homalg

A homological algebra meta-package for computable Abelian categories

Version 2015.03.31

March 2015

Mohamed Barakat

Markus Lange-Hegermann

(*this manual is still under construction*)

This manual is best viewed as an HTML document. The latest version is available ONLINE at:

http://homalg.math.rwth-aachen.de/~barakat/homalg-project/homalg/chap0.html

An OFFLINE version should be included in the documentation subfolder of the package. This package is part of the homalg-project:

http://homalg.math.rwth-aachen.de/index.php/core-packages/homalg-package

Mohamed Barakat
Email: barakat@mathematik.uni-kl.de
Homepage: http://www.mathematik.uni-kl.de/~barakat/
Address: Department of Mathematics,
University of Kaiserslautern,
67653 Kaiserslautern,
Germany

Markus Lange-Hegermann
Email: markus.lange.hegermann@rwth-aachen.de
Homepage: http://wwwb.math.rwth-aachen.de/~markus
Address: Lehrstuhl B für Mathematik, RWTH Aachen, Templergraben 64, 52056 Aachen, Germany

Copyright

Acknowledgements

Max Neunhöffer not only taught me the philosophy of object-oriented programming in GAP4, but also to what extent this philosophy is still unique among programming languages (→ B.2). The slides (in German) to his talk in our seminar on 30.10.2006 can be found on his homepage.

He, Frank Lübeck, and Thomas Breuer patiently answered trillions of specific questions, even those I was too lazy to look up in the excellent reference manual. Without their continuous and tireless help and advice, not only this package but the as a whole homalg project would have remained on my todo list.

A lot of ideas that make up this package and the whole homalg project came out of intensive discussions with Daniel Robertz during our early collaboration, where we developed our philosophy of a meta package for homological algebra and implemented it in Maple.

In the fall of 2007 I began collaborating with Simon Görtzen to further pursue and extend these ideas preparing the transition to GAP4. With his help homalg became an extendable multi-package project.

Max Neunhöffer convinced me to use his wonderful IO package to start communicating with external computer algebra systems. This was crucial to remedy the yet missing support for important rings in GAP. Max provided the first piece of code to access the computer algebra system Singular. This was the starting point of the packages HomalgToCAS and IO_ForHomalg, which were further abstracted by Simon and myself enabling homalg to communicate with virtually any external (computer algebra) system.

Thomas Bächler wrote the package MapleForHomalg to directly access Maple via its C-interface. It offers an alternative to the package IO_ForHomalg, which requires Maple's terminal interface `cmaple`.

The basic support for Sage was added by Simon, and the support for Singular was initiated by Markus Lange-Hegermann and continued by him and Simon, while Markus Kirschmer contributed the complete support for MAGMA. This formed the beginning of the RingsForHomalg package. Recently, Daniel added the support for Macaulay2.

My concerns about how to handle the garbage collection in the external computer algebra systems were evaporated with the idea of Thomas Breuer using the so-called weak pointers in GAP4 to keep track of all the external objects that became obsolete for homalg. This idea took shape in a discussion with him and Frank Lübeck and finally found its way into the package HomalgToCAS.

My gratitude to all with whom I worked together to develop extension packages and those who developed their own packages within the homalg project (→ Appendix E). Without their contributions the package homalg would have remained a core without a body:

- Thomas Bächler
- Barbara Bremer
- Thomas Breuer
- Anna Fabiańska
- Simon Görtzen
- Markus Kirschmer
- Markus Lange-Hegermann
- Frank Lübeck
- Max Neunhöffer

- Daniel Robertz

I would also like to thank Alban Quadrat for supporting the homalg project and for all the wonderful discussions we had. At several places in the code I was happy to add the comment: “I learned this from Alban”.

My teacher Wilhelm Plesken remains an inexhaustible source of extremely broad and deep knowledge. Thank you for being such a magnificent person.

This manual was created using the GAPDoc package of Max Neunhöffer and Frank Lübeck.

Last but not least, thanks to *Miriam*, *Josef*, *Jonas*, and *Irene* for the endless love and support.

Mohamed Barakat

Contents

Chapter 1

Introduction

1.1 What is the role of the homalg package in the homalg project?

1.1.1 Philosophy

The package homalg is meant to be the first part of a continuously growing open source multi volume book about homological and homotopical algebra. homalg is an attempt to translate as much as possible of homological algebra, as can be found in books like [CE99], [ML63], [HS97], [Rot79], [Wei94], and [GM03], into a language that a computer can directly understand. But just like the aforementioned books, homalg should, to a great extent, be readable by a mathematician, even without deep programming knowledge. For the reasons mentioned in (→ Appendix B.2) GAP4 was chosen as the language of homalg.

1.1.2 homalg provides ...

The package homalg is the foundational part of the project. It provides procedures to construct basic objects in homological algebra:

- filtrations of objects
- complexes (of objects and of complexes)
- chain morphisms
- bicomplexes
- bigraded (differential) objects
- spectral sequences
- functors

Beside these so-called constructors homalg provides *operations* to perform computations with these objects. The list of operations includes:

- computation of subfactor objects
- applying functors (like `Ext`, `Tor`, ...) to objects, morphisms, complexes and chain morphisms

- derivation and composition of functors
- horse shoe resolution of short exact sequences of objects
- connecting homomorphisms and long exact sequences
- Cartan-Eilenberg resolution of complexes
- hyper (co)homology
- spectral sequences of bicomplexes
- the Grothendieck spectral sequences associated to two composable functors
- test if an object is torsion-free, reflexive, projective, stably free, pure
- determine the rank, grade, projective dimension, degree of torsion-freeness, and codegree of purity of an object

Using the philosophy of GAP4, one or more methods are *installed* for each operation, depending on *properties* and *attributes* of these objects. These properties and attributes can themselves be computed by methods installed for this purpose.

1.1.3 Building upon the homalg package

As mentioned above, the package homalg should only be the first and foundational part of the homalg project. On the one hand it is designed independently of the details of the different matrix operations, which other packages are meant to provide. Typically, these packages (like RingsForHomalg) heavily rely on existing, well tested, and optimized systems like Singular, Macaulay2, or MAGMA. On the other hand other packages can be built upon or extend the homalg package in different ways:

- add constructors (sheaves, schemes, simplicial sets, ...)
- add methods for basic operation (Yoneda products, Massey products, Steenrod operations, ...)
- add methods to compute sheaf cohomology, local cohomology, Hochschild (co)homology, cyclic (co)homology...
- provide algorithms for holonomic D-modules based on the restriction algorithm: localization, computing tensor products, `Hom`, `Ext`, de Rham cohomology, ...
- support change of rings, Lyndon/Hochschild-Serre spectral sequence, base change spectral sequences, ...
- support perturbation techniques, Serre and Eilenberg-Moore spectral sequence of simplicial spaces of infinite type, ...
- ...

The project will remain open and contributions are highly welcome. The different packages will be attributed to their respective authors. The whole project will be attributed to the "homalg team", i.e. the authors and contributers of all packages in the project.

1.2 This manual

Chapter 2 describes the installation of this package. The remaining chapters are each devoted to one of the homalg objects (→ 1.1.2) with its constructors, properties, attributes, and operations.

Chapter 2

Installation of the homalg Package

To install this package just extract the package's archive file to the GAP `pkg` directory.

By default the homalg package is not automatically loaded by GAP when it is installed. You must load the package with

```
LoadPackage( "homalg" );
```

before its functions become available.

Please, send me an e-mail if you have any questions, remarks, suggestions, etc. concerning this package. Also, I would be pleased to hear about applications of this package.

Mohamed Barakat

Chapter 3

Objects

3.1 Objects: Category and Representations

3.1.1 IsHomalgObject

▷ `IsHomalgObject(`*F*`)` (Category)

Returns: `true` or `false`

This is the super GAP-category which will include the GAP-categories `IsHomalgStaticObject` (3.1.2), `IsHomalgComplex` (6.1.1), `IsHomalgBicomplex` (8.1.1), `IsHomalgBigradedObject` (9.1.1), and `IsHomalgSpectralSequence` (10.1.1). We need this GAP-category to be able to build complexes with *objects* being objects of homalg categories or again complexes.

Code

```
DeclareCategory( "IsHomalgObject",
        IsHomalgObjectOrMorphism and
        IsStructureObjectOrObject and
        IsAdditiveElementWithZero );
```

3.1.2 IsHomalgStaticObject

▷ `IsHomalgStaticObject(`*F*`)` (Category)

Returns: `true` or `false`

This is the super GAP-category which will include the GAP-categories `IsHomalgModule`, etc.

Code

```
DeclareCategory( "IsHomalgStaticObject",
        IsHomalgStaticObjectOrMorphism and
        IsHomalgObject );
```

3.1.3 IsFinitelyPresentedObjectRep

▷ `IsFinitelyPresentedObjectRep(`*M*`)` (Representation)

Returns: `true` or `false`

The GAP representation of finitley presented homalg objects.

(It is a representation of the GAP category `IsHomalgObject` (3.1.1), which is a subrepresentation of the GAP representations `IsStructureObjectOrFinitelyPresentedObjectRep`.)

Code

```
DeclareRepresentation( "IsFinitelyPresentedObjectRep",
        IsHomalgObject and
        IsStructureObjectOrFinitelyPresentedObjectRep,
        [ ] );
```

3.1.4 IsStaticFinitelyPresentedObjectOrSubobjectRep

▷ IsStaticFinitelyPresentedObjectOrSubobjectRep(*M*) (Representation)

Returns: true or false

The GAP representation of finitley presented homalg static objects.

(It is a representation of the GAP category IsHomalgStaticObject (3.1.2).)

Code

```
DeclareRepresentation( "IsStaticFinitelyPresentedObjectOrSubobjectRep",
        IsHomalgStaticObject,
        [ ] );
```

3.1.5 IsStaticFinitelyPresentedObjectRep

▷ IsStaticFinitelyPresentedObjectRep(*M*) (Representation)

Returns: true or false

The GAP representation of finitley presented homalg static objects.

(It is a representation of the GAP category IsHomalgStaticObject (3.1.2), which is a subrepresentation of the GAP representations IsStaticFinitelyPresentedObjectOrSubobjectRep and IsFinitelyPresentedObjectRep.)

Code

```
DeclareRepresentation( "IsStaticFinitelyPresentedObjectRep",
        IsStaticFinitelyPresentedObjectOrSubobjectRep and
        IsFinitelyPresentedObjectRep,
        [ ] );
```

3.1.6 IsStaticFinitelyPresentedSubobjectRep

▷ IsStaticFinitelyPresentedSubobjectRep(*M*) (Representation)

Returns: true or false

The GAP representation of finitley presented homalg subobjects of static objects.

(It is a representation of the GAP category IsHomalgStaticObject (3.1.2), which is a subrepresentation of the GAP representations IsStaticFinitelyPresentedObjectOrSubobjectRep and IsFinitelyPresentedObjectRep.)

Code

```
DeclareRepresentation( "IsStaticFinitelyPresentedSubobjectRep",
        IsStaticFinitelyPresentedObjectOrSubobjectRep and
        IsFinitelyPresentedObjectRep,
        [ ] );
```

3.2 Objects: Constructors

3.2.1 Subobject (constructor for subobjects using morphisms)

▷ `Subobject(phi)` (operation)
Returns: a homalg subobject
A synonym of `ImageSubobject` (4.4.7).

3.3 Objects: Properties

3.3.1 IsFree

▷ `IsFree(M)` (property)
Returns: `true` or `false`
Check if the homalg object `M` is free.

3.3.2 IsStablyFree

▷ `IsStablyFree(M)` (property)
Returns: `true` or `false`
Check if the homalg object `M` is stably free.

3.3.3 IsProjective

▷ `IsProjective(M)` (property)
Returns: `true` or `false`
Check if the homalg object `M` is projective.

3.3.4 IsProjectiveOfConstantRank

▷ `IsProjectiveOfConstantRank(M)` (property)
Returns: `true` or `false`
Check if the homalg object `M` is projective of constant rank.

3.3.5 IsInjective

▷ `IsInjective(M)` (property)
Returns: `true` or `false`
Check if the homalg object `M` is (marked) injective.

3.3.6 IsInjectiveCogenerator

▷ `IsInjectiveCogenerator(M)` (property)
Returns: `true` or `false`
Check if the homalg object `M` is (marked) an injective cogenerator.

3.3.7 FiniteFreeResolutionExists

▷ FiniteFreeResolutionExists(*M*) (property)
Returns: true or false
Check if the homalg object *M* allows a finite free resolution.
(no method installed)

3.3.8 IsReflexive

▷ IsReflexive(*M*) (property)
Returns: true or false
Check if the homalg object *M* is reflexive.

3.3.9 IsTorsionFree

▷ IsTorsionFree(*M*) (property)
Returns: true or false
Check if the homalg object *M* is torsion-free.

3.3.10 IsArtinian

▷ IsArtinian(*M*) (property)
Returns: true or false
Check if the homalg object *M* is artinian.

3.3.11 IsTorsion

▷ IsTorsion(*M*) (property)
Returns: true or false
Check if the homalg object *M* is torsion.

3.3.12 IsPure

▷ IsPure(*M*) (property)
Returns: true or false
Check if the homalg object *M* is pure.

3.3.13 IsCohenMacaulay

▷ IsCohenMacaulay(*M*) (property)
Returns: true or false
Check if the homalg object *M* is Cohen-Macaulay (depends on the specific Abelian category).

3.3.14 IsGorenstein

▷ IsGorenstein(*M*) (property)
Returns: true or false
Check if the homalg object *M* is Gorenstein (depends on the specific Abelian category).

3.3.15 IsKoszul

▷ `IsKoszul(M)` (property)
Returns: `true` or `false`
Check if the homalg object `M` is Koszul (depends on the specific Abelian category).

3.3.16 HasConstantRank

▷ `HasConstantRank(M)` (property)
Returns: `true` or `false`
Check if the homalg object `M` has constant rank.
(no method installed)

3.3.17 ConstructedAsAnIdeal

▷ `ConstructedAsAnIdeal(J)` (property)
Returns: `true` or `false`
Check if the homalg subobject `J` was constructed as an ideal.
(no method installed)

3.4 Objects: Attributes

3.4.1 TorsionSubobject

▷ `TorsionSubobject(M)` (attribute)
Returns: a homalg subobject
This constructor returns the finitely generated torsion subobject of the homalg object `M`.

3.4.2 TheMorphismToZero

▷ `TheMorphismToZero(M)` (attribute)
Returns: a homalg map
The zero morphism from the homalg object `M` to zero.

3.4.3 TheIdentityMorphism

▷ `TheIdentityMorphism(M)` (attribute)
Returns: a homalg map
The identity automorphism of the homalg object `M`.

3.4.4 FullSubobject

▷ `FullSubobject(M)` (attribute)
Returns: a homalg subobject
The homalg object `M` as a subobject of itself.

3.4.5 ZeroSubobject

▷ `ZeroSubobject(M)` (attribute)

Returns: a homalg subobject

The zero subobject of the homalg object `M`.

3.4.6 EmbeddingInSuperObject

▷ `EmbeddingInSuperObject(N)` (attribute)

Returns: a homalg map

In case `N` was defined as a subobject of some object L the embedding of `N` in L is returned.

3.4.7 SuperObject (for subobjects)

▷ `SuperObject(M)` (attribute)

Returns: a homalg object

In case `M` was defined as a subobject of some object L the super object L is returned.

3.4.8 FactorObject

▷ `FactorObject(N)` (attribute)

Returns: a homalg object

In case `N` was defined as a subobject of some object L the factor object L/N is returned.

3.4.9 UnderlyingSubobject

▷ `UnderlyingSubobject(M)` (attribute)

Returns: a homalg subobject

In case `M` was defined as the object underlying a subobject L then L is returned.

