

Modules

**A homalg based package for the Abelian
category of finitely presented modules
over computable rings**

2026.04-01

10 April 2026

**Thomas Bächler
Mohamed Barakat
Florian Diebold
Sebastian Gutsche
Markus Lange-Hegermann
Vinay Wagh**

Thomas Bächler Email: thomas@momo.math.rwth-aachen.de

Homepage: <http://wwwb.math.rwth-aachen.de/~thomas/>

Address: Thomas Bächler
Lehrstuhl B für Mathematik
RWTH Aachen
Templergraben 64
52062 Aachen
Germany

Mohamed Barakat Email: mohamed.barakat@uni-siegen.de

Homepage: <https://mohamed-barakat.github.io>

Address: Walter-Flex-Str. 3
57072 Siegen
Germany

Florian Diebold Email: diebold@mathematik.uni-kl.de

Address: Department of Mathematics
University of Kaiserslautern
67653 Kaiserslautern
Germany

Sebastian Gutsche Email: gutsche@mathematik.uni-siegen.de

Homepage: <https://sebasguts.github.io>

Address: Department Mathematik
Universität Siegen
Walter-Flex-Straße 3
57072 Siegen
Germany

Markus Lange-Hegermann Email: markus.lange-hegermann@hs-owl.de

Homepage: <https://www.th-owl.de/eecs/fachbereich/team/markus-lange-hegermann/>

Address: Markus Lange-Hegermann
Hochschule Ostwestfalen-Lippe
Liebigstraße 87
32657 Lemgo
Germany

Vinay Wagh Email: waghoba@gmail.com

Homepage: <http://www.iitg.ernet.in/vinay.wagh/>

Address: E-102, Department of Mathematics,
Indian Institute of Technology Guwahati,
Guwahati, Assam, India.
PIN: 781 039.
India

Copyright

© 2007-2015 by Mohamed Barakat and Markus Lange-Hegermann This package may be distributed under the terms and conditions of the GNU Public License Version 2 or (at your option) any later version.

Acknowledgements

We are very much indebted to Alban Quadrat.

Contents

1	Introduction	6
1.1	What is the role of the Modules package in the homalg project?	6
1.2	This manual	9
2	Installation of the Modules Package	10
3	Quick Start	11
3.1	Why are all examples in this manual over \mathbb{Z} or $\mathbb{Z}/m\mathbb{Z}$?	11
3.2	gap> ExamplesForHomalg();	11
3.3	A typical example	11
4	Ring Maps	17
4.1	Ring Maps: Attributes	17
4.2	Ring Maps: Operations and Functions	17
5	Relations	18
5.1	Relations: Categories and Representations	18
5.2	Relations: Constructors	19
5.3	Relations: Properties	19
5.4	Relations: Attributes	19
5.5	Relations: Operations and Functions	19
6	Generators	20
6.1	Generators: Categories and Representations	20
6.2	Generators: Constructors	21
6.3	Generators: Properties	21
6.4	Generators: Attributes	21
6.5	Generators: Operations and Functions	21
7	Modules	22
7.1	Modules: Category and Representations	22
7.2	Modules: Constructors	24
7.3	Modules: Properties	29
7.4	Modules: Attributes	30
7.5	Modules: Operations and Functions	33

8	Maps	35
8.1	Maps: Categories and Representations	35
8.2	Maps: Constructors	36
8.3	Maps: Properties	39
8.4	Maps: Attributes	39
8.5	Maps: Operations and Functions	39
9	Module Elements	40
9.1	Module Elements: Category and Representations	40
9.2	Module Elements: Constructors	40
9.3	Module Elements: Properties	40
9.4	Module Elements: Attributes	41
9.5	Module Elements: Operations and Functions	41
10	Functors	42
10.1	Functors: Category and Representations	42
10.2	Functors: Constructors	42
10.3	Functors: Attributes	42
10.4	Basic Functors	42
10.5	Tool Functors	63
10.6	Other Functors	63
10.7	Functors: Operations and Functions	63
11	Symmetric Algebra and Koszul Complex	64
11.1	Symmetric Algebra: Constructor	64
11.2	Symmetric Algebra: Properties and Attributes	64
12	Exterior Algebra and Koszul Complex	65
12.1	Exterior Algebra: Constructor	65
12.2	Exterior Algebra: Properties and Attributes	65
12.3	Exterior Algebra: Element Properties	66
12.4	Exterior Algebra: Element Operations	66
12.5	Koszul complex and Cayley determinant	66
13	Examples	68
13.1	ExtExt	68
13.2	Purity	69
13.3	TorExt-Grothendieck	71
13.4	TorExt	73
A	The Mathematical Idea behind Modules	76
B	Logic Subpackages	77
B.1	LIMOD: Logical Implications for Modules	77
B.2	LIHOM: Logical Implications for Homomorphisms of Modules	77

C	Overview of the Modules Package Source Code	78
C.1	Relations and Generators	78
C.2	The Basic Objects	79
C.3	The High Level Homological Algorithms	79
C.4	Logical Implications for homalg Objects	80
	References	81
	Index	82

Chapter 1

Introduction

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

1.1.1 **Modules** provides ...

It provides procedures to construct basic objects in homological algebra:

- modules (generators, relations)
- submodules (as images of maps)
- maps

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

- resolution of modules
- images of maps
- the functors `Hom` and `TensorProduct` (`Ext` and `Tor` are then provided by **homalg**)
- test if a module is torsion-free, reflexive, projective, stably free, free, pure
- determine the rank, grade, projective dimension, degree of torsion-freeness, and codegree of purity of a module

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.

1.1.2 **Rings supported in a sufficient way**

Through out this manual the following terminology is used. We say that a computer algebra system “sufficiently supports” a ring R , if it contains procedures to effectively solve one-sided inhomogeneous linear systems $XA = B$ and $AX = B$ with coefficients over R (\rightarrow [722Moduleschapter.7-limitation](#))

1.1.3 Principal limitation

Note that the solution space of the one-sided finite dimensional system $YA = 0$ (resp. $AY = 0$) over a left (resp. right) noetherian ring R is a finitely generated left (resp. right) R -module, even if R is not commutative. The solution space of the linear system $X_1A_1 + A_2X_2 + A_3X_3A_4 = 0$ is in general not an R -module, and worse, in general not finitely generated over the center of R . **Modules** can only handle homological problems that lead to *one sided finite dimensional* homogeneous or inhomogeneous systems over the underlying ring R . Such problems are called problems of *finite type* over R . Typically, the computation of $\text{Hom}(M, N)$ of two (even) finitely generated modules over a *noncommutative* ring R is generally *not* of finite type over R , unless at least one of the two modules is an R -bimodule. Also note that over a commutative ring any linear system can be easily brought to a one-sided form. For more details see [BR08].

1.1.4 Ring dictionaries (technical)

Modules uses the so-called `homalgTable`, which is stored in the ring, to know how to delegate the necessary matrix operations. I.e. the `homalgTable` serves as a small dictionary that enables **Modules** to speak (as much as needed of) the language of the computer algebra system which hosts the ring and the matrices. The **GAP** internal ring of integers is the only ring which **Modules** endows with a `homalgTable`. Other packages like **GaussForHomalg** and **RingsForHomalg** provide dictionaries for further rings. While **GaussForHomalg** defines internal rings and matrices, the package **RingsForHomalg** enables defining external rings and matrices in a wide range of (external) computer algebra systems (Singular, Sage, Macaulay2, MAGMA, Maple) by providing appropriate dictionaries.

Since these dictionaries are all what is needed to handle matrix operations, **Modules** does not distinguish between handling internal and handling external matrices. Even the physical communication with the external systems is not at all a concern of **Modules**. This is the job of the package **IO_ForHomalg**, which is based on the powerful **IO** package of Max Neunhöffer. Furthermore, for all structures beyond matrices (from relations, generators, and modules, to functors and spectral sequences) **Modules** no longer distinguishes between internal and external.

1.1.5 The advantages of the outsourcing concept

Linking different systems to achieve one task is a highly attractive idea, especially if it helps to avoid reinventing wheels over and over again. This was essential for **homalg**, since **Singular** and **MAGMA** provide the fastest and most advanced Gröbner basis algorithms, while **GAP4** is by far the most convenient programming language to realize complex mathematical structures (\rightarrow Appendix (**homalg: Why GAP4?**)). Second, the implementation of the homological constructions is automatically universal, since it is independent of where the matrices reside and how the several matrix operations are realized. In particular, **homalg** will always be able to use the system with the fastest Gröbner basis implementation. In this respect is **homalg** and all packages that build upon it future proof.

1.1.6 Does this mean that **homalg** has only algorithms for the generic case?

No, on the contrary. There are a lot of specialized algorithms installed in **homalg**. These algorithms are based on properties and attributes that -- thanks to **GAP4** -- **homalg** objects can carry (\rightarrow Appendix (**homalg: GAP4 is a mathematical object-oriented programming language**)): Not only can **homalg** take the special nature of the underlying ring into account, it also deals with modules,

complexes, ... depending on their special properties. Still, these special algorithms, like all algorithms in `homalg`, are independent of the computer algebra system which hosts the matrices and which will perform the several matrix operations.

1.1.7 The principle of least communication (technical)

Linking different systems can also be highly problematic. The following two points are often among the major sources of difficulties:

- Different systems use different languages:
It takes a huge amount of time and effort to teach systems the dialects of each others. These dialects are also rarely fixed forever, and might very well be subject to slight modifications. So the larger the dictionary, the more difficult is its maintenance.
- Data has to be transferred from one system to another:
Even if there is a unified data format, transferring data between systems can lead to performance losses, especially when a big amount of data has to be transferred.

Solving these two difficulties is an important part of `Modules`'s design. `Modules` splits homological computations into two parts. The matrices reside in a system which provides fast matrix operations (addition, multiplication, bases and normal form computations), while the higher structures (modules, maps, complexes, chain morphisms, spectral sequences, functors, ...) with their properties, attributes, and algorithms live in `GAP4`, as the system where one can easily create such complex structures and handle all their logical dependencies. With this split there is no need to transfer each sort of data outside of its system. The remaining communication between `GAP4` and the system hosting the matrices gets along with a tiny dictionary. Moreover, `GAP4`, as it manages and delegates all computations, also manages the whole data flow, while the other system does not even recognize that it is part of a bidirectional communication.

The existence of such a clear cut is certainly to some extent due to the special nature of homological computations.

1.1.8 Frequently asked questions

- Q: Does outsourcing the matrices mean that `Modules` is able to compute spectral sequences, for example, without ever seeing the matrices involved in the computation?

A: Yes.

- Q: Can `Modules` profit from the implementation of homological constructions like `Hom`, `Ext`, ... in `Singular`?

A: No. This is for a lot of reasons incompatible with the idea and design ([→ 1](#)) of `Modules`.

- Q: Are the external systems involved in the higher algorithms?

A: No. They host all the matrices and do all matrix operations delegated to them without knowing what for. The meaning of the matrices and their logical interrelation is only known to `GAP4`.

- Q: Do developers of packages building upon **Modules** need to know anything about the communication with the external systems?

A: No, unless they want to use more features of the external systems than those reflected by **Modules**. For this purpose, developers can use the unified communication interface provided by **HomalgToCAS**. This is the interface used by **Modules**.

1.2 This manual

Chapter 2 describes the installation of this package, while Chapter 3 provides a short quick guide to build your first own example, using the package **ExamplesForHomalg**. The remaining chapters are each devoted to one of the **homalg** objects (→ [722Moduleschapter.7-providesModules-provides](#) its constructors, properties, attributes, and operations).

Chapter 2

Installation of the Modules Package

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

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

```
LoadPackage( "Modules" );
```

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 and Markus Lange-Hegermann

Chapter 3

Quick Start

This chapter should give you a quick guide to create your first example in `homalg`.

3.1 Why are all examples in this manual over \mathbb{Z} or $\mathbb{Z}/m\mathbb{Z}$?

As the reader might notice, all examples in this manual will be either over \mathbb{Z} or over one of its residue class rings $\mathbb{Z}/m\mathbb{Z}$. There are two reasons for this. The first reason is that **GAP** does not natively support rings other than \mathbb{Z} in a *sufficient* way (\rightarrow 1.1.2).

