] & & F_2 \arrow[r, dashed] & \cdots \\
& & Q_0 \arrow[ur, dashed, "i"'] & & Q_1 \arrow[ur, dashed, "i"'] & & Q_2 \arrow[ur, dashed, "i"'] & &
\end{tikzcd}$$

Constructing this quotient module and creating an injection to a cofree comodule is the most important part of the algorithm we have written. How we instructed the computer to calculate this is what we will explain after we show how we can deduce Ext in a special case.

3.2 Ext for this coresolution

What we will see is that using our algorithm, the induced morphisms will be zero and when we Hom with A-comodule k we get that $\mathrm{Hom}_A(k, A \otimes V) \cong V$[11]. This makes it very easy to calculate Ext, because $\mathrm{Ext}(k, M)^s \cong V_s$, where $F_s = A \otimes V_s$ the cofree comodule in index s in the coresolution of M.

To start, we will give the full proof of Lemma 3.17, which says there is a bijection between $\mathrm{Hom}_A(M, A \otimes V) \leftrightarrow \mathrm{Hom}_k(M, V)$.

Proof. We already showed the construction from right to left, which takes $f : M \to V$ to $(1 \otimes f) \circ \Delta_M$.

$$F : \left(M \overset{f}{\to} V\right) \mapsto \left(M \overset{\Delta_M}{\to} A \otimes M \overset{1\otimes f}{\to} A \otimes V\right)$$

The construction from left to right is as follows.

We have an A-comodule morphism $g : M \to A \otimes V$, and we take it to $(\varepsilon \otimes 1) \circ g$.

$$G : \left(M \overset{g}{\to} A \otimes V\right) \mapsto \left(M \overset{g}{\to} A \otimes V \overset{\varepsilon\otimes 1}{\to} k \otimes V \cong V\right)$$

These are clearly all linear maps thus $(\varepsilon \otimes 1) \circ g$ is a k-linear map.

What we will now prove is that F and G are left and right inverses.

First $F \circ G$**,**

This comes down to checking whether $g = (1 \otimes ((\varepsilon \otimes 1) \circ g)) \circ \Delta_M(g)$. We have already shown in the partial proof of this Lemma 3.17, that $(1 \otimes ((\varepsilon \otimes 1) \circ g)) = (1 \otimes \varepsilon \otimes 1) \circ (1 \otimes g)$. Consider the following diagram,

$$\begin{tikzcd}
M \arrow[rr, "\Delta_M"] \arrow[d, "g"'] & & A \otimes M \arrow[d, "1\otimes g"] \\
A \otimes V \arrow[rr, "\Delta_A \otimes 1"'] \arrow[dr, Leftrightarrow, "\cong"'] & & A \otimes A \otimes V \arrow[dl, "1\otimes\varepsilon\otimes 1"] \\
& A \otimes_k k \otimes V &
\end{tikzcd}$$

The commutativity of the top square is guaranteed by the fact that g is an A-comodule morphism. We know that the in a coalgebra the following diagram must commute,

$$\begin{tikzcd}
& A \arrow[dl, Leftrightarrow, "\cong"'] \arrow[dr, "\Delta"] & \\
A \otimes_k k & & A \otimes A \arrow[ll, "1\otimes\varepsilon"]
\end{tikzcd}$$

[11] In general this holds whenever the coresolution is minimal. What we will show is that this also holds for our construction of the coresolution.

Because we only take the identity between each V, we get that the bottom part of our previous diagram also commutes. Showing that $F \circ G = id$.

Now $G \circ F$**,**

Here we verify for $f : M \to V$ that $f = (\varepsilon \otimes 1) \circ (1 \otimes f) \circ \Delta_M(f)$. It is trivial to see that $(\varepsilon \otimes 1) \circ (1 \otimes f) = \varepsilon \otimes f = (1 \otimes f) \circ (\varepsilon \otimes 1)$. Now consider the following diagram,

$$\begin{tikzcd}
M \arrow[rr, "\Delta_M"] \arrow[dr, Leftrightarrow, "\cong"'] \arrow[dd, "f"'] & & A \otimes M \arrow[dl, "\varepsilon\otimes 1"'] \arrow[dr, "1\otimes f"] & \\
& k \otimes_k M \arrow[dr, "1\otimes f"'] & & A \otimes V \arrow[dl, "\varepsilon\otimes 1"] \\
V \arrow[rr, Leftrightarrow, "\cong"] & & k \otimes_k V &
\end{tikzcd}$$

The commutativity of the top triangle follows from the fact the M is a comodule. The commutativity of the right square is what we have just shown. This together implies that $G \circ F = id$.

And with that we see that we have a bijection $\mathrm{Hom}_A(M, A \otimes V) \leftrightarrow \mathrm{Hom}_k(M, V)$. □

Using the lemma above, we will show that $\mathrm{Hom}_A(k, A \otimes V)$ is (linearly) isomorphic to V.

Lemma 3.18. *$Hom_A(k, A \otimes V) \cong_{Set} Hom_k(k, V) \cong_k V$*

Proof. We have already shown the first isomorphism, what remains to show is the second isomorphism. Define the map $p : \mathrm{Hom}_k(k, V) \to V$ as $f \mapsto f(1)$. This map is clearly linear and surjective because we can construct a map to which sends 1 to a specific element in V. What remains to show is that it is injective.

Assume we have an $f, g \in \mathrm{Hom}_k(k, V)$ such that $f(1) = g(1)$. Then we have that for all $j \in k$,

$$f(j) - g(j) = j \cdot (f(1) - g(1)) = j \cdot 0 = 0$$

Because f and g are equal on their domain, they are the same function.

Thus, we have a linear isomorphism $\mathrm{Hom}_k(k, V) \cong_k V$. □

Now to give a slightly different view on the proof,

Example 3.19. One can write $f \in \mathrm{Hom}_k(k, V)$ as the following matrix,

	k
v_1	m_1
v_2	m_2
$\cdots$	
v_n	m_n

Now this shows that in the general case we only have to choose what $m_1, \cdots, m_n$ are. This makes $\{m_1, \cdots, m_n\}$ into an n dimensional vector space. Because V has dimension n as well, the two vector spaces are isomorphic.

Graded

For the graded case, nothing special happens. Here we will look in each degree individually, where we use the results we have already seen,

$$(\mathrm{Hom}_A(k, A\otimes V))^t = \mathrm{Hom}_A(k[t], A\otimes V) \cong_{Set} \mathrm{Hom}_k(k[t], V) = (\mathrm{Hom}_k(k, V))^t$$

V is a graded vector space, thus $V = \bigoplus_{t\in\mathbb{N}} V_t$ For the second equality we see that $(\mathrm{Hom}_k(k, V))^t = \mathrm{Hom}_k(k[t], \bigoplus_{t\in\mathbb{N}} V_t) = \mathrm{Hom}_k(k[t], V_t) \cong_k V_t$, where V_t is the vector space in degree t. Now the complete equality becomes as follows,

$$\mathrm{Hom}_A(k, A\otimes V) \cong_{Set}^{Graded} \mathrm{Hom}_k(k, V) \cong_k^{Graded} V$$

What remains is to show that in our coresolution the induced morphisms are zero. Thus, for a coresolution,

$$0 \longrightarrow N \xrightarrow{\epsilon} F_0 \xrightarrow{d_1} F_1 \xrightarrow{d_2} F_2 \xrightarrow{d_3} \cdots$$

After applying $\mathrm{Hom}_A(k, \_)$ and removing N,

$$0 \longrightarrow \mathrm{Hom}_A(k, F_0) \xrightarrow{d_1^*} \mathrm{Hom}_A(k, F_1) \xrightarrow{d_2^*} \mathrm{Hom}_A(k, F_2) \xrightarrow{d_3^*} \cdots$$

We will show that d_i^*, which takes $f \in \mathrm{Hom}_A(k, F_{i-1})$ to $d_i \circ f \in \mathrm{Hom}_A(k, F_i)$, is equal to zero.

Lemma 3.20. *For a connected coalgebra A, we have that $d_i^* = 0$.*

Proof. To prove this, we have to know a bit more about the structure of a connected coalgebra.

Recall that a connected coalgebra is graded and $A^0 \cong k$. Define $A^+ = A \setminus A^0$. For a connected coalgebra we must have that the counit $\varepsilon(A^+) = 0$ and $\varepsilon(A^0) = k$.