(no method installed)

3.4.10 NatTrIdToHomHom_R (for morphisms)

▷ `NatTrIdToHomHom_R(M)` (attribute)

Returns: a homalg morphism

The natural evaluation morphism from the homalg object `M` to its double dual `HomHom(M)`.

3.4.11 Annihilator (for static objects)

▷ `Annihilator(M)` (attribute)

Returns: a homalg subobject

The annihilator of the object `M` as a subobject of the structure object.

3.4.12 EndomorphismRing (for static objects)

▷ `EndomorphismRing(M)` (attribute)

Returns: a homalg object

The endomorphism ring of the object `M`.

3.4.13 UnitObject

▷ `UnitObject(M)` (property)
Returns: a Chern character
M is a homalg object.

3.4.14 RankOfObject

▷ `RankOfObject(M)` (attribute)
Returns: a nonnegative integer
The projective rank of the homalg object M.

3.4.15 ProjectiveDimension

▷ `ProjectiveDimension(M)` (attribute)
Returns: a nonnegative integer
The projective dimension of the homalg object M.

3.4.16 DegreeOfTorsionFreeness

▷ `DegreeOfTorsionFreeness(M)` (attribute)
Returns: a nonnegative integer of infinity
Auslander's degree of torsion-freeness of the homalg object M. It is set to infinity only for $M = 0$.

3.4.17 Grade

▷ `Grade(M)` (attribute)
Returns: a nonnegative integer of infinity
The grade of the homalg object M. It is set to infinity if $M = 0$. Another name for this operation is `Depth`.

3.4.18 PurityFiltration

▷ `PurityFiltration(M)` (attribute)
Returns: a homalg filtration
The purity filtration of the homalg object M.

3.4.19 CodegreeOfPurity

▷ `CodegreeOfPurity(M)` (attribute)
Returns: a list of nonnegative integers
The codegree of purity of the homalg object M.

3.4.20 HilbertPolynomial

▷ `HilbertPolynomial(M)` (attribute)
Returns: a univariate polynomial with rational coefficients
M is a homalg object.

3.4.21 AffineDimension

▷ `AffineDimension(M)` (attribute)
Returns: a nonnegative integer
`M` is a homalg object.

3.4.22 ProjectiveDegree

▷ `ProjectiveDegree(M)` (attribute)
Returns: a nonnegative integer
`M` is a homalg object.

3.4.23 ConstantTermOfHilbertPolynomialn

▷ `ConstantTermOfHilbertPolynomialn(M)` (attribute)
Returns: an integer
`M` is a homalg object.

3.4.24 ElementOfGrothendieckGroup

▷ `ElementOfGrothendieckGroup(M)` (property)
Returns: an element of the Grothendieck group of a projective space
`M` is a homalg object.

3.4.25 ChernPolynomial

▷ `ChernPolynomial(M)` (property)
Returns: a Chern polynomial with rank
`M` is a homalg object.

3.4.26 ChernCharacter

▷ `ChernCharacter(M)` (property)
Returns: a Chern character
`M` is a homalg object.

3.5 Objects: Operations and Functions

3.5.1 CurrentResolution

▷ `CurrentResolution(M)` (attribute)
Returns: a homalg complex
The computed (part of a) resolution of the static object `M`.

3.5.2 UnderlyingObject (for subobjects)

▷ `UnderlyingObject(M)` (operation)

Returns: a homalg object

In case M was defined as a subobject of some object L the object underlying the subobject M is returned.

3.5.3 Saturate (for ideals)

▷ `Saturate(K, J)` (operation)

Returns: a homalg ideal

Compute the saturation ideal $K : J^{\infty}$ of the ideals K and J.

Example

```
gap> ZZ := HomalgRingOfIntegers( );
Z
gap> Display( ZZ );
<An internal ring>
gap> m := LeftSubmodule( "2", ZZ );
<A principal (left) ideal given by a cyclic generator>
gap> Display( m );
[ [  2 ] ]

A (left) ideal generated by the entry of the above matrix
gap> J := LeftSubmodule( "3", ZZ );
<A principal (left) ideal given by a cyclic generator>
gap> Display( J );
[ [  3 ] ]

A (left) ideal generated by the entry of the above matrix
gap> I := Intersect( J, m^3 );
<A principal (left) ideal given by a cyclic generator>
gap> Display( I );
[ [  -24 ] ]

A (left) ideal generated by the entry of the above matrix
gap> Im := SubobjectQuotient( I, m );
<A principal (left) ideal of rank 1 on a free generator>
gap> Display( Im );
[ [  -12 ] ]

A (left) ideal generated by the entry of the above matrix
gap> I_m := Saturate( I, m );
<A principal (left) ideal of rank 1 on a free generator>
gap> Display( I_m );
[ [  -3 ] ]

A (left) ideal generated by the entry of the above matrix
gap> I_m = J;
true
```

Code

```
InstallMethod( Saturate,
        "for homalg subobjects of static objects",
```

```
        [ IsStaticFinitelyPresentedSubobjectRep, IsStaticFinitelyPresentedSubobjectRep ],

  function( K, J )
    local quotient_last, quotient;

    quotient_last := SubobjectQuotient( K, J );

    quotient := SubobjectQuotient( quotient_last, J );

    while not IsSubset( quotient_last, quotient ) do
        quotient_last := quotient;
        quotient := SubobjectQuotient( quotient_last, J );
    od;

    return quotient_last;

end );


InstallMethod( \-,        ## a geometrically motivated definition
        "for homalg subobjects of static objects",
        [ IsStaticFinitelyPresentedSubobjectRep, IsStaticFinitelyPresentedSubobjectRep ],

  function( K, J )

    return Saturate( K, J );

end );
```

Chapter 4

Morphisms

4.1 Morphisms: Categories and Representations

4.1.1 IsHomalgMorphism

▷ IsHomalgMorphism(*phi*) (Category)

Returns: true or false

This is the super GAP-category which will include the GAP-categories IsHomalgStaticMorphism (4.1.2) and IsHomalgChainMorphism (7.1.1). We need this GAP-category to be able to build complexes with *objects* being objects of homalg categories or again complexes. We need this GAP-category to be able to build chain morphisms with *morphisms* being morphisms of homalg categories or again chain morphisms.
CAUTION: Never let homalg morphisms (which are not endomorphisms) be multiplicative elements!!

Code

```
DeclareCategory( "IsHomalgMorphism",
        IsHomalgStaticObjectOrMorphism and
        IsAdditiveElementWithInverse );
```

4.1.2 IsHomalgStaticMorphism

▷ IsHomalgStaticMorphism(*phi*) (Category)

Returns: true or false

This is the super GAP-category which will include the GAP-categories IsHomalgMap, etc.
CAUTION: Never let homalg morphisms (which are not endomorphisms) be multiplicative elements!!

Code

```
DeclareCategory( "IsHomalgStaticMorphism",
        IsHomalgMorphism );
```

4.1.3 IsHomalgEndomorphism

▷ IsHomalgEndomorphism(*phi*) (Category)

Returns: true or false

This is the super GAP-category which will include the GAP-categories IsHomalgSelfMap, IsHomalgChainEndomorphism (7.1.2), etc. be multiplicative elements!!

Code
```
DeclareCategory( "IsHomalgEndomorphism",
        IsHomalgMorphism and
        IsMultiplicativeElementWithInverse );
```

4.1.4 IsMorphismOfFinitelyGeneratedObjectsRep

▷ IsMorphismOfFinitelyGeneratedObjectsRep(*phi*) (Representation)

Returns: true or false

The GAP representation of morphisms of finitley generated homalg objects.

(It is a representation of the GAP category IsHomalgMorphism (4.1.1).)

Code
```
DeclareRepresentation( "IsMorphismOfFinitelyGeneratedObjectsRep",
        IsHomalgMorphism,
        [ ] );
```

4.1.5 IsStaticMorphismOfFinitelyGeneratedObjectsRep

▷ IsStaticMorphismOfFinitelyGeneratedObjectsRep(*phi*) (Representation)

Returns: true or false

The GAP representation of static morphisms of finitley generated homalg static objects.

(It is a representation of the GAP category IsHomalgStaticMorphism (4.1.2), which is a subrepresentation of the GAP representation IsMorphismOfFinitelyGeneratedObjectsRep (4.1.4).)

Code
```
DeclareRepresentation( "IsStaticMorphismOfFinitelyGeneratedObjectsRep",
        IsHomalgStaticMorphism and
        IsMorphismOfFinitelyGeneratedObjectsRep,
        [ ] );
```

4.2 Morphisms: Constructors

4.3 Morphisms: Properties

4.3.1 IsMorphism

▷ IsMorphism(*phi*) (property)

Returns: true or false

Check if *phi* is a well-defined map, i.e. independent of all involved presentations.

4.3.2 IsGeneralizedMorphism

▷ IsGeneralizedMorphism(*phi*) (property)

Returns: true or false

Check if *phi* is a generalized morphism.

4.3.3 IsGeneralizedEpimorphism

▷ `IsGeneralizedEpimorphism(`*phi*`)` (property)
Returns: `true` or `false`
Check if *phi* is a generalized epimorphism.

4.3.4 IsGeneralizedMonomorphism

▷ `IsGeneralizedMonomorphism(`*phi*`)` (property)
Returns: `true` or `false`
Check if *phi* is a generalized monomorphism.

4.3.5 IsGeneralizedIsomorphism

▷ `IsGeneralizedIsomorphism(`*phi*`)` (property)
Returns: `true` or `false`
Check if *phi* is a generalized isomorphism.

4.3.6 IsOne

▷ `IsOne(`*phi*`)` (property)
Returns: `true` or `false`
Check if the homalg morphism *phi* is the identity morphism.

4.3.7 IsIdempotent

▷ `IsIdempotent(`*phi*`)` (property)
Returns: `true` or `false`
Check if the homalg morphism *phi* is an automorphism.

4.3.8 IsMonomorphism

▷ `IsMonomorphism(`*phi*`)` (property)
Returns: `true` or `false`
Check if the homalg morphism *phi* is a monomorphism.

4.3.9 IsEpimorphism

▷ `IsEpimorphism(`*phi*`)` (property)
Returns: `true` or `false`
Check if the homalg morphism *phi* is an epimorphism.

4.3.10 IsSplitMonomorphism

▷ `IsSplitMonomorphism(`*phi*`)` (property)
Returns: `true` or `false`
Check if the homalg morphism *phi* is a split monomorphism.

4.3.11 IsSplitEpimorphism

▷ IsSplitEpimorphism(*phi*) (property)
Returns: true or false
Check if the homalg morphism *phi* is a split epimorphism.

4.3.12 IsIsomorphism

▷ IsIsomorphism(*phi*) (property)
Returns: true or false
Check if the homalg morphism *phi* is an isomorphism.

4.3.13 IsAutomorphism

▷ IsAutomorphism(*phi*) (property)
Returns: true or false
Check if the homalg morphism *phi* is an automorphism.

4.4 Morphisms: Attributes

4.4.1 Source

▷ Source(*phi*) (attribute)
Returns: a homalg object
The source of the homalg morphism *phi*.

4.4.2 Range

▷ Range(*phi*) (attribute)
Returns: a homalg object
The target (range) of the homalg morphism *phi*.

4.4.3 CokernelEpi (for morphisms)

▷ CokernelEpi(*phi*) (attribute)
Returns: a homalg morphism
The natural epimorphism from the Range(*phi*) onto the Cokernel(*phi*).

4.4.4 CokernelNaturalGeneralizedIsomorphism (for morphisms)

▷ CokernelNaturalGeneralizedIsomorphism(*phi*) (attribute)
Returns: a homalg morphism
The natural generalized isomorphism from the Cokernel(*phi*) onto the Range(*phi*).

4.4.5 KernelSubobject

▷ `KernelSubobject(`*phi*`)` (attribute)

Returns: a homalg subobject

This constructor returns the finitely generated kernel of the homalg morphism *phi* as a subobject of the homalg object `Source(`*phi*`)` with generators given by the syzygies of *phi*.

4.4.6 KernelEmb (for morphisms)

▷ `KernelEmb(`*phi*`)` (attribute)

Returns: a homalg morphism

The natural embedding of the `Kernel(`*phi*`)` into the `Source(`*phi*`)`.

4.4.7 ImageSubobject

▷ `ImageSubobject(`*phi*`)` (attribute)

Returns: a homalg subobject

This constructor returns the finitely generated image of the homalg morphism *phi* as a subobject of the homalg object `Range(`*phi*`)` with generators given by *phi* applied to the generators of its source object.

4.4.8 ImageObjectEmb (for morphisms)

▷ `ImageObjectEmb(`*phi*`)` (attribute)

Returns: a homalg morphism

The natural embedding of the `ImageObject(`*phi*`)` into the `Range(`*phi*`)`.

4.4.9 ImageObjectEpi (for morphisms)

▷ `ImageObjectEpi(`*phi*`)` (attribute)

Returns: a homalg morphism

The natural epimorphism from the `Source(`*phi*`)` onto the `ImageObject(`*phi*`)`.

4.4.10 MorphismAid

▷ `MorphismAid(`*phi*`)` (attribute)

Returns: a homalg morphism

The morphism aid map of a true generalized map.

(no method installed)

4.4.11 GeneralizedInverse

▷ `GeneralizedInverse(`*phi*`)` (attribute)

Returns: a homalg morphism

The generalized inverse of the epimorphism *phi* (cf. [Bar, Cor. 4.8])).

4.4.12 DegreeOfMorphism

▷ DegreeOfMorphism(*phi*) (attribute)

Returns: an integer

The degree of the morphism *phi* between graded objects.

(no method installed)

4.5 Morphisms: Operations and Functions

4.5.1 ByASmallerPresentation (for morphisms)

▷ ByASmallerPresentation(*phi*) (method)

Returns: a homalg map

It invokes ByASmallerPresentation for homalg (static) objects.

Code

```
InstallMethod( ByASmallerPresentation,
        "for homalg morphisms",
        [ IsStaticMorphismOfFinitelyGeneratedObjectsRep ],

  function( phi )

    ByASmallerPresentation( Source( phi ) );
    ByASmallerPresentation( Range( phi ) );

    return DecideZero( phi );

end );
```

This method performs side effects on its argument *phi* and returns it.

Example

```
gap> ZZ := HomalgRingOfIntegers( );
Z
gap> M := HomalgMatrix( "[ 2, 3, 4,   5, 6, 7 ]", 2, 3, ZZ );
<A 2 x 3 matrix over an internal ring>
gap> M := LeftPresentation( M );
<A non-torsion left module presented by 2 relations for 3 generators>
gap> N := HomalgMatrix( "[ 2, 3, 4, 5,   6, 7, 8, 9 ]", 2, 4, ZZ );
<A 2 x 4 matrix over an internal ring>
gap> N := LeftPresentation( N );
<A non-torsion left module presented by 2 relations for 4 generators>
gap> mat := HomalgMatrix( "[ \
> 1, 0, -2, -4, \
> 0, 1,  4,  7, \
> 1, 0, -2, -4  \
> ]", 3, 4, ZZ );
<A 3 x 4 matrix over an internal ring>
gap> phi := HomalgMap( mat, M, N );
<A "homomorphism" of left modules>
gap> IsMorphism( phi );
true
gap> phi;
```

```
<A homomorphism of left modules>
gap> Display( phi );
[ [   1,   0,  -2,  -4 ],
  [   0,   1,   4,   7 ],
  [   1,   0,  -2,  -4 ] ]

the map is currently represented by the above 3 x 4 matrix
gap> ByASmallerPresentation( phi );
<A non-zero homomorphism of left modules>
gap> Display( phi );
[ [   0,   0,   0 ],
  [   1,  -1,  -2 ] ]

the map is currently represented by the above 2 x 3 matrix
gap> M;
<A rank 1 left module presented by 1 relation for 2 generators>
gap> Display( M );
Z/< 3 > + Z^(1 x 1)
gap> N;
<A rank 2 left module presented by 1 relation for 3 generators>
gap> Display( N );
Z/< 4 > + Z^(1 x 2)
```

Chapter 5

Elements

An element of an object M is internally represented by a morphism from the "structure object" to the object M. In particular, the data structure for object elements automatically profits from the intrinsic realization of morphisms in the homalg project.

