SBStrips

discrete models of special biserial algebras, string modules and their syzygies

version 0.6.0

Joe Allen

Copyright

Contents

Chapter 1

Introduction

1.1 Why "strips", not "strings"?

First, some context. Representation theorists use the word *string* to mean a decorated graph that, in a particular fashion, describes a module; it is accordingly called a *string module*. Liu and Morin [**?**] showed that the syzygy of a string module over a special biserial (SB) algebra is a direct sum of string modules. Their proof is constructive, detailing how to obtain the strings indexing the syzygy summands from the string indexing the original module. Their language explains how to spot patterns appearing "from one syzygy to the next", but it does not scale in a particularly transparent way. For example, I believe it does not lend itself to clearly seeing asymptotic behaviour of syzygies of string modules. My research has aimed, in part, to provide a more robust language: one which lays bare more patterns in the syzygies of string modules over SB algebras.

One key ingredient is a slight refinement of the definition of a string. Really, this differs from the established definition only in technical ways, the effect being to disambiguate how the graph is decorated so that the syzygy calculation is streamlined. In my thesis, I propose the term *strip* for this refined notion of a string. A happy side-effect of this name change is that it avoids the clash with what GAP already thinks "string" means.

In brief: if whenever you read the word "strip" here, you imagine that it means the kind of decorated graph that representation theorists call a "string", then you won't go too far wrong.

1.2 Aims

1.3 Installation

Chapter 2

Worked example

2.1 Strips, aka "strings for special biserial algebras"

2.2 Calculations with strips

2.3 A look under the bonnet

Chapter 3

Quivers and special biserial algebras

3.1 Introduction

Quivers are finite directed graphs. Paths in a given quiver Q can be concatenated in an obvious way, and this concatenation can be extended K-linearly (over a field K) to give an associative, unital algebra KQ called a *path algebra*. A path algebra is infinite-dimensional iff its underlying quiver Q is acyclic. Finite-dimensional *quiver algebras* – that is, finite-dimensional quotient algebras KQ/I of a path algebra KQ by some (frequently admissible) ideal I – are a very important class of rings, whose representation theory has been much studied.

The excellent QPA package implements these objects in GAP. The (far more humdrum) SB-Strips package extends QPA's functionality. Quivers constructed using the QPA function `Quiver` (**QPA: Quivers**) belong to the filter `IsQuiver` (**QPA: IsQuiver**), and special biserial algebras are those quiver algebras for which the property `IsSpecialBiserialAlgebra` (**QPA: IsSpecialBiserialAlgebra**) returns `true`.

In this section, we explain some added functionality for quivers and special biserial algebras.

3.2 New property of quivers

3.2.1 Is1RegQuiver

▷ `Is1RegQuiver(`*`quiver`*`)` (property)

Argument: *`quiver`*, a quiver
Returns: either `true` or `false`, depending on whether or not *`quiver`* is 1-regular.

3.2.2 IsOverquiver

▷ `IsOverquiver(`*`quiver`*`)` (property)

Argument: *`quiver`*, a quiver
Returns: `true` if *`quiver`* was constructed by **??**, and `false` otherwise.

3.3 New attributes of quivers

3.3.1 1RegQuivIntAct

▷ `1RegQuivIntAct(x, k)` (operation)

Arguments: `x`, which is either a vertex or an arrow of a 1-regular quiver; `k`, an integer.

Returns: the path $x+k$, as per the $\mathbb{Z}$-action (see below).

Recall that a quiver is 1-regular iff the source and target functions s,t are bijections from the arrow set to the vertex set (in which case the inverse t^{-1} is well-defined). The generator $1 \in \mathbb{Z}$ acts as "t^{-1} then s" on vertices and "s then t^{-1}" on arrows.

This operation figures out from `x` the quiver to which `x` belongs and applies `1RegQuivIntActionFunction` (3.3.2) of tha quiver. For this reason, it is more user-friendly.