Because A is a coalgebra we have the following commuting diagram,

$$\begin{tikzcd}
k \otimes_k A & & A \otimes A \arrow[ll, "\varepsilon\otimes 1"'] \\
& A \arrow[ul, Leftrightarrow, "\cong"] \arrow[ur, "\Delta"'] \arrow[dl, Leftrightarrow, "\cong"'] \arrow[dr, "\Delta"] & \\
A \otimes_k k & & A \otimes A \arrow[ll, "1\otimes\varepsilon"]
\end{tikzcd}$$

Here we see that we must have that $a \otimes 1 = (1\otimes\varepsilon)(\Delta_A(a))$. With this we conclude for $a \in A^+$ that $\Delta_A(a) = a\otimes 1 + 1\otimes a + \sum_i a_{i1}\otimes a_{i2}$, where $a_{i1}, a_{i2} \in A^+$.

For degree reasons and the fact the k is a comodule (so it must adhere to the counit diagram) we must have that $\Delta_k(1) = 1\otimes 1$.

Now for any $f \in \text{Hom}_A(k, F_i)$, with $F_i = A \otimes V_i$ we can not have that $f(1) = \sum_j a_j \otimes v_j$, where any $0 \neq a_j \in A^+$. Because f is an A-comodule morphism we have that,

$$\begin{tikzcd}
k \arrow[r, "\Delta_k"] \arrow[d, "f"'] & A \otimes k \arrow[d, "1\otimes f"] \\
A \otimes V \arrow[r, "\Delta_A \otimes 1"'] & A \otimes A \otimes V
\end{tikzcd}$$

We now see that $(1 \otimes f)(\Delta_k(1)) = 1 \otimes (\sum_i a_i \otimes v_i) = \sum_i 1 \otimes a_i \otimes v_i$, but going through $A \otimes V$ we see that,

$$(\Delta_A \otimes 1)(f(1)) = \sum_i (a_i \otimes 1 \otimes v_i + 1 \otimes a_i \otimes v_i + \cdots)$$

where $\cdots \in A^+ \otimes A^+ \otimes V$.

But because there exists an i with $0 \neq a_i \in A^+$ we must have that $(1 \otimes f)(\Delta_k(1)) \neq (\Delta_A \otimes 1)(f(1))$. That means that we can only have $f(1) = \sum 1 \otimes v_j$.

What remains to show is that $d_{i+1}(1 \otimes v_j) = 0$. When we were constructing our coresolution we had the following diagram,

$$\begin{tikzcd}
F_{i-1} \arrow[rr, dashed, "i\circ\pi = d_i"] \arrow[dr, "\pi"'] & & F_i \arrow[r, dashed, "d_{i+1}"] & F_{i+1} \\
& Q \arrow[ur, "i"'] & &
\end{tikzcd}$$

Because Q is a comodule and A a connected coalgebra we must have that, $\Delta_Q(q) = 1 \otimes q + \sum_i a_i \otimes q_i$, where $a_i \in A^+$.

If we look at the construction of i, we see that when we construct the cofree comodule $A \otimes \{v_n\}$. We start with an $\bar{f} : Q \to \{v_n\}$ which sends a $q \in Q$ to v_n.

Now if we construct the A-comodule morphism we get that, $f := (1 \otimes \bar{f})(\Delta_Q)$ and $f(q) = 1 \otimes v_n + \sum_i a_i \otimes f(q_i)$, but because q_i is in a different degree then q we must have that $f(q_i) = 0$. This also means that $i(q) = 1 \otimes e_n$.

Because π is surjective we now have that $\exists x \in F_{i-1}$ such that $d_i(x) = 1 \otimes v_j$. This guarantees that all the basis elements of the form $1 \otimes v_j \in A \otimes V_i$ are always hit by d_i.

Now we see that in that when we construct the quotient ${F_i}/{\text{im } d_i}$, we have that $\pi(1 \otimes v_j) = 0$. This finally gives us that $d_{i+1}(1 \otimes v_j) = 0$.

We combine the fact that any $f \in \text{Hom}_A(k, F_i)$ has $f(1) = \sum 1 \otimes v_j$ and that $d_{i+1}(\sum_j 1 \otimes v_j) = 0$. Now this gives us that, $d^*_{i+1}(f)(1) = (d_{i+1} \circ f)(1) = d_{i+1}(\sum 1 \otimes v_j) = 0$. Because f was arbitrary and f and d_{i+1} are linear maps, we have that $d^*_{i+1} = 0$. □

Let us remind the reader of what we have now done.

With these lemmas we have that for a cofree coresolution of N, an A-comodule,

$$\begin{tikzcd}
0 \arrow[r] & N \arrow[r, "\epsilon"] & A \otimes V_0 \arrow[r, "d_1"] & A \otimes V_1 \arrow[r, "d_2"] & A \otimes V_2 \arrow[r, "d_3"] & \cdots
\end{tikzcd}$$

applying $\mathrm{Hom}(k, \_)$ and removing N,

$$0 \longrightarrow \mathrm{Hom}_A(k, A \otimes V_0) \xrightarrow{0} \mathrm{Hom}_A(k, A \otimes V_1) \xrightarrow{0} \mathrm{Hom}_A(k, A \otimes V_2) \xrightarrow{0} \cdots$$

Ext, in the graded sense, is defined as,

$$\mathrm{Ext}_A^{s,*}(D, M) = {}^{\ker d^*_{s+1}}\big/_{\mathrm{im}\ d^*_s}$$

Now if we look in each degree t we see,

$$\mathrm{Ext}_A^{s,t}(D, M) = \left({}^{\ker d^*_{s+1}}\big/_{\mathrm{im}\ d^*_s}\right)^t = (\ker d^*_{s+1})^t = \mathrm{Hom}_A(k[t], A \otimes V_s) \cong V_s^t$$

4 The algorithm

Now that we have all the definitions we can finally start describing how we make the computer calculate Ext for us. We will start by describing the data structures we use to remember all the information we have to store. Note that from this point on we will only be considering comodules and coalgebras of finite type. We also introduce a cutoff point for the grading, meaning that our graded vector spaces will always be finite. We will also only work with connected coalgebras.

4.1 Data structures

First we start of with some basic support types which will help us define more complex data structures.

```
type Grading = int

struct BasisElement {
    name: string,
    degree: Grading,
    generator: boolean,
};

type Basis = List[BasisElement]
type BasisIndex = int

type Matrix = ...
```

This is not really special yet, we are just defining how we store a basis for our vector space, which is list of elements with a grading and a name. The generator boolean here is what we will use in the end to deduce how many cofree copies we have generated and in what degree they are in. This generator boolean also only makes sense when we are either defining a coalgebra or making a cofree comodule.

The type of Matrix, is any type which stores finite field elements in a 2-dimensional array and for which there are certain built in linear algebra routines. Explicitly we require the Matrix type to support finding the row reduced echelon form of a matrix, taking the transpose, and finding the null space matrix. Where the null space matrix is a matrix which, when we span the rows give us a basis for the kernel of our matrix.

Our matrix type should also support slicing. Slicing means looking at a subpart of a matrix, and the syntax is as follows,

```
subM = M[ i : j ,  k : l ]
```

As an example,

```
subM = M[ 1 : 3 ,  2 : 4 ]
```

Here M is a matrix with dimension at least $3x4$, recalling that we start our indexing at zero. Now subM is a 2x2 matrix, meaning we don't include the third and fourth index in the sub matrix.

Example 4.1.

$$M = \begin{pmatrix} a & b & c & d \\ \begin{matrix} e & f \\ i & j \end{matrix} & & \multicolumn{2}{c}{\begin{pmatrix} g & h \\ k & l \end{pmatrix}} \end{pmatrix} \qquad subM = \begin{pmatrix} g & h \\ k & l \end{pmatrix}$$

If there is no integer to either the left or right of :, then we will automatically insert 0 on the left and the maximum possible index on the right. This means that $M = M[:, :]$.

Coalgebra

```
struct Coalgebra {
    basis: Basis,
    coaction: Matrix,
};
```

The basis represents the vector space basis of the coalgebra. Note that we also have an implicit ordering in the basis, which is based on its index in the list, which we use to know what elements map to where in our coaction matrix. The matrix object represents the coaction on the coalgebra A.

The coaction is the morphism $\Delta : A \to A \otimes A$. Because we are only working over finite coalgebras, we can represent the coaction action in a matrix.