5.1 Elements: Category and Representations

5.1.1 IsHomalgElement

▷ `IsHomalgElement(M)` (Category)

Returns: `true` or `false`

The GAP category of object elements.

5.1.2 IsElementOfAnObjectGivenByAMorphismRep

▷ `IsElementOfAnObjectGivenByAMorphismRep(M)` (Representation)

Returns: `true` or `false`

The GAP representation of elements of finitley presented objects.
(It is a representation of the GAP category `IsHomalgElement` (5.1.1).)

5.2 Elements: Constructors

5.3 Elements: Properties

5.3.1 IsZero (for elements)

▷ `IsZero(m)` (property)

Returns: `true` or `false`

Check if the object element `m` is zero.

5.3.2 IsCyclicGenerator

▷ `IsCyclicGenerator(m)` (property)

Returns: `true` or `false`

Check if the object element `m` is a cyclic generator.

5.3.3 IsTorsion

▷ `IsTorsion(m)` (property)

Returns: `true` or `false`

Check if the object element m is a torsion element.

5.4 Elements: Attributes

5.4.1 Annihilator (for elements)

▷ `Annihilator(e)` (attribute)

Returns: a homalg subobject

The annihilator of the object element `e` as a subobject of the structure object.

5.5 Elements: Operations and Functions

5.5.1 in (for elements)

▷ `in(m, N)` (attribute)

Returns: `true` or `false`

Is the element m of the object M included in the subobject $N \leq M$, i.e., does the morphism (with the unit object as source and M as target) underling the element m of M factor over the subobject morphism $N \to M$?

Example

```
gap> ZZ := HomalgRingOfIntegers( );
Z
gap> M := 2 * ZZ;
<A free left module of rank 2 on free generators>
gap> a := HomalgModuleElement( "[ 6, 0 ]", M );
( 6, 0 )
gap> N := Subobject( HomalgMap( "[ 2, 0 ]", 1 * ZZ, M ) );
<A free left submodule given by a cyclic generator>
gap> K := Subobject( HomalgMap( "[ 4, 0 ]", 1 * ZZ, M ) );
<A free left submodule given by a cyclic generator>
gap> a in M;
true
gap> a in N;
true
gap> a in UnderlyingObject( N );
true
gap> a in K;
false
gap> a in UnderlyingObject( K );
false
gap> a in 3 * ZZ;
false
```

Code

```
InstallMethod( \in,
        "for homalg elements",
        [ IsHomalgElement, IsStaticFinitelyPresentedSubobjectRep ],
```

```
  function( m, N )
    local phi, psi;

    phi := UnderlyingMorphism( m );

    psi := MorphismHavingSubobjectAsItsImage( N );

    if not IsIdenticalObj( Range( phi ), Range( psi ) ) then
        Error( "the super object of the subobject and the range ",
               "of the morphism underlying the element do not coincide\n" );
    fi;

    return IsZero( PreCompose( phi, CokernelEpi( psi ) ) );

end );
```

Chapter 6

Complexes

6.1 Complexes: Category and Representations

6.1.1 IsHomalgComplex

▷ `IsHomalgComplex(C)` (Category)

Returns: `true` or `false`

The GAP category of homalg (co)complexes.

(It is a subcategory of the GAP category `IsHomalgObject`.)

6.1.2 IsComplexOfFinitelyPresentedObjectsRep

▷ `IsComplexOfFinitelyPresentedObjectsRep(C)` (Representation)

Returns: `true` or `false`

The GAP representation of complexes of finitley presented homalg objects.

(It is a representation of the GAP category `IsHomalgComplex` (6.1.1), which is a subrepresentation of the GAP representation `IsFinitelyPresentedObjectRep`.)

6.1.3 IsCocomplexOfFinitelyPresentedObjectsRep

▷ `IsCocomplexOfFinitelyPresentedObjectsRep(C)` (Representation)

Returns: `true` or `false`

The GAP representation of cocomplexes of finitley presented homalg objects.

(It is a representation of the GAP category `IsHomalgComplex` (6.1.1), which is a subrepresentation of the GAP representation `IsFinitelyPresentedObjectRep`.)

6.2 Complexes: Constructors

6.2.1 HomalgComplex (constructor for complexes given an object)

▷ `HomalgComplex(M[, d])` (function)

▷ `HomalgComplex(phi[, d])` (function)

▷ `HomalgComplex(C[, d])` (function)

▷ `HomalgComplex(cm[, d])` (function)

Returns: a homalg complex

The first syntax creates a complex (i.e. chain complex) with the single homalg object *M* at (homological) degree *d*.

The second syntax creates a complex with the single homalg morphism *phi*, its source placed at (homological) degree *d* (and its target at *d*−1).

The third syntax creates a complex (i.e. chain complex) with the single homalg (co)complex *C* at (homological) degree *d*.

The fourth syntax creates a complex with the single homalg (co)chain morphism *cm* (→ HomalgChainMorphism (7.2.1)), its source placed at (homological) degree *d* (and its target at *d*−1).

If *d* is not provided it defaults to zero in all cases.

To add a morphism (resp. (co)chain morphism) to a complex use Add (6.5.1).

Example

```
gap> ZZ := HomalgRingOfIntegers( );
Z
gap> M := HomalgMatrix( "[ 2, 3, 4,   5, 6, 7 ]", 2, 3, ZZ );
<A 2 x 3 matrix over an internal ring>
gap> M := LeftPresentation( M );
<A non-torsion left module presented by 2 relations for 3 generators>
gap> N := HomalgMatrix( "[ 2, 3, 4, 5,   6, 7, 8, 9 ]", 2, 4, ZZ );
<A 2 x 4 matrix over an internal ring>
gap> N := LeftPresentation( N );
<A non-torsion left module presented by 2 relations for 4 generators>
gap> mat := HomalgMatrix( "[ \
> 0, 3, 6, 9, \
> 0, 2, 4, 6, \
> 0, 3, 6, 9  \
> ]", 3, 4, ZZ );
<A 3 x 4 matrix over an internal ring>
gap> phi := HomalgMap( mat, M, N );
<A "homomorphism" of left modules>
gap> IsMorphism( phi );
true
gap> phi;
<A homomorphism of left modules>
```

The first possibility:

Example

```
<A homomorphism of left modules>
gap> C := HomalgComplex( N );
<A non-zero graded homology object consisting of a single left module at degre\
e 0>
gap> Add( C, phi );
gap> C;
<A complex containing a single morphism of left modules at degrees [ 0 .. 1 ]>
```

The second possibility:

Example

```
gap> C := HomalgComplex( phi );
<A non-zero acyclic complex containing a single morphism of left modules at de\
grees [ 0 .. 1 ]>
```

6.2.2 HomalgCocomplex (constructor for cocomplexes given a object)

▷ HomalgCocomplex(*M[, d]*) (function)
▷ HomalgCocomplex(*phi[, d]*) (function)
▷ HomalgCocomplex(*C[, d]*) (function)
▷ HomalgCocomplex(*cm[, d]*) (function)

Returns: a homalg complex

The first syntax creates a cocomplex (i.e. cochain complex) with the single homalg object *M* at (cohomological) degree *d*.

The second syntax creates a cocomplex with the single homalg morphism *phi*, its source placed at (cohomological) degree *d* (and its target at *d*+1).

The third syntax creates a cocomplex (i.e. cochain complex) with the single homalg cocomplex *C* at (cohomological) degree *d*.

The fourth syntax creates a cocomplex with the single homalg (co)chain morphism *cm* (→ HomalgChainMorphism (7.2.1)), its source placed at (cohomological) degree *d* (and its target at *d*+1).

If *d* is not provided it defaults to zero in all cases.

To add a morphism (resp. (co)chain morphism) to a cocomplex use Add (6.5.1).

Example

```
gap> ZZ := HomalgRingOfIntegers( );
Z
gap> M := HomalgMatrix( "[ 2, 3, 4,   5, 6, 7 ]", 2, 3, ZZ );
<A 2 x 3 matrix over an internal ring>
gap> M := RightPresentation( Involution( M ) );
<A non-torsion right module on 3 generators satisfying 2 relations>
gap> N := HomalgMatrix( "[ 2, 3, 4, 5,   6, 7, 8, 9 ]", 2, 4, ZZ );
<A 2 x 4 matrix over an internal ring>
gap> N := RightPresentation( Involution( N ) );
<A non-torsion right module on 4 generators satisfying 2 relations>
gap> mat := HomalgMatrix( "[ \
> 0, 3, 6, 9, \
> 0, 2, 4, 6, \
> 0, 3, 6, 9  \
> ]", 3, 4, ZZ );
<A 3 x 4 matrix over an internal ring>
gap> phi := HomalgMap( Involution( mat ), M, N );
<A "homomorphism" of right modules>
gap> IsMorphism( phi );
true
gap> phi;
<A homomorphism of right modules>
```

The first possibility:

Example

```
<A homomorphism of right modules>
gap> C := HomalgCocomplex( M );
<A non-zero graded cohomology object consisting of a single right module at de\
gree 0>
gap> Add( C, phi );
gap> C;
```

```
<A cocomplex containing a single morphism of right modules at degrees
[ 0 .. 1 ]>
```

The second possibility:

Example

```
gap> C := HomalgCocomplex( phi );
<A non-zero acyclic cocomplex containing a single morphism of right modules at\
 degrees [ 0 .. 1 ]>
```

6.3 Complexes: Properties

6.3.1 IsSequence

▷ IsSequence(*C*) (property)

Returns: true or false

Check if all maps in *C* are well-defined.

6.3.2 IsComplex

▷ IsComplex(*C*) (property)

Returns: true or false

Check if *C* is complex.

6.3.3 IsAcyclic

▷ IsAcyclic(*C*) (property)

Returns: true or false

Check if the homalg complex *C* is acyclic, i.e. exact except at its boundaries.

6.3.4 IsRightAcyclic

▷ IsRightAcyclic(*C*) (property)

Returns: true or false

Check if the homalg complex *C* is acyclic, i.e. exact except at its left boundary.

6.3.5 IsLeftAcyclic

▷ IsLeftAcyclic(*C*) (property)

Returns: true or false

Check if the homalg complex *C* is acyclic, i.e. exact except at its right boundary.

6.3.6 IsGradedObject

▷ IsGradedObject(*C*) (property)

Returns: true or false

Check if the homalg complex *C* is a graded object, i.e. if all maps between the objects in *C* vanish.

6.3.7 IsExactSequence

▷ `IsExactSequence(C)` (property)
Returns: `true` or `false`
Check if the homalg complex `C` is exact.

6.3.8 IsShortExactSequence

▷ `IsShortExactSequence(C)` (property)
Returns: `true` or `false`
Check if the homalg complex `C` is a short exact sequence.

6.3.9 IsSplitShortExactSequence

▷ `IsSplitShortExactSequence(C)` (property)
Returns: `true` or `false`
Check if the homalg complex `C` is a split short exact sequence.

6.3.10 IsTriangle

▷ `IsTriangle(C)` (property)
Returns: `true` or `false`
Set to true if the homalg complex `C` is a triangle.

6.3.11 IsExactTriangle

▷ `IsExactTriangle(C)` (property)
Returns: `true` or `false`
Check if the homalg complex `C` is an exact triangle.

6.4 Complexes: Attributes

6.4.1 BettiTable (for complexes)

▷ `BettiTable(C)` (attribute)
Returns: a homalg diagram
The Betti diagram of the homalg complex `C` of graded modules.

6.4.2 FiltrationByShortExactSequence (for complexes)

▷ `FiltrationByShortExactSequence(C)` (attribute)
Returns: a homalg diagram
The filtration induced by the short exact sequence `C` on its middle object.

6.5 Complexes: Operations and Functions

6.5.1 Add (to complexes given a morphism)

▷ Add(*C*, *phi*) (operation)
▷ Add(*C*, *mat*) (operation)

Returns: a homalg complex

In the first syntax the morphism *phi* is added to the (co)chain complex *C* (→ 6.2) as the new *highest* degree morphism and the altered argument *C* is returned. In case *C* is a chain complex, the highest degree object in *C* and the target of *phi* must be *identical*. In case *C* is a *co*chain complex, the highest degree object in *C* and the source of *phi* must be *identical*.

In the second syntax the matrix *mat* is interpreted as the matrix of the new *highest* degree morphism *psi*, created according to the following rules: In case *C* is a chain complex, the highest degree left (resp. right) object C_d in *C* is declared as the target of *psi*, while its source is taken to be a free left (resp. right) object of rank equal to NrRows(*mat*) (resp. NrColumns(*mat*)). For this NrColumns(*mat*) (resp. NrRows(*mat*)) must coincide with the NrGenerators(C_d). In case *C* is a *co*chain complex, the highest degree left (resp. right) object C^d in *C* is declared as the source of *psi*, while its target is taken to be a free left (resp. right) object of rank equal to NrColumns(*mat*) (resp. NrRows(*mat*)). For this NrRows(*mat*) (resp. Columns(*mat*)) must coincide with the NrGenerators(C^d).

Example

```
gap> ZZ := HomalgRingOfIntegers( );
Z
gap> mat := HomalgMatrix( "[ 0, 1,   0, 0 ]", 2, 2, ZZ );
<A 2 x 2 matrix over an internal ring>
gap> phi := HomalgMap( mat );
<A homomorphism of left modules>
gap> C := HomalgComplex( phi );
<A non-zero acyclic complex containing a single morphism of left modules at de\
grees [ 0 .. 1 ]>
gap> Add( C, mat );
gap> C;
<A sequence containing 2 morphisms of left modules at degrees [ 0 .. 2 ]>
gap> Display( C );
--------------------------
at homology degree: 2
Z^(1 x 2)
--------------------------
[ [  0,  1 ],
  [  0,  0 ] ]

the map is currently represented by the above 2 x 2 matrix
------------v------------
at homology degree: 1
Z^(1 x 2)
--------------------------
[ [  0,  1 ],
  [  0,  0 ] ]

the map is currently represented by the above 2 x 2 matrix
------------v------------
at homology degree: 0
```

```
Z^(1 x 2)
-------------------------
gap> IsComplex( C );
true
gap> IsAcyclic( C );
true
gap> IsExactSequence( C );
false
gap> C;
<A non-zero acyclic complex containing 2 morphisms of left modules at degrees
[ 0 .. 2 ]>
```

6.5.2 ByASmallerPresentation (for complexes)

▷ ByASmallerPresentation(*C*) (method)

Returns: a homalg complex

It invokes ByASmallerPresentation for homalg (static) objects.

Code

```
InstallMethod( ByASmallerPresentation,
        "for homalg complexes",
        [ IsHomalgComplex ],

  function( C )

    List( ObjectsOfComplex( C ), ByASmallerPresentation );

    if Length( ObjectDegreesOfComplex( C ) ) > 1 then
        List( MorphismsOfComplex( C ), DecideZero );
    fi;

    IsZero( C );

    return C;

end );
```

This method performs side effects on its argument *C* and returns it.

