Algorithm to find all submodules of a module

Studforsk

Didrik Fosse

2018

Contents

1 Introduction

In this text I will provide an algorithm in GAP to find all submodules of any finite dimensional module over a finite dimensional algebra over a finite field.

2 The Algorithm

Assume that $\Lambda = kQ/I$ for a finite field k, a (finite) quiver Q and an admissible ideal I. Given a finite dimensional Λ-module M, our goal is to find all submodules of M. The first step is to find all the simple submodules of M. To do so, it is sufficient to consider the socle of M, since $\mathrm{soc}M$ is precisely the submodule of M generated by all the simple submodules of M. We have that $\mathrm{soc}M$ can be written as follows

$$\mathrm{soc}M \simeq S_1^{m_1} \oplus S_2^{m_2} \oplus \cdots \oplus S_n^{m_n}$$

where S_i is simple for all i, and $S_i \not\simeq S_j$ for $i \neq j$. Now, the simple submodules of M are all the one dimensional submodules of each of the direct summands $S_i^{m_i}$ in $\mathrm{soc}M$. Since each $S_i^{m_i}$ is m_i dimensional as a k-vector space, we can find the one dimensional submodules by considering the different lines through the origin in k^{m_i}. Of course, for any given i, these submodules are all isomorphic as Λ-modules, but they are different submodules of M nonetheless. This gives us all simple submodules of M. Now, for each simple submodule S in M look at the inclusion

$$S \hookrightarrow M$$

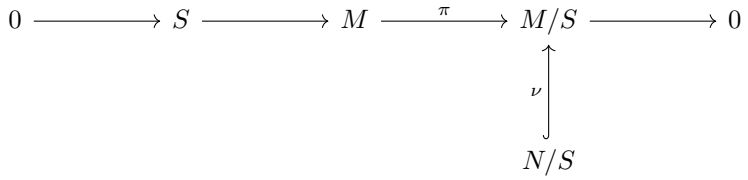
Take the cokernel of the inclusion to get the short exact sequence

$$0 \longrightarrow S \longrightarrow M \xrightarrow{\pi} M/S \longrightarrow 0$$

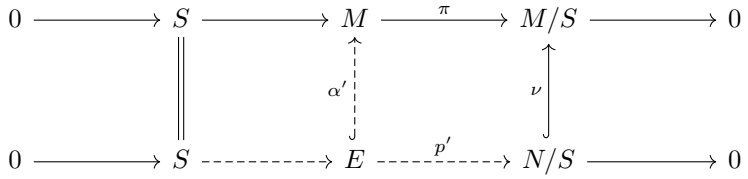
Now, by recursively calling the algorithm, we find all submodules of M/S. We know that all submodules of M/S are of the form N/S, where N is a submodule of M which cointains S. So for each submodule $N/S \subseteq M/S$ we can find $N \subseteq M$ such that the following diagram commutes.

$$\begin{array}{ccccccccc} 0 & \longrightarrow & S & \longrightarrow & M & \xrightarrow{\pi} & M/S & \longrightarrow & 0 \\ & & \| & & \uparrow{\scriptstyle \alpha} & & \uparrow{\scriptstyle \nu} & & \\ 0 & \longrightarrow & S & \longrightarrow & N & \xrightarrow{p} & N/S & \longrightarrow & 0 \end{array}$$

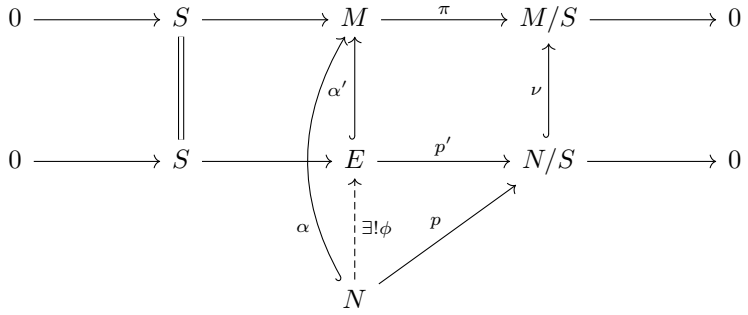
In order to find this N, we take the pullback of the following diagram.



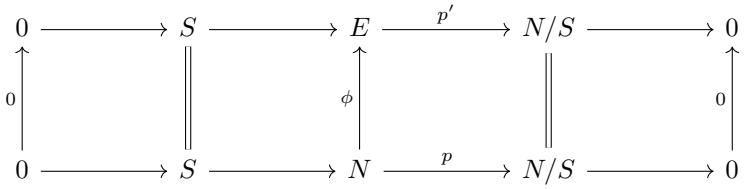
Since taking pullback preserves kernels, we get the following diagram.