The matrix is of dimensions $len(basis)^2$ by $len(basis)$, with $len(basis)$ meaning the size of the list of basis elements, which is the dimension of the vector space of A.

Note that we do not explicitly store a basis for this tensor product $A \otimes A$. But instead we will make use of the basis of the coalgebra and implicitly use the following basis for the tensor, $\{a_i \otimes a_j \mid 1 \leq i, j \leq len(basis)\}$. In the matrix we should also be explicit about the ordering of both basis, which will be giving as follows,

$$\begin{array}{c|cccc} & a_1 & a_2 & \cdots & a_n \\ \hline a_1 \otimes a_1 & & & & \\ a_2 \otimes a_1 & & & & \\ \cdots & & & & \\ a_n \otimes a_1 & & & M & \\ A \otimes a_2 & & & & \\ \cdots & & & & \\ A \otimes a_n & & & & \end{array}$$

In the top row we see the basis we have defined in our Coalgebra struct. In the column on the left the computer only knows what its numerical index is, however it doesn't know which two elements the element of the tensor product is made up of.

So now we would like to know what basis element of the tensor product this column index is. To find out the left hand side of the tensor we take the column index modulo $len(basis)$ and for the right hand side we take the column index divided by $len(basis)$. In code,

```
left_coalgebra_index = column_index % len(basis)
right_coalgebra_index = column_index // len(basis)
```

Here % is taking i modulo (%) j, and // is integer division (this is the same as normal division and afterwards flooring).

Example 4.2. So assume our vector space is only 2 dimensional, with basis $\{a_0, a_1\}$[12]. We would then get a coaction matrix of 4x2, which can look (for example) as follows,

	a_0	a_1
$a_0 \otimes a_0$	1	0
$a_1 \otimes a_0$	0	1
$a_0 \otimes a_1$	0	1
$a_1 \otimes a_1$	0	0

Now if we have we are interested in which tensor product is at column index 2, we see that we get,

```
left_coalgebra_index = column_index % len(basis) = 2 % 2 = 0
right_coalgebra_index = column_index // len(basis) = 2 // 2 = 1
```

This means that at column index 2 (remember that we start counting at zero) we have the tensor $a_0 \otimes a_1$.

If we compare this to the definition of a coalgebra we see some things missing. First we don't store the unital function, this is because the unital function can be deduced by the fact that we are always working over connected coalgebras. We also require that the coaction the user provides adheres to the coassociativity diagram and that the other implicit diagrams also commute.

Comodule

```
struct Comodule {
    coalgebra: Coalgebra,
    basis: Basis,
    coaction: Matrix,
};
```

The basis again represents the (finite) vector space basis, and the coalgebra references a coalgebra

[12] In programming it is normal to start counting from zero and this also somewhat simplifies the formulas for calculating indices

object. Now the coaction is again a matrix and is represented as follows,

$$\begin{array}{c|cccc}
 & m_1 & m_2 & \cdots & m_n \\
\hline
a_1 \otimes m_1 & & & & \\
a_2 \otimes m_1 & & & & \\
\cdots & & & & \\
a_n \otimes m_1 & & & M & \\
A \otimes m_2 & & & & \\
\cdots & & & & \\
A \otimes m_n & & & &
\end{array}$$

However, we now take a tensor product of $A \otimes_k M$. We do (almost) the same process to understand which column index represents which tensor product element. But now we have to divide by the length of the coalgebra vector space basis,

```
comodule_index = column_index % len(coalgebra.basis)
coalgebra_index = column_index // len(coalgebra.basis)
```

Morphisms

We would also like to remember morphisms between A-comodules. The only thing we really have to remember is the linear map between these two comodules.

```
struct Morphism {
    domain: Comodule,
    codomain: Comodule,
    map: Matrix,
};
```

Here we don't explicitly store that the map is an A-comodule map, we only remember the linear map. We have proven in Chapter 3 that the way in which we construct the map guarantees that the map commutes the A-comodule morphism diagram.

Example 4.3. If the domain C has basis $\{c_1, \cdots, c_m\}$ and the codomain D has basis $\{d_1, \cdots, d_n\}$. Then the map is matrix might looks as follows,

$$\begin{array}{c|cccc}
 & c_1 & c_2 & \cdots & c_m \\
\hline
d_1 & 1 & 1 & \cdots & 0 \\
d_2 & 0 & 1 & \cdots & 1 \\
d_3 & 0 & 0 & \cdots & 0 \\
\cdots & \cdots & \cdots & \cdots & \cdots \\
d_n & 1 & 1 & \cdots & 1
\end{array}$$

Resolution

```
struct Resolution {
    comodule: Comodule,
    morphisms: List[Morphism],
};
```

The resolution object merely the original comodule for which it makes a resolution and it remembers all morphisms from the resolution. The comodule object contains the coalgebra as well. And the list of morphisms remember their domains and codomains, thus we also know what all the specific free comodules are.

4.2 Routines

To construct the actual coresolution we will use the description of the algorithm we have given in Chapter 3.1. That means we should make two routines, one to construct the cokernel and one to create a minimal injection from a comodule to a cofree comodule.

The routines we provide are not the most efficient algorithms yet, and there are optimizations which will make them run faster, some of which we will go over later in this thesis. But for understandability we have chosen to give the most basic algorithm which explains the idea the best.

Cokernel routine

```
function Cokernel(f: Morphism) -> Morphism {
    P = f.map.transpose().null_space().row_reduce()

    pivots: List[int] = find_pivots(P)

    // Make basis
    Q_basis: List[BasisElement] = []
    for pivot in pivots:
        Q_basis.push(f.codomain.basis[pivot])

    // Get coalgebra dimension for coaction matrix
    coalg = f.codomain.coalgebra
    a = len(coalg.basis)


    // Make coaction
    coaction = zero_matrix(a*len(Q_basis), len(Q_basis))

    for q_i in len(Q_basis):                                  // 1
        f_i = pivots[q_i]                                     // 2
        for c_f_i in len(f.codomain.basis):                   // 3
            for c_q_i in len(Q_basis):                        // 4
                coaction[c_q_i*a:(c_q_i+1)*a, q_i] +=        // 5
                    P[c_q_i, c_f_i] *                         // 6
                    f.codomain.coaction[c_f_i*a:(c_f_i+1)*a, f_i] // 7


    Q = Comodule(coalg, Q_basis, coaction)
    return Morphism(f.codomain, Q, P)
}
```

First we will prove that Q_basis is actually the coker f and that M is the correct (linear) map to Q. For this part we will only be worried about the vector space structure and not yet about the A-comodule structure. If we look at the definition 3.15 of the cokernel, we see that both the kernel and cokernel can be defined as the objects which make the following sequence exact,

$$\begin{tikzcd} 0 \arrow[r] & \ker f \arrow[r, hook, "i"] & M \arrow[r, "f"] & N \arrow[r, two heads, "\pi"] & \text{coker } f \arrow[r] & 0 \end{tikzcd}$$

We know that when dualizing, injective maps become surjective maps and vice versa. Here we will also be using the fact that $\mathrm{Hom}_k(\_, k)$, which is dualizing ($M^* := \mathrm{Hom}_k(M, k)$), is a left exact functor. Because dualizing is left exact we have that if the following sequence is exact,[13]

$$\begin{tikzcd} M \arrow[r, "f"] & N \arrow[r, two heads, "\pi"] & \text{coker } f \arrow[r] & 0 \end{tikzcd}$$

Then the following sequence is also exact,

$$\begin{tikzcd} M^* & N^* \arrow[l, "f^*"'] & (\text{coker } f)^* \arrow[l, hook', "\pi^*"'] & 0 \arrow[l] \end{tikzcd}$$

Because M and N are finite dimensional, $M^* \cong M$ and $N^* \cong N$. This implies we also have the exact sequence,

$$\begin{tikzcd} M & N \arrow[l, "f^*"'] & (\text{coker } f)^* \arrow[l, hook', "\pi^*"'] & 0 \arrow[l] \end{tikzcd}$$

We also know that the object D which makes the sequence $0 \to D \to N \overset{f^*}{\to} M$ exact is the $\ker f^*$. Thus, we have that $(\text{coker } f)^* \cong D \cong \ker f^*$. Because we have chosen a basis for both M and N we have that $f^* = f^T$, the transpose of the matrix f.