Example

```
gap> ZZ := HomalgRingOfIntegers( );
Z
gap> M := HomalgMatrix( "[ 2, 3, 4,   5, 6, 7 ]", 2, 3, ZZ );
<A 2 x 3 matrix over an internal ring>
gap> M := LeftPresentation( M );
<A non-torsion left module presented by 2 relations for 3 generators>
gap> N := HomalgMatrix( "[ 2, 3, 4, 5,   6, 7, 8, 9 ]", 2, 4, ZZ );
<A 2 x 4 matrix over an internal ring>
gap> N := LeftPresentation( N );
<A non-torsion left module presented by 2 relations for 4 generators>
gap> mat := HomalgMatrix( "[ \
> 0, 3, 6, 9, \
> 0, 2, 4, 6, \
> 0, 3, 6, 9  \
```

```
> ]", 3, 4, ZZ );
<A 3 x 4 matrix over an internal ring>
gap> phi := HomalgMap( mat, M, N );
<A "homomorphism" of left modules>
gap> IsMorphism( phi );
true
gap> phi;
<A homomorphism of left modules>
gap> C := HomalgComplex( phi );
<A non-zero acyclic complex containing a single morphism of left modules at de\
grees [ 0 .. 1 ]>
gap> Display( C );
-------------------------
at homology degree: 1
[ [  2,  3,  4 ],
  [  5,  6,  7 ] ]

Cokernel of the map

Z^(1x2) --> Z^(1x3),

currently represented by the above matrix
-------------------------
[ [  0,  3,  6,  9 ],
  [  0,  2,  4,  6 ],
  [  0,  3,  6,  9 ] ]

the map is currently represented by the above 3 x 4 matrix
------------v------------
at homology degree: 0
[ [  2,  3,  4,  5 ],
  [  6,  7,  8,  9 ] ]

Cokernel of the map

Z^(1x2) --> Z^(1x4),

currently represented by the above matrix
-------------------------
```

And now:

Example

```
gap> ByASmallerPresentation( C );
<A non-zero acyclic complex containing a single morphism of left modules at de\
grees [ 0 .. 1 ]>
gap> Display( C );
-------------------------
at homology degree: 1
Z/< 3 > + Z^(1 x 1)
-------------------------
[ [  0,  0,  0 ],
  [  2,  0,  0 ] ]
```

```
the map is currently represented by the above 2 x 3 matrix
------------v------------
at homology degree: 0
Z/< 4 > + Z^(1 x 2)
-------------------------
```

Chapter 7

Chain Morphisms

7.1 ChainMorphisms: Categories and Representations

7.1.1 IsHomalgChainMorphism

▷ `IsHomalgChainMorphism(`*cm*`)` (Category)

Returns: `true` or `false`

The GAP category of homalg (co)chain morphisms.

(It is a subcategory of the GAP category `IsHomalgMorphism`.)

7.1.2 IsHomalgChainEndomorphism

▷ `IsHomalgChainEndomorphism(`*cm*`)` (Category)

Returns: `true` or `false`

The GAP category of homalg (co)chain endomorphisms.

(It is a subcategory of the GAP categories `IsHomalgChainMorphism` and `IsHomalgEndomorphism`.)

7.1.3 IsChainMorphismOfFinitelyPresentedObjectsRep

▷ `IsChainMorphismOfFinitelyPresentedObjectsRep(`*c*`)` (Representation)

Returns: `true` or `false`

The GAP representation of chain morphisms of finitely presented homalg objects.

(It is a representation of the GAP category `IsHomalgChainMorphism` (7.1.1), which is a subrepresentation of the GAP representation `IsMorphismOfFinitelyGeneratedObjectsRep`.)

7.1.4 IsCochainMorphismOfFinitelyPresentedObjectsRep

▷ `IsCochainMorphismOfFinitelyPresentedObjectsRep(`*c*`)` (Representation)

Returns: `true` or `false`

The GAP representation of cochain morphisms of finitely presented homalg objects.

(It is a representation of the GAP category `IsHomalgChainMorphism` (7.1.1), which is a subrepresentation of the GAP representation `IsMorphismOfFinitelyGeneratedObjectsRep`.)

7.2 Chain Morphisms: Constructors

7.2.1 HomalgChainMorphism (constructor for chain morphisms given a morphism)

▷ HomalgChainMorphism(*phi[, C][, D][, d]*) (function)

Returns: a homalg chain morphism

The constructor creates a (co)chain morphism given a source homalg (co)chain complex *C*, a target homalg (co)chain complex *D*, and a homalg morphism *phi* at (co)homological degree *d*. The returned (co)chain morphism will cautiously be indicated using parenthesis: "chain morphism". To verify if the result is indeed a (co)chain morphism use IsMorphism (7.3.1). If source and target are identical objects, and only then, the (co)chain morphism is created as a (co)chain endomorphism.

The following examples shows a chain morphism that induces the zero morphism on homology, but is itself *not* zero in the derived category:

Example

```
gap> ZZ := HomalgRingOfIntegers( );
Z
gap> M := 1 * ZZ;
<The free left module of rank 1 on a free generator>
gap> Display( M );
Z^(1 x 1)
gap> N := HomalgMatrix( "[3]", 1, 1, ZZ );;
gap> N := LeftPresentation( N );
<A cyclic left module presented by 1 relation for a cyclic generator>
gap> Display( N );
Z/< 3 >
gap> a := HomalgMap( HomalgMatrix( "[2]", 1, 1, ZZ ), M, M );
<An endomorphism of a left module>
gap> c := HomalgMap( HomalgMatrix( "[2]", 1, 1, ZZ ), M, N );
<A homomorphism of left modules>
gap> b := HomalgMap( HomalgMatrix( "[1]", 1, 1, ZZ ), M, M );
<An endomorphism of a left module>
gap> d := HomalgMap( HomalgMatrix( "[1]", 1, 1, ZZ ), M, N );
<A homomorphism of left modules>
gap> C1 := HomalgComplex( a );
<A non-zero acyclic complex containing a single morphism of left modules at de\
grees [ 0 .. 1 ]>
gap> C2 := HomalgComplex( c );
<A non-zero acyclic complex containing a single morphism of left modules at de\
grees [ 0 .. 1 ]>
gap> cm := HomalgChainMorphism( d, C1, C2 );
<A "chain morphism" containing a single left morphism at degree 0>
gap> Add( cm, b );
gap> IsMorphism( cm );
true
gap> cm;
<A chain morphism containing 2 morphisms of left modules at degrees
[ 0 .. 1 ]>
gap> hcm := DefectOfExactness( cm );
<A chain morphism of graded objects containing
2 morphisms of left modules at degrees [ 0 .. 1 ]>
gap> IsZero( hcm );
true
```

```
gap> IsZero( Source( hcm ) );
false
gap> IsZero( Range( hcm ) );
false
```

7.3 Chain Morphisms: Properties

7.3.1 IsMorphism (for chain morphisms)

▷ IsMorphism(*cm*) (property)
Returns: true or false
Check if *cm* is a well-defined chain morphism, i.e. independent of all involved presentations.

7.3.2 IsGeneralizedMorphism (for chain morphisms)

▷ IsGeneralizedMorphism(*cm*) (property)
Returns: true or false
Check if *cm* is a generalized morphism.

7.3.3 IsGeneralizedEpimorphism (for chain morphisms)

▷ IsGeneralizedEpimorphism(*cm*) (property)
Returns: true or false
Check if *cm* is a generalized epimorphism.

7.3.4 IsGeneralizedMonomorphism (for chain morphisms)

▷ IsGeneralizedMonomorphism(*cm*) (property)
Returns: true or false
Check if *cm* is a generalized monomorphism.

7.3.5 IsGeneralizedIsomorphism (for chain morphisms)

▷ IsGeneralizedIsomorphism(*cm*) (property)
Returns: true or false
Check if *cm* is a generalized isomorphism.

7.3.6 IsOne (for chain morphisms)

▷ IsOne(*cm*) (property)
Returns: true or false
Check if the homalg chain morphism *cm* is the identity chain morphism.

7.3.7 IsMonomorphism (for chain morphisms)

▷ IsMonomorphism(*cm*) (property)
Returns: true or false
Check if the homalg chain morphism *cm* is a monomorphism.

7.3.8 IsEpimorphism (for chain morphisms)

▷ `IsEpimorphism(`*cm*`)` (property)
Returns: `true` or `false`
Check if the homalg chain morphism *cm* is an epimorphism.

7.3.9 IsSplitMonomorphism (for chain morphisms)

▷ `IsSplitMonomorphism(`*cm*`)` (property)
Returns: `true` or `false`
Check if the homalg chain morphism *cm* is a split monomorphism.

7.3.10 IsSplitEpimorphism (for chain morphisms)

▷ `IsSplitEpimorphism(`*cm*`)` (property)
Returns: `true` or `false`
Check if the homalg chain morphism *cm* is a split epimorphism.

7.3.11 IsIsomorphism (for chain morphisms)

▷ `IsIsomorphism(`*cm*`)` (property)
Returns: `true` or `false`
Check if the homalg chain morphism *cm* is an isomorphism.

7.3.12 IsAutomorphism (for chain morphisms)

▷ `IsAutomorphism(`*cm*`)` (property)
Returns: `true` or `false`
Check if the homalg chain morphism *cm* is an automorphism.

7.3.13 IsGradedMorphism (for chain morphisms)

▷ `IsGradedMorphism(`*cm*`)` (property)
Returns: `true` or `false`
Check if the source and target complex of the homalg chain morphism *cm* are graded objects, i.e. if all their morphisms vanish.

7.3.14 IsQuasiIsomorphism (for chain morphisms)

▷ `IsQuasiIsomorphism(`*cm*`)` (property)
Returns: `true` or `false`
Check if the homalg chain morphism *cm* is a quasi-isomorphism.

7.4 Chain Morphisms: Attributes

7.4.1 Source (for chain morphisms)

▷ `Source(cm)` (attribute)

Returns: a homalg complex

The source of the homalg chain morphism *cm*.

7.4.2 Range (for chain morphisms)

▷ `Range(cm)` (attribute)

Returns: a homalg complex

The target (range) of the homalg chain morphism *cm*.

7.5 Chain Morphisms: Operations and Functions

7.5.1 ByASmallerPresentation (for chain morphisms)

▷ `ByASmallerPresentation(cm)` (method)

Returns: a homalg complex

See `ByASmallerPresentation` (6.5.2) on complexes.

Code

```
InstallMethod( ByASmallerPresentation,
        "for homalg chain morphisms",
        [ IsHomalgChainMorphism ],

  function( cm )

    ByASmallerPresentation( Source( cm ) );
    ByASmallerPresentation( Range( cm ) );

    List( MorphismsOfChainMorphism( cm ), DecideZero );

    return cm;

end );
```

This method performs side effects on its argument *cm* and returns it.

Chapter 8

Bicomplexes

Each bicomplex in homalg has an underlying complex of complexes. The bicomplex structure is simply the addition of the known sign trick which induces the obvious equivalence between the category of bicomplexes and the category of complexes with complexes as objects and chain morphisms as morphisms. The majority of filtered complexes in algebra and geometry (unlike topology) arise as the total complex of a bicomplex. Hence, most spectral sequences in algebra are spectral sequences of bicomplexes. Indeed, bicomplexes in homalg are mainly used as an input for the spectral sequence machinery.

8.1 Bicomplexes: Category and Representations

8.1.1 IsHomalgBicomplex

▷ `IsHomalgBicomplex(`*BC*`)` (Category)

Returns: `true` or `false`

The GAP category of homalg bi(co)complexes.

(It is a subcategory of the GAP category `IsHomalgObject`.)

8.1.2 IsBicomplexOfFinitelyPresentedObjectsRep

▷ `IsBicomplexOfFinitelyPresentedObjectsRep(`*BC*`)` (Representation)

Returns: `true` or `false`

The GAP representation of bicomplexes (homological bicomplexes) of finitley generated homalg objects.

(It is a representation of the GAP category `IsHomalgBicomplex` (8.1.1), which is a subrepresentation of the GAP representation `IsFinitelyPresentedObjectRep`.)

8.1.3 IsBicocomplexOfFinitelyPresentedObjectsRep

▷ `IsBicocomplexOfFinitelyPresentedObjectsRep(`*BC*`)` (Representation)

Returns: `true` or `false`

The GAP representation of bicocomplexes (cohomological bicomplexes) of finitley generated homalg objects.

(It is a representation of the GAP category `IsHomalgBicomplex` (8.1.1), which is a subrepresentation of the GAP representation `IsFinitelyPresentedObjectRep`.)

8.2 Bicomplexes: Constructors

8.2.1 HomalgBicomplex (constructor for bicomplexes given a complex of complexes)

▷ HomalgBicomplex(*C*) (function)

Returns: a homalg bicomplex

This constructor creates a bicomplex (homological bicomplex) given a homalg complex of (co)complexes *C* (→ HomalgComplex (6.2.1)), resp. creates a bicocomplex (cohomological bicomplex) given a homalg cocomplex of (co)complexes *C* (→ HomalgCocomplex (6.2.2)). Using the usual sign-trick a complex of complexes gives rise to a bicomplex and vice versa.

Example

```
gap> ZZ := HomalgRingOfIntegers( );
Z
gap> M := HomalgMatrix( "[ 2, 3, 4,   5, 6, 7 ]", 2, 3, ZZ );
<A 2 x 3 matrix over an internal ring>
gap> M := LeftPresentation( M );
<A non-torsion left module presented by 2 relations for 3 generators>
gap> d := Resolution( M );
<A non-zero right acyclic complex containing a single morphism of left modules\
 at degrees [ 0 .. 1 ]>
gap> dd := Hom( d );
<A non-zero acyclic cocomplex containing a single morphism of right modules at\
 degrees [ 0 .. 1 ]>
gap> C := Resolution( dd );
<An acyclic cocomplex containing a single morphism of right complexes at degre\
es [ 0 .. 1 ]>
gap> CC := Hom( C );
<A non-zero acyclic complex containing a single morphism of left cocomplexes a\
t degrees [ 0 .. 1 ]>
gap> BC := HomalgBicomplex( CC );
<A non-zero bicomplex containing left modules at bidegrees [ 0 .. 1 ]x
[ -1 .. 0 ]>
gap> Display( BC );
 * *
 * *
gap> UU := UnderlyingComplex( BC );
<A non-zero acyclic complex containing a single morphism of left cocomplexes a\
t degrees [ 0 .. 1 ]>
gap> IsIdenticalObj( UU, CC );
true
gap> tBC := TransposedBicomplex( BC );
<A non-zero bicomplex containing left modules at bidegrees [ -1 .. 0 ]x
[ 0 .. 1 ]>
gap> Display( tBC );
 * *
 * *
```

8.3 Bicomplexes: Properties

8.3.1 IsBisequence

▷ `IsBisequence(`*BC*`)` (property)
Returns: `true` or `false`
Check if all maps in *BC* are well-defined.

8.3.2 IsBicomplex

▷ `IsBicomplex(`*BC*`)` (property)
Returns: `true` or `false`
Check if *BC* is bicomplex.

8.3.3 IsTransposedWRTTheAssociatedComplex

▷ `IsTransposedWRTTheAssociatedComplex(`*BC*`)` (property)
Returns: `true` or `false`
Check if *BC* is transposed with respect to the associated complex of complexes.
(no method installed).

8.4 Bicomplexes: Attributes

8.4.1 TotalComplex

▷ `TotalComplex(`*BC*`)` (attribute)
Returns: a homalg (co)complex
The associated total complex.

8.4.2 SpectralSequence (for bicomplexes)

▷ `SpectralSequence(`*BC*`)` (attribute)
Returns: a homalg (co)homological spectral sequence
The associated spectral sequence.

8.5 Bicomplexes: Operations and Functions

8.5.1 UnderlyingComplex

▷ `UnderlyingComplex(`*BC*`)` (function)
Returns: a homalg complex
The (co)complex of (co)complexes underlying the (co)homological bicomplex *BC*.

8.5.2 ByASmallerPresentation (for bicomplexes)

▷ `ByASmallerPresentation(`*B*`)` (method)
Returns: a homalg bicomplex
See `ByASmallerPresentation` (6.5.2) on complexes.

Code

```
InstallMethod( ByASmallerPresentation,
        "for homalg bicomplexes",
        [ IsHomalgBicomplex ],

  function( B )

    ByASmallerPresentation( UnderlyingComplex( B ) );

    IsZero( B );

    return B;

end );
```

This method performs side effects on its argument *B* and returns it.