The second and more important reason is to underline the fact the all effective homological constructions that are relevant for **Modules** have only as much to do with the Gröbnerbasis algorithm as they do with the Hermite algorithm for the ring \mathbb{Z} ; both algorithms are used to effectively solve inhomogeneous linear systems over the respective ring. And **Modules** is designed to use rings and matrices over these rings together with all their operations as a black box. In other words: Because **Modules** works for \mathbb{Z} , it works by its design for all other computable rings.

3.2 `gap> ExamplesForHomalg();`

To quickly create a ring for use with **Modules** enter

```
ExamplesForHomalg();
```

which will load the package `ExamplesForHomalg` (if installed) and provide a step by step guide to create the ring. For the full core functionality you need to install the packages `homalg`, `HomalgToCAS`, `IO_ForHomalg`, `RingsForHomalg`, `Gauss`, and `GaussForHomalg`. They are part of the `homalg` project.

3.3 A typical example

3.3.1 `HomHom`

The following example is taken from Section 2 of [BR06].

The computation takes place over the residue class ring $R = \mathbb{Z}/2^8\mathbb{Z}$ using the generic support

for residue class rings provided by the subpackage `ResidueClassRingForHomalg` of the `MatricesForHomalg` package. For a native support of the rings $R = \mathbb{Z}/p^n\mathbb{Z}$ use the `GaussForHomalg` package.

Here we compute the (infinite) long exact homology sequence of the covariant functor $\text{Hom}(\text{Hom}(-, \mathbb{Z}/2^7\mathbb{Z}), \mathbb{Z}/2^4\mathbb{Z})$ (and its left derived functors) applied to the short exact sequence

$$0 \longrightarrow M_- = \mathbb{Z}/2^2\mathbb{Z} \xrightarrow{\alpha_1} M = \mathbb{Z}/2^5\mathbb{Z} \xrightarrow{\alpha_2} {}_M = \mathbb{Z}/2^3\mathbb{Z} \longrightarrow 0.$$

Example

```
gap> zz := HomalgRingOfIntegers( );
Z
gap> Display( zz );
<An internal ring>
gap> R := zz / 2^8;
Z/( 256 )
gap> Display( R );
<A residue class ring>
gap> M := LeftPresentation( [ 2^5 ], R );
<A cyclic left module presented by 1 relation for a cyclic generator>
gap> Display( M );
Z/( 256 )/< |[ 32 ]| >
gap> _M := LeftPresentation( [ 2^3 ], R );
<A cyclic left module presented by 1 relation for a cyclic generator>
gap> Display( _M );
Z/( 256 )/< |[ 8 ]| >
gap> alpha2 := HomalgMap( [ 1 ], M, _M );
<A "homomorphism" of left modules>
gap> IsMorphism( alpha2 );
true
gap> alpha2;
<A homomorphism of left modules>
gap> Display( alpha2 );
[ [ 1 ] ]

modulo [ 256 ]

the map is currently represented by the above 1 x 1 matrix
gap> M_ := Kernel( alpha2 );
<A cyclic left module presented by yet unknown relations for a cyclic generator>
gap> alpha1 := KernelEmb( alpha2 );
<A monomorphism of left modules>
gap> seq := HomalgComplex( alpha2 );
<An acyclic complex containing a single morphism of left modules at degrees
[ 0 .. 1 ]>
gap> Add( seq, alpha1 );
gap> seq;
<A sequence containing 2 morphisms of left modules at degrees [ 0 .. 2 ]>
gap> IsShortExactSequence( seq );
true
gap> seq;
<A short exact sequence containing 2 morphisms of left modules at degrees
[ 0 .. 2 ]>
```

```

gap> Display( seq );
-----
at homology degree: 2
Z/( 256 )/< |[ 4 ]| >
-----
[ [ 8 ] ]

modulo [ 256 ]

the map is currently represented by the above 1 x 1 matrix
-----v-----
at homology degree: 1
Z/( 256 )/< |[ 32 ]| >
-----
[ [ 1 ] ]

modulo [ 256 ]

the map is currently represented by the above 1 x 1 matrix
-----v-----
at homology degree: 0
Z/( 256 )/< |[ 8 ]| >
-----
gap> K := LeftPresentation( [ 2^7 ], R );
<A cyclic left module presented by 1 relation for a cyclic generator>
gap> L := RightPresentation( [ 2^4 ], R );
<A cyclic right module on a cyclic generator satisfying 1 relation>
gap> triangle := LHomHom( 4, seq, K, L, "t" );
<An exact triangle containing 3 morphisms of left complexes at degrees
[ 1, 2, 3, 1 ]>
gap> lehs := LongSequence( triangle );
<A sequence containing 14 morphisms of left modules at degrees [ 0 .. 14 ]>
gap> ByASmallerPresentation( lehs );
<A non-zero sequence containing 14 morphisms of left modules at degrees
[ 0 .. 14 ]>
gap> IsExactSequence( lehs );
false
gap> lehs;
<A non-zero left acyclic complex containing
14 morphisms of left modules at degrees [ 0 .. 14 ]>
gap> Assert( 0, IsLeftAcyclic( lehs ) );
gap> Display( lehs );
-----
at homology degree: 14
Z/( 256 )/< |[ 4 ]| >
-----
[ [ 4 ] ]

modulo [ 256 ]

the map is currently represented by the above 1 x 1 matrix
-----v-----

```

```

at homology degree: 13
Z/( 256 )/< |[ 8 ]| >
-----
[ [ 2 ] ]

modulo [ 256 ]

the map is currently represented by the above 1 x 1 matrix
-----v-----
at homology degree: 12
Z/( 256 )/< |[ 8 ]| >
-----
[ [ 2 ] ]

modulo [ 256 ]

the map is currently represented by the above 1 x 1 matrix
-----v-----
at homology degree: 11
Z/( 256 )/< |[ 4 ]| >
-----
[ [ 4 ] ]

modulo [ 256 ]

the map is currently represented by the above 1 x 1 matrix
-----v-----
at homology degree: 10
Z/( 256 )/< |[ 8 ]| >
-----
[ [ 2 ] ]

modulo [ 256 ]

the map is currently represented by the above 1 x 1 matrix
-----v-----
at homology degree: 9
Z/( 256 )/< |[ 8 ]| >
-----
[ [ 2 ] ]

modulo [ 256 ]

the map is currently represented by the above 1 x 1 matrix
-----v-----
at homology degree: 8
Z/( 256 )/< |[ 4 ]| >
-----
[ [ 4 ] ]

modulo [ 256 ]

```

```

the map is currently represented by the above 1 x 1 matrix
-----v-----
at homology degree: 7
Z/( 256 )/< |[ 8 ]| >
-----
[ [ 2 ] ]

modulo [ 256 ]

the map is currently represented by the above 1 x 1 matrix
-----v-----
at homology degree: 6
Z/( 256 )/< |[ 8 ]| >
-----
[ [ 2 ] ]

modulo [ 256 ]

the map is currently represented by the above 1 x 1 matrix
-----v-----
at homology degree: 5
Z/( 256 )/< |[ 4 ]| >
-----
[ [ 4 ] ]

modulo [ 256 ]

the map is currently represented by the above 1 x 1 matrix
-----v-----
at homology degree: 4
Z/( 256 )/< |[ 8 ]| >
-----
[ [ 2 ] ]

modulo [ 256 ]

the map is currently represented by the above 1 x 1 matrix
-----v-----
at homology degree: 3
Z/( 256 )/< |[ 8 ]| >
-----
[ [ 2 ] ]

modulo [ 256 ]

the map is currently represented by the above 1 x 1 matrix
-----v-----
at homology degree: 2
Z/( 256 )/< |[ 4 ]| >
-----
[ [ 8 ] ]

```



```
modulo [ 256 ]

the map is currently represented by the above 1 x 1 matrix
-----v-----
at homology degree: 1
Z/( 256 )/< |[ 16 ]| >
-----
[ [ 1 ] ]

modulo [ 256 ]

the map is currently represented by the above 1 x 1 matrix
-----v-----
at homology degree: 0
Z/( 256 )/< |[ 8 ]| >
-----
```

Chapter 4

Ring Maps

4.1 Ring Maps: Attributes

4.1.1 KernelSubobject (for ring maps)

▷ `KernelSubobject(ϕ)` (attribute)
Returns: a `homalg` submodule
The kernel ideal of the ring map ϕ .

4.1.2 KernelEmb (for ring maps)

▷ `KernelEmb(ϕ)` (attribute)
Returns: a `homalg` map
The embedding of the kernel ideal `Kernel(ϕ)` into the `Source(ϕ)`, both viewed as modules over the ring $R := \text{Source}(\phi)$ (cf. `Kernel` (4.2.1)).

4.2 Ring Maps: Operations and Functions

4.2.1 Kernel (for ring maps)

▷ `Kernel(ϕ)` (method)
Returns: a `homalg` module
The kernel ideal of the ring map ϕ as an abstract module.

Chapter 5

Relations

A finite presentation of a module is given by a finite set of generators and a finite set of relations among these generators. In `homalg` a set of relations of a left/right module is given by a matrix `rel`, the rows/columns of which are interpreted as relations among n generators, n being the number of columns/rows of the matrix `rel`.

The data structure of a module in `homalg` is designed to contain not only one but several sets of relations (together with corresponding sets of generators (\rightarrow Chapter 6)). The different sets of relations are linked with so-called transition matrices (\rightarrow Chapter 7).

The relations of a `homalg` module are evaluated in a lazy way. This avoids unnecessary computations.

5.1 Relations: Categories and Representations

5.1.1 IsHomalgRelations

▷ `IsHomalgRelations(rel)` (Category)
Returns: true or false
The GAP category of `homalg` relations.

5.1.2 IsHomalgRelationsOfLeftModule

▷ `IsHomalgRelationsOfLeftModule(rel)` (Category)
Returns: true or false
The GAP category of `homalg` relations of a left module.
(It is a subcategory of the GAP category `IsHomalgRelations`.)

5.1.3 IsHomalgRelationsOfRightModule

▷ `IsHomalgRelationsOfRightModule(rel)` (Category)
Returns: true or false
The GAP category of `homalg` relations of a right module.
(It is a subcategory of the GAP category `IsHomalgRelations`.)

5.1.4 IsRelationsOfFinitelyPresentedModuleRep

▷ `IsRelationsOfFinitelyPresentedModuleRep(rel)` (Representation)

Returns: true or false

The GAP representation of a finite set of relations of a finitely presented homalg module.

(It is a representation of the GAP category `IsHomalgRelations` (5.1.1))

5.2 Relations: Constructors

5.3 Relations: Properties

5.3.1 CanBeUsedToDecideZeroEffectively

▷ `CanBeUsedToDecideZeroEffectively(rel)` (property)

Returns: true or false

Check if the homalg set of relations *rel* can be used for normal form reductions.

(no method installed)

5.3.2 IsInjectivePresentation

▷ `IsInjectivePresentation(rel)` (property)

Returns: true or false

Check if the homalg set of relations *rel* has zero syzygies.

5.4 Relations: Attributes

5.5 Relations: Operations and Functions

Chapter 6

Generators

To present a left/right module it suffices to take a matrix *rel* and interpret its rows/columns as relations among *n abstract* generators, where *n* is the number of columns/rows of *rel*. Only that these abstract generators are useless when it comes to specific modules like modules of homomorphisms, where one expects the generators to be maps between modules. For this reason a presentation of a module in *homalg* is not merely a matrix of relations, but together with a set of generators.

In *homalg* a set of generators of a left/right module is given by a matrix *gen* with rows/columns being interpreted as the generators.

The data structure of a module in *homalg* is designed to contain not only one but several sets of generators (together with their sets of relations (\rightarrow Chapter 5)). The different sets of generators are linked with so-called transition matrices (\rightarrow Chapter 7).

6.1 Generators: Categories and Representations

6.1.1 IsHomalgGenerators

▷ `IsHomalgGenerators(rel)` (Category)
Returns: true or false
The GAP category of *homalg* generators.