Now all that is left is to show that there exists a surjective matrix P from N to $kerf^T$ for which it holds that $f^T \cdot P^T = 0$. Meaning that P maps element from the kernel of f^T to a basis for $kerf^T$. This P is exactly given by null_space() function. One can think of P as the matrix for which its rows span a basis for $\ker f^T \subset N$. With that we see,

$$\begin{tikzcd} & & \text{coker } f \arrow[d, Leftrightarrow] & \\ M^* & N^* \arrow[l, "f^*"'] & (\text{coker } f)^* \arrow[l, hook', "\pi^*"'] \arrow[d, Leftrightarrow] & 0 \arrow[l] \\ M & N \arrow[l, "f^T"'] \arrow[r, bend right, two heads, "P"'] & kerf^* \arrow[l, hook', "\pi^*"'] & 0 \arrow[l] \end{tikzcd}$$

Where we have a map P which goes from N to the coker f. There is one extra transformation which we will apply which will help us later, and that is that we take the row reduced echelon form. This will make it easier to find an element in N which maps to a specific basis in coker f. This row reduced echelon form can be seen as a basis transformation on coker f.

Example 4.4. A row reduced echelon matrix has the following form,

$$\begin{bmatrix} 1 & 0 & a_1 & 0 & b_1 \\ 0 & 1 & a_2 & 0 & b_2 \\ 0 & 0 & 0 & 1 & b_3 \end{bmatrix}$$

[13] The details can be found in Dummit and Foote [2], theorem 33 chapter 10.5.

Now we for each row in this matrix P represents an element of a specific degree in N. And we will create Q_basis depending on this degree. We also don't care about a name for this basis element, because in our final coresolution we will not remember this intermediate coker object.

Now to make the coaction we will do the following,

$$\begin{tikzcd}
N \arrow[r, "\Delta_N"] \arrow[d, "P"'] & A \otimes N \arrow[d, "1\otimes P"'] \\
\text{coker } f \arrow[r, dotted, "\Delta_{\text{coker } f}"] & A \otimes \text{coker } f
\end{tikzcd}$$

We will make the coaction on coker f by looking at what it does for Δ_N. We know that P is surjective. Thus, for each element in the coker f we have a preimage in N, these are step $(1), (2)$. Now we iterate over all elements in N as well as all elements in coker f, step $(3), (4)$. The most difficult step is now as follows.

For each c_f_i we look at the coaction for f_i with respect to a part of the tensor. This means that we are interested in $\Delta_N(f_i)$, but only in elements with respect to $A\otimes$ c_f_i, step (7). This gives us a vector where its size is the dimension of A. Now we look at what P does to c_f_i in each basis c_q_i of coker f, step (6). This gives us a scalar. Now we multiply this scalar with the vector we got, and we add that to the coaction of $\Delta_{\text{coker } f}(q_i)$, where we again only look at elements with respect to $A\otimes$ c_q_i, step (5).

This gives us a coaction $\Delta_{\text{coker } f}$, and because we have deduced this coaction from the coaction on N, we can deduce the commutativity of the coassociativity and the counit diagrams by using the fact that Δ_N already makes these commute. Thus we have constructed the coker f correctly.

We also used this auxiliary routine to find pivot elements in a surjective row reduced matrix,

```
function find_pivots(surj_rref_map: Matrix) -> List[int] {
    // Get the dimensions of the matrix
    rows, cols = surj_rref_map.shape

    pivots = []
    for r in rows:
        for c in cols:
            if surj_rref_map[r,c] == 1:
                pivots.push(c)
                break
    return pivots
}
```

Injection to cofree comodule routine

```
function injection(M: Comodule) -> Morphism {
    growing = zero_morphism(M, zero_comodule(M.coalgebra))
    a = len(M.coalgebra.basis)

    while True:
        kernel = growing.map.null_space()

        m_1 = lowest_degree_element(kernel, M)

        if m_1 == -1:
            return growing

        F = cofree(M.basis[m_1].grading)
        M_to_F = M.coaction[m_1*a:(m_1+1)*a]
        m_1_morph = Morphism(M, F, M_to_F)

        growing = combine_morphisms(growing, m_1_morph)
}
```

Recall 3.1, where we describe the initialization and the iteration of the algorithm.

We start with a morphism to a zero cofree comodule and the iteration each step goes as follows,

- First we take the lowest graded element $q \neq 0 \in \ker i$, (if we are in the non-graded case, take any element). If such a q does not exist, $\ker i = 0$ and we are done.
- Make a morphism $f : M \to A \otimes e_{n+1}$ for which $f(q) \neq 0$. This guarantees that q won't be in the kernel.
- Now we define $\bar{i}$ to be $M \to A \otimes \{e_1, \cdots, e_n\} \cup \{e_{n+1}\}$, with $\bar{i}(m) := i(m) + f(m)$.
- We repeat the process with $i = \bar{i}$ until the $\ker i = 0$.

If we look at the routine above we see that we start with a zero morphism (called growing) from M.

Then for the iteration, We take the null space, the kernel, of the current growing morphism matrix. And then using the lowest_grade_element subroutine we get the element m_1, a basis element in M which we also got in the proof 3.1.

Now we construct $A \otimes \{e_{n+1}\}$, where e_{n+1} has the same degree as m_1. This is just copying the basis of A and shifting the degree of this basis with the degree of m_1.

If we look at the proof again we see that we constructed the map to $A \otimes \{e_{n+1}\}$ as $(1 \otimes \bar{f}) \circ \Delta_M$, where $f(m_1) = e_{n+1}$ and $f(m_{i>1}) = 0$. We see that this function $(1 \otimes f) \circ \Delta_M$ does the following to our coaction matrix on M,

$$\begin{array}{c|cccc} & m_1 & m_2 & \cdots & m_n \\ \hline a_1 \otimes f(m_1) & & & & \\ a_2 \otimes f(m_1) & & & & \\ \cdots & & & & \\ a_n \otimes f(m_1) & & \Delta_M & & \\ A \otimes f(m_2) & & & & \\ \cdots & & & & \\ A \otimes f(m_n) & & & & \end{array} = \begin{array}{c|cccc} & m_1 & m_2 & \cdots & m_n \\ \hline a_1 \otimes f(m_1) & & & & \\ a_2 \otimes f(m_1) & & & & \\ \cdots & & & & \\ a_n \otimes f(m_1) & & \Delta_M & & \\ A \otimes 0 & & & & \\ \cdots & & & & \\ A \otimes 0 & & & & \end{array} =$$

Now to construct the mapping from $M \to A \otimes \{e_{n+1}\}$ we only need to consider all basis elements which are non zero. Thus we can just copy the top part of the matrix as follows,

$$\begin{array}{c|cccc} & m_1 & m_2 & \cdots & m_n \\ \hline a_1 \otimes e_{n+1} & & & & \\ a_2 \otimes e_{n+1} & & \text{M\_to\_F} & & \\ \cdots & & & & \\ a_n \otimes e_{n+1} & & & & \end{array}$$

And this is the same as the array slicing we do in our routine. Then finally we combine this morphism with the growing morphism we already have. Which happens in one of the following subroutines. First the lowest degree element,

```
function lowest_degree_element(null: Matrix, M: Comodule) -> BasisIndex {
    rows, cols = null.shape

    basis_index = -1
    degree = 0

    for r in range(rows):
        for c in range(cols):
            if null[r,c] != 0:
                if M.basis[c] < degree or basis_index = -1:
                    basis_index = c
                    degree = M.basis[c]
                break
    return basis_index
}
```

Here we construct $A \otimes \{e_{n+1}\}$,

```
function cofree(degree: Grading, coalg: Coalgebra) -> Comodule {
    basis = []
    for b in coalg.basis:
        b.degree += grading
        basis.push(b)
    return Comodule(coalg, basis, coalg.coaction)
}
```

Here we construct a zero A-comodule,

```
function zero_comodule(coalgebra: Coalgebra) -> Comodule {
    return Comodule(coalgebra, [], zero_matrix(0,0))
}
```

A morphism between two comodules which sends everything to zero,

```
function zero_morphism(domain: Comodule, codomain: Comodule) -> Comodule {
    map = zero_matrix(len(codomain.basis), len(domain.basis))
    return Morphism(domain, codomain, map)
}
```

The direct sum just takes the two basis and sums them together. It also takes a block sum of the coaction matrixes

```
function direct_sum(M: Comodule, N: Comodule) -> Comodule {
    basis = M.basis + N.basis
    coaction = M.coaction.block_sum(other.coaction)
    return Comodule(M.coalgebra, basis, coaction)
}
```