Chapter 9

Bigraded Objects

Bigraded objects in homalg provide a data structure for the sheets (or pages) of spectral sequences.

9.1 BigradedObjects: Categories and Representations

9.1.1 IsHomalgBigradedObject

▷ `IsHomalgBigradedObject(`*Er*`)` (Category)

Returns: `true` or `false`

The GAP category of homalg bigraded objects.

(It is a subcategory of the GAP category `IsHomalgObject`.)

9.1.2 IsHomalgBigradedObjectAssociatedToAnExactCouple

▷ `IsHomalgBigradedObjectAssociatedToAnExactCouple(`*Er*`)` (Category)

Returns: `true` or `false`

The GAP category of homalg bigraded objects associated to an exact couple.

(It is a subcategory of the GAP category `IsHomalgBigradedObject`.)

9.1.3 IsHomalgBigradedObjectAssociatedToAFilteredComplex

▷ `IsHomalgBigradedObjectAssociatedToAFilteredComplex(`*Er*`)` (Category)

Returns: `true` or `false`

The GAP category of homalg bigraded objects associated to a filtered complex.

The 0-th spectral sheet E_0 stemming from a filtration is a bigraded (differential) object, which, in general, does not stem from an exact couple (although E_1, E_2, ... do).

(It is a subcategory of the GAP category `IsHomalgBigradedObject`.)

9.1.4 IsHomalgBigradedObjectAssociatedToABicomplex

▷ `IsHomalgBigradedObjectAssociatedToABicomplex(`*Er*`)` (Category)

Returns: `true` or `false`

The GAP category of homalg bigraded objects associated to a bicmplex.

(It is a subcategory of the GAP category `IsHomalgBigradedObjectAssociatedToAFilteredComplex`.)

9.1.5 IsBigradedObjectOfFinitelyPresentedObjectsRep

▷ IsBigradedObjectOfFinitelyPresentedObjectsRep(*Er*) (Representation)

Returns: true or false

The GAP representation of bigraded objects of finitley generated homalg objects.

(It is a representation of the GAP category IsHomalgBigradedObject (9.1.1), which is a subrepresentation of the GAP representation IsFinitelyPresentedObjectRep.)

9.2 Bigraded Objects: Constructors

9.2.1 HomalgBigradedObject (constructor for bigraded objects given a bicomplex)

▷ HomalgBigradedObject(*B*) (operation)

Returns: a homalg bigraded object

This constructor creates a homological (resp. cohomological) bigraded object given a homological (resp. cohomological) homalg bicomplex *B* (→ HomalgBicomplex (8.2.1)). This is nothing but the level zero sheet (without differential) of the spectral sequence associated to the bicomplex *B*. So it is the double array of homalg objects (i.e. static objects or complexes) in *B* forgetting the morphisms.

Example

```
gap> ZZ := HomalgRingOfIntegers( );
Z
gap> M := HomalgMatrix( "[ 2, 3, 4,   5, 6, 7 ]", 2, 3, ZZ );;
gap> M := LeftPresentation( M );
<A non-torsion left module presented by 2 relations for 3 generators>
gap> d := Resolution( M );;
gap> dd := Hom( d );;
gap> C := Resolution( dd );;
gap> CC := Hom( C );
<A non-zero acyclic complex containing a single morphism of left cocomplexes a\
t degrees [ 0 .. 1 ]>
gap> B := HomalgBicomplex( CC );
<A non-zero bicomplex containing left modules at bidegrees [ 0 .. 1 ]x
[ -1 .. 0 ]>
gap> E0 := HomalgBigradedObject( B );
<A bigraded object containing left modules at bidegrees [ 0 .. 1 ]x
[ -1 .. 0 ]>
gap> Display( E0 );
Level 0:

 * *
 * *
```

9.2.2 AsDifferentialObject (for homalg bigraded objects stemming from a bicomplex)

▷ AsDifferentialObject(*Er*) (method)

Returns: a homalg bigraded object

Add the induced bidegree $(-r, r-1)$ (resp. $(r, -r+1)$) differential to the level *r* homological (resp. cohomological) bigraded object stemming from a homological (resp. cohomological) bicomplex. This method performs side effects on its argument *Er* and returns it.

For an example see DefectOfExactness (9.2.3) below.

9.2.3 DefectOfExactness (for homalg differential bigraded objects)

▷ `DefectOfExactness(Er)` (method)

Returns: a homalg bigraded object

Homological: Compute the homology of a level `r` *differential* homological bigraded object, that is the `r`-th sheet of a homological spectral sequence endowed with a bidegree $(-r, r-1)$ differential. The result is a level $r+1$ homological bigraded object *without* its differential.

Cohomological: Compute the cohomology of a level `r` *differential* cohomological bigraded object, that is the `r`-th sheet of a cohomological spectral sequence endowed with a bidegree $(r, -r+1)$ differential. The result is a level $r+1$ cohomological bigraded object *without* its differential.

The differential of the resulting level $r+1$ object can a posteriori be computed using `AsDifferentialObject` (9.2.2). The objects in the result are subquotients of the objects in `Er`. An object in `Er` (at a spot (p,q)) is called *stable* if no passage to a true subquotient occurs at any higher level. Of course, a zero object (at a spot (p,q)) is always stable.

Example

```
gap> ZZ := HomalgRingOfIntegers( );
Z
gap> M := HomalgMatrix( "[ 2, 3, 4,   5, 6, 7 ]", 2, 3, ZZ );;
gap> M := LeftPresentation( M );
<A non-torsion left module presented by 2 relations for 3 generators>
gap> d := Resolution( M );;
gap> dd := Hom( d );;
gap> C := Resolution( dd );;
gap> CC := Hom( C );
<A non-zero acyclic complex containing a single morphism of left cocomplexes a\
t degrees [ 0 .. 1 ]>
gap> B := HomalgBicomplex( CC );
<A non-zero bicomplex containing left modules at bidegrees [ 0 .. 1 ]x
[ -1 .. 0 ]>
```

Now we construct the spectral sequence associated to the bicomplex B, also called the *first* spectral sequence:

Example

```
gap> I_E0 := HomalgBigradedObject( B );
<A bigraded object containing left modules at bidegrees [ 0 .. 1 ]x
[ -1 .. 0 ]>
gap> Display( I_E0 );
Level 0:

 * *
 * *
gap> AsDifferentialObject( I_E0 );
<A bigraded object with a differential of bidegree
[ 0, -1 ] containing left modules at bidegrees [ 0 .. 1 ]x[ -1 .. 0 ]>
gap> I_E0;
<A bigraded object with a differential of bidegree
[ 0, -1 ] containing left modules at bidegrees [ 0 .. 1 ]x[ -1 .. 0 ]>
gap> AsDifferentialObject( I_E0 );
<A bigraded object with a differential of bidegree
[ 0, -1 ] containing left modules at bidegrees [ 0 .. 1 ]x[ -1 .. 0 ]>
gap> I_E1 := DefectOfExactness( I_E0 );
```

```
<A bigraded object containing left modules at bidegrees [ 0 .. 1 ]x
[ -1 .. 0 ]>
gap> Display( I_E1 );
Level 1:

 * *
 . .
gap> AsDifferentialObject( I_E1 );
<A bigraded object with a differential of bidegree
[ -1, 0 ] containing left modules at bidegrees [ 0 .. 1 ]x[ -1 .. 0 ]>
gap> I_E2 := DefectOfExactness( I_E1 );
<A bigraded object containing left modules at bidegrees [ 0 .. 1 ]x
[ -1 .. 0 ]>
gap> Display( I_E2 );
Level 2:

 s .
 . .
```

Legend:

- A star $*$ stands for a nonzero object.
- A dot . stands for a zero object.
- The letter s stands for a nonzero object that became stable.

The *second* spectral sequence of the bicomplex is, by definition, the spectral sequence associated to the transposed bicomplex:

Example

```
gap> tB := TransposedBicomplex( B );
<A non-zero bicomplex containing left modules at bidegrees [ -1 .. 0 ]x
[ 0 .. 1 ]>
gap> II_E0 := HomalgBigradedObject( tB );
<A bigraded object containing left modules at bidegrees [ -1 .. 0 ]x
[ 0 .. 1 ]>
gap> Display( II_E0 );
Level 0:

 * *
 * *
gap> AsDifferentialObject( II_E0 );
<A bigraded object with a differential of bidegree
[ 0, -1 ] containing left modules at bidegrees [ -1 .. 0 ]x[ 0 .. 1 ]>
gap> II_E1 := DefectOfExactness( II_E0 );
<A bigraded object containing left modules at bidegrees [ -1 .. 0 ]x
[ 0 .. 1 ]>
gap> Display( II_E1 );
Level 1:

 * *
 . s
gap> AsDifferentialObject( II_E1 );
```

```
<A bigraded object with a differential of bidegree
[ -1, 0 ] containing left modules at bidegrees [ -1 .. 0 ]x[ 0 .. 1 ]>
gap> II_E2 := DefectOfExactness( II_E1 );
<A bigraded object containing left modules at bidegrees [ -1 .. 0 ]x
[ 0 .. 1 ]>
gap> Display( II_E2 );
Level 2:

 s .
 . s
```

9.3 Bigraded Objects: Properties

9.3.1 IsEndowedWithDifferential

▷ IsEndowedWithDifferential(*Er*) (property)

Returns: true or false

Check if *Er* is a differential bigraded object.

(no method installed)

9.3.2 IsStableSheet

▷ IsStableSheet(*Er*) (property)

Returns: true or false

Check if *Er* is stable.

(no method installed)

9.4 Bigraded Objects: Operations and Functions

9.4.1 ByASmallerPresentation (for bigraded objects)

▷ ByASmallerPresentation(*Er*) (method)

Returns: a homalg bigraded object

It invokes ByASmallerPresentation for homalg (static) objects.

Code

```
InstallMethod( ByASmallerPresentation,
        "for homalg bigraded objects",
        [ IsHomalgBigradedObject ],

  function( Er )

    List( Flat( ObjectsOfBigradedObject( Er ) ), ByASmallerPresentation );

    return Er;

end );
```

This method performs side effects on its argument *Er* and returns it.

Chapter 10

Spectral Sequences

Spectral sequences are regarded as the computational sledgehammer in homological algebra. Quoting the last lines of Rotman's book [Rot79]:

"The reader should now be convinced that virtually every purely homological result may be proved with spectral sequences. Even though "elementary" proofs may exist for many of these results, spectral sequences offer a systematic approach in place of sporadic success."

10.1 SpectralSequences: Categorie and Representations

10.1.1 IsHomalgSpectralSequence

▷ `IsHomalgSpectralSequence(`*E*`)` (Category)

Returns: `true` or `false`

The GAP category of homalg (co)homological spectral sequences.
(It is a subcategory of the GAP category `IsHomalgObject`.)

10.1.2 IsHomalgSpectralSequenceAssociatedToAnExactCouple

▷ `IsHomalgSpectralSequenceAssociatedToAnExactCouple(`*E*`)` (Category)

Returns: `true` or `false`

The GAP category of homalg associated to an exact couple.
(It is a subcategory of the GAP category `IsHomalgSpectralSequence`.)

10.1.3 IsHomalgSpectralSequenceAssociatedToAFilteredComplex

▷ `IsHomalgSpectralSequenceAssociatedToAFilteredComplex(`*E*`)` (Category)

Returns: `true` or `false`

The GAP category of homalg associated to a filtered complex.
(It is a subcategory of the GAP category `IsHomalgSpectralSequence`.)

The 0-th spectral sheet E_0 stemming from a filtration is a bigraded (differential) object, which, in general, does not stem from an exact couple (although E_1, E_2, ... do).

10.1.4 IsHomalgSpectralSequenceAssociatedToABicomplex

▷ `IsHomalgSpectralSequenceAssociatedToABicomplex(E)` (Category)

Returns: `true` or `false`

The GAP category of homalg associated to a bicomplex.

(It is a subcategory of the GAP category `IsHomalgSpectralSequenceAssociatedToAFilteredComplex`.)

10.1.5 IsSpectralSequenceOfFinitelyPresentedObjectsRep

▷ `IsSpectralSequenceOfFinitelyPresentedObjectsRep(E)` (Representation)

Returns: `true` or `false`

The GAP representation of homological spectral sequences of finitley generated homalg objects.

(It is a representation of the GAP category `IsHomalgSpectralSequence` (10.1.1), which is a subrepresentation of the GAP representation `IsFinitelyPresentedObjectRep`.)

10.1.6 IsSpectralCosequenceOfFinitelyPresentedObjectsRep

▷ `IsSpectralCosequenceOfFinitelyPresentedObjectsRep(E)` (Representation)

Returns: `true` or `false`

The GAP representation of cohomological spectral sequences of finitley generated homalg objects.

(It is a representation of the GAP category `IsHomalgSpectralSequence` (10.1.1), which is a subrepresentation of the GAP representation `IsFinitelyPresentedObjectRep`.)

10.2 Spectral Sequences: Constructors

10.2.1 HomalgSpectralSequence (constructor for spectral sequences given a bicomplex)

▷ `HomalgSpectralSequence(r, B, a)` (operation)
▷ `HomalgSpectralSequence(r, B)` (operation)
▷ `HomalgSpectralSequence(B, a)` (operation)
▷ `HomalgSpectralSequence(B)` (operation)

Returns: a homalg spectral sequence

The first syntax is the main constructor. It creates the homological (resp. cohomological) spectral sequence associated to the homological (resp. cohomological) bicomplex `B` starting at level 0 and ending at level $r \geq 0$ (regardless if the spectral sequence stabilizes earlier). The generalized embeddings into the objects of 0-th sheet are always computed for each higher sheet Er and stored as a record under the component Er!.absolute_embeddings. If `a` is greater than 0 the generalized embeddings into the objects of the `a`-th sheet also get computed for each higher sheet Er and stored as a record under the component Er!.relative_embeddings. The level `a` at which the spectral sequence becomes intrinsic is a natural candidate for `a`. The `a`-th sheet is called the *special* sheet.

If $r = -1$ it computes all the sheets of the spectral sequence until the sequence stabilizes, i.e. until all higher arrows become zero.

If $a = -1$ no special sheet is specified.

In the second syntax `a` is set to -1.

In the third syntax `r` is set to -1.

In the fourth syntax both `r` and `a` are set to -1.

The following example demonstrates the computation of a $Tor - Ext$ spectral sequence:

Example

```
gap> ZZ := HomalgRingOfIntegers( );
Z
gap> M := HomalgMatrix( "[ 2, 3, 4,   5, 6, 7 ]", 2, 3, ZZ );;
gap> M := LeftPresentation( M );
<A non-torsion left module presented by 2 relations for 3 generators>
gap> dM := Resolution( M );
<A non-zero right acyclic complex containing a single morphism of left modules\
 at degrees [ 0 .. 1 ]>
gap> CC := Hom( dM, dM );
<A non-zero acyclic cocomplex containing a single morphism of right complexes \
at degrees [ 0 .. 1 ]>
gap> B := HomalgBicomplex( CC );
<A non-zero bicocomplex containing right modules at bidegrees [ 0 .. 1 ]x
[ -1 .. 0 ]>
```

Now we construct the spectral sequence associated to the bicomplex B, also called the *first* spectral sequence:

Example

```
gap> I_E := HomalgSpectralSequence( 2, B );
<A stable cohomological spectral sequence with sheets at levels
[ 0 .. 2 ] each consisting of right modules at bidegrees [ 0 .. 1 ]x
[ -1 .. 0 ]>
gap> Display( I_E );
a cohomological spectral sequence at bidegrees
[ [ 0 .. 1 ], [ -1 .. 0 ] ]
---------
Level 0:

 * *
 * *
---------
Level 1:

 * *
 . .
---------
Level 2:

 s s
 . .
```

Legend:

- A star `*` stands for a nonzero object.
- A dot `.` stands for a zero object.
- The letter `s` stands for a nonzero object that became stable.