6.1.2 IsHomalgGeneratorsOfLeftModule

▷ `IsHomalgGeneratorsOfLeftModule(rel)` (Category)
Returns: true or false
The GAP category of *homalg* generators of a left module.
(It is a subcategory of the GAP category `IsHomalgGenerators`.)

6.1.3 IsHomalgGeneratorsOfRightModule

▷ `IsHomalgGeneratorsOfRightModule(rel)` (Category)
Returns: true or false
The GAP category of *homalg* generators of a right module.
(It is a subcategory of the GAP category `IsHomalgGenerators`.)

6.1.4 IsGeneratorsOfModuleRep

▷ `IsGeneratorsOfModuleRep(rel)` (Representation)

Returns: true or false

The GAP representation of a finite set of generators of a `homalg` module.

(It is a representation of the GAP category `IsHomalgGenerators` (6.1.1))

```
Code
DeclareRepresentation( "IsGeneratorsOfModuleRep",
  IsHomalgGenerators,
  [ "generators" ] );
```

6.1.5 IsGeneratorsOfFinitelyGeneratedModuleRep

▷ `IsGeneratorsOfFinitelyGeneratedModuleRep(rel)` (Representation)

Returns: true or false

The GAP representation of a finite set of generators of a finitely generated `homalg` module.

(It is a representation of the GAP representation `IsGeneratorsOfModuleRep` (6.1.4))

```
Code
DeclareRepresentation( "IsGeneratorsOfFinitelyGeneratedModuleRep",
  IsGeneratorsOfModuleRep,
  [ "generators", "relations_of_hullmodule" ] );
```

6.2 Generators: Constructors

6.3 Generators: Properties

6.3.1 IsReduced (for generators)

▷ `IsReduced(gen)` (property)

Returns: true or false

Check if the `homalg` set of generators *gen* is marked reduced.

(no method installed)

6.4 Generators: Attributes

6.4.1 ProcedureToReadjustGenerators

▷ `ProcedureToReadjustGenerators(gen)` (attribute)

Returns: a function

A function that takes the rows/columns of *gen* and returns an object (e.g. a matrix) that can be interpreted as a generator (this is important for modules of homomorphisms).

6.5 Generators: Operations and Functions

Chapter 7

Modules

A `homalg` module is a data structure for a finitely presented module. A presentation is given by a set of generators and a set of relations among these generators. The data structure for modules in `homalg` has two novel features:

- The data structure allows several presentations linked with so-called transition matrices. One of the presentations is marked as the default presentation, which is usually the last added one. A new presentation can always be added provided it is linked to the default presentation by a transition matrix. If needed, the user can reset the default presentation by choosing one of the other presentations saved in the data structure of the `homalg` module. Effectively, a module is then given by “all” its presentations (as “coordinates”) together with isomorphisms between them (as “coordinate changes”). Being able to “change coordinates” makes the realization of a module in `homalg` *intrinsic* (or “coordinate free”).
- To present a left/right module it suffices to take a matrix M and interpret its rows/columns as relations among n *abstract* generators, where n is the number of columns/rows of M . Only that these abstract generators are useless when it comes to specific modules like modules of homomorphisms, where one expects the generators to be maps between modules. For this reason a presentation of a module in `homalg` is not merely a matrix of relations, but together with a set of generators.

7.1 Modules: Category and Representations

7.1.1 IsHomalgModule

▷ `IsHomalgModule(M)` (Category)

Returns: true or false

The GAP category of `homalg` modules.

(It is a subcategory of the GAP categories `IsHomalgSemiringOrModule` and `IsHomalgStaticObject`.)

Code

```
DeclareCategory( "IsHomalgModule",
  IsHomalgSemiringOrModule and
  IsHomalgModuleOrMap and
  IsHomalgStaticObject );
```

7.1.2 IsFinitelyPresentedModuleOrSubmoduleRep

▷ IsFinitelyPresentedModuleOrSubmoduleRep(M) (Representation)

Returns: true or false

The GAP representation of finitely presented homalg modules or submodules.

(It is a representation of the GAP category IsHomalgModule (7.1.1), which is a subrepresentation of the GAP representations IsStaticFinitelyPresentedObjectOrSubobjectRep.)

Code

```
DeclareRepresentation( "IsFinitelyPresentedModuleOrSubmoduleRep",
  IsHomalgModule and
  IsStaticFinitelyPresentedObjectOrSubobjectRep,
  [ ] );
```

7.1.3 IsFinitelyPresentedModuleRep

▷ IsFinitelyPresentedModuleRep(M) (Representation)

Returns: true or false

The GAP representation of finitely presented homalg modules.

(It is a representation of the GAP category IsHomalgModule (7.1.1), which is a subrepresentation of the GAP representations IsFinitelyPresentedModuleOrSubmoduleRep, IsStaticFinitelyPresentedObjectRep, and IsHomalgSemiringOrFinitelyPresentedModuleRep.)

Code

```
DeclareRepresentation( "IsFinitelyPresentedModuleRep",
  IsFinitelyPresentedModuleOrSubmoduleRep and
  IsStaticFinitelyPresentedObjectRep and
  IsHomalgSemiringOrFinitelyPresentedModuleRep,
  [ "SetsOfGenerators", "SetsOfRelations",
    "PresentationMorphisms",
    "Resolutions",
    "TransitionMatrices",
    "PositionOfTheDefaultPresentation" ] );
```

7.1.4 IsFinitelyPresentedSubmoduleRep

▷ IsFinitelyPresentedSubmoduleRep(M) (Representation)

Returns: true or false

The GAP representation of finitely generated homalg submodules.

(It is a representation of the GAP category IsHomalgModule (7.1.1), which is a subrepresentation of the GAP representations IsFinitelyPresentedModuleOrSubmoduleRep, IsStaticFinitelyPresentedSubobjectRep, and IsHomalgSemiringOrFinitelyPresentedModuleRep.)

Code

```
DeclareRepresentation( "IsFinitelyPresentedSubmoduleRep",
  IsFinitelyPresentedModuleOrSubmoduleRep and
  IsStaticFinitelyPresentedSubobjectRep and
  IsHomalgSemiringOrFinitelyPresentedModuleRep,
  [ "map_having_subobject_as_its_image" ] );
```


7.2 Modules: Constructors

7.2.1 LeftPresentation (constructor for left modules)

▷ `LeftPresentation(mat)` (operation)

Returns: a `homalg` module

This constructor returns the finitely presented left module with relations given by the rows of the `homalg` matrix `mat`.

Example

```
gap> zz := HomalgRingOfIntegers( );;
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> Display( M );
[ [ 2, 3, 4 ],
  [ 5, 6, 7 ] ]

Cokernel of the map

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

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

7.2.2 RightPresentation (constructor for right modules)

▷ `RightPresentation(mat)` (operation)

Returns: a `homalg` module

This constructor returns the finitely presented right module with relations given by the columns of the `homalg` matrix `mat`.

Example

```
gap> zz := HomalgRingOfIntegers( );;
gap> M := HomalgMatrix( "[ \
> 2, 3, 4, \
> 5, 6, 7 \
> ]", 2, 3, zz );
<A 2 x 3 matrix over an internal ring>
gap> M := RightPresentation( M );
<A right module on 2 generators satisfying 3 relations>
gap> ByASmallerPresentation( M );
<A cyclic torsion right module on a cyclic generator satisfying 1 relation>
gap> Display( last );
Z/< 3 >
```

7.2.3 HomalgFreeLeftModule (constructor for free left modules)

▷ HomalgFreeLeftModule(r , R) (operation)

Returns: a homalg module

This constructor returns a free left module of rank r over the homalg ring R .

Example

```
gap> zz := HomalgRingOfIntegers( );;
gap> F := HomalgFreeLeftModule( 1, zz );
<A free left module of rank 1 on a free generator>
gap> 1 * zz;
<The free left module of rank 1 on a free generator>
gap> F := HomalgFreeLeftModule( 2, zz );
<A free left module of rank 2 on free generators>
gap> 2 * zz;
<A free left module of rank 2 on free generators>
```

7.2.4 HomalgFreeRightModule (constructor for free right modules)

▷ HomalgFreeRightModule(r , R) (operation)

Returns: a homalg module

This constructor returns a free right module of rank r over the homalg ring R .

Example

```
gap> zz := HomalgRingOfIntegers( );;
gap> F := HomalgFreeRightModule( 1, zz );
<A free right module of rank 1 on a free generator>
gap> zz * 1;
<The free right module of rank 1 on a free generator>
gap> F := HomalgFreeRightModule( 2, zz );
<A free right module of rank 2 on free generators>
gap> zz * 2;
<A free right module of rank 2 on free generators>
```

7.2.5 HomalgZeroLeftModule (constructor for zero left modules)

▷ HomalgZeroLeftModule(r , R) (operation)

Returns: a homalg module

This constructor returns a zero left module of rank r over the homalg ring R .

Example

```
gap> zz := HomalgRingOfIntegers( );;
gap> F := HomalgZeroLeftModule( zz );
<A zero left module>
gap> 0 * zz;
<The zero left module>
```

7.2.6 HomalgZeroRightModule (constructor for zero right modules)

▷ HomalgZeroRightModule(r , R) (operation)

Returns: a homalg module

This constructor returns a zero right module of rank r over the homalg ring R .

Example

```
gap> zz := HomalgRingOfIntegers( );
gap> F := HomalgZeroRightModule( zz );
<A zero right module>
gap> zz * 0;
<The zero right module>
```

7.2.7 * (transfer a module over a different ring)

- ▷ $\backslash*(R, M)$ (operation)
 ▷ $\backslash*(M, R)$ (operation)

Returns: a homalg module

Transfers the S -module M over the homalg ring R . This works only in three cases:

1. S is a subring of R .
2. R is a residue class ring of S constructed using $/$.
3. R is a subring of S and the entries of the current matrix of S -relations of M lie in R .

CAUTION: So it is not suited for general base change.

Example

```
gap> zz := HomalgRingOfIntegers( );
Z
gap> Display( zz );
<An internal ring>
gap> Z4 := zz / 4;
Z/( 4 )
gap> Display( Z4 );
<A residue class ring>
gap> M := HomalgDiagonalMatrix( [ 2 .. 4 ], zz );
<An unevaluated diagonal 3 x 3 matrix over an internal ring>
gap> M := LeftPresentation( M );
<A torsion left module presented by 3 relations for 3 generators>
gap> Display( M );
Z/< 2 > + Z/< 3 > + Z/< 4 >
gap> M;
<A torsion left module presented by 3 relations for 3 generators>
gap> N := Z4 * M; ## or N := M * Z4;
<A non-torsion left module presented by 2 relations for 3 generators>
gap> ByASmallerPresentation( N );
<A non-torsion left module presented by 1 relation for 2 generators>
gap> Display( N );
Z/( 4 )/< |[ 2 ]| > + Z/( 4 )^(1 x 1)
gap> N;
<A non-torsion left module presented by 1 relation for 2 generators>
```

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> Z4 := zz / 4;
Z/( 4 )
gap> Display( Z4 );
<A residue class ring>
gap> M4 := Z4 * M;
<A non-torsion left module presented by 2 relations for 3 generators>
gap> Display( M4 );
[ [ 2, 3, 4 ],
  [ 5, 6, 7 ] ]

modulo [ 4 ]

Cokernel of the map

Z/( 4 )^(1x2) --> Z/( 4 )^(1x3),

currently represented by the above matrix
gap> d := Resolution( 2, M4 );
<A right acyclic complex containing 2 morphisms of left modules at degrees
[ 0 .. 2 ]>
gap> dd := Hom( d, Z4 );
<A cocomplex containing 2 morphisms of right modules at degrees [ 0 .. 2 ]>
gap> DD := Resolution( 2, dd );
<A cocomplex containing 2 morphisms of right complexes at degrees [ 0 .. 2 ]>
gap> D := Hom( DD, Z4 );
<A complex containing 2 morphisms of left cocomplexes at degrees [ 0 .. 2 ]>
gap> C := zz * D;
<A "complex" containing 2 morphisms of left cocomplexes at degrees [ 0 .. 2 ]>
gap> LowestDegreeObject( C );
<A "cocomplex" containing 2 morphisms of left modules at degrees [ 0 .. 2 ]>
gap> Display( last );
-----
at cohomology degree: 2
0
-----^-----
(an empty 1 x 0 matrix)

the map is currently represented by the above 1 x 0 matrix
-----
at cohomology degree: 1
Z/< 4 >
-----^-----
[ [ 0 ],
  [ 1 ],
  [ 2 ],
  [ 1 ] ]

the map is currently represented by the above 4 x 1 matrix

```

```
-----
at cohomology degree: 0
Z/< 4 > + Z/< 4 > + Z/< 4 > + Z/< 4 >
-----
```

7.2.8 Subobject (constructor for submodules using matrices)

▷ `Subobject(mat, M)` (operation)

Returns: a `homalg` submodule

This constructor returns the finitely generated left/right submodule of the `homalg` module M with generators given by the rows/columns of the `homalg` matrix mat .