Example 4.5. Assume A is a coalgebra with two a basis of two elements $\{a_0, a_1\}$ If we had two comodules M and N with basis $\{m_0, m_1\}$ and $\{n_0, n_1\}$ with the following coactions,

$$\Delta_M = \begin{array}{r|cc} & m_0 & m_1 \\ \hline a_0 \otimes m_0 & 1 & 0 \\ a_1 \otimes m_0 & 0 & 1 \\ a_0 \otimes m_1 & 0 & 1 \\ a_1 \otimes m_1 & 0 & 0 \end{array} \qquad \Delta_N = \begin{array}{r|cc} & n_0 & n_1 \\ \hline a_0 \otimes n_0 & 1 & 0 \\ a_1 \otimes n_0 & 1 & 1 \\ a_0 \otimes n_1 & 0 & 1 \\ a_1 \otimes n_1 & 0 & 1 \end{array}$$

The direct sum would have basis $\{m_0, m_1, n_0, n_1\}$ and coaction,

$$\Delta_{M+N} = \begin{array}{r|cccc} & m_0 & m_1 & n_0 & n_1 \\ \hline a_0 \otimes m_0 & 1 & 0 & 0 & 0 \\ a_1 \otimes m_0 & 0 & 1 & 0 & 0 \\ a_0 \otimes m_1 & 0 & 1 & 0 & 0 \\ a_1 \otimes m_1 & 0 & 0 & 0 & 0 \\ a_0 \otimes n_0 & 0 & 0 & 1 & 0 \\ a_1 \otimes n_0 & 0 & 0 & 1 & 1 \\ a_0 \otimes n_1 & 0 & 0 & 0 & 1 \\ a_1 \otimes n_1 & 0 & 0 & 0 & 1 \end{array}$$

Here we combine two morphisms with the same domain. We take a direct sum of the codomains and we (vertically) stack the two matrices.

```
function combine_morphisms(a: Morphism, b: Morphism) -> Morphism {
    map = a.map.stack(b.map)
    codomain = direct_sum(a.codomain, b.codomain)
    return Morphism(a.domain, codomain, map)
}
```

Example 4.6. assume the domain of a and b is $\{m_0, m_1\}$ and the codomain of a is $\{a_0\}$ and the codomain of b is $\{b_0, b_1\}$. The maps of a and b are,

$$a = \begin{array}{c|cc} & m_0 & m_1 \\ \hline a_0 & 1 & 0 \end{array} \qquad b = \begin{array}{c|cc} & m_0 & m_1 \\ \hline b_0 & 1 & 0 \\ b_1 & 1 & 1 \end{array}$$

Then the stack becomes,

$$\begin{array}{c|cc} & m_0 & m_1 \\ \hline a_0 & 1 & 0 \\ b_0 & 1 & 0 \\ b_1 & 1 & 1 \end{array}$$

The coresolution

Now the routine to construct the coresolution is not special anymore.

```
function resolution(M: Comodule) -> Coresolution {
    zero = zero_morphism(zero_comodule(), M)
    morphisms = [zero]

    // Make the coresolution
    for n in range(FILTRATION_MAX):
        // Get the latest morphism from our list of morphisms
        morph = morphisms[-1]

        // Get the cokernel
        coker = cokernel(morph)

        // Injection from the cokernel to a cofree comodule
        injec = injection(coker.codomain)

        // Compose the two morphisms
        final = injec @ coker

        // Add the composed map to the list
        morphisms.append(final)

    return Resolution(M, morphisms)
}
```

Now we have shown in chapter 3.2 that $\mathrm{Ext}_A^{s,t} = V_s^t$. And to find the generators of V_s^t, we do the following,

```
function v_s_t(cores: Coresolution, s: int, t: Grading) -> List[BasisElement] {
    v_s = cores.morphisms[s].codomain.basis

    generators = []
    for b in v_s:
        if b.generator and b.degree = t:
            generators.push(b)

    return generators
}
```

Our program now gives us the following output for $N = k$ as a comodule, and $A = A(1)_*$ the dual mod 2 steenrod subcoalgebra,

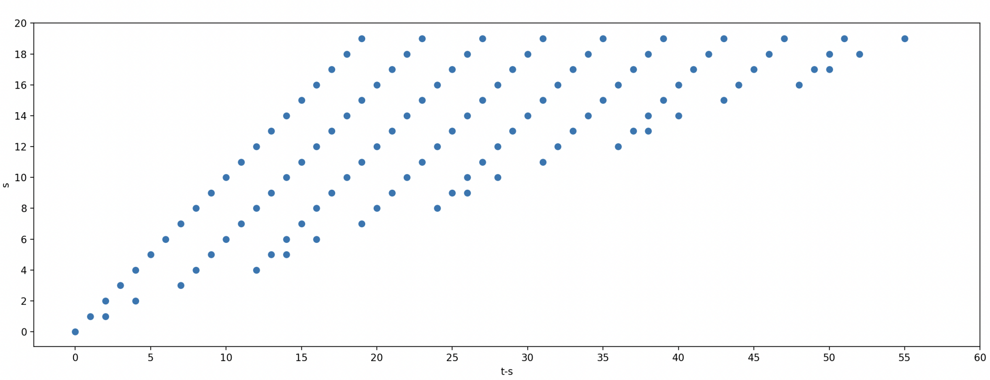

Figure 1: $A(1)_*$ with x axis equal to t

Here the y axis represents s and the x axis represents t. We draw a dot whenever $V_s^t \neq 0$. In the literature we usually let the x axis represent $t - s$, that would look as follows,

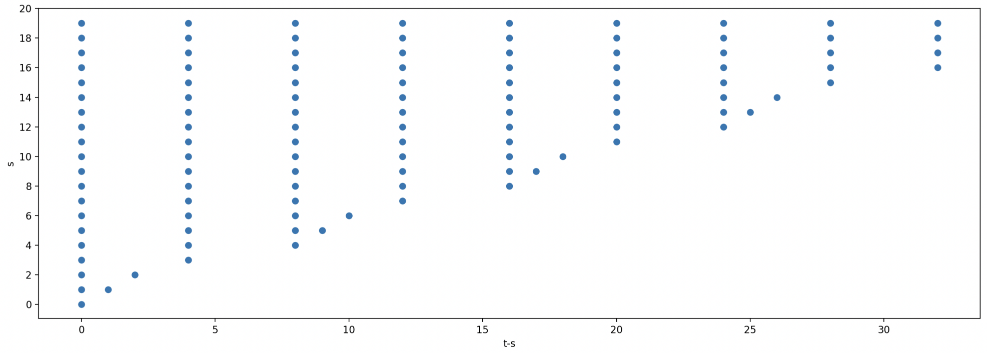

Figure 2: $A(1)_*$ with x axis equal to $t - s$

5 Structure Lines

Up until this point we have only calculated vector basis generators for Ext. However, there also exists a ring structure on Ext. The precise definition of this ring structure goes beyond the scope of this thesis.

This ring structure can be (partially) displayed in the Ext page by drawing lines between Ext^n and Ext^{n+1}. Where to draw these lines follows some rules. We will naively use these rules to display the structure lines and thus ring structure of Ext.

Definition 5.1. A primitive element of a coalgebra A, where 1 is the multiplicative identity of A, is an element a for which the coaction is,

$$\Delta_A(a) = a \otimes 1 + 1 \otimes a$$

We will also refer to a primitive element x in a cofree comodule if $x = a \otimes v_i \in A \otimes V$, where a is primitive in the coalgebra. This means that there can be multiple primitive elements in a cofree comodule which get represented by the same primitive element a in the coalgebra.

Every primitive element of A has its own class of structure lines. In the coresolution for N we have $F_i \overset{d_{i+1}}{\to} F_{i+1}$, with F_i and F_{i+1} both cofree. We can rewrite this to $A \otimes V_i \overset{d_{i+1}}{\to} A \otimes V_{i+1}$. Let V_i have basis $\{v_{ik}\}_{k \in I}$ and V_{i+1} have basis $\left\{v_{(i+1)l}\right\}_{l \in J}$. Remember as well that we only consider A coalgebras which are connected. Thus, we will represent the basis of A^0 as 1. Also remember that after applying the Hom functor with k we only keep the V_i part. Thus, $\text{Hom}_A(k, A \otimes V_i) \cong V_i$.