The *second* spectral sequence of the bicomplex is, by definition, the spectral sequence associated to the transposed bicomplex:

Example

```
gap> tB := TransposedBicomplex( B );
<A non-zero bicocomplex containing right modules at bidegrees [ -1 .. 0 ]x
[ 0 .. 1 ]>
gap> II_E := HomalgSpectralSequence( tB, 2 );
<A stable cohomological spectral sequence with sheets at levels
[ 0 .. 2 ] each consisting of right modules at bidegrees [ -1 .. 0 ]x
[ 0 .. 1 ]>
gap> Display( II_E );
a cohomological spectral sequence at bidegrees
[ [ -1 .. 0 ], [ 0 .. 1 ] ]
---------
Level 0:

 * *
 * *
---------
Level 1:

 * *
 * *
---------
Level 2:

 s s
 . s
```

10.3 Spectral Sequences: Attributes

10.3.1 GeneralizedEmbeddingsInTotalObjects

▷ `GeneralizedEmbeddingsInTotalObjects(`*E*`)` (attribute)

Returns: a record containing homalg maps

The generalized embbedings of the objects in the stable sheet into the objects of the associated total complex.

10.3.2 GeneralizedEmbeddingsInTotalDefects

▷ `GeneralizedEmbeddingsInTotalDefects(`*E*`)` (attribute)

Returns: a record containing homalg maps

The generalized embbedings of the objects in the stable sheet into the defects of the associated total complex.

10.4 Spectral Sequences: Operations and Functions

10.4.1 ByASmallerPresentation (for spectral sequences)

▷ `ByASmallerPresentation(`*E*`)` (method)

Returns: a homalg spectral sequence

See ByASmallerPresentation (9.4.1) on bigraded object.

Code

```
InstallMethod( ByASmallerPresentation,
        "for homalg spectral sequences",
        [ IsHomalgSpectralSequence ],

  function( E )

    ByASmallerPresentation( HighestLevelSheetInSpectralSequence( E ) );

    if IsBound( E!.TransposedSpectralSequence ) then
        ByASmallerPresentation( E!.TransposedSpectralSequence );
    fi;

    return E;

end );
```

This method performs side effects on its argument E and returns it.

Chapter 11

Functors

Functors and their natural transformations form the heart of the homalg package. Usually, a functor is realized in computer algebra systems as a procedure which can be applied to a certain type of objects. In [BR08] it was explained how to implement a functor of Abelian categories – by itself – as an object which can be further manipulated (composed, derived, ...). So in addition to the constructor `CreateHomalgFunctor` (11.2.1) which is used to create functors from scratch, homalg provides further easy-to-use constructors to create new functors out of existing ones:

- `InsertObjectInMultiFunctor` (11.2.2)
- `RightSatelliteOfCofunctor` (11.2.3)
- `LeftSatelliteOfFunctor` (11.2.4)
- `RightDerivedCofunctor` (11.2.5)
- `LeftDerivedFunctor` (11.2.6)
- `ComposeFunctors` (11.2.7)

In homalg each functor is implemented as a GAP4 object.

So-called installers (→ `InstallFunctor` (11.7.1) and `InstallDeltaFunctor` (11.7.2)) take such a functor object and create operations in order to apply the functor on objects, morphisms, complexes (of objects or again of complexes), and chain morphisms. The installer `InstallDeltaFunctor` (11.7.2) creates additional operations for δ-functors in order to compute connecting homomorphisms, exact triangles, and associated long exact sequences by starting with a short exact sequence.

In homalg special emphasis is laid on the action of functors on *morphisms*, as an essential part of the very definition of a functor. This is for no obvious reason often neglected in computer algebra systems. Starting from a functor where the action on morphisms is also defined, all the above constructors again create functors with actions both on objects and on morphisms (and hence on chain complexes and chain morphisms).

It turned out that in a variety of situations a caching mechanism for functors is not only extremely useful (e.g. to avoid repeated expensive computations) but also an absolute necessity for the coherence of data. Functors in homalg are therefore endowed with a caching mechanism.

If R is a homalg ring in which the component R!.`ByASmallerPresentation` is set to true

```
R!.ByASmallerPresentation := true;
```

any functor which returns an object over R will first apply `ByASmallerPresentation` to its result before returning it.

One of the highlights in homalg is the computation of Grothendieck's spectral sequences connecting the composition of the derivations of two functors with the derived functor of their composite.

11.1 Functors: Category and Representations

11.1.1 IsHomalgFunctor

▷ `IsHomalgFunctor(F)` (Category)

Returns: `true` or `false`

The GAP category of homalg (multi-)functors.

11.1.2 IsHomalgFunctorRep

▷ `IsHomalgFunctorRep(E)` (Representation)

Returns: `true` or `false`

The GAP representation of homalg (multi-)functors.

(It is a representation of the GAP category `IsHomalgFunctor` (11.1.1).)

11.2 Functors: Constructors

11.2.1 CreateHomalgFunctor (constructor for functors)

▷ `CreateHomalgFunctor(list1, list2, ...)` (function)

Returns: a homalg functor

This constructor is used to create functors for homalg from scratch. *listN* is of the form *listN = [stringN, valueN]*. *stringN* will be the name of a component of the created functor and *valueN* will be its value. This constructor is listed here for the sake of completeness. Its documentation is rather better placed in a homalg programmers guide. The remaining constructors create new functors out of existing ones and are probably more interesting for end users.

The constructor does *not* invoke `InstallFunctor` (11.7.1). This has to be done manually!

11.2.2 InsertObjectInMultiFunctor (constructor for functors given a multi-functor and an object)

▷ `InsertObjectInMultiFunctor(F, p, obj, H)` (operation)

Returns: a homalg functor

Given a homalg multi-functor F with multiplicity m and a string H return the functor `Functor_H` $:= F(...,obj,...)$, where *obj* is inserted at the p-th position. Of course *obj* must be an object (e.g. ring, module, ...) that can be inserted at this particular position. The string H becomes the name of the returned functor ($\rightarrow$ `NameOfFunctor` (11.3.1)). The variable `Functor_H` will automatically be assigned if free, otherwise a warning is issued.

The constructor automatically invokes `InstallFunctor` (11.7.1) which installs several necessary operations under the name H.

Example

```
gap> ZZ := HomalgRingOfIntegers( );
Z
gap> ZZ * 1;
<The free right module of rank 1 on a free generator>
gap> InsertObjectInMultiFunctor( Functor_Hom_for_fp_modules, 2, ZZ * 1, "Hom_ZZ" );
<The functor Hom_ZZ for f.p. modules and their maps over computable rings>
gap> Functor_Hom_ZZ_for_fp_modules;        ## got automatically defined
<The functor Hom_ZZ for f.p. modules and their maps over computable rings>
gap> Hom_ZZ;                ## got automatically defined
<Operation "Hom_ZZ">
```

11.2.3 RightSatelliteOfCofunctor (constructor of the right satellite of a contravariant functor)

▷ `RightSatelliteOfCofunctor(F[, p][, H])` (operation)

Returns: a homalg functor

Given a homalg (multi-)functor F and a string H return the right satellite of F with respect to its p-th argument. F is assumed contravariant in its p-th argument. The string H becomes the name of the returned functor (→ `NameOfFunctor` (11.3.1)). The variable `Functor_`H will automatically be assigned if free, otherwise a warning is issued.

If p is not specified it is assumed 1. If the string H is not specified the letter 'S' is added to the left of the name of F (→ `NameOfFunctor` (11.3.1)).

The constructor automatically invokes `InstallFunctor` (11.7.1) which installs several necessary operations under the name H.

11.2.4 LeftSatelliteOfFunctor (constructor of the left satellite of a covariant functor)

▷ `LeftSatelliteOfFunctor(F[, p][, H])` (operation)

Returns: a homalg functor

Given a homalg (multi-)functor F and a string H return the left satellite of F with respect to its p-th argument. F is assumed covariant in its p-th argument. The string H becomes the name of the returned functor (→ `NameOfFunctor` (11.3.1)). The variable `Functor_`H will automatically be assigned if free, otherwise a warning is issued.

If p is not specified it is assumed 1. If the string H is not specified the string "S_" is added to the left of the name of F (→ `NameOfFunctor` (11.3.1)).

The constructor automatically invokes `InstallFunctor` (11.7.1) which installs several necessary operations under the name H.

11.2.5 RightDerivedCofunctor (constructor of the right derived functor of a contravariant functor)

▷ `RightDerivedCofunctor(F[, p][, H])` (operation)

Returns: a homalg functor

Given a homalg (multi-)functor F and a string H return the right derived functor of F with respect to its p-th argument. F is assumed contravariant in its p-th argument. The string H becomes the name of the returned functor (→ `NameOfFunctor` (11.3.1)). The variable `Functor_`H will automatically be assigned if free, otherwise a warning is issued.

If *p* is not specified it is assumed 1. If the string *H* is not specified the letter 'R' is added to the left of the name of *F* (→ NameOfFunctor (11.3.1)).

The constructor automatically invokes InstallFunctor (11.7.1) and InstallDeltaFunctor (11.7.2) which install several necessary operations under the name *H*.

11.2.6 LeftDerivedFunctor (constructor of the left derived functor of a covariant functor)

▷ LeftDerivedFunctor(*F*[, *p*][, *H*]) (operation)

Returns: a homalg functor

Given a homalg (multi-)functor *F* and a string *H* return the left derived functor of *F* with respect to its *p*-th argument. *F* is assumed covariant in its *p*-th argument. The string *H* becomes the name of the returned functor (→ NameOfFunctor (11.3.1)). The variable Functor_*H* will automatically be assigned if free, otherwise a warning is issued.

If *p* is not specified it is assumed 1. If the string *H* is not specified the letter "S_" is added to the left of the name of *F* (→ NameOfFunctor (11.3.1)).

The constructor automatically invokes InstallFunctor (11.7.1) and InstallDeltaFunctor (11.7.2) which install several necessary operations under the name *H*.

11.2.7 ComposeFunctors (constructor for functors given two functors)

▷ ComposeFunctors(*F*[, *p*], *G*[, *H*]) (operation)

Returns: a homalg functor

Given two homalg (multi-)functors *F* and *G* and a string *H* return the composed functor Functor_*H* := $F(...,G(...),...)$, where *G* is inserted at the *p*-th position. Of course *G* must be a functor that can be inserted at this particular position. The string *H* becomes the name of the returned functor (→ NameOfFunctor (11.3.1)). The variable Functor_*H* will automatically be assigned if free, otherwise a warning is issued.

If *p* is not specified it is assumed 1. If the string *H* is not specified the names of *F* and *G* are concatenated in this order (→ NameOfFunctor (11.3.1)).

F * *G* is a shortcut for ComposeFunctors(*F*,1,*G*).

The constructor automatically invokes InstallFunctor (11.7.1) which installs several necessary operations under the name *H*.

Check this:

Example

```
gap> Functor_Hom_for_fp_modules * Functor_TensorProduct_for_fp_modules;
<The functor HomTensorProduct for f.p. modules and their maps over computable \
rings>
gap> Functor_HomTensorProduct_for_fp_modules;        ## got automatically defined
<The functor HomTensorProduct for f.p. modules and their maps over computable \
rings>
gap> HomTensorProduct;                ## got automatically defined
<Operation "HomTensorProduct">
```

11.3 Functors: Attributes

11.3.1 NameOfFunctor

▷ NameOfFunctor(*F*) (attribute)

Returns: a string

The name of the homalg functor *F*.

Example

```
gap> NameOfFunctor( Functor_Ext_for_fp_modules );
"Ext"
gap> Display( Functor_Ext_for_fp_modules );
Ext
```

11.3.2 OperationOfFunctor

▷ OperationOfFunctor(*F*) (attribute)

Returns: an operation

The operation of the functor *F*.

Example

```
gap> Functor_Ext_for_fp_modules;
<The functor Ext for f.p. modules and their maps over computable rings>
gap> OperationOfFunctor( Functor_Ext_for_fp_modules );
<Operation "Ext">
```

11.3.3 Genesis

▷ Genesis(*F*) (attribute)

Returns: a list

The first entry of the returned list is the name of the constructor used to create the functor *F*. The reset of the list contains arguments that were passed to this constructor for creating *F*.

These are examples of different functors created using the different constructors:

- CreateHomalgFunctor:

  Example

  ```
  gap> Functor_Hom_for_fp_modules;
  <The functor Hom for f.p. modules and their maps over computable rings>
  ```

- InsertObjectInMultiFunctor:

  Example

  ```
  gap> ZZ := HomalgRingOfIntegers( );
  Z
  gap> LeftDualizingFunctor( ZZ, "ZZ_Hom" );
  <The functor ZZ_Hom for f.p. modules and their maps over computable rings>
  gap> Functor_ZZ_Hom_for_fp_modules;        ## got automatically defined
  <The functor ZZ_Hom for f.p. modules and their maps over computable rings>
  gap> ZZ_Hom;                ## got automatically defined
  <Operation "ZZ_Hom">
  gap> Genesis( Functor_ZZ_Hom_for_fp_modules );
  [ "InsertObjectInMultiFunctor",
    <The functor Hom for f.p. modules and their maps over computable rings>, 2,
  ```

```
    <The free left module of rank 1 on a free generator> ]
gap> 1 * ZZ;
<The free left module of rank 1 on a free generator>
```

- LeftDerivedFunctor:

Example

```
gap> Functor_TensorProduct_for_fp_modules;
<The functor TensorProduct for f.p. modules and their maps over computable rin\
gs>
gap> Genesis( Functor_LTensorProduct_for_fp_modules );
[ "LeftDerivedFunctor",
  <The functor TensorProduct for f.p. modules and their maps over computable r\
ings>, 1 ]
```

- RightDerivedCofunctor:

Example

```
gap> Genesis( Functor_RHom_for_fp_modules );
[ "RightDerivedCofunctor",
  <The functor Hom for f.p. modules and their maps over computable rings>, 1 ]
```

- LeftSatelliteOfFunctor:

Example

```
gap> Genesis( Functor_Tor_for_fp_modules );
[ "LeftSatelliteOfFunctor",
  <The functor TensorProduct for f.p. modules and their maps over computable r\
ings>, 1 ]
```

- RightSatelliteOfCofunctor:

Example

```
gap> Genesis( Functor_Ext_for_fp_modules );
[ "RightSatelliteOfCofunctor",
  <The functor Hom for f.p. modules and their maps over computable rings>, 1 ]
```

- ComposeFunctors:

Example

```
gap> Genesis( Functor_HomHom_for_fp_modules );
[ "ComposeFunctors",
  [ <The functor Hom for f.p. modules and their maps over computable rings>,
      <The functor Hom for f.p. modules and their maps over computable rings>
    ], 1 ]
gap> ValueGlobal( "ComposeFunctors" );
<Operation "ComposeFunctors">
```

11.3.4 ProcedureToReadjustGenerators (for functors)

▷ ProcedureToReadjustGenerators(*Functor*) (attribute)

Returns: a function

11.4 Basic Functors

11.4.1 functor_Kernel

▷ functor_Kernel (global variable)

The functor that associates to a map its kernel.

Code

```
InstallValue( functor_Kernel,
        CreateHomalgFunctor(
                [ "name", "Kernel" ],
                [ "category", HOMALG.category ],
                [ "operation", "Kernel" ],
                [ "natural_transformation", "KernelEmb" ],
                [ "special", true ],
                [ "number_of_arguments", 1 ],
                [ "1", [ [ "covariant" ],
                        [ IsStaticMorphismOfFinitelyGeneratedObjectsRep,
                          [ IsHomalgChainMorphism, IsKernelSquare ] ] ] ],
                [ "OnObjects", _Functor_Kernel_OnObjects ]
                )
        );
```

11.4.2 functor_DefectOfExactness

▷ functor_DefectOfExactness (global variable)

The functor that associates to a pair of composable maps with a zero compositum the defect of exactness, i.e. the kernel of the outer map modulo the image of the inner map.