3.3.2 1RegQuivIntActionFunction

▷ `1RegQuivIntActionFunction(quiver)` (attribute)

Argument: `quiver`, a 1-regular quiver (as tested by `Is1RegQuiver` (3.2.1))

Returns: a single function `f` describing the $\mathbb{Z}$-actions on the vertices and the arrows of `quiver`

Recall that a quiver is 1-regular iff the source and target functions s,t are bijections from the arrow set to the vertex set (in which case the inverse t^{-1} is well-defined). The generator $1 \in \mathbb{Z}$ acts as "t^{-1} then s" on vertices and "s then t^{-1}" on arrows.

In practice you will probably want to use `1RegQuivIntAct` (3.4.1), since it saves you having to remind SBStrips which quiver you intend to act on.

3.3.3 2RegAugmentationOfQuiver

▷ `2RegAugmentationOfQuiver(ground_quiv)` (attribute)

Argument: `ground_quiv`, a sub2-regular quiver (as tested by `IsSpecialBiserialQuiver` (**QPA: IsSpecialBiserialQuiver**))

Returns: a 2-regular quiver of which `ground_quiv` may naturally be seen as a subquiver

If `ground_quiv` is itself sub-2-regular, then this attribute returns `ground_quiv` identically. If not, then this attribute constructs a brand new quiver object which has vertices and arrows having the same names as those of `ground_quiv`, but also has arrows with names `augarr1`, `augarr2` and so on.

3.4 Operations on vertices and arrows of quivers

3.4.1 1RegQuivIntAct

▷ `1RegQuivIntAct(x, k)` (operation)

Arguments: `x`, which is either a vertex or an arrow of a 1-regular quiver; `k`, an integer.

Returns: the path $x+k$, as per the $\mathbb{Z}$-action (see below).

Recall that a quiver is 1-regular iff the source and target functions s,t are bijections from the arrow set to the vertex set (in which case the inverse t^{-1} is well-defined). The generator $1 \in \mathbb{Z}$ acts as

"t^{-1} then s" on vertices and "s then t^{-1}" on arrows.
This operation figures out from `x` the quiver to which `x` belongs and applies `1RegQuivIntActionFunction` (3.3.2) of tha quiver. For this reason, it is more user-friendly.

3.4.2 PathBySourceAndLength

▷ `PathBySourceAndLength(vert, len)` (operation)

Arguments: *vert*, a vertex of a 1-regular quiver Q; *len*, a nonnegative integer.
Returns: the unique path in Q which has source *vert* and length *len*.

3.4.3 PathByTargetAndLength

▷ `PathByTargetAndLength(vert, len)` (operation)

Arguments: *vert*, a vertex of a 1-regular quiver Q; *len*, a nonnegative integer.
Returns: the unique path in Q which has target *vert* and length *len*.

3.5 New attributes for special biserial algebras

3.5.1 OverquiverOfSbAlg

▷ `OverquiverOfSbAlg(sba)` (attribute)

Argument: *sba*, a special biserial algebra
Returns: a quiver `oquiv` with which uniserial *sba*-modules can be conveniently (and unambiguously) represented.

3.5.2 SimpleStripsOfSbAlg

▷ `SimpleStripsOfSbAlg(sba)` (attribute)

Argument: *sba*, a special biserial algebra (ie, `IsSpecialBiserialAlgebra` (**QPA: IsSpecialBiserialAlgebra**) returs `true`)
Returns: a list `simple_list`, whose jth entry is the simple strip corresponding to the jth vertex of *sba*.
You will have specified *sba* to GAP via some quiver. The vertices of that quiver are ordered; `SimpleStripsOfSbAlg` adopts that order for strips of simple modules.