We draw a **structure line** between two basis elements v_{ik} and $v_{(i+1)l}$ for the primitive element a whenever $1 \otimes v_{(i+1)l}$ is in the sum $d_{i+1}(a \otimes v_{ik}) = \sum_b a_b \otimes v_{(i+1)b} \in A \otimes V_{i+1}$. In words, when a gets mapped to 1 we draw a structure line between their respective basis elements.

Routines

To programmatically find these structure lines we should first know which elements are primitive in the coalgebra. To do this we append the basis element data structure as follows,

```
struct BasisElement {
    name: string,
    degree: Grading,
    generator: boolean,
    primitive_index: None | int,
    generated_index: int,
};
```

Each primitive element in the basis of our coalgebra has a **unique** primitive_index. In a cofree comodule, generated_index represents to which basis element v_{ij} of V_i it belongs. Before structure lines, we did not need to keep apart multiple basis elements in the same degree. We were only interested in the amount of generators in each degree. This generated index is now necessary because we also want to know to exactly which basis element in $\text{Hom}_A(k, F_i)$ we map.

This also slightly changes the way in which we make a cofree comodule in the Injection to cofree comodule routine. We keep track of how many basis elements there implicitly are in V_i.

```
function resolve(Q: Comodule) -> Morphism {

    growing = zero_morphism(Q, zero_comodule(Q.coalgebra))
    a = len(Q.coalgebra.basis)

    generated_index = 0
    while True:
        // ......

        F = cofree(Q.basis[low_el].grading,
                        Q.coalgebra, generated_index)

        // ......

        generated_index += 1
}

function cofree(degree: Grading, coalg: Coalgebra,
                    index: int) -> Comodule {
    basis = []
    for b in coalg.basis:
        b.degree += grading
        b.generated_index = index
        basis.push(b)
    return Comodule(coalg, basis, coalg.coaction)
}
```

The routine

Now we will discuss the routine to find these structure lines,

```
function structure_lines(domain: Basis, codomain: Basis, map: Matrix)
                            -> List[(BasisIndex,BasisIndex,int)] {
    lines = []
    for prim_id in primitive_indices(domain):
        for target_id in range(len(codomain)):
            // If we don't map to this element, continue
            if map[target_id, prim_id] == 0:
                continue

            // Check if the element we map to is a generator
            target = codomain.basis[target_id]
            if target.generator == False:
                continue

            primitive = domain[prim_id].primitive_index

            // Get the generator element belonging
            // to this primitive element
            prim_generated_index = domain[prim_id].generated_index
            prim_generator = domain.find_generator(prim_generated_index)
            lines.push((prim_gen, el_id, primitive))
    return lines
}

function primitive_indices(basis: Basis) -> List[BasisIndex] {
    prims = []
    for p_id in range(len(basis)):
            if basis[p_id].primitive != None:
                prims.push(p_id)
    return prims
}

function find_generator(basis: Basis, generator_index: int) -> BasisIndex {
    for el_id in range(len(basis)):
        el = basis[el_id]
        if el.generator and el.generated_index == generated_index:
            return el_id
}
```

Here prim_id is an index of a primitive element in the basis list of the domain. Such an element is thus (implicitly) of the form $a_p \otimes v_i$ with a_p primitive. Then we check for each element it send to, if that element is a generator. An element in a cofree comodule is a generator if it (implicitly) represents $1 \otimes v_i$. And if it is an element we collect find the generator corresponding to the primitive element we started with and add the structure line it produces to the list of structure lines.

Example 5.2. We again look at the previous example $A(1)_*$, which is isomorphic to the polynomial algebra $\mathbb{F}_2[\xi_1, \xi_2] / \langle \xi_1^4, \xi_2^2 \rangle$. The coaction for the indeterminates is,

$$\Delta_A(\xi_1) = \xi_1 \otimes 1 + 1 \otimes \xi_1$$

$$\Delta_A(\xi_2) = \xi_2 \otimes 1 + 1 \otimes \xi_2 + \xi_1^2 \otimes \xi_1$$

One can easily verify for this coalgebra that ξ_1 and ξ_1^2 are primitive elements. The output of our program now becomes, where the red lines represent a ξ_1 going to 1 and the blue lines represent ξ_1^2 going to 1,

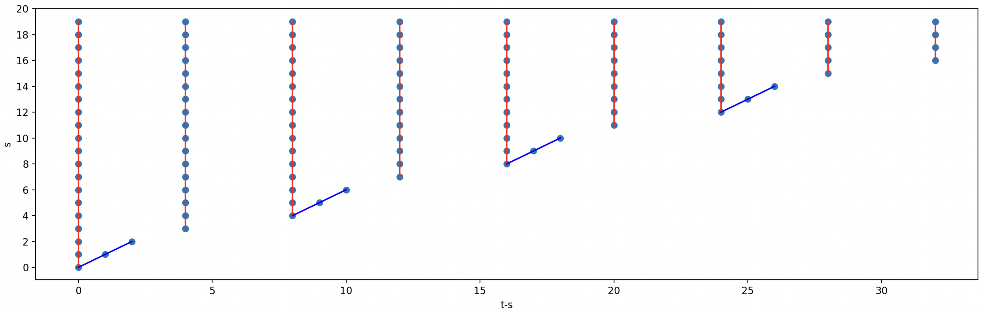

Figure 3: $A(1)_*$ with structure lines

6 Grading

Until now our algorithm has not used the fact that we are working over graded vector spaces. This means that there is a lot of redundant information in our Matrix object. Because for graded vector space V and W and a map f between them, we have that $f(V^i) \subset W^i$. This means that in our matrix we have a lot of zeros. This fact is something we will use. More specifically, if we sort the basis on V and the basis on W and reorder our matrix using this sorting, we see that we get a block diagonal matrix.

Definition 6.1. A block diagonal matrix is a matrix M, with submatrices M_i which only occur on the diagonal. Thus,

$$M = \begin{pmatrix} M_1 & & & 0 \\ & M_2 & & \\ & & \dots & \\ 0 & & & M_n \end{pmatrix}$$

For example,

$$M = \begin{pmatrix} 1 & 1 & 2 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 4 \end{pmatrix}$$

is a block diagonal matrix where,

$$M_1 = \begin{pmatrix} 1 & 1 & 2 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix} \quad M_2 = \begin{pmatrix} 4 \end{pmatrix}$$

are the submatrices.

The advantage of these block diagonal matrices is that all the operations we do on M can be done over the submatrices. This greatly reduces the computational complexity, increasing the speed of our algorithm. We see that,

$$M^T = \begin{pmatrix} M_1 & & & 0 \\ & M_2 & & \\ & & \dots & \\ 0 & & & M_n \end{pmatrix}^T = \begin{pmatrix} M_1^T & & & 0 \\ & M_2^T & & \\ & & \dots & \\ 0 & & & M_n^T \end{pmatrix}$$

Finding the basis matrix for the kernel of M can also be done by looking in each submatrix, as well as finding the row reduced echelon form.

The big downside of using diagonal matrices is that we now have to sort the basis for a graded vector space. This becomes slightly more difficult when the graded vector space we are looking at is actually the tensor product of two graded vector spaces. Before, we had a closed formula to calculate what column index in the coaction matrix represent which basis element in our tensor. Now we will have to remember a lookup table, which remembers what column index represents what element. Now we will explain how our code changes to incorporate these changes.

Graded Routines

The helper data structures we have change slightly,

```
struct BasisElement {
    name: string,
    generator: boolean,
    primitive_index: None | int,
    generated_index: int,
};

type Basis = Dict[Grading, List[BasisElement]]
type BasisIndex = (Grading, int)

type GradedMap = Dict[Grading, Matrix]
```

Here we remember the basis of our graded vector space as a dictionary, where we index depending on the grading and what we get back is a list of basis elements belonging to that grading. We now also have graded linear maps which is again a dictionary for the grading together with a matrix object. This GradedMap type inherits the routines for the null space, transpose and row reduced echelon form from the underlying Matrix type.

For both the coalgebra and the comodule we now also have to remember the basis for the tensor product.