7.2.9 Subobject (constructor for submodules using a list of ring elements)

▷ `Subobject(gens, M)` (operation)

Returns: a `homalg` submodule

This constructor returns the finitely generated left/right submodule of the `homalg` cyclic left/right module M with generators given by the entries of the list $gens$.

7.2.10 LeftSubmodule (constructor for left submodules)

▷ `LeftSubmodule(mat)` (operation)

Returns: a `homalg` submodule

This constructor returns the finitely generated left submodule with generators given by the rows of the `homalg` matrix mat .

```
----- Code -----
InstallMethod( LeftSubmodule,
               "constructor for homalg submodules",
               [ IsHomalgMatrix ],

               function( gen )
                 local R;

                 R := HomalgRing( gen );

                 return Subobject( gen, NumberColumns( gen ) * R );

               end );
```

```
----- Example -----
gap> Z4 := HomalgRingOfIntegers( ) / 4;
Z/( 4 )
gap> I := HomalgMatrix( "[ 2 ]", 1, 1, Z4 );
<A 1 x 1 matrix over a residue class ring>
gap> I := LeftSubmodule( I );
<A principal torsion-free (left) ideal given by a cyclic generator>
gap> IsFree( I );
false
gap> I;
<A principal reflexive non-projective (left) ideal given by a cyclic generator\
>
```

7.2.11 RightSubmodule (constructor for right submodules)

▷ `RightSubmodule(mat)` (operation)

Returns: a `homalg` submodule

This constructor returns the finitely generated right submodule with generators given by the columns of the `homalg` matrix `mat`.

```

Code
InstallMethod( RightSubmodule,
               "constructor for homalg submodules",
               [ IsHomalgMatrix ],

               function( gen )
                 local R;

                 R := HomalgRing( gen );

                 return Subobject( gen, R * NumberRows( gen ) );

               end );

```

```

Example
gap> Z4 := HomalgRingOfIntegers( ) / 4;
Z/( 4 )
gap> I := HomalgMatrix( "[ 2 ]", 1, 1, Z4 );
<A 1 x 1 matrix over a residue class ring>
gap> I := RightSubmodule( I );
<A principal torsion-free (right) ideal given by a cyclic generator>
gap> IsFree( I );
false
gap> I;
<A principal reflexive non-projective (right) ideal given by a cyclic generato\
r>

```

7.3 Modules: Properties

7.3.1 IsCyclic

▷ `IsCyclic(M)` (property)

Returns: true or false

Check if the `homalg` module M is cyclic.

7.3.2 IsHolonomic

▷ `IsHolonomic(M)` (property)

Returns: true or false

Check if the `homalg` module M is holonomic.

7.3.3 IsReduced (for modules)

▷ `IsReduced(M)` (property)

Returns: true or false

Check if the `homalg` module M is reduced.

7.3.4 IsPrimeIdeal

- ▷ `IsPrimeIdeal(J)` (property)
Returns: `true` or `false`
 Check if the `homalg` submodule J is a prime ideal. The ring has to be commutative.
 (no method installed)

7.3.5 IsPrimeModule (for modules)

- ▷ `IsPrimeModule(M)` (property)
Returns: `true` or `false`
 Check if the `homalg` module M is prime.
 For more properties see the corresponding section (**homalg: Objects: Properties**) in the documentation of the `homalg` package.

7.4 Modules: Attributes

7.4.1 ResidueClassRing

- ▷ `ResidueClassRing(J)` (attribute)
Returns: a `homalg` ring
 In case J was defined as a (left/right) ideal of the ring R the residue class ring R/J is returned.

7.4.2 PrimaryDecomposition

- ▷ `PrimaryDecomposition(J)` (attribute)
Returns: a list
 The primary decomposition of the ideal J . The ring has to be commutative.
 (no method installed)

7.4.3 RadicalDecomposition

- ▷ `RadicalDecomposition(J)` (attribute)
Returns: a list
 The prime decomposition of the radical of the ideal J . The ring has to be commutative.
 (no method installed)

7.4.4 ModuleOfKaehlerDifferentials

- ▷ `ModuleOfKaehlerDifferentials(R)` (attribute)
Returns: a `homalg` module
 The module of Kaehler differentials of the (residue class ring) R .
 (method installed in package `GradedModules`)

7.4.5 RadicalSubobject

- ▷ `RadicalSubobject(M)` (property)
Returns: a function
 M is a homalg module.

7.4.6 SymmetricAlgebra

- ▷ `SymmetricAlgebra(M)` (attribute)
Returns: a homalg ring
The symmetric algebra of the module M .

7.4.7 ExteriorAlgebra

- ▷ `ExteriorAlgebra(M)` (attribute)
Returns: a homalg ring
The exterior algebra of the module M .

7.4.8 ElementaryDivisors

- ▷ `ElementaryDivisors(M)` (attribute)
Returns: a list of ring elements
The list of elementary divisors of the homalg module M , in case they exist.
(no method installed)

7.4.9 FittingIdeal

- ▷ `FittingIdeal(M)` (attribute)
Returns: a list
The Fitting ideal of M .

7.4.10 NonFlatLocus

- ▷ `NonFlatLocus(M)` (attribute)
Returns: a list
The non flat locus of M .

7.4.11 LargestMinimalNumberOfLocalGenerators

- ▷ `LargestMinimalNumberOfLocalGenerators(M)` (attribute)
Returns: a nonnegative integer
The minimal number of *local* generators of the module M .

7.4.12 CoefficientsOfUnreducedNumeratorOfHilbertPoincareSeries

- ▷ `CoefficientsOfUnreducedNumeratorOfHilbertPoincareSeries(M)` (attribute)
Returns: a list of integers
 M is a homalg module.

7.4.13 CoefficientsOfNumeratorOfHilbertPoincareSeries

- ▷ `CoefficientsOfNumeratorOfHilbertPoincareSeries(M)` (attribute)
Returns: a list of integers
 M is a `homalg` module.

7.4.14 UnreducedNumeratorOfHilbertPoincareSeries

- ▷ `UnreducedNumeratorOfHilbertPoincareSeries(M)` (attribute)
Returns: a univariate polynomial with rational coefficients
 M is a `homalg` module.

7.4.15 NumeratorOfHilbertPoincareSeries

- ▷ `NumeratorOfHilbertPoincareSeries(M)` (attribute)
Returns: a univariate polynomial with rational coefficients
 M is a `homalg` module.

7.4.16 HilbertPoincareSeries

- ▷ `HilbertPoincareSeries(M)` (attribute)
Returns: a univariate rational function with rational coefficients
 M is a `homalg` module.

7.4.17 AffineDegree

- ▷ `AffineDegree(M)` (attribute)
Returns: a nonnegative integer
 M is a `homalg` module.

7.4.18 DataOfHilbertFunction

- ▷ `DataOfHilbertFunction(M)` (property)
Returns: a function
 M is a `homalg` module.

7.4.19 HilbertFunction

- ▷ `HilbertFunction(M)` (property)
Returns: a function
 M is a `homalg` module.

7.4.20 IndexOfRegularity

- ▷ `IndexOfRegularity(M)` (property)
Returns: a function
 M is a `homalg` module.

For more attributes see the corresponding section (**homalg: Objects: Attributes**) in the documentation of the `homalg` package.

7.5 Modules: Operations and Functions

7.5.1 HomalgRing (for modules)

- ▷ `HomalgRing(M)` (operation)
Returns: a homalg ring
 The homalg ring of the homalg module M .

Example

```
gap> zz := HomalgRingOfIntegers( );
Z
gap> M := zz * 4;
<A free right module of rank 4 on free generators>
gap> R := HomalgRing( M );
Z
gap> IsIdenticalObj( R, zz );
true
```

7.5.2 ByASmallerPresentation (for modules)

- ▷ `ByASmallerPresentation(M)` (method)
Returns: a homalg module
 Use different strategies to reduce the presentation of the given homalg module M . This method performs side effects on its argument M and returns it.

Example

```
gap> zz := HomalgRingOfIntegers( );
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> Display( M );
[ [ 2, 3, 4 ],
  [ 5, 6, 7 ] ]

Cokernel of the map

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

currently represented by the above matrix
gap> ByASmallerPresentation( M );
<A rank 1 left module presented by 1 relation for 2 generators>
gap> Display( last );
Z/< 3 > + Z^(1 x 1)
gap> SetsOfGenerators( M );
<A set containing 2 sets of generators of a homalg module>
gap> SetsOfRelations( M );
<A set containing 2 sets of relations of a homalg module>
gap> M;
<A rank 1 left module presented by 1 relation for 2 generators>
gap> SetPositionOfTheDefaultPresentation( M, 1 );
```

```
gap> M;
<A rank 1 left module presented by 2 relations for 3 generators>
```

7.5.3 $\backslash*$ (constructor for ideal multiples)

- ▷ $\backslash*(J, M)$ (operation)
Returns: a homalg submodule
 Compute the submodule JM (resp. MJ) of the given left (resp. right) R -module M , where J is a left (resp. right) ideal in R .

7.5.4 SubobjectQuotient (for submodules)

- ▷ $\text{SubobjectQuotient}(K, J)$ (operation)
Returns: a homalg ideal
 Compute the submodule quotient ideal $K : J$ of the submodules K and J of a common R -module M .

Chapter 8

Maps

A `homalg` map is a data structures for maps (module homomorphisms) between finitely generated modules. Each map in `homalg` knows its source (\rightarrow `Source` (**homalg: Source**)) and its target (\rightarrow `Range` (**homalg: Range**)). A map is represented by a `homalg` matrix relative to the current set of generators of the source resp. target `homalg` module. As with modules (\rightarrow Chapter 7), maps in `homalg` are realized in an intrinsic manner: If the presentations of the source or/and target module are altered after the map was constructed, a new adapted representation matrix of the map is automatically computed whenever needed. For this the internal transition matrices of the modules are used. `homalg` uses the so-called *associative* convention for maps. This means that maps of left modules are applied from the right, whereas maps of right modules from the left.

8.1 Maps: Categories and Representations

8.1.1 IsHomalgMap

▷ `IsHomalgMap(phi)` (Category)
Returns: true or false
The GAP category of `homalg` maps.
(It is a subcategory of the GAP categories `IsHomalgModuleOrMap` and `IsHomalgStaticMorphism`.)

Code

```
DeclareCategory( "IsHomalgMap",  
                IsHomalgModuleOrMap and  
                IsHomalgStaticMorphism );
```

8.1.2 IsHomalgSelfMap

▷ `IsHomalgSelfMap(phi)` (Category)
Returns: true or false
The GAP category of `homalg` self-maps.
(It is a subcategory of the GAP categories `IsHomalgMap` and `IsHomalgEndomorphism`.)

Code

```
DeclareCategory( "IsHomalgSelfMap",  
                IsHomalgMap and  
                IsHomalgEndomorphism );
```

8.1.3 IsMapOfFinitelyGeneratedModulesRep

▷ IsMapOfFinitelyGeneratedModulesRep(*phi*) (Representation)

Returns: true or false

The GAP representation of maps between finitely generated homalg modules.

(It is a representation of the GAP category IsHomalgChainMorphism (**homalg: IsHomalgChainMorphism**), which is a subrepresentation of the GAP representation IsStaticMorphismOfFinitelyGeneratedObjectsRep.)