In QPA, submodules of M are represented as the image of inclusions into M. So to show that the submodule given by taking the pullback is equal to the submodule N, we must show that $\operatorname{Im}\alpha' = \operatorname{Im}\alpha$. In order to do so, we insert the module N in the diagram as follows.



By the universal property of pullbacks, we know that there is a unique map $\phi : N \to E$ such that the diagram commutes. Since $\alpha = \alpha' \circ \phi$, it is obvious that $\operatorname{Im}\alpha \subseteq \operatorname{Im}\alpha'$. To show that we have equality, it suffices to show that ϕ is an isomorphism. Consider the following diagram.



By the Five Lemma we get that ϕ is an isomorphism. This shows that $\operatorname{Im}\alpha' = \operatorname{Im}\alpha' \circ \phi = \operatorname{Im}\alpha$, which is what we wanted to show. In other words, given a submodule $N/S \subseteq M/S$, taking pullback will return the corresponding submodule $N \subseteq M$. This allows us to define an algorithm to find all proper nonzero submodules of a given module M:

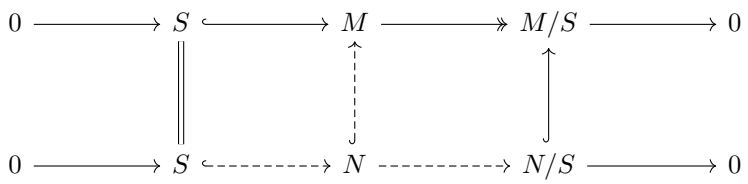
```
begin
    create an empty list called submodules;
    find socM, the socle of M;
    for each simple submodule S in socM do
        add S to the list submodules;
        create the short exact sequence S → M → M/S;
        recursively call this algorithm to find all submodules of M/S;
        for each submodule of M/S do
            take the pullback of the morphisms M → M/S and
              N/S → M/S;
            add the image of the pullback of N/S → M/S to the list
              submodules;
        end
    end
    return submodules
end
```

Be aware that the algorithm ensures that the list `submodules` contains at least one copy of each proper nonzero submodule, but it doesn't limit it to one copy of each. This means that the list may contain multiple copies of each submodule of M, and hence we need a way to get rid of the duplicates. Additionally, the algorithm given above finds only the proper nonzero submodules of M. Since we actually want all submodules, we must not forget to add M and (0). Before we address these two issues, we will prove that this algorithm indeed does find all proper nonzero submodules of M.

Theorem 1. *For any module M which is finite dimensional over a finite dimensional algebra over a finite field, the algorithm described in the pseudocode above returns all proper nonzero submodules of M.*

Proof. In order to show that this method actually gives all submodules of M, we will use induction on the length of M. We know that $\ell(M) = 1$ if and only if M is a simple module. Since the algorithm finds all simple submodules, this gives the base case of the induction. Now, let us assume that we can find all submodules of any module of length k, and let M be a module of length $\ell(M) = k + 1$. We need to show that we can find all submodules of M. Let $0 \neq N \subsetneq M$ be an arbitrary submodule. Since N is finite and nonzero, it must contain a simple submodule. And since we can find all simple submodules of M, we can find a simple submodule S such that $S \subseteq N \subsetneq M$. Now, since $\ell(M) = \ell(S) + \ell(M/S)$, we have that $\ell(M/S) = \ell(M) - \ell(S) = k + 1 - 1 = k$. Thus, by our induction hypothesis, we can find all submodules of M/S. Since

N is a submodule of M, we know that N/S must be a submodule of M/S. So we find $N/S \subseteq M/S$, and by applying our pullback construction



we get the submodule N. Since N was an arbitrary submodule of M, this concludes the proof by induction. □

The algorithm itself is divided into two functions, `AllSubModules` and `AllSubModulesRecursive`. The method described in the pseudocode above is implemented in the function `AllSubModulesRecursive`, which creates a list of all proper nonzero submodules of the given module M. The submodules in this list are represented in QPA as the image of inclusions of submodules into M. Unfortunately, the list contains multiple copies of many of the submodules, and we only want each of them to appear once. To solve this issue, we use the function `AllSubModules`. This function calls `AllSubModulesRecursive` to generate a list called `submodules`, and then it creates the new list `differentsubmodules` picking out one copy of each of the different submodules in `submodules`. It does so by iterating through `submodules`, and for each submodule it checks if that submodule is equal to any of the submodules in `differentsubmodules`. If not, then that submodule is added to `differentsubmodules`. The function `AllSubModulesRecursive` also adds the submodules 0 and M to the list `differentsubmodules`, because `AllSubModulesRecursive` only finds the proper nonzero submodules of M.