For the graded tensor product of $V \otimes W$ we want to have two maps. One which takes two basis elements from the two underlying graded vector space basis and gives us the basis element in the graded tensor product, and we also want a map which takes a basis element from our graded tensor product back to their original basis elements. This first map is what we will refer to as Constructor and the second as Deconstructor.

```
// Comodule BasisIndex -> Coalgebra BasisIndex -> Tensor BasisIndex
type Constructor = Dict[Grading, Dict[Grading, List[BasisIndex]]]

// Tensor BasisIndex -> (Comodule BasisIndex, Coalgebra BasisIndex)
type Deconstructor = Dict[Grading, List[(BasisIndex,BasisIndex)]]

struct Coalgebra {
    basis: Basis,
    coaction: GradedMap,
    construct: Constructor,
    deconstruct: Deconstructor

};

struct Comodule {
    coalgebra: Coalgebra,
    basis: Basis,
    coaction: GradedMap,
    construct: Constructor,
    deconstruct: Deconstructor
};
```

Now we should inspect what this does for our two routines, constructing the cokernel and making an injection to a cofree comodule.

```
function Cokernel(f: Morphism) -> Morphism {
    // P and f.map are now GradedMaps
    P = f.map.transpose().null_space().row_reduce()

    // Pivots are now found in each grading
    pivots: Dict[Grading, List[int]] = find_pivots(P)

    // Make basis
    Q_basis: Dict[Grading, List[BasisElement]] = {}
    for degree in pivots:
        for pivot in pivots[degree]:
            Q_basis[degree].push(f.codomain.basis[degree][pivot])


    // Make the tensor construct and deconstruct maps
    coalg = f.codomain.coalgebra
    construct, deconstruct = make_tensor_maps(coalg.basis, Q_basis)

    // Make coaction
    coaction: GradedMap = {}
    for degree in Q_basis:                                              // 1
        coaction[degree] =
            zero_matrix(len(deconstruct[degree]), len(Q_basis))         // 2

        for q_i in len(Q_basis[degree]):                                // 3
            f_i = pivots[degree][q_i]                                   // 4
            for f_tens in len(f.codomain.deconstruct[degree]):          // 5
                ((coalg_deg, coalg_id), (comod_deg, comod_id)) =        // 6
                        f.codomain.deconstruct[degree][f_tens]
                for c_q_i in len(Q_basis[comod_deg]):                   // 7
                    tensor_gr, tensor_id =                              // 8
                        construct[mod_gr][c_q_i][alg_gr][alg_id]
                    coaction[tensor_gr][tensor_id] +=                   // 9
                        P[comod_deg][c_q_i, comod_id]
                        * f.codomain.coaction[degree][f_tens, f_i]

    Q = Comodule(coalg, Q_basis, coaction, construct, deconstruct)
    return Morphism(f.codomain, Q, P)
}
```

For the cokernel we see that up until the making the coaction nothing scary happens. We just do everything in a graded way. Now to explain exactly what happens in the creation of the coaction. At step (1) we loop over all degrees in our Q_basis. Then we initialize a zero matrix with the correct dimensions for the coaction in this degree in step (2). We use the deconstruct map to find out how many basis elements there are for the tensor product.

Then we iterate for every basis element in the specific degree for Q and get element an in F, the codomain, that maps to that basis element in Q, steps (3), (4). We loop over every element in the tensor product belonging to the codomain of F. Using the deconstruct from F we get the original coalgebra degree and element and comodule degree and element, steps $(5),(6)$. So now we have an element of the form $a_i \otimes f_j \in A \otimes F$. We use P in the degree of f_j to map to an element in Q, (7). Then using the construct map on Q we reconstruct the basis element for the tensor product on Q, (8). Finally, multiply the value of the map P from comod_id to c_q_i with the coaction of F from f_i to f_tens, step (9).

This just comes down to chasing the following diagram for basis elements in Q, but now in a graded way,

$$\begin{tikzcd}
F \arrow[r, "\Delta_F"] \arrow[d, "P"'] & A \otimes F \arrow[d, "1\otimes P"] \\
Q \arrow[r, dashed, "\Delta_Q"'] & A \otimes Q
\end{tikzcd}$$

For the injection we see,

```
function injection(M: Comodule) -> Morphism {
    growing = zero_morphism(M, zero_comodule(M.coalgebra))
    a = len(M.coalgebra.basis)

    while True:
        // Take the kernel, but now in a graded way
        kernel = growing.map.null_space()

        // Still returns a BasisIndex,
        // but that now consists of both a degree and index
        m_1_degree, m_1_index = lowest_degree_element(kernel, M)

        // If no lowest degree element is found, we are done
        if m_1_index == -1:
            return growing

        // Create a cofree copy with a specific grading
        F = cofree(M.basis[m_1_degree][m_1_index].grading)

        // Construct the A-comodule morphism to F
        M_to_F: GradedMap = {}

        coalgebra_to_tensor = Q.construct[m_1_degree][m_1_index]

        // We go through all coalgebra grades
        for a_gr in coalgebra_to_tensor:
            tensor_grade = add_grade(m_1_degree, a_gr)


            zero_matrix = zero(len(coalgebra_to_tensor[a_gr]),
                                   len(M.basis[tensor_grade]))
            for a_id in len(coalgebra_to_tensor[a_gr]):
                (_, t_id) = coalgebra_to_tensor[a_gr][a_id]
                zero_matrix[a_id] = M.coaction[tensor_grade][t_id]

            mapping_to_F[tensor_grade] = zero_matrix

        m_1_morph = Morphism(M, F, M_to_F)

        growing = combine_morphisms(growing, m_1_morph)
}
```

Again nothing special happens here, except that we have to do more bookkeeping. We loop over all elements $a_i \otimes m\_1$, for basis elements $a_i \in A$, the coalgebra, and create the map to F depending on what the coaction on M is. The only place we should take notice is when taking the direct sum of the codomains. There we also have to be careful to update the construct and deconstruct maps to refer to the correct basis elements.

7 Example

We will calculate $\mathrm{Ext}^{s,t}_{A(2)_*}(\mathbb{F}_2, \mathbb{F}_2)$, for $0 \leq s \leq 20$ and $0 \leq t - s \leq 63$. Here $A(2)_*$ is the dual mod 2 steenrod subcoalgebra and this coalgebra is isomorphic to,

$$\mathbb{F}_2[\xi_1, \xi_2, \xi_3] \big/ \langle \xi_1^8, \xi_2^4, \xi_3^2 \rangle$$

For the general dual steenrod algebra, and also for the subcoalgebra, we have that the degree of $\xi_n = 2^n - 1$ and the coaction is,

$$\Delta_A(\xi_n) = \sum_{0 \leq i \leq n} \xi_{n-i}^{2^i} \otimes \xi_i$$

Because the steenrod algebra and its dual are both hopf algebras it suffices to define the coaction on only the indeterminates. The coaction is a ring homomorphism and that way we can find what the coaction does on all basis elements.

If we run our program we will see the following output,

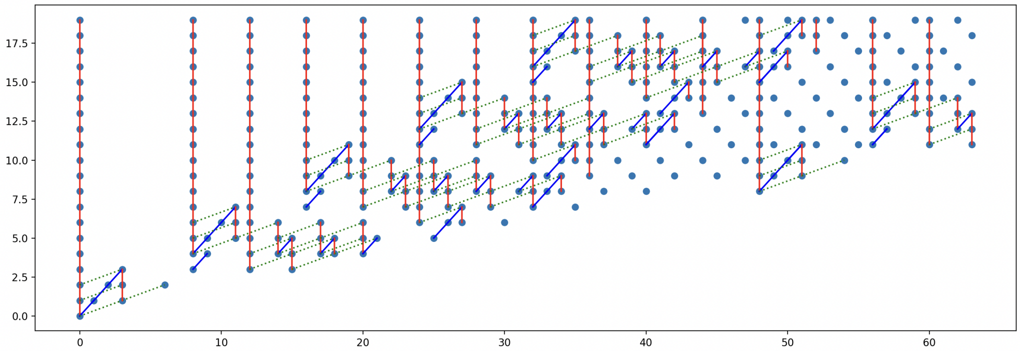

Figure 4: Our output

Assume we had the following coresolution,

$$0 \longrightarrow N \xrightarrow{\epsilon} A \otimes V_0 \xrightarrow{d_1} A \otimes V_1 \xrightarrow{d_2} A \otimes V_2 \xrightarrow{d_3} \cdots$$

Here, each dot represents a graded vector space basis element in a V_s. If a dot is in location x, y that would mean that we have a basis element in V_x in the degree $x + y$.