8.2 Maps: Constructors

8.2.1 HomalgMap (constructor for maps)

▷ HomalgMap(*mat*, *M*, *N*) (function)

▷ HomalgMap(*mat*[, *string*]) (function)

Returns: a homalg map

This constructor returns a map (homomorphism) of finitely presented modules. It is represented by the homalg matrix *mat* relative to the current set of generators of the source homalg module *M* and target module *N* (\rightarrow 7.2). Unless the source module is free *and* given on free generators the returned map will cautiously be indicated using parenthesis: “homomorphism”. To verify if the result is indeed a well defined map use IsMorphism (**homalg: IsMorphism**). If the presentations of the source or/and target module are altered after the map was constructed, a new adapted representation matrix of the map is automatically computed whenever needed. For this the internal transition matrices of the modules are used. If source and target are identical objects, and only then, the map is created as a selfmap (endomorphism). homalg uses the so-called *associative* convention for maps. This means that maps of left modules are applied from the right, whereas maps of right modules from the left.

— Example —

```
gap> zz := HomalgRingOfIntegers( );;
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( M );
<A rank 1 left module presented by 1 relation for 2 generators>
gap> Display( last );
Z/< 3 > + Z^(1 x 1)
gap> Display( phi );
[ [ 2, 1, 0, -1 ],
  [ 1, 0, -2, -4 ] ]
```

the map is currently represented by the above 2 x 4 matrix

```
gap> ByASmallerPresentation( N );
<A rank 2 left module presented by 1 relation for 3 generators>
gap> Display( N );
Z/< 4 > + Z^(1 x 2)
gap> Display( phi );
[ [ -8, 0, 0 ],
  [ -3, -1, -2 ] ]
```

the map is currently represented by the above 2 x 3 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

To construct a map with source being a not yet specified free module

Example

```
gap> N;
<A rank 2 left module presented by 1 relation for 3 generators>
gap> SetPositionOfTheDefaultSetOfGenerators( N, 1 );
gap> N;
<A rank 2 left module presented by 2 relations for 4 generators>
gap> psi := HomalgMap( mat, "free", N );
<A homomorphism of left modules>
gap> Source( psi );
<A free left module of rank 3 on free generators>
```

To construct a map between not yet specified free left modules

Example

```
gap> chi := HomalgMap( mat ); ## or chi := HomalgMap( mat, "1" );
<A homomorphism of left modules>
gap> Source( chi );
<A free left module of rank 3 on free generators>
gap> Range( chi );
<A free left module of rank 4 on free generators>
```

To construct a map between not yet specified free right modules

Example

```
gap> kappa := HomalgMap( mat, "r" );
<A homomorphism of right modules>
gap> Source( kappa );
<A free right module of rank 4 on free generators>
gap> Range( kappa );
<A free right module of rank 3 on free generators>
```

8.2.2 HomalgZeroMap (constructor for zero maps)

▷ HomalgZeroMap(M , N)

(function)

Returns: a homalg map

The constructor returns the zero map between the source homalg module M and the target homalg module N .

Example

```
gap> zz := HomalgRingOfIntegers( );
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> HomalgZeroMap( M, N );
<The zero morphism of left modules>
```

8.2.3 HomalgIdentityMap (constructor for identity maps)

▷ HomalgIdentityMap(M , N)

(function)

Returns: a homalg map

The constructor returns the identity map of the homalg module M .

Example

```
gap> zz := HomalgRingOfIntegers( );
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> HomalgIdentityMap( M );
<The identity morphism of a non-zero left module>
```

8.3 Maps: Properties

8.4 Maps: Attributes

8.5 Maps: Operations and Functions

8.5.1 HomalgRing

▷ `HomalgRing(ϕ)` (operation)

Returns: a `homalg` ring

The `homalg` ring of the `homalg` map ϕ .

Example

```
gap> zz := HomalgRingOfIntegers( );
Z
gap> phi := HomalgIdentityMap( 2 * zz );
<The identity morphism of a non-zero left module>
gap> R := HomalgRing( phi );
Z
gap> IsIdenticalObj( R, zz );
true
```

8.5.2 PreInverse

▷ `PreInverse(ϕ)` (operation)

Returns: a `homalg` map, `false`, or `fail`

Compute a pre-inverse of the morphism ϕ in case one exists. For a pre-inverse to exist ϕ must be an epimorphism. For *commutative* rings `homalg` has an algorithm installed which decides the existence and returns a pre-inverse in case one exists. If a pre-inverse does not exist then `false` is returned. The algorithm finds a particular solution of a two-side inhomogeneous linear system over $R := \text{HomalgRing}(\phi)$. For *noncommutative* rings a heuristic method is installed. If it finds a pre-inverse it returns it, otherwise it returns `fail` (\rightarrow [722Moduleschapter.7-limitationModules-limitation](#) operation `PreInverse` is used to install a method for the property `IsSplitEpimorphism` (**homalg: IsSplitEpimorphism**)).

`PreInverse` checks if it can decide the projectivity of `Range(ϕ)`.

Chapter 9

Module Elements

An element of a module M is internally represented by a module map from the (distinguished) rank 1 free module to the module M . In particular, the data structure for module elements automatically profits from the intrinsic realization of morphisms in the `homalg` project.

9.1 Module Elements: Category and Representations

9.1.1 IsHomalgElement

▷ `IsHomalgElement(M)` (Category)
Returns: true or false
The GAP category of module elements.

9.1.2 IsElementOfAModuleGivenByAMorphismRep

▷ `IsElementOfAModuleGivenByAMorphismRep(M)` (Representation)
Returns: true or false
The GAP representation of elements of modules.
(It is a subrepresentation of `IsElementOfAnObjectGivenByAMorphismRep` (**homalg**: `IsElementOfAnObjectGivenByAMorphismRep`).)

9.2 Module Elements: Constructors

9.3 Module Elements: Properties

9.3.1 IsElementOfIntegers

▷ `IsElementOfIntegers(m)` (property)
Returns: true or false
Check if the m is an element of the integers viewed as a module over itself.

Example

```
gap> zz := HomalgRingOfIntegers( );
Z
gap> a := HomalgElement( HomalgMap( "[[2]]", 1 * zz, 1 * zz ) );
2
```

```

gap> IsElementOfIntegers( a );
true
gap> Z4 := zz / 4;
Z/( 4 )
gap> b := HomalgElement( HomalgMap( "[[-1]]", 1 * Z4, 1 * Z4 ) );
|[ 3 ]|
gap> IsElementOfIntegers( b );
false

```

9.4 Module Elements: Attributes

9.5 Module Elements: Operations and Functions

9.5.1 HomalgRing (for module elements)

▷ `HomalgRing(m)` (operation)

Returns: a homalg ring

The homalg ring of the homalg module element *m*.

Example

```

gap> zz := HomalgRingOfIntegers( );
Z
gap> a := HomalgElement( HomalgMap( "[[2]]", 1 * zz, 1 * zz ) );
2
gap> IsIdenticalObj( zz, HomalgRing( a ) );
true

```

Chapter 10

Functors

10.1 Functors: Category and Representations

10.2 Functors: Constructors

10.3 Functors: Attributes

10.4 Basic Functors

10.4.1 functor_Cokernel

▷ functor_Cokernel (global variable)

The functor that associates to a map its cokernel.

```
Code
BindGlobal( "functor_Cokernel_for_fp_modules",
  CreateHomalgFunctor(
    [ "name", "Cokernel" ],
    [ "category", HOMALG_MODULES.category ],
    [ "operation", "Cokernel" ],
    [ "natural_transformation", "CokernelEpi" ],
    [ "special", true ],
    [ "number_of_arguments", 1 ],
    [ "1", [ [ "covariant" ],
      [ IsMapOffinitelyGeneratedModulesRep,
        [ IsHomalgChainMorphism, IsImageSquare ] ] ] ],
    [ "OnObjects", _Functor_Cokernel_OnModules ]
  )
);
```

10.4.2 Cokernel

▷ Cokernel(ϕ) (operation)

The following example also makes use of the natural transformation CokernelEpi.

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> N := HomalgMatrix( "[ 2, 3, 4, 5, 6, 7, 8, 9 ]", 2, 4, zz );
gap> N := LeftPresentation( N );
<A non-torsion left module presented by 2 relations for 4 generators>
gap> mat := HomalgMatrix( "[ \
> 1, 0, -3, -6, \
> 0, 1, 6, 11, \
> 1, 0, -3, -6 \
> ]", 3, 4, zz );
gap> phi := HomalgMap( mat, M, N );
gap> IsMorphism( phi );
true
gap> phi;
<A homomorphism of left modules>
gap> coker := Cokernel( phi );
<A left module presented by 5 relations for 4 generators>
gap> ByASmallerPresentation( coker );
<A rank 1 left module presented by 1 relation for 2 generators>
gap> Display( coker );
Z/< 8 > + Z^(1 x 1)
gap> nu := CokernelEpi( phi );
<An epimorphism of left modules>
gap> Display( nu );
[ [ -5, 0 ],
  [ -6, 1 ],
  [ 1, -2 ],
  [ 0, 1 ] ]

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

the map is currently represented by the above 3 x 2 matrix
gap> PreInverse( nu );
false

```

10.4.3 functor_ImageObject

▷ functor_ImageObject

(global variable)

The functor that associates to a map its image.

```

Code
BindGlobal( "functor_ImageObject_for_fp_modules",
  CreateHomalgFunctor(
    [ "name", "ImageObject for modules" ],
    [ "category", HOMALG_MODULES.category ],
    [ "operation", "ImageObject" ],
    [ "natural_transformation", "ImageObjectEmb" ],
    [ "number_of_arguments", 1 ],
    [ "1", [ [ "covariant" ],
      [ IsMapOfFinitelyGeneratedModulesRep and
        AdmissibleInputForHomalgFunctors ] ] ],
    [ "OnObjects", _Functor_ImageObject_OnModules ]
  )
);

```

10.4.4 ImageObject

▷ ImageObject(ϕ)

(operation)

The following example also makes use of the natural transformations ImageObjectEpi and ImageObjectEmb.

```

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> N := HomalgMatrix( "[ 2, 3, 4, 5, 6, 7, 8, 9 ]", 2, 4, zz );
gap> N := LeftPresentation( N );
<A non-torsion left module presented by 2 relations for 4 generators>
gap> mat := HomalgMatrix( "[ \
> 1, 0, -3, -6, \
> 0, 1, 6, 11, \
> 1, 0, -3, -6 \
> ]", 3, 4, zz );
gap> phi := HomalgMap( mat, M, N );
gap> IsMorphism( phi );
true
gap> phi;
<A homomorphism of left modules>
gap> im := ImageObject( phi );
<A left module presented by yet unknown relations for 3 generators>
gap> ByASmallerPresentation( im );
<A free left module of rank 1 on a free generator>
gap> pi := ImageObjectEpi( phi );
<A non-zero split epimorphism of left modules>
gap> epsilon := ImageObjectEmb( phi );
<A monomorphism of left modules>
gap> phi = pi * epsilon;
true

```

10.4.5 Kernel (for maps)

▷ `Kernel(phi)`

(operation)

The following example also makes use of the natural transformation `KernelEmb`.

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> N := HomalgMatrix( "[ 2, 3, 4, 5, 6, 7, 8, 9 ]", 2, 4, zz );
gap> N := LeftPresentation( N );
<A non-torsion left module presented by 2 relations for 4 generators>
gap> mat := HomalgMatrix( "[ \
> 1, 0, -3, -6, \
> 0, 1, 6, 11, \
> 1, 0, -3, -6 \
> ]", 3, 4, zz );
gap> phi := HomalgMap( mat, M, N );
gap> IsMorphism( phi );
true
gap> phi;
<A homomorphism of left modules>
gap> ker := Kernel( phi );
<A cyclic left module presented by yet unknown relations for a cyclic generator>
r>
gap> Display( ker );
Z/< -3 >
gap> ByASmallerPresentation( last );
<A cyclic torsion left module presented by 1 relation for a cyclic generator>
gap> Display( ker );
Z/< 3 >
gap> iota := KernelEmb( phi );
<A monomorphism of left modules>
gap> Display( iota );
[ [ 0, 1, 2 ] ]

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

the map is currently represented by the above 1 x 2 matrix
gap> PostInverse( iota );
<A non-zero split epimorphism of left modules>
```

10.4.6 DefectOfExactness

▷ DefectOfExactness(*phi*, *psi*)

(operation)

We follow the associative convention for applying maps. For left modules *phi* is applied first and from the right. For right modules *psi* is applied first and from the left.

