Software and algorithms for representations of algebras and categories

Peter Webb

University of Minnesota

16 February, 2015

Outline:

Why do representations of categories?

How I encode a category in GAP

Algorithms that are available so far

What needs to be done?

Computing Auslander-Reiten sequences

References

- P.J. Webb, *An introduction to the representations and cohomology of categories,* pp. 149-173 in: M. Geck, D. Testerman and J. Thévenaz (eds.), Group Representation Theory, EPFL Press (Lausanne) 2007.
- The tutorials on representations of categories accessible from Webb's home page.

Categories of interest

Many categories arise from finite groups.

- the fusion category or Frobenius category
- linking systems associated to p-local finite groups
- Quillen's subcategory of the fusion category
- the orbit category (objects are finite G-sets, morphisms are equivariant maps)
- posets

Other categories:

- abstract EI categories (Endomorphisms are Isomorphisms)
- categories used to form diagrams of topological spaces, where the Bousfield-Kan spectral sequence computes the homology of the homotopy colimit.

What needs to be computed, and why?

- Cohomology:
  - limits are H^0 and H_0
  - the vanishing of Ext is important in the existence and uniqueness of linking systems (resolved by Chermak, Glauberman-Lynd in a different way).
  - Fei Xu's counterexample to the conjecture of Snashall and Solberg is an application of the theory.
  - Is the orbit category of a finite group Koszul?
- Indecomposable representations:
  - computations may have been useful in Liping Li's (partial) determination of EI categories of finite representation type.
  - Auslander-Reiten quivers

Why keep the category, rather than simply regarding the category algebra as an algebra?

Representations of a category $\mathcal{C}$ over R are the same thing as modules for the category algebra $R\mathcal{C}$.

- Like group algebras, category algebras have a multiplicative basis: we have a $\otimes$ on representations and there is a distinguished representation $\underline{R}$.
- The category algebra comes with a set of orthogonal idempotents, one for each object in the category. Endomorphisms can be stored as lists of matrices which preserve the images of the idempotents.
- It is convenient to have the structure of the category in the computer when constructing a representation.
- For posets or EI categories with p-groups as endomorphism groups in characteristic p, condensation does nothing.

Concrete categories

A concrete category is one in which all the objects are (disjoint) sets and the morphisms are morphisms of sets.

Theorem (Analogue of Cayley's theorem)

Every finite category is isomorphic to a concrete category.

Encoding a category

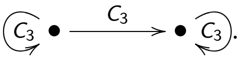
I encode the objects of a category as a list of lists:

$$\{\{1,\ldots,a_1\},\{a_1+1,\ldots,a_1+a_2\},\ldots\}$$

Each morphism is encoded as the (partially defined) list of values of a function from one of the sets to another. To enter a category we enter a list of generating morphisms.

Example

```
gap> c3c3:=ConcreteCategory([[2,3,1],[4,5,6],[,,,5,6,4]]);
rec( generators := [ [ 2, 3, 1 ], [ 4, 5, 6 ], [ ,,, 5, 6, 4 ] ],
objects := [ [ 1, 2, 3 ], [ 4, 5, 6 ] ],
domain := [ 1, 1, 2 ], codomain := [ 1, 2, 2 ] )
```

Picture: 

Other possibilities

A category could be stored as

- An abstract list of objects, a list of morphisms between them which generate a free category, and a list of relations between them. This seems like a good idea and would be appropriate when quivers are part of the theory.
- A matrix representation?

These are not what I have done!

Representations in the computer

A representation of a category $\mathcal{C}$ over R is a functor $M : \mathcal{C} \to R$-mod. Thus, for each object of $\mathcal{C}$ an R-module is specified and for each morphism a linear map is specified between the corresponding R-modules.

A representation is entered as a list of matrices which represent the action of the category generators.

Example

```
gap> one:=One(GF(3));;
gap>
d:=[[1,1,0,0,0],[0,1,1,0,0],[0,0,1,0,0],[0,0,0,1,1],[0,0,0,0,1]]*one;;
gap> e:=[[0,1,0,0],[0,0,1,0],[0,0,0,0],[0,1,0,1],[0,0,1,0]]*one;;
gap> f:=[[1,1,0,0],[0,1,1,0],[0,0,1,0],[0,0,0,1]]*one;;

gap> nine:=CatRep(c3c3,[d,e,f],GF(3));
rec(category := rec( generators := [ [ 2, 3, 1 ], [ 4, 5, 6 ], [
,,, 5, 6, 4 ] ], objects := [ [ 1, 2, 3 ], [ 4, 5, 6 ] ],
domain := [ 1, 1, 2 ], codomain := [ 1, 2, 2 ] ),
genimages := [ [ [ Z(3)^0, Z(3)^0, 0*Z(3), 0*Z(3), 0*Z(3) ],
[ 0*Z(3), Z(3)^0, Z(3)^0, 0*Z(3), 0*Z(3) ],
```

etc

```
[ 0*Z(3), 0*Z(3), 0*Z(3), Z(3)^0 ] ] ], field := GF(3),
dimension := [ 5, 4 ] )
```