3.5.3 ProjectiveStripsOfSbAlg

▷ `ProjectiveStripsOfSbAlg(sba)` (attribute)

Argument: *sba*, a special biserial algebra (ie, `IsSpecialBiserialAlgebra` (**QPA: IsSpecialBiserialAlgebra**) returs `true`)
Returns: a list `proj_list`, whose entry are either strips or the boolean `fail`.
You will have specified *sba* to GAP via some quiver. The vertices of that quiver are ordered; `ProjectiveStripsOfSbAlg` adopts that order for strips of projective modules.

If the projective module corresponding to the `j`th vertex of *sba* is a string module, then `ProjectiveStripsOfSbAlg( sba )[j]` returns the strip describing that string module. If not, then it returns `fail`.

3.5.4 InjectiveStripsOfSbAlg

▷ `InjectiveStripsOfSbAlg(sba)` (attribute)

Argument: *sba*, a special biserial algebra (ie, `IsSpecialBiserialAlgebra` (**QPA: IsSpecialBiserialAlgebra**) returs `true`)

Returns: a list `inj_list`, whose entry are either strips or the boolean `fail`.

You will have specified *sba* to GAP via some quiver. The vertices of that quiver are ordered; `InjectiveStripsOfSbAlg` adopts that order for strips of projective modules.

If the injective module corresponding to the `j`th vertex of *sba* is a string module, then `InjectiveStripsOfSbAlg( sba )[j]` returns the strip describing that string module. If not, then it returns `fail`.

3.6 New function for special biserial algebras

3.6.1 TestInjectiveStripsUpToNthSyzygy

▷ `TestInjectiveStripsUpToNthSyzygy(sba, N)` (function)

Arguments: *sba* a special biserial algebra (ie, `IsSpecialBiserialAlgebra` (**QPA: IsSpecialBiserialAlgebra**) returs `true`); *N*, a positive integer

Returns: `true`, if all strips of injective string modules have finite syzygy type by the *N*th syzygy, and `false` otherwise.

This function calls `InjectiveStripsOfSbAlg` (3.5.4) for *sba*, filters out all the `fail`s, and then checks each remaining strip individually using `IsFiniteSyzygyTypeStripByNthSyzygy` (7.4.1) (with second argument *N*).

Author's note. For every special biserial algebra I test, this function returns true for sufficiently large *N*. It suggests that the injective cogenerator of a SB algebra always has finite syzygy type. This condition implies many homological conditions of interest (including the big finitistic dimension conjecture)!

Chapter 4

Permissible data

Chapter 5

Syllables

5.1 Introduction

5.2 Properties of syllables

5.2.1 IsStationarySyllable

▷ `IsStationarySyllable(sy)` (property)

Argument: `sy`, a syllable

Returns: either `true` or `false`, depending on whether or not the underlying path of `sy` is a stationary path.

Chapter 6

Patches

Chapter 7

Strips

7.1 Introduction

7.2 Constructing strips

7.3 Attributes of strips

7.3.1 WidthOfStrip

▷ `WidthOfStrip(`*`strip`*`)` (operation)

Argument: *`strip`*, a strip
Returns: a nonnegative integer, counting the number (with multiplicity) of syllables of *`strip`* are nonstationary.

7.4 Operation on strips

7.4.1 IsFiniteSyzygyTypeStripByNthSyzygy

▷ `IsFiniteSyzygyTypeStripByNthSyzygy(`*`strip, N`*`)` (operation)

Arguments: *`strip`*, a strip; *`N`*, a positive integer
Returns: `true` if the strips appearing in the *`N`*th syzygy of *`strip`* have all appeared among earlier syzygies, and `false` otherwise.

If the call to this function returns `true`, then it will also print the smallest *`N`* for which it would return `true`.

7.4.2 IsPeriodicStripByNthSyzygy

▷ `IsPeriodicStripByNthSyzygy(`*`strip, N`*`)` (operation)

Arguments: *`strip`*, a strip; *`N`*, a positive integer
Returns: `true` if *`strip`* is appears among its own first *`N`* syzygies, and `false` otherwise.

If the call to this function returns `true`, then it will also print the index of the syzygy at which *`strip`* first appears.