The following example also makes use of the natural transformation KernelEmb.

Example

```
gap> zz := HomalgRingOfIntegers( );
Z
gap> M := HomalgMatrix( "[ 2, 3, 4, 0, 5, 6, 7, 0 ]", 2, 4, zz );
gap> M := LeftPresentation( M );
<A non-torsion left module presented by 2 relations for 4 generators>
gap> N := HomalgMatrix( "[ 2, 3, 4, 5, 6, 7, 8, 9 ]", 2, 4, zz );
gap> N := LeftPresentation( N );
<A non-torsion left module presented by 2 relations for 4 generators>
gap> mat := HomalgMatrix( "[ \
> 1, 3, 3, 3, \
> 0, 3, 10, 17, \
> 1, 3, 3, 3, \
> 0, 0, 0, 0 \
> ]", 4, 4, zz );
gap> phi := HomalgMap( mat, M, N );
gap> IsMorphism( phi );
true
gap> phi;
<A homomorphism of left modules>
gap> iota := KernelEmb( phi );
<A monomorphism of left modules>
gap> DefectOfExactness( iota, phi );
<A zero left module>
gap> hom_iota := Hom( iota ); ## a shorthand for Hom( iota, zz );
<A homomorphism of right modules>
gap> hom_phi := Hom( phi ); ## a shorthand for Hom( phi, zz );
<A homomorphism of right modules>
gap> DefectOfExactness( hom_iota, hom_phi );
<A cyclic right module on a cyclic generator satisfying yet unknown relations>
gap> ByASmallerPresentation( last );
<A cyclic torsion right module on a cyclic generator satisfying 1 relation>
gap> Display( last );
Z/< 2 >
```

10.4.7 Functor_Hom

▷ Functor_Hom

(global variable)

The bifunctor Hom.

Code

```
BindGlobal( "Functor_Hom_for_fp_modules",
  CreateHomalgFunction(
    [ "name", "Hom" ],
    [ "category", HOMALG_MODULES.category ],
```

```

[ "operation", "Hom" ],
[ "number_of_arguments", 2 ],
[ "1", [ [ "contravariant", "right adjoint", "distinguished" ] ] ],
[ "2", [ [ "covariant", "left exact" ] ] ],
[ "OnObjects", _Functor_Hom_OnModules ],
[ "OnMorphisms", _Functor_Hom_OnMaps ],
[ "MorphismConstructor", HOMALG_MODULES.category.MorphismConstructor ]
)
);

```

10.4.8 Hom

▷ Hom(o1, o2)

(operation)

o1 resp. o2 could be a module, a map, a complex (of modules or of again of complexes), or a chain morphism.

Each generator of a module of homomorphisms is displayed as a matrix of appropriate dimensions.

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> N := HomalgMatrix( "[ 2, 3, 4, 5, 6, 7, 8, 9 ]", 2, 4, zz );
gap> N := LeftPresentation( N );
<A non-torsion left module presented by 2 relations for 4 generators>
gap> mat := HomalgMatrix( "[ \
> 1, 0, -3, -6, \
> 0, 1, 6, 11, \
> 1, 0, -3, -6 \
> ]", 3, 4, zz );
gap> phi := HomalgMap( mat, M, N );
gap> IsMorphism( phi );
true
gap> phi;
<A homomorphism of left modules>
gap> psi := Hom( phi, M );
<A homomorphism of right modules>
gap> ByASmallerPresentation( psi );
<A non-zero homomorphism of right modules>
gap> Display( psi );
[ [ 1, 1, 1, 0 ],
  [ 2, 2, 0, 0 ],
  [ 0, 0, -2, 0 ] ]

the map is currently represented by the above 3 x 4 matrix
gap> homNM := Source( psi );
<A rank 2 right module on 4 generators satisfying 2 relations>
gap> IsIdenticalObj( homNM, Hom( N, M ) ); ## the caching at work
true
gap> homMM := Range( psi );

```



```

<A rank 1 right module on 3 generators satisfying 2 relations>
gap> IsIdenticalObj( homMM, Hom( M, M ) ); ## the caching at work
true
gap> Display( homNM );
Z/< 3 > + Z/< 3 > + Z^(2 x 1)
gap> Display( homMM );
Z/< 3 > + Z/< 3 > + Z^(1 x 1)
gap> IsMonomorphism( psi );
false
gap> IsEpimorphism( psi );
false
gap> GeneratorsOfModule( homMM );
<A set of 3 generators of a homalg right module>
gap> Display( last );
[ [ 0, 0, 0 ],
  [ 0, 1, 2 ],
  [ 0, 0, 0 ] ]

the map is currently represented by the above 3 x 3 matrix

[ [ 0, 2, 4 ],
  [ 0, 0, 0 ],
  [ 0, 2, 4 ] ]

the map is currently represented by the above 3 x 3 matrix

[ [ 0, 0, 1 ],
  [ 0, 2, 2 ],
  [ 0, 0, 1 ] ]

the map is currently represented by the above 3 x 3 matrix

a set of 3 generators given by the the above matrices
gap> GeneratorsOfModule( homNM );
<A set of 4 generators of a homalg right module>
gap> Display( last );
[ [ 0, 1, 2 ],
  [ 0, 1, 2 ],
  [ 0, 1, 2 ],
  [ 0, 0, 0 ] ]

the map is currently represented by the above 4 x 3 matrix

[ [ 0, 1, 2 ],
  [ 0, 0, 0 ],
  [ 0, 0, 0 ],
  [ 0, 2, 4 ] ]

the map is currently represented by the above 4 x 3 matrix

[ [ 0, 0, 1 ],
  [ 0, 1, -1 ],

```

```
[ 0, 1, 5 ],
[ 0, 1, 1 ] ]
```

the map is currently represented by the above 4 x 3 matrix

```
[ [ 0, 0, 0 ],
  [ 0, 0, 1 ],
  [ 0, 0, -2 ],
  [ 0, 0, 1 ] ]
```

the map is currently represented by the above 4 x 3 matrix

a set of 4 generators given by the the above matrices

If for example the source N gets a new presentation, you will see the effect on the generators:

Example

```
gap> ByASmallerPresentation( N );
<A rank 2 left module presented by 1 relation for 3 generators>
gap> GeneratorsOfModule( homNM );
<A set of 4 generators of a homalg right module>
gap> Display( last );
[ [ 0, 3, 6 ],
  [ 0, 1, 2 ],
  [ 0, 0, 0 ] ]
```

the map is currently represented by the above 3 x 3 matrix

```
[ [ 0, 9, 18 ],
  [ 0, 0, 0 ],
  [ 0, 2, 4 ] ]
```

the map is currently represented by the above 3 x 3 matrix

```
[ [ 0, 9, 18 ],
  [ 0, 1, 5 ],
  [ 0, 1, 1 ] ]
```

the map is currently represented by the above 3 x 3 matrix

```
[ [ 0, 0, 0 ],
  [ 0, 0, -2 ],
  [ 0, 0, 1 ] ]
```

the map is currently represented by the above 3 x 3 matrix

a set of 4 generators given by the the above matrices

Now we compute a certain natural filtration on $\text{Hom}(M, M)$:

Example

```
gap> dM := Resolution( M );
<A non-zero right acyclic complex containing a single morphism of left modules\
at degrees [ 0 .. 1 ]>
```

```

gap> hMM := Hom( dM, dM );
<A non-zero acyclic cocomplex containing a single morphism of right complexes \
at degrees [ 0 .. 1 ]>
gap> BMM := HomalgBicomplex( hMM );
<A non-zero bicomplex containing right modules at bidegrees [ 0 .. 1 ]x
[ -1 .. 0 ]>
gap> II_E := SecondSpectralSequenceWithFiltration( BMM );
<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 );
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 );
<A descending filtration with degrees [ -1 .. 0 ] and graded parts:

```

```

-1:  <A non-zero cyclic torsion right module on a cyclic generator satisfying
      yet unknown relations>
  0:  <A rank 1 right module on 3 generators satisfying 2 relations>
of
<A right module on 4 generators satisfying yet unknown relations>>
gap> ByASmallerPresentation( filt );
<A descending filtration with degrees [ -1 .. 0 ] and graded parts:

-1:  <A non-zero cyclic torsion right module on a cyclic generator satisfying 1\
      relation>
  0:  <A rank 1 right module on 2 generators satisfying 1 relation>
of
<A rank 1 right module on 3 generators satisfying 2 relations>>
gap> Display( filt );
Degree -1:

Z/< 3 >
-----
Degree 0:

Z/< 3 > + Z^(1 x 1)
gap> Display( homMM );
Z/< 3 > + Z/< 3 > + Z^(1 x 1)

```

10.4.9 Functor_TensorProduct

▷ Functor_TensorProduct

(global variable)

The tensor product bifunctor.

```

Code
BindGlobal( "Functor_TensorProduct_for_fp_modules",
  CreateHomalgFunctor(
    [ "name", "TensorProduct" ],
    [ "category", HOMALG_MODULES.category ],
    [ "operation", "TensorProductOp" ],
    [ "number_of_arguments", 2 ],
    [ "1", [ [ "covariant", "left adjoint", "distinguished" ] ] ],
    [ "2", [ [ "covariant", "left adjoint" ] ] ],
    [ "OnObjects", _Functor_TensorProduct_OnModules ],
    [ "OnMorphisms", _Functor_TensorProduct_OnMaps ],
    [ "MorphismConstructor", HOMALG_MODULES.category.MorphismConstructor ]
  )
);

```

10.4.10 TensorProduct

▷ TensorProduct(o1, o2)

(operation)

▷ *(o1, o2)

(operation)

$o1$ resp. $o2$ could be a module, a map, a complex (of modules or of again of complexes), or a chain morphism.

The symbol $*$ is a shorthand for several operations associated with the functor `Functor_TensorProduct_for_fp_modules` installed under the name `TensorProduct`.

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, -3, -6, \
> 0, 1, 6, 11, \
> 1, 0, -3, -6 \
> ]", 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> L := Hom( zz, M );
<A rank 1 right module on 3 generators satisfying yet unknown relations>
gap> ByASmallerPresentation( L );
<A rank 1 right module on 2 generators satisfying 1 relation>
gap> Display( L );
Z/< 3 > + Z^(1 x 1)
gap> L;
<A rank 1 right module on 2 generators satisfying 1 relation>
gap> psi := phi * L;
<A homomorphism of right modules>
gap> ByASmallerPresentation( psi );
<A non-zero homomorphism of right modules>
gap> Display( psi );
[ [ 0, 0, 1, 1 ],
  [ 0, 0, 8, 1 ],
  [ 0, 0, 0, -2 ],
  [ 0, 0, 0, 2 ] ]

the map is currently represented by the above 4 x 4 matrix
gap> ML := Source( psi );
<A rank 1 right module on 4 generators satisfying 3 relations>
gap> IsIdenticalObj( ML, M * L ); ## the caching at work
true
gap> NL := Range( psi );
<A rank 2 right module on 4 generators satisfying 2 relations>
gap> IsIdenticalObj( NL, N * L ); ## the caching at work
```

```

true
gap> Display( ML );
Z/< 3 > + Z/< 3 > + Z/< 3 > + Z^(1 x 1)
gap> Display( NL );
Z/< 3 > + Z/< 12 > + Z^(2 x 1)

```

Now we compute a certain natural filtration on the tensor product $M*L$:

Example

```

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> CE := Resolution( GP );
<An acyclic cocomplex containing a single morphism of right complexes at degree\
  es [ 0 .. 1 ]>
gap> FCE := Hom( CE, L );
<A non-zero acyclic complex containing a single morphism of left cocomplexes a\
  t degrees [ 0 .. 1 ]>
gap> BC := HomalgBicomplex( FCE );
<A non-zero bicomplex containing left modules at bidegrees [ 0 .. 1 ]x
  [ -1 .. 0 ]>
gap> II_E := SecondSpectralSequenceWithFiltration( BC );
<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 );
<An ascending filtration with degrees [ -1 .. 0 ] and graded parts:
  0:  <A rank 1 left module presented by 1 relation for 2 generators>
 -1:  <A non-zero left module presented by 2 relations for 2 generators>
of
<A non-zero left module presented by 4 relations for 4 generators>>
gap> ByASmallerPresentation( filt );
<An ascending filtration with degrees [ -1 .. 0 ] and graded parts:
  0:  <A rank 1 left module presented by 1 relation for 2 generators>
 -1:  <A non-zero torsion left module presented by 2 relations
       for 2 generators>
of
<A rank 1 left module presented by 3 relations for 4 generators>>
gap> Display( filt );
Degree 0:

Z/< 3 > + Z^(1 x 1)
-----
Degree -1:

Z/< 3 > + Z/< 3 >
gap> Display( ML );
Z/< 3 > + Z/< 3 > + Z/< 3 > + Z^(1 x 1)

```

10.4.11 Functor_Ext

▷ Functor_Ext

(global variable)

The bifunctor Ext.