If we compare our output to the output from Robert R. Bruner's ext solver[1] we see,

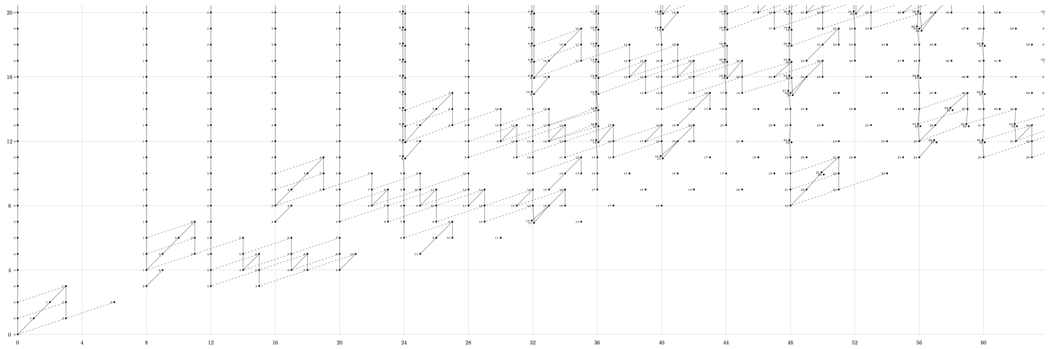

Figure 5: Robert R. Bruner's ext for $A(2)$

On careful inspection we can see that these are the same. Giving extra confidence that our program is correct.

Our program also has an advantage over the program from Robert R. Bruner because our program can calculate Ext for an arbitrary connected coalgebra. While Robert R. Bruner's program can only calculate Ext for A and $A(2)$, the steenrod algebra and subalgebra.

8 Future work

There are two places where one can look to improve the work so far,

- Work over polynomial rings instead of fields.
- Make the algorithm in Cpp instead of Python, to speed up calculations and to provide true parallelism.

Polynomial rings

When we started the thesis our goal was to write a piece of code which calculates Ext for the equivariant case. This would mean that instead of looking at a k-coalgebra we would be looking over $k[X, Y]$-coalgebras. Here X and Y both have a degree and the grading overall would become a bigrading, meaning the index set for the grading would become $\mathbb{N}^2$.

In this thesis we haven't succeeded in doing so yet. But we have written an algorithm which calculates Ext for specific coalgebras and comodules, and can create coresolutions in the more general case.

Future work can focus on expanding the current algorithm to accommodate for coalgebras over a polynomial ring in any number of unknowns. It should not be incredibly difficult to do this. Because it is enough to know where the $1 \in K[X_1, \cdots, X_n]$ of a polynomial ring is sent to. The rest of the map is now dictated by the fact that we have $k[X_1, \cdots, X_n]$-"linear" maps. Most of the work will probably be in verifying if all the math checks out. Where it is most important to look at what happens after Hom and what that implies for Ext. The programming will probably just be more bookkeeping.

Cpp

Right now we can run our code with the galois package or the sage environment. Although sage is roughly twice as fast as the galois package, sage is quite difficult to install, especially on Windows. This is why our algorithm supports both implementations

Both of these packages work with python, and sadly, python is not the fastest language. These packages were also not primarily written for speed, but for exactness and correctness. It is also not clear what translation layers are necessary from python to both of the libraries.

Next to that, our algorithm is quite suited to parallelism, because we can calculate most things in a graded way. Python sadly does not provide any thread parallelism due to its global interpreter lock. One could circumvent this by launching multiple processes. But they sadly can't share any memory, thus you would need to copy whole objects to do any parallelism. Both the process coordination and copying of objects turned out to take longer then the serial algorithm would take. Cpp however, provides enough options for true thread (or even 'green thread') parallelism.

All of this combined makes it worth it to switch over to the Cpp programming language, where we will use the linbox library to provide us with finite field linear algebra. Our estimations are that this will increase speed anywhere from 20 up until 1000 times. Which is not uncommon for number crunching programs which get translated from Python to a fast compiled language like Cpp.

9 Appendix

All code we have written can be found in the following Github repository[14]. To run the program we require a python3.11 installation, with the following modules installed: matplotlib, numpy and galois.

In our program the user can define their own coalgebra in two ways,

- Define the whole graded vector space basis and the coaction on each basis
- If the coalgebra is a polynomial hopf algebra, it suffices to define what the coaction and degree are for each indeterminate, and what the (monomial) relations[15] are.

To define the whole coalgebra we create a .txt file as follows,

```
– FIELD
p

– BASIS
name : degree
....

– GENERATOR
name_of_generator

– COACTION
name : k_i * name_1 | name_2 + ...
....
```

[14] The word GitHub should have a hyperlink, but here it is again. https://github.com/Chrisvossetje/ext-solver/

[15] The software does not support relations which are sums of different elements, for example, $\xi_1 + \xi_2$.

Here p is the prime field over which to work. For the basis we give each element a name and its degree. We also have to define what basis element is responsible for the connected part. Finally, we define for each basis element, by referencing its name, the coaction on that basis element. If $p = 2$, we don't write what k_i is, else k_i is the coefficient and name_1 | name_2 reference the tensor basis $v_1 \otimes v_2$.

Example 9.1. $A(1)_*$, the dual steenrod subcoalgebra is encoded as follows,

```
– FIELD
2

– BASIS
1 : (0, 0)
xi1^1 : (1, 0)
xi2^1 : (3, 0)
xi1^3 : (3, 0)
xi1^2 : (2, 0)
xi1^1xi2^1 : (4, 0)
xi1^2xi2^1 : (5, 0)
xi1^3xi2^1 : (6, 0)

– GENERATOR
1

– COACTION
1 : 1|1
xi1^1 : xi1^1|1 + 1|xi1^1
xi2^1 : xi2^1|1 + xi1^2|xi1^1 + 1|xi2^1
xi1^3 : xi1^3|1 + xi1^2|xi1^1 + 1|xi1^3 + xi1^1|xi1^2
xi1^2 : xi1^2|1 + 1|xi1^2
xi1^1xi2^1 : xi1^1xi2^1|1 + xi2^1|xi1^1 + xi1^3|xi1^1 + more_terms
xi1^2xi2^1 : xi1^2xi2^1|1 + xi1^2|xi2^1 + xi1^2|xi1^3 +  more_terms
xi1^3xi2^1 : xi1^3xi2^1|1 + xi1^2xi2^1|xi1^1 + xi1^3|xi2^1 + more_terms
```

If the coalgebra is a polynomial hopf algebra we define it as follows,

```
– FIELD
p

– GENERATOR
name : degree
....

– RELATION
name_1^i*name_2^j*....

– COACTION
name : k_i * name_1 | name_2 + ...
```

Now it is enough to define the coaction on each generator, because in a hopf algebra we have that the coaction must also respect the multiplicative structure.

Example 9.2. $A(2)_*$, another dual steenrod subcoalgebra is encoded as a hopf algebra as follows,

```
– FIELD
2

– GENERATOR
xi1: (1,0)
xi2: (3,0)
xi3: (7,0)

– RELATION
xi1^8
xi2^4
xi3^2


– COACTION
xi1: 1|xi1 + xi1|1
xi2: 1|xi2 + xi2|1 + xi1^2|xi1
xi3: 1|xi3 + xi3|1 + xi2^2|xi1 + xi1^4|xi2
```

Our program also has the ability to make a coresolution for any N, where N is an A-comodule. When no comodule is provided the coresolution will be made for k, as an A-comodule.

A comodule is also defined in a .txt file as follows,

```
– FIELD
p

– BASIS
name : degree
....

– COACTION
name : k_i * name_1 | name_2 + ...
```

Here name_1 comes from the coalgebra we have just defined, and name_2 comes from the basis of the comodule.

In globals.py the user can define what the maximum filtration index and the maximum stem index is.

References

[1] Robert R. Bruner. *ext1.9.5, ext calculator for the steenrod algebra.* 2022. URL: `http://www.rrb.wayne.edu/papers/#code`.

[2] David S. Dummit and Richard M. Foote. *Abstract Algebra, third edition.* John Wiley & Sons, Inc., 2004.

[3] Dan Isaksen. *Classical and C-motivic Adams charts.* 2022. URL: `https://s.wayne.edu/isaksen/adams-charts/`.

[4] John Milnor. "The Steenrod Algebra and Its Dual". In: *Annals of Mathematics* 67.1 (1958).

[5] John W. Milnor and John C. Moore. “On the Structure of Hopf Algebras”. In: *Mathematics Department, Princeton University* 81.2 (1965).

[6] *Sage CAS*. URL: `https://www.sagemath.org/`.