Code

```
InstallValue( functor_DefectOfExactness,
        CreateHomalgFunctor(
                [ "name", "DefectOfExactness" ],
                [ "category", HOMALG.category ],
                [ "operation", "DefectOfExactness" ],
                [ "special", true ],
                [ "number_of_arguments", 2 ],
                [ "1", [ [ "covariant" ],
                        [ IsStaticMorphismOfFinitelyGeneratedObjectsRep,
                          [ IsHomalgChainMorphism, IsLambekPairOfSquares ] ] ] ],
                [ "2", [ [ "covariant" ],
                        [ IsStaticMorphismOfFinitelyGeneratedObjectsRep ] ] ],
                [ "OnObjects", _Functor_DefectOfExactness_OnObjects ]
                )
        );
```

11.5 Tool Functors

11.6 Other Functors

11.7 Functors: Operations and Functions

11.7.1 InstallFunctor

▷ `InstallFunctor(`*F*`)` (operation)

Install several methods for GAP operations that get declared under the name of the homalg (multi-)functor *F* (→ `NameOfFunctor` (11.3.1)). These methods are used to apply the functor to objects, morphisms, (co)complexes of objects, and (co)chain morphisms. The objects in the (co)complexes might again be (co)complexes.

(For purely technical reasons the multiplicity of the functor might at most be three. This restriction should disappear in future versions.)

Code

```
InstallMethod( InstallFunctor,
        "for homalg functors",
        [ IsHomalgFunctorRep ],

  function( Functor )

    InstallFunctorOnObjects( Functor );

    if IsSpecialFunctor( Functor ) then

        InstallSpecialFunctorOnMorphisms( Functor );

    else

        InstallFunctorOnMorphisms( Functor );

        InstallFunctorOnComplexes( Functor );

        InstallFunctorOnChainMorphisms( Functor );

    fi;

end );
```

The method does not return anything.

11.7.2 InstallDeltaFunctor

▷ `InstallDeltaFunctor(`*F*`)` (operation)

In case *F* is a δ-functor in the sense of Grothendieck the procedure installs several operations under the name of the homalg (multi-)functor *F* (→ `NameOfFunctor` (11.3.1)) allowing one to compute

connecting homomorphisms, exact triangles, and associated long exact sequences. The input of these operations is a short exact sequence.

(For purely technical reasons the multiplicity of the functor might at most be three. This restriction should disappear in future versions.)

Code

```
InstallMethod( InstallDeltaFunctor,
        "for homalg functors",
        [ IsHomalgFunctorRep ],

  function( Functor )
    local number_of_arguments;

    number_of_arguments := MultiplicityOfFunctor( Functor );

    if number_of_arguments = 1 then

        HelperToInstallUnivariateDeltaFunctor( Functor );

    elif number_of_arguments = 2 then

        HelperToInstallFirstArgumentOfBivariateDeltaFunctor( Functor );
        HelperToInstallSecondArgumentOfBivariateDeltaFunctor( Functor );

    elif number_of_arguments = 3 then

        HelperToInstallFirstArgumentOfTrivariateDeltaFunctor( Functor );
        HelperToInstallSecondArgumentOfTrivariateDeltaFunctor( Functor );
        HelperToInstallThirdArgumentOfTrivariateDeltaFunctor( Functor );

    fi;

end );
```

The method does not return anything.

Chapter 12

Examples

12.1 ExtExt

This corresponds to Example B.2 in [Bar].

Example

```
gap> ZZ := HomalgRingOfIntegers( );
Z
gap> imat := HomalgMatrix( "[ \
>   262,  -33,   75,  -40, \
>   682,  -86,  196, -104, \
>  1186, -151,  341, -180, \
> -1932,  248, -556,  292, \
>  1018, -127,  293, -156  \
> ]", 5, 4, ZZ );
<A 5 x 4 matrix over an internal ring>
gap> M := LeftPresentation( imat );
<A left module presented by 5 relations for 4 generators>
gap> N := Hom( ZZ, M );
<A rank 1 right module on 4 generators satisfying yet unknown relations>
gap> F := InsertObjectInMultiFunctor( Functor_Hom_for_fp_modules, 2, N, "TensorN" );
<The functor TensorN for f.p. modules and their maps over computable rings>
gap> G := LeftDualizingFunctor( ZZ );;
gap> II_E := GrothendieckSpectralSequence( F, G, M );
<A stable homological spectral sequence with sheets at levels
[ 0 .. 2 ] each consisting of left modules at bidegrees [ -1 .. 0 ]x
[ 0 .. 1 ]>
gap> Display( II_E );
The associated transposed spectral sequence:

a homological spectral sequence at bidegrees
[ [ 0 .. 1 ], [ -1 .. 0 ] ]
---------
Level 0:

 * *
 * *
---------
Level 1:
```

```
 * *
 . .
---------
Level 2:

 s s
 . .

Now the spectral sequence of the bicomplex:

a homological spectral sequence at bidegrees
[ [ -1 .. 0 ], [ 0 .. 1 ] ]
---------
Level 0:

 * *
 * *
---------
Level 1:

 * *
 . s
---------
Level 2:

 s s
 . s
gap> filt := FiltrationBySpectralSequence( II_E, 0 );
<An ascending filtration with degrees [ -1 .. 0 ] and graded parts:
   0:        <A non-torsion left module presented by 3 relations for 4 generators>
  -1:        <A non-zero left module presented by 33 relations for 8 generators>
of
<A non-zero left module presented by 27 relations for 19 generators>>
gap> ByASmallerPresentation( filt );
<An ascending filtration with degrees [ -1 .. 0 ] and graded parts:
   0:        <A non-torsion left module presented by 2 relations for 3 generators>

-1:        <A non-zero torsion left module presented by 6 relations for 6 generators>
of
<A rank 1 left module presented by 8 relations for 9 generators>>
gap> m := IsomorphismOfFiltration( filt );
<A non-zero isomorphism of left modules>
```

12.2 Purity

This corresponds to Example B.3 in [Bar].

Example

```
gap> ZZ := HomalgRingOfIntegers( );
Z
gap> imat := HomalgMatrix( "[ \
>   262,  -33,   75,  -40, \
```

```
>   682,  -86,  196, -104, \
>  1186, -151,  341, -180, \
> -1932,  248, -556,  292, \
>  1018, -127,  293, -156  \
> ]", 5, 4, ZZ );
<A 5 x 4 matrix over an internal ring>
gap> M := LeftPresentation( imat );
<A left module presented by 5 relations for 4 generators>
gap> filt := PurityFiltration( M );
<The ascending purity filtration with degrees [ -1 .. 0 ] and graded parts:
   0:        <A free left module of rank 1 on a free generator>

-1:        <A non-zero torsion left module presented by 2 relations for 2 generators>
of
<A non-pure rank 1 left module presented by 2 relations for 3 generators>>
gap> M;
<A non-pure rank 1 left module presented by 2 relations for 3 generators>
gap> II_E := SpectralSequence( filt );
<A stable homological spectral sequence with sheets at levels
[ 0 .. 2 ] each consisting of left modules at bidegrees [ -1 .. 0 ]x
[ 0 .. 1 ]>
gap> Display( II_E );
The associated transposed spectral sequence:

a homological spectral sequence at bidegrees
[ [ 0 .. 1 ], [ -1 .. 0 ] ]
---------
Level 0:

 * *
 * *
---------
Level 1:

 * *
 . .
---------
Level 2:

 s .
 . .

Now the spectral sequence of the bicomplex:

a homological spectral sequence at bidegrees
[ [ -1 .. 0 ], [ 0 .. 1 ] ]
---------
Level 0:

 * *
 * *
---------
Level 1:
```

```
 * *
 . s
---------
Level 2:

 s .
 . s
gap> m := IsomorphismOfFiltration( filt );
<A non-zero isomorphism of left modules>
gap> IsIdenticalObj( Range( m ), M );
true
gap> Source( m );
<A non-torsion left module presented by 2 relations for 3 generators (locked)>
gap> Display( last );
[ [   0,   2,   0 ],
  [   0,   0,  12 ] ]

Cokernel of the map

Z^(1x2) --> Z^(1x3),

currently represented by the above matrix
gap> Display( filt );
Degree 0:

Z^(1 x 1)
----------
Degree -1:

Z/< 2 > + Z/< 12 >
```

12.3 TorExt-Grothendieck

This corresponds to Example B.5 in [Bar].

Example

```
gap> ZZ := HomalgRingOfIntegers( );
Z
gap> imat := HomalgMatrix( "[ \
>   262,  -33,   75,  -40, \
>   682,  -86,  196, -104, \
>  1186, -151,  341, -180, \
> -1932,  248, -556,  292, \
>  1018, -127,  293, -156  \
> ]", 5, 4, ZZ );
<A 5 x 4 matrix over an internal ring>
gap> M := LeftPresentation( imat );
<A left module presented by 5 relations for 4 generators>
gap> F := InsertObjectInMultiFunctor( Functor_TensorProduct_for_fp_modules, 2, M, "TensorM" );
<The functor TensorM for f.p. modules and their maps over computable rings>
gap> G := LeftDualizingFunctor( ZZ );;
```

```
gap> II_E := GrothendieckSpectralSequence( F, G, M );
<A stable cohomological spectral sequence with sheets at levels
[ 0 .. 2 ] each consisting of left modules at bidegrees [ -1 .. 0 ]x
[ 0 .. 1 ]>
gap> Display( II_E );
The associated transposed spectral sequence:

a cohomological spectral sequence at bidegrees
[ [ 0 .. 1 ], [ -1 .. 0 ] ]
---------
Level 0:

 * *
 * *
---------
Level 1:

 * *
 . .
---------
Level 2:

 s s
 . .

Now the spectral sequence of the bicomplex:

a cohomological spectral sequence at bidegrees
[ [ -1 .. 0 ], [ 0 .. 1 ] ]
---------
Level 0:

 * *
 * *
---------
Level 1:

 * *
 . s
---------
Level 2:

 s s
 . s
gap> filt := FiltrationBySpectralSequence( II_E, 0 );
<A descending filtration with degrees [ -1 .. 0 ] and graded parts:

-1:        <A non-zero left module presented by yet unknown relations for 9 generator\
s>

0:        <A non-zero left module presented by yet unknown relations for 4 generators\
>
of
```

```
<A left module presented by yet unknown relations for 29 generators>>
gap> ByASmallerPresentation( filt );
<A descending filtration with degrees [ -1 .. 0 ] and graded parts:
  -1:        <A non-zero left module presented by 4 relations for 4 generators>
   0:        <A non-torsion left module presented by 2 relations for 3 generators>
of
<A non-torsion left module presented by 6 relations for 7 generators>>
gap> m := IsomorphismOfFiltration( filt );
<A non-zero isomorphism of left modules>
```

12.4 TorExt

This corresponds to Example B.6 in [Bar].

Example

```
gap> ZZ := HomalgRingOfIntegers( );
Z
gap> imat := HomalgMatrix( "[ \
>   262,  -33,   75,  -40, \
>   682,  -86,  196, -104, \
>  1186, -151,  341, -180, \
> -1932,  248, -556,  292, \
>  1018, -127,  293, -156  \
> ]", 5, 4, ZZ );
<A 5 x 4 matrix over an internal ring>
gap> M := LeftPresentation( imat );
<A left module presented by 5 relations for 4 generators>
gap> P := Resolution( M );
<A non-zero right acyclic complex containing a single morphism of left modules\
 at degrees [ 0 .. 1 ]>
gap> GP := Hom( P );
<A non-zero acyclic cocomplex containing a single morphism of right modules at\
 degrees [ 0 .. 1 ]>
gap> FGP := GP * P;
<A non-zero acyclic cocomplex containing a single morphism of left complexes a\
t degrees [ 0 .. 1 ]>
gap> BC := HomalgBicomplex( FGP );
<A non-zero bicocomplex containing left modules at bidegrees [ 0 .. 1 ]x
[ -1 .. 0 ]>
gap> p_degrees := ObjectDegreesOfBicomplex( BC )[1];
[ 0, 1 ]
gap> II_E := SecondSpectralSequenceWithFiltration( BC, p_degrees );
<A stable cohomological spectral sequence with sheets at levels
[ 0 .. 2 ] each consisting of left modules at bidegrees [ -1 .. 0 ]x
[ 0 .. 1 ]>
gap> Display( II_E );
The associated transposed spectral sequence:

a cohomological spectral sequence at bidegrees
[ [ 0 .. 1 ], [ -1 .. 0 ] ]
---------
Level 0:
```

```
 * *
 * *
---------
Level 1:

 * *
 . .
---------
Level 2:

 s s
 . .

Now the spectral sequence of the bicomplex:

a cohomological spectral sequence at bidegrees
[ [ -1 .. 0 ], [ 0 .. 1 ] ]
---------
Level 0:

 * *
 * *
---------
Level 1:

 * *
 * *
---------
Level 2:

 s s
 . s
gap> filt := FiltrationBySpectralSequence( II_E, 0 );
<A descending filtration with degrees [ -1 .. 0 ] and graded parts:

-1:        <A non-zero left module presented by yet unknown relations for 10 generato\
rs>
   0:        <A rank 1 left module presented by 3 relations for 4 generators>
of
<A left module presented by yet unknown relations for 13 generators>>
gap> ByASmallerPresentation( filt );
<A descending filtration with degrees [ -1 .. 0 ] and graded parts:
  -1:        <A non-zero left module presented by 4 relations for 4 generators>
   0:        <A rank 1 left module presented by 2 relations for 3 generators>
of
<A non-torsion left module presented by 6 relations for 7 generators>>
gap> m := IsomorphismOfFiltration( filt );
<A non-zero isomorphism of left modules>
```

Appendix A

The Mathematical Idea behind homalg

Appendix B

Development

B.1 Why was homalg discontinued in Maple?

The original implementation of homalg in Maple by Daniel Robertz and myself hit several walls. The speed of the Gröbner basis routines in Maple was the smallest issue. The rising complexity of data structures for high level algorithms (bicomplexes, functors, spectral sequences, ...) became the main problem. We very much felt the need for an object-oriented programming language, a language that allows defining complicated mathematical objects carrying properties and attributes and even containing other objects as subobjects.

As we were pushed to look for an alternative to Maple, our wish list grew even further. Section B.2 is a summary of this wish list.

B.2 Why GAP4?

B.2.1 GAP is free and open software

In 1993 J. Neubüser addressed the necessity of free software in mathematics:

"You can read Sylow's Theorem and its proof in Huppert's book in the library without even buying the book and then you can use Sylow's Theorem for the rest of your life free of charge, but - and for understandable reasons of getting funds for the maintenance, the necessity of which I have pointed out [...] - for many computer algebra systems license fees have to be paid regularly for the total time of their use. In order to protect what you pay for, you do not get the source, but only an executable, i.e. a black box. You can press buttons and you get answers in the same way as you get the bright pictures from your television set but you cannot control how they were made in either case.

With this situation two of the most basic rules of conduct in mathematics are violated. In mathematics information is passed on free of charge and everything is laid open for checking. Not applying these rules to computer algebra systems that are made for mathematical research [...] means moving in a most undesirable direction. Most important: Can we expect somebody to believe a result of a program that he is not allowed to see? [...] And even: If O'Nan and Scott would have to pay a license fee for using an implementation of their ideas about primitive groups, should not they in turn be entitled to charge a license fee for using their ideas in the implementation?"

I had the pleasure of being one of his students.

The detailed copyright for GAP can found on the GAP homepage under Start – Download – Copyright.