Below is the only *specific* line of code used to define Functor_Ext_for_fp_modules and all the different operations Ext in homalg.

Code

```

RightSatelliteOfCofunctor( Functor_Hom_for_fp_modules, "Ext" );

```

10.4.12 Ext

▷ `Ext([c,]o1, o2[, str])` (operation)

Compute the c -th extension object of $o1$ with $o2$ where c is a nonnegative integer and $o1$ resp. $o2$ could be a module, a map, a complex (of modules or of again of complexes), or a chain morphism. If $str="a"$ then the (cohomologically) graded object $Ext^i(o1, o2)$ for $0 \leq i \leq c$ is computed. If neither c nor str is specified then the cohomologically graded object $Ext^i(o1, o2)$ for $0 \leq i \leq d$ is computed, where d is the length of the internally computed free resolution of $o1$.

Each generator of a module of extensions is displayed as a matrix of appropriate dimensions.

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> N := TorsionObject( M );
<A cyclic torsion left module presented by yet unknown relations for a cyclic \
generator>
gap> iota := TorsionObjectEmb( M );
<A monomorphism of left modules>
gap> psi := Ext( 1, iota, N );
<A homomorphism of right modules>
gap> ByASmallerPresentation( psi );
<A non-zero homomorphism of right modules>
gap> Display( psi );
[ [ 1 ] ]

the map is currently represented by the above 1 x 1 matrix
gap> extNN := Range( psi );
<A non-zero cyclic torsion right module on a cyclic generator satisfying 1 rel\
ation>
gap> IsIdenticalObj( extNN, Ext( 1, N, N ) ); ## the caching at work
true
gap> extMN := Source( psi );
<A non-zero cyclic torsion right module on a cyclic generator satisfying 1 rel\
ation>
gap> IsIdenticalObj( extMN, Ext( 1, M, N ) ); ## the caching at work
true
gap> Display( extNN );
Z/< 3 >
gap> Display( extMN );
Z/< 3 >
```

10.4.13 Functor_Tor

▷ `Functor_Tor` (global variable)

The bifunctor `Tor`.

Below is the only *specific* line of code used to define `Functor_Tor_for_fp_modules` and all the different operations `Tor` in `homalg`.

Code

```
LeftSatelliteOfFunctor( Functor_TensorProduct_for_fp_modules, "Tor" );
```

10.4.14 Tor

▷ `Tor([c,]o1, o2[, str])` (operation)

Compute the c -th torsion object of $o1$ with $o2$ where c is a nonnegative integer and $o1$ resp. $o2$ could be a module, a map, a complex (of modules or of again of complexes), or a chain morphism. If $str="a"$ then the (cohomologically) graded object $Tor_i(o1,o2)$ for $0 \leq i \leq c$ is computed. If neither c nor str is specified then the cohomologically graded object $Tor_i(o1,o2)$ for $0 \leq i \leq d$ is computed, where d is the length of the internally computed free resolution of $o1$.

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> N := TorsionObject( M );
<A cyclic torsion left module presented by yet unknown relations for a cyclic \
generator>
gap> iota := TorsionObjectEmb( M );
<A monomorphism of left modules>
gap> psi := Tor( 1, iota, N );
<A homomorphism of left modules>
gap> ByASmallerPresentation( psi );
<A non-zero homomorphism of left modules>
gap> Display( psi );
[ [ 1 ] ]

the map is currently represented by the above 1 x 1 matrix
gap> torNN := Source( psi );
<A non-zero cyclic torsion left module presented by 1 relation for a cyclic ge\
nerator>
gap> IsIdenticalObj( torNN, Tor( 1, N, N ) ); ## the caching at work
true
gap> torMN := Range( psi );
<A non-zero cyclic torsion left module presented by 1 relation for a cyclic ge\
nerator>
gap> IsIdenticalObj( torMN, Tor( 1, M, N ) ); ## the caching at work
true
gap> Display( torNN );
Z/< 3 >
gap> Display( torMN );
Z/< 3 >
```

10.4.15 Functor_RHom

▷ `Functor_RHom` (global variable)

The bifunctor RHom .

Below is the only *specific* line of code used to define `Functor_RHom_for_fp_modules` and all the different operations RHom in `homalg`.

Code

```
RightDerivedCofunctor( Functor_Hom_for_fp_modules );
```

10.4.16 RHom

▷ $\text{RHom}([c,]o1, o2[, str])$ (operation)

Compute the c -th extension object of $o1$ with $o2$ where c is a nonnegative integer and $o1$ resp. $o2$ could be a module, a map, a complex (of modules or of again of complexes), or a chain morphism. The string str may take different values:

- If $str="a"$ then $R^i\text{Hom}(o1, o2)$ for $0 \leq i \leq c$ is computed.
- If $str="c"$ then the c -th connecting homomorphism with respect to the short exact sequence $o1$ is computed.
- If $str="t"$ then the exact triangle upto cohomological degree c with respect to the short exact sequence $o1$ is computed.

If neither c nor str is specified then the cohomologically graded object $R^i\text{Hom}(o1, o2)$ for $0 \leq i \leq d$ is computed, where d is the length of the internally computed free resolution of $o1$.

Each generator of a module of derived homomorphisms is displayed as a matrix of appropriate dimensions.

Example

```
gap> zz := HomalgRingOfIntegers( );
Z
gap> m := HomalgMatrix( [ [ 8, 0 ], [ 0, 2 ] ], zz );
gap> M := LeftPresentation( m );
<A left module presented by 2 relations for 2 generators>
gap> Display( M );
Z/< 8 > + Z/< 2 >
gap> M;
<A torsion left module presented by 2 relations for 2 generators>
gap> a := HomalgMatrix( [ [ 2, 0 ] ], zz );
gap> alpha := HomalgMap( a, "free", M );
<A homomorphism of left modules>
gap> pi := CokernelEpi( alpha );
<An epimorphism of left modules>
gap> Display( pi );
[ [ 1, 0 ],
  [ 0, 1 ] ]

the map is currently represented by the above 2 x 2 matrix
gap> iota := KernelEmb( pi );
<A monomorphism of left modules>
gap> Display( iota );
[ [ 2, 0 ] ]
```

```

the map is currently represented by the above 1 x 2 matrix
gap> N := Kernel( pi );
<A cyclic torsion left module presented by yet unknown relations for a cyclic \
generator>
gap> Display( N );
Z/< 4 >
gap> C := HomalgComplex( pi );
<A left acyclic complex containing a single morphism of left modules at degree\
s [ 0 .. 1 ]>
gap> Add( C, iota );
gap> ByASmallerPresentation( C );
<A non-zero short exact sequence containing
2 morphisms of left modules at degrees [ 0 .. 2 ]>
gap> Display( C );
-----
at homology degree: 2
Z/< 4 >
-----
[ [ 0, 6 ] ]

the map is currently represented by the above 1 x 2 matrix
-----v-----
at homology degree: 1
Z/< 2 > + Z/< 8 >
-----
[ [ 0, 1 ],
  [ 1, 1 ] ]

the map is currently represented by the above 2 x 2 matrix
-----v-----
at homology degree: 0
Z/< 2 > + Z/< 2 >
-----
gap> T := RHom( C, N );
<An exact cotriangle containing 3 morphisms of right cocomplexes at degrees
[ 0, 1, 2, 0 ]>
gap> ByASmallerPresentation( T );
<A non-zero exact cotriangle containing
3 morphisms of right cocomplexes at degrees [ 0, 1, 2, 0 ]>
gap> L := LongSequence( T );
<A cosequence containing 5 morphisms of right modules at degrees [ 0 .. 5 ]>
gap> Display( L );
-----
at cohomology degree: 5
Z/< 4 >
-----^-----
[ [ 0, 3 ] ]

the map is currently represented by the above 1 x 2 matrix
-----
at cohomology degree: 4
Z/< 2 > + Z/< 4 >

```

```

-----^-----
[ [ 0, 1 ],
  [ 0, 0 ] ]

the map is currently represented by the above 2 x 2 matrix
-----
at cohomology degree: 3
Z/< 2 > + Z/< 2 >
-----^-----
[ [ 1 ],
  [ 0 ] ]

the map is currently represented by the above 2 x 1 matrix
-----
at cohomology degree: 2
Z/< 4 >
-----^-----
[ [ 0, 2 ] ]

the map is currently represented by the above 1 x 2 matrix
-----
at cohomology degree: 1
Z/< 2 > + Z/< 4 >
-----^-----
[ [ 0, 1 ],
  [ 2, 2 ] ]

the map is currently represented by the above 2 x 2 matrix
-----
at cohomology degree: 0
Z/< 2 > + Z/< 2 >
-----
gap> IsExactSequence( L );
true
gap> L;
<An exact cosequence containing 5 morphisms of right modules at degrees
[ 0 .. 5 ]>

```

10.4.17 Functor_LTensorProduct

▷ Functor_LTensorProduct

(global variable)

The bifunctor LTensorProduct.

Below is the only *specific* line of code used to define Functor_LTensorProduct_for_fp_modules and all the different operations LTensorProduct in homalg.

Code

```

LeftDerivedFunctor( Functor_TensorProduct_for_fp_modules );

```

10.4.18 LTensorProduct

▷ `LTensorProduct([c,]o1, o2[, str])`

(operation)

Compute the c -th torsion object of $o1$ with $o2$ where c is a nonnegative integer and $o1$ resp. $o2$ could be a module, a map, a complex (of modules or of again of complexes), or a chain morphism. The string str may take different values:

- If $str="a"$ then $L_iTensorProduct(o1,o2)$ for $0 \leq i \leq c$ is computed.
- If $str="c"$ then the c -th connecting homomorphism with respect to the short exact sequence $o1$ is computed.
- If $str="t"$ then the exact triangle upto cohomological degree c with respect to the short exact sequence $o1$ is computed.

If neither c nor str is specified then the cohomologically graded object $L_iTensorProduct(o1,o2)$ for $0 \leq i \leq d$ is computed, where d is the length of the internally computed free resolution of $o1$.

Each generator of a module of derived homomorphisms is displayed as a matrix of appropriate dimensions.