3 Code

```
1   DeclareOperation( "AllSubModules" , [IsPathAlgebraMatModule]);
2   InstallMethod( AllSubModules, "for a representation",
                                    [IsPathAlgebraMatModule], function(M)
3     local   submodules, Vspaces,  differentsubmodules,  f,  V;
4
5           submodules := AllSubModulesRecursive( M );
6           Vspaces := [ ];
7           differentsubmodules := [ ];
8           for f in submodules do
9             V := Subspace( UnderlyingLeftModule( M ), List( BasisVectors( Basis(
                          Source( f ) ) ), b -> ExtRepOfObj( ImageElm( f, b ) ) ) );
10            if not ( V in Vspaces ) then
11              Add( Vspaces, V );
12              Add( differentsubmodules, f );
13            fi;
14          od;
15
16
17        Add( differentsubmodules, SubRepresentationInclusion( M, [] ), 1 );
18        Add( differentsubmodules, IdentityMapping( M ) );
19        Print("Length of submodules: ", Length(submodules), "\n");
20        Print("Length of differentsubmodules: ", Length(differentsubmodules), "\n");
21        return differentsubmodules;
22
23  end
24  );
```

```
25  DeclareOperation( "AllSubModulesRecursive" , [IsPathAlgebraMatModule]);
26  InstallMethod( AllSubModulesRecursive, "for a representation",
                                         [IsPathAlgebraMatModule], function(M)
27        local submodules, field, simples, soc, socDecMult, socDims, k, i, dims,
                  simpleSocleSummands, socleSummands, mats, j, inclusionInSocle,
                  s, ms, u, newSubModule;
28
29        submodules:=[];
30        if IsZero( M ) then
```

```
31              return;
32      fi;
33      if IsSimpleQPAModule( M ) then
34              return submodules;
35      else
36              field := LeftActingDomain( M );
37              simples := [];
38              soc := SocleOfModule( M );
39              socDecMult := DecomposeModuleWithMultiplicities( soc );
40              socDims := DimensionVector( soc );
41              k := Length( socDecMult[ 1 ] );
42              for i in [ 1..k ] do
43                      simpleSocleSummands := ListWithIdenticalEntries(
                                socDecMult[ 2 ][ i ], socDecMult[ 1 ][ i ] );
44                      socleSummands := DirectSumOfQPAModules( simpleSocleSummands );
45                      dims := DimensionVector( socleSummands );
46                      mats := [];
47                      for j in [ 1..Length(dims) ] do
48                              if ( dims[j] = 0 ) and ( socDims[ j ] = 0 ) then
49                                      Add( mats, NullMat( 1, 1, field ) );
50                              elif ( dims[ j ] = 0 ) and ( socDims[ j ] > 0 ) then
51                                      Add( mats, NullMat( 1, socDims[ j ], field ) );
52                              elif ( dims[ j ] = socDims[ j ] ) then
53                                      Add( mats, IdentityMat( dims[ j ], field ) );
54                              fi;
55                      od;
56                      inclusionInSocle := RightModuleHomOverAlgebra( socleSummands,
                                                                        soc, mats );
57                      for s in Subspaces( socleSummands, 1 ) do
58                              Add( simples, RightModuleHomOverAlgebra(
                                      socDecMult[ 1 ][ i ], socleSummands, List(
                                      ExtRepOfObj( ExtRepOfObj( BasisVectors(
                                      Basis( s ) )[ 1 ] ) ),
                                      b -> [ b ] ) ) * inclusionInSocle );
59                      od;
60              od;
61              for s in simples do
62                      Add( submodules, s * SocleOfModuleInclusion( M ) );
63                      ms := CoKernelProjection( s * SocleOfModuleInclusion( M ) );
64                      for u in AllSubModulesRecursive( Image( ms ) ) do
65                              newSubModule := PullBack( ms, u )[ 2 ];
66                              Add( submodules, newSubModule );
67                      od;
68              od;
69      fi;
70  return submodules;
```

```
71  end
72  );
```