B.2.2 GAP has an area of expertise

Not only does GAP have the potential of natively supporting a wide range of mathematical structures, but finite groups and their representation theory are already an area of expertise. So there are at least some areas where one does not need to start from scratch.

But one could argue that rings are more central for homological algebra than finite groups, and that GAP4, as for the time when the homalg project was shaping, does not seriously support important rings in a manner that enables homological computations. This drawback would favor, for example, Singular (with its subsystem Plural) over GAP4. Point B.2.3 indicates how this drawback was overcome in a way, that even gave the lead back to GAP4.

One of my future plans for the homalg project is to address moduli problems in algebraic geometry (favorably via orbifold stacks), where discrete groups (and especially finite groups) play a central role. As of the time of writing these lines, discrete groups, finite groups, and orbifolds are already in the focus of part of the project: The package SCO by Simon Görtzen to compute the cohomology of orbifolds is part of the currently available homalg project.

For the remaining points the choice of GAP4 as the programming language for developing homalg was unavoidable.

B.2.3 GAP4 can communicate

With the excellent IO package of Max Neunhöffer GAP4 is able to communicate in an extremely efficient way with the outer world via bidirectional streams. This allows homalg to delegate things that cannot be done in GAP to an external system such as Singular, Sage, Macaulay2, MAGMA, or Maple.

B.2.4 GAP4 is a *mathematical* object-oriented programming language

The object-oriented programming philosophy of GAP4 was developed by mathematicians who wanted to handle complex mathematical objects carrying *properties* and *attributes*, as often encountered in algebra and geometry. GAP4 was thus designed to address the needs of *mathematical* object-oriented programming more than any other language designed by computer scientists. This was primarily achieved by the advanced *method selection* techniques that very much resemble the mathematical way of thinking.

Unlike the common object-oriented programming languages, methods in GAP4 are not bound to objects but to operations. In particular, one can also install methods that depend on two or more arguments. The index of a subgroup is an easy example of an operation illustrating this. While it would be sufficient to bind a method for computing the order of a group to the object representing the group, it is not clear what to do with the index, since its definition involves two objects: a group G and a subgroup U. Note that the index of U in a subgroup of G containing U might also be of interest. Things become even more complicated when the arguments of the operation are unrelated objects. Moreover, binding methods to operations makes it possible for the programming language to support the installation of one or more methods for the same operation, depending on already known properties or attributes of the involved objects.

Moreover GAP4 supports so-called *immediate and true methods*. This considerably simplifies teaching theorems to the computer. For example it takes one line of code to teach GAP4 that a reflexive left module over a ring with left global dimension less or equal to two is projective. These logical implications are installed globally and GAP4 immediately uses them as soon as the respective assumptions are fulfilled. This mechanism enables GAP4 to draw arbitrary long lines of conclusions.

The more one knows about the objects involved in the computation the more specialized efficient algorithms can be utilized, while other computations can be completely avoided. homalg is equipped with plenty of logical implications for rings, matrices, modules, morphisms, and complexes.

When all these features become relevant to what you want to do, there is hardly an alternative to GAP4.

B.2.5 GAP4 packages are easily extendible

Being able to install several methods for a single operation (→ B.2.4) has the additional advantage of making GAP4 packages easily extendible. If you have an algorithm that, in a special case, performs better than existing algorithms you can install it as a method which gets triggered when the special case occurs. You don't need to break existing code to insert an additional `elif` section contributing to an increasing unreadability of the code. Even better, you don't even need to know *anything* about the code of other existing methods. In addition to that, you can add (maybe missing) properties and attributes (along with methods to compute them) to existing objects.

B.3 Why not Sage?

Although the python-based Sage fulfills most of the above requirements, it was primarily the points expressed in B.2.4 that finally favored GAP4 over Sage: The object-orientedness of python, although very modern, does not cover the needs of the homalg package. At this place I would like to thank William Stein for the helpful discussion about Sage during the early stage of developing homalg, and to Max Neunhöffer who explained me the advantages of the object-oriented programming in GAP4.

B.4 How does homalg compare to Sage?

In what follows homalg often refers to the whole homalg project.

B.4.1 They differ in objectives and scale

First of all, Sage is a huge project, that, among other things, is intended to replace commercial, general purpose computer algebra systems like Maple and Mathematica. So while Sage targets (a growing number of) different fields of computer algebra, homalg only focuses on homological, and hopefully in the near future also homotopical techniques (applicable to some of these different fields). The two projects simply follow different goals and are different in scale.

B.4.2 They differ in the programming language

Sage is based on python and the C-extension cython while homalg is based on GAP4. Quoting from an email response William Stein sent me on the 25. of February, 2008: "Sage *is* Python + a library". Although I seriously considered developing homalg as part of Sage, for the reason mentioned in B.2.4 I finally decided to use GAP4 as the programming language.

B.4.3 They differ in the way they communicate with the outer world

Both Sage and homalg rely for many things on external computer algebra systems. But although one can simply invoke a GAP shell or a Singular shell from within Sage, Sage normally runs

the external computer algebra systems in the background and tries to understand the internals of the objects residing in them. An object in the external computer algebra system is wrapped by an object in Sage and supporting this external object involves understanding its details in the external system. homalg follows a different strategy: The only external objects homalg needs (beside rings) are non-empty matrices. And being zero or not is basically the only thing homalg wants to know about a matrix after knowing its dimension. I myself was stunned by this insight, which culminated in *the principle of least communication* (→ **Modules: The principle of least communication (technical)**).

In particular, Sage can make use of all of homalg, but for in order to make full use, Sage needs to understand the internals of the homalg objects. On the contrary, homalg can only make limited use of Sage (or of virtually any computer algebra system that supports rings in a sufficient way (→ (**Modules: Rings supported in a sufficient way**))), but without the need to delve into the inner life of the Sage objects.

Appendix C

Logic Subpackages

C.1 LIOBJ: Logical Implications for Objects of Abelian Categories

C.2 LIMOR: Logical Implications for Morphisms of Abelian Categories

C.3 LICPX: Logical Implications for Complexes in Abelian Categories

Appendix D

Debugging homalg

Beside the GAP builtin debugging facilities (→ (**Reference: Debugging and Profiling Facilities**)) homalg provides two ways to debug the computations.

D.1 Increase the assertion level

homalg comes with numerous builtin assertion checks. They are activated if the user increases the assertion level using

`SetAssertionLevel( level );`

(→ (**Reference: SetAssertionLevel**)), where *level* is one of the values below:

level	description
0	no assertion checks whatsoever
3	"high"-level homological assertions are checked
4	"mid"-level homological assertions are checked
5	"low"-level homological assertions are checked
6	assertions about basic matrix operations are checked (→ Appendices of the MatricesForHomalg package) (these are among the operations often delegated to external systems)

In particular, if homalg delegates matrix operations to an external system then `SetAssertionLevel( 4 );` can be used to let homalg debug the external system.

Appendix E

The Core Packages and the Idea behind their Splitting

I will try to explain the idea behind splitting the 6 *core packages*:

1. homalg
2. Modules
3. HomalgToCAS
4. IO_ForHomalg
5. RingsForHomalg
6. ExamplesForHomalg

E.1 The 6=2+4 split

The following is an attempt to explain the 6=2+4 split.

E.1.1 Logically independent

The package homalg is logically independent from all other packages in the project. And among the six core packages it is together with Modules the only package that has to do with mathematics. The remaining four packages are of technical nature. More precisely, homalg is a stand alone package, that offers abstract homological constructions for computable Abelian categories. But since the ring of integers (at least up till now) is the only ring which for the purposes of homological algebra is *sufficiently supported* in GAP ($\rightarrow$ (**Modules: Rings supported in a sufficient way**)), Modules can put the above mentioned abstract constructions into action only for the ring of integers and by generic (but of course non-efficient) methods for any of its residue class rings (Simon Görtzen's package Gauss adds the missing sufficient support for $\mathbb{Z}/p^n$ and $\mathbb{Q}$ to GAP and his other package GaussForHomalg makes this support visible to Modules).

E.1.2 Black boxes

The package Modules uses rings and matrices over these rings as a black box, enabling other packages to "abuse" homalg to compute over rings other than the ring of integers by simply providing the appropriate black boxes. And whether these rings and matrices are inside or outside GAP is not at all the concern of homalg. Even the GAP representation for external rings, external ring elements, and external matrices are declared in the package HomalgToCAS and not in homalg.

E.1.3 Summing up

One of the main concepts of the homalg project is that high level and low level computations in homological algebra can and *should* be separated. So splitting homalg from the remaining 4 core packages is just emphasizing this concept. Moreover, homalg is up till now by far the biggest package in the project and will probably keep growing by supporting more basic homological constructions, whereas the other 4 packages will remain stable over longer time intervals.

E.2 The 4=1+1+1+1 split

The following is meant to justify the remaining 4=1+1+1+1 split.

E.2.1 HomalgToCAS

The package HomalgToCAS (which needs the homalg package) includes all what is needed to let the black boxes used by homalg reside in external computer algebra systems. So as mentioned above, HomalgToCAS is the right place to declare the three GAP representations external rings, external ring elements, and external matrices. Still, HomalgToCAS is independent from the external computer algebra system with which GAP will communicate *and* independent of how this communication physically looks like.

E.2.2 IO_ForHomalg and Alternatives

The package IO_ForHomalg (which needs HomalgToCAS) allows GAP to communicate via I/O-streams with computer algebra systems that come with a terminal interface. IO_ForHomalg uses Max Neunhöffer's IO package, yet it is independent from the specific computer algebra system, as long as the latter provides a terminal interface. Splitting IO_ForHomalg from HomalgToCAS gives the freedom to replace the former by another package that lets GAP communicate with an external system using a different technology. So making IO_ForHomalg a package of its own makes it clear for developers of a new communication method which package of the homalg project has to be imitated/replaced. To be concrete, Thomas Bächler wrote a package called MapleForHomalg that enables GAP to communicate with Maple without the need for a terminal interface.

E.2.3 RingsForHomalg

The package RingsForHomalg (which needs HomalgToCAS) provides the details of the black boxes homalg relies on. The details of the black boxes of course depend on the external computer algebra system (Singular, MAGMA, Macaulay2, Maple, Sage, ...), but are independent from the way the communication takes place. So it can be used either with IO_ForHomalg, with MapleForHomalg, or with any future communication package.

E.2.4 Your own RingsForHomalg

If someone needs to support a ring in some computer algebra system that GAP can already communicate with, but where the ring is not supported by RingsForHomalg yet, she or he needs to imitate/replace RingsForHomalg (as Simon Görtzen did with his GaussForHomalg, where the computer algebra system was GAP itself, extended by his package Gauss). Any substitute for RingsForHomalg – as it only needs HomalgToCAS – will again be independent from the way how GAP communicates with the computer algebra system that hosts the ring. This should encourage people to link more external systems to homalg without being forced to join the development of the package RingsForHomalg. They can simply write their own package and get the full credit for it.

E.2.5 ExamplesForHomalg

The package ExamplesForHomalg (which needs RingsForHomalg) contains example scripts over various rings that are written in a universal way, i.e. independent from the system that hosts the rings. These examples cannot be part of the homalg package as they are defined over rings that GAP does not support. The package ExamplesForHomalg is meant to be the package where anyone can contribute interesting examples using homalg without necessarily contributing to the code of any of the remaining core packages.

E.2.6 Documentation

Splitting the core packages is part of documenting the project. The complete manuals of the homalg and ExamplesForHomalg packages (maybe apart from the appendices) can then be free from any non-mathematical technicalities the average user is not interested in. A documentation of the three packages HomalgToCAS, IO_ForHomalg, and RingsForHomalg will be rather technical and of interest mainly for developers.

E.2.7 Crediting

Everyone is encouraged to contribute to the homalg project. The project follows the philosophy of avoiding huge monolithic packages and splitting unrelated tasks. This should enable contributers to write their own packages (building on other existing packages) and getting the full credit for their work, which can then be easily distinguished from the work of others.

E.2.8 Stability

A huge monolithic package can never stabilize, even though parts of it will stay frozen for a long period of time. The splitting makes it likely that parts of the project together with their documentation quickly reach a stable state.

Appendix F

Overview of the homalg Package Source Code

The homalg package reached more than 50.000 lines of GAP4 code (excluding the documentation) before the first release was made. To keep this amount of code tracebale, the package was split in several files.

F.1 The Basic Objects

Filename .gd/.gi	Content
HomalgObject	objects of Abelian categories
HomalgSubobject	subobject of objects of Abelian categories
HomalgMorphism	morphisms of Abelian categories
HomalgElement	elements are morphisms from "structure objects"
HomalgFiltration	filtrations of objects of Abelian categories
HomalgComplex	(co)complexes of objects or of (co)complexes
HomalgChainMorphism	chain morphisms of (co)complexes consisting of morphisms or chain morphisms
HomalgBicomplex	bicomplexes of objects or of (co)complexes
HomalgBigradedObject	(differential) bigraded objects
HomalgSpectralSequence	homological and cohomological spectral sequences
HomalgFunctor	constructors of (multi) functors of Abelian categories, left derivation of covariant functors, right derivation of contravariant functors, left satellites of covariant functors, right satellites of contravariant functors, and composition of functors
HomalgDiagram	basic diagrams

Table: *The homalg package files (continued)*

F.2 The High Level Homological Algorithms

Filename .gd/.gi	Content
StaticObjects	subfactors, syzygy objects, shorten resolutions, saturations
Morphisms	resolutions, (co)kernel sequences
Complexes	(co)homology, horse shoe lemma, connecting homomorphisms, Cartan-Eilenberg resolution
ChainMorphisms	(co)homology
SpectralSequences	Grothendieck bicomplexes associated to two composable functors, spectral sequences of bicomplexes, Grothendieck spectral sequences
Filtrations	spectral filtrations, i.e. filtrations induced by spectral sequences of bicomplexes, purity filtration
ToolFunctors	composition, addition, substraction, stacking, augmentation, and post dividing maps
BasicFunctors	kernel, defect of exactness
OtherFunctors	torsion submodule, torsion free factor, pullback, pushout, Auslander dual

Table: *The homalg package files (continued)*

F.3 Logical Implications for homalg Objects

Filename .gd/.gi	Content
LIOBJ	logical implications for objects of an Abelian category
LIMOR	logical implications for morphisms of an Abelian category
LICPX	logical implications for complexes

Table: *The homalg package files (continued)*

References

[Bar] M. Barakat. Spectral Filtrations via Generalized Morphisms. arxiv.org/abs/0904.0240. 25, 68, 69, 71, 73

[BR08] M. Barakat and D. Robertz. homalg – A Meta-Package for Homological Algebra. *J. Algebra Appl.*, 7(3):299–317, 2008. arXiv:math.AC/0701146. 59

[CE99] H. Cartan and S. Eilenberg. *Homological algebra*. Princeton Landmarks in Mathematics. Princeton University Press, Princeton, NJ, 1999. With an appendix by David A. Buchsbaum, Reprint of the 1956 original. 7

[GM03] S. I. Gelfand and Y. I. Manin. *Methods of homological algebra*. Springer Monographs in Mathematics. Springer-Verlag, Berlin, 2. edition, 2003. 7

[HS97] P. J. Hilton and U. Stammbach. *A course in homological algebra*, volume 4 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, second edition, 1997. 7

[ML63] S. Mac Lane. *Homology*. Die Grundlehren der mathematischen Wissenschaften, Bd. 114. Academic Press Inc., Publishers, New York, 1963. 7

[Rot79] J. J. Rotman. *An introduction to homological algebra*, volume 85 of *Pure and Applied Mathematics*. Academic Press Inc. [Harcourt Brace Jovanovich Publishers], New York, 1979. 7, 54

[Wei94] C. A. Weibel. *An introduction to homological algebra*, volume 38 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, Cambridge, 1994. 7

Index