Example

```
gap> zz := HomalgRingOfIntegers( );
Z
gap> m := HomalgMatrix( [ [ 8, 0 ], [ 0, 2 ] ], zz );
gap> M := LeftPresentation( m );
<A left module presented by 2 relations for 2 generators>
gap> Display( M );
Z/< 8 > + Z/< 2 >
gap> M;
<A torsion left module presented by 2 relations for 2 generators>
gap> a := HomalgMatrix( [ [ 2, 0 ] ], zz );
gap> alpha := HomalgMap( a, "free", M );
<A homomorphism of left modules>
gap> pi := CokernelEpi( alpha );
<An epimorphism of left modules>
gap> Display( pi );
[ [ 1, 0 ],
  [ 0, 1 ] ]

the map is currently represented by the above 2 x 2 matrix
gap> iota := KernelEmb( pi );
<A monomorphism of left modules>
gap> Display( iota );
[ [ 2, 0 ] ]

the map is currently represented by the above 1 x 2 matrix
gap> N := Kernel( pi );
<A cyclic torsion left module presented by yet unknown relations for a cyclic \
generator>
gap> Display( N );
Z/< 4 >
gap> C := HomalgComplex( pi );
<A left acyclic complex containing a single morphism of left modules at degree\
```

```

s [ 0 .. 1 ]>
gap> Add( C, iota );
gap> ByASmallerPresentation( C );
<A non-zero short exact sequence containing
2 morphisms of left modules at degrees [ 0 .. 2 ]>
gap> Display( C );
-----
at homology degree: 2
Z/< 4 >
-----
[ [ 0, 6 ] ]

the map is currently represented by the above 1 x 2 matrix
-----v-----
at homology degree: 1
Z/< 2 > + Z/< 8 >
-----
[ [ 0, 1 ],
  [ 1, 1 ] ]

the map is currently represented by the above 2 x 2 matrix
-----v-----
at homology degree: 0
Z/< 2 > + Z/< 2 >
-----
gap> T := LTensorProduct( C, N );
<An exact triangle containing 3 morphisms of left complexes at degrees
[ 1, 2, 3, 1 ]>
gap> ByASmallerPresentation( T );
<A non-zero exact triangle containing
3 morphisms of left complexes at degrees [ 1, 2, 3, 1 ]>
gap> L := LongSequence( T );
<A sequence containing 5 morphisms of left modules at degrees [ 0 .. 5 ]>
gap> Display( L );
-----
at homology degree: 5
Z/< 4 >
-----
[ [ 0, 3 ] ]

the map is currently represented by the above 1 x 2 matrix
-----v-----
at homology degree: 4
Z/< 2 > + Z/< 4 >
-----
[ [ 0, 1 ],
  [ 0, 0 ] ]

the map is currently represented by the above 2 x 2 matrix
-----v-----
at homology degree: 3
Z/< 2 > + Z/< 2 >

```

```

-----
[ [ 2 ],
  [ 0 ] ]

the map is currently represented by the above 2 x 1 matrix
-----v-----
at homology degree: 2
Z/< 4 >
-----
[ [ 0, 2 ] ]

the map is currently represented by the above 1 x 2 matrix
-----v-----
at homology degree: 1
Z/< 2 > + Z/< 4 >
-----
[ [ 0, 1 ],
  [ 1, 1 ] ]

the map is currently represented by the above 2 x 2 matrix
-----v-----
at homology degree: 0
Z/< 2 > + Z/< 2 >
-----
gap> IsExactSequence( L );
true
gap> L;
<An exact sequence containing 5 morphisms of left modules at degrees
[ 0 .. 5 ]>

```

10.4.19 Functor_HomHom

▷ Functor_HomHom

(global variable)

The bifunctor HomHom.

Below is the only *specific* line of code used to define Functor_HomHom_for_fp_modules and all the different operations HomHom in homalg.

```

----- Code -----
Functor_Hom_for_fp_modules * Functor_Hom_for_fp_modules;

```

10.4.20 Functor_LHomHom

▷ Functor_LHomHom

(global variable)

The bifunctor LHomHom.

Below is the only *specific* line of code used to define Functor_LHomHom_for_fp_modules and all the different operations LHomHom in homalg.

```

----- Code -----
LeftDerivedFunctor( Functor_HomHom_for_fp_modules );

```

10.5 Tool Functors

10.6 Other Functors

10.7 Functors: Operations and Functions

Chapter 11

Symmetric Algebra and Koszul Complex

11.1 Symmetric Algebra: Constructor

11.1.1 SymmetricPower

- ▷ `SymmetricPower(k , M)` (operation)
Returns: a homalg module
Construct the k -th exterior power of module M .

11.2 Symmetric Algebra: Properties and Attributes

11.2.1 IsSymmetricPower

- ▷ `IsSymmetricPower(M)` (property)
Returns: true or false
Marks a module as an symmetric power of another module.

11.2.2 SymmetricPowerExponent

- ▷ `SymmetricPowerExponent(M)` (attribute)
Returns: an integer
The exponent of the symmetric power.

11.2.3 SymmetricPowerBaseModule

- ▷ `SymmetricPowerBaseModule(M)` (attribute)
Returns: a homalg module
The module that M is an symmetric power of.

Chapter 12

Exterior Algebra and Koszul Complex

What follows are several operations related to the exterior algebra of a free module:

- A constructor for the graded parts of the exterior algebra (“exterior powers”)
- Several Operations on elements of these exterior powers
- A constructor for the “Koszul complex”
- An implementation of the “Cayley determinant” as defined in [CQ11], which allows calculating greatest common divisors from finite free resolutions.

12.1 Exterior Algebra: Constructor

12.1.1 ExteriorPower

▷ `ExteriorPower(k , M)` (operation)
Returns: a `homalg` module
Construct the k -th exterior power of module M .

12.2 Exterior Algebra: Properties and Attributes

12.2.1 IsExteriorPower

▷ `IsExteriorPower(M)` (property)
Returns: `true` or `false`
Marks a module as an exterior power of another module.

12.2.2 ExteriorPowerExponent

▷ `ExteriorPowerExponent(M)` (attribute)
Returns: an integer
The exponent of the exterior power.

12.2.3 ExteriorPowerBaseModule

- ▷ `ExteriorPowerBaseModule(M)` (attribute)
Returns: a homalg module
 The module that M is an exterior power of.

12.3 Exterior Algebra: Element Properties

12.3.1 IsExteriorPowerElement

- ▷ `IsExteriorPowerElement(x)` (property)
Returns: true or false
 Checks if the element x is from an exterior power.

12.4 Exterior Algebra: Element Operations

12.4.1 Wedge (for elements of exterior powers of free modules)

- ▷ `Wedge(x , y)` (operation)
Returns: an element of an exterior power
 Calculate $x \wedge y$.

12.4.2 ExteriorPowerElementDual

- ▷ `ExteriorPowerElementDual(x)` (operation)
Returns: an element of an exterior power
 For x in a q -th exterior power of a free module of rank n , return x^* in the $(n-q)$ -th exterior power, as defined in [CQ11].

12.4.3 SingleValueOfExteriorPowerElement

- ▷ `SingleValueOfExteriorPowerElement(x)` (operation)
Returns: a ring element
 For x in a highest exterior power, returns its single coordinate in the canonical basis; i.e. $[x]$ as defined in [CQ11].

12.5 Koszul complex and Cayley determinant

12.5.1 KoszulCocomplex

- ▷ `KoszulCocomplex(a , E)` (operation)
Returns: a homalg cocomplex
 Calculate the E -valued Koszul complex of a .

12.5.2 CayleyDeterminant

▷ `CayleyDeterminant(\mathcal{C})` (operation)

Returns: a ring element

Calculate the Cayley determinant of the complex \mathcal{C} , as defined in [CQ11].

12.5.3 Gcd_UsingCayleyDeterminant

▷ `Gcd_UsingCayleyDeterminant(x , y [, ...])` (function)

Returns: a ring element

Returns the greatest common divisor of the given ring elements, calculated using the Cayley determinant.

Chapter 13

Examples

13.1 ExtExt

This corresponds to Example B.2 in [Bar09].

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 21 relations for 8 generators>
of
<A non-zero left module presented by 31 relations for 19 generators>>
gap> ByASmallerPresentation( filt );
<An ascending filtration with degrees [ -1 .. 0 ] and graded parts:
  0:  <A rank 1 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>

```

13.2 Purity

This corresponds to Example B.3 in [Bar09].

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 >

```

13.3 TorExt-Grothendieck

This corresponds to Example B.5 in [Bar09].

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 6 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 14 generators>>
gap> ByASmallerPresentation( filt );
<A descending filtration with degrees [ -1 .. 0 ] and graded parts:
  -1:  <A non-zero torsion left module presented by 4 relations
        for 4 generators>
  0:   <A rank 1 left module presented by 2 relations for 3 generators>
of
<A rank 1 left module presented by 6 relations for 7 generators>>
gap> m := IsomorphismOfFiltration( filt );
<A non-zero isomorphism of left modules>

```

13.4 TorExt

This corresponds to Example B.6 in [Bar09].

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 bicomplex 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 torsion left module presented by yet unknown relations for
      4 generators>
  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 torsion left module presented by 4 relations
      for 4 generators>
  0:  <A rank 1 left module presented by 2 relations for 3 generators>
of
<A rank 1 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 Modules

As finite dimensional constructions in linear algebra over a field k boil down to solving (in)homogeneous linear systems over k , the Gaussian algorithm makes the whole theory perfectly computable.

Hence, for homological algebra (viewed as linear algebra over general rings) to be computable one needs to find appropriate substitutes for the Gaussian algorithm, where finite dimensionality has to be replaced by finite generatedness.

Luckily such substitutes exist for many rings of interest. Beside the well-known Hermite normal form algorithm for principal ideal rings it turns out that appropriate generalizations of the classical Gröbner basis algorithm for polynomial rings provide the desired substitute for a wide class of commutative *and* noncommutative rings. Note that for noncommutative rings the above discussion has to be restricted to homological constructions leading to one-sided linear systems $XA = B$ resp. $AX = B$ (\rightarrow [722Moduleschapter.7](#)-limitation)

Appendix B

Logic Subpackages

B.1 LIMOD: Logical Implications for Modules

B.2 LIHOM: Logical Implications for Homomorphisms of Modules

Appendix C

Overview of the Modules Package Source Code

C.1 Relations and Generators

Filename .gd/.gi	Content
HomalgRingMap	operations and methods ring maps
HomalgRelations	a set of relations
SetsOfRelations	several sets of relations
HomalgGenerators	a set of generators
SetsOfGenerators	several sets of generators

Table: *The homalg package files*

C.2 The Basic Objects

Filename .gd/.gi	Content
HomalgModule	modules allowing several presentations linked with transition matrices
HomalgSubmodule	submodules allowing several sets of generators
HomalgMap	maps allowing several presentations of their source and target
HomalgFiltration	filtration of a module
HomalgComplex	(co)complexes of modules or of (co)complexes
HomalgChainMap	chain maps of (co)complexes consisting of maps or chain maps
HomalgBicomplex	bicomplexes of modules or of (co)complexes
HomalgBigradedObject	(differential) bigraded modules
HomalgSpectralSequence	homological and cohomological spectral sequences
HomalgDiagram	Betti diagrams

Table: *The homalg package files (continued)*

C.3 The High Level Homological Algorithms

Filename .gd/.gi	Content
Modules	subfactors, resolutions, parameterizations, intersections, annihilators
ToolFunctors	zero map, composition, addition, subtraction, stacking, augmentation, and post dividing maps
BasicFunctors	cokernel, image, kernel, tensor product, Hom, Ext, Tor, RHom, LTensorProduct, HomHom, LHomHom, BaseChange (preliminary)
OtherFunctors	direct sum

Table: *The homalg package files (continued)*

C.4 Logical Implications for homalg Objects

Filename .gd/.gi	Content
LIMOD	logical implications for modules
LIHOM	logical implications for module homomorphisms

Table: *The homalg package files (continued)*

References

- [Bar09] Mohamed Barakat. Spectral filtrations via generalized morphisms. ([arXiv:0904.0240](#)), 2009. [68](#), [69](#), [71](#), [73](#)
- [BR06] Mohamed Barakat and Daniel Robertz. [homalg](#): First steps to an abstract package for homological algebra. In *Proceedings of the X meeting on computational algebra and its applications - EACA 2006*, pages 29–32, Sevilla, Spain, September 2006. [11](#)
- [BR08] Mohamed Barakat and Daniel Robertz. `homalg` – A meta-package for homological algebra. *J. Algebra Appl.*, 7(3):299–317, 2008. ([arXiv:math.AC/0701146](#)). [7](#)
- [CQ11] Thierry Coquand and Claude Quitté. Constructive finite free resolutions. *manuscripta mathematica*, pages 1–15, 2011. [65](#), [66](#), [67](#)

Index

Modules, 6

*

 constructor for ideal multiples, 34

 TensorProduct, 51

 transfer a module over