Note that although the total dimension of the representation space is 9, the matrices which represent the action of the generators are smaller than 9×9.

Algorithms which stay within the category

- Book-keeping: set up the category, compute all morphisms within the category, find the origin or terminus of a morphism, find whether two morphisms are composable, etc.
- Compute homology of the nerve as a topological space (in effect, the bar resolution).

Example

```
gap> for i in [1..3] do Print(Homology(c3c3,i)); od;
[ 3 ][ ][ 3 ]
gap> for i in [1..3] do Print(HomologyDimension(c3c3,i,3)); od;

111
```

The first calculation finds the first three homology groups over $\mathbb{Z}$ of the nerve of c3c3, returning invariant factors. The second calculation does the same over $\mathbb{F}_3$, returning dimensions.

Available algorithms for representations

- Construct: the constant functor, the representable projectives, the corepresentable injectives.
- Form the tensor product of two representations.
- Find the submodule generated by a set of vectors.
- Find the quotient module by a submodule.
- Find the space of homomorphsms between two modules.
- Find the sum of the images under all homomorphisms between two modules.
- Find the decomposition of a module into indecomposable summands.

A further calculation with c3c3

We return to the representation `nine` of the category c3c3. It is indecomposable:

```
gap> Length(Decompose(nine));
1
```

```
gap> fortyone:=TensorProductRep(nine,nine);;
```

This produces a representation of dimension 25 at object 1, and a space of dimension 16 at object 2. We examine its indecomposable summands.

```
gap> d:=Decompose(fortyone);;
gap> List(d,x->List(x,Length));
```

```
[ [ 3, 0 ], [ 3, 1 ], [ 3, 3 ], [ 3, 3 ], [ 0, 3 ], [ 3, 0 ], [
3, 0 ], [ 3, 0 ], [ 1, 3 ], [ 3, 3 ] ]
```

Thus fortyone has 10 indecomposable summands, with dimension lists as above. In the tutorial available from my home page we do a further calculation with the structure of the third summand, determining completely its structure.

What needs to be done?

- Compute cohomology
  - using resolutions and

Theorem
$H^*(|\mathcal{C}|) \cong \mathrm{Ext}^*_{R\mathcal{C}}(\underline{R}, \underline{R})$

  - other approaches to Ext^1.

- 'Character' theory
- chain complexes

An algorithm for Auslander-Reiten sequences

See: P.J. Webb, *Consequences of the existence of Auslander-Reiten triangles...*, arXiv:1301.4701

Let C be an indecomposable module for a finite-dimensional algebra.

1. construct a projective presentation $P_1 \to P_0 (\to C \to 0)$;
2. compute the effect of the Nakayama functor: $\nu P_1 \to \nu P_0$;
3. compute a map $f : \mathcal{P} \to \mathcal{I}$ which is in the socle of the action of $\mathrm{End}(C)$ on homotopy classes of these maps;
4. compute the mapping cone $C(f)$. We obtain a short exact sequence $0 \to \mathcal{I} \to C(f) \to \mathcal{P} \to 0$;
5. compute H_1 of this sequence.

In (2) we construct the Auslander-Reiten translate $\tau(C)$ as the kernel of $\nu P_1 \to \nu P_0$. For some algebras the Nakayama functor is straightforward; for instance if the algebra is symmetric, $\nu = 1$. Steps 4 and 5 are routine. Step 3 is a problem.

Step 3. compute a map $f : \mathcal{P} \to \mathcal{I}$ which is in the socle of the action of $\mathrm{End}(C)$ on homotopy classes of these maps

Quite often the map f can be constructed as

$$\begin{array}{ccc} (P_1 & \longrightarrow & P_0) \\ {\scriptstyle 0}\big\downarrow & & {\scriptstyle \alpha}\big\downarrow \\ (\nu P_1 & \longrightarrow & \nu P_0) \end{array}$$

where α maps P_0 to a simple submodule of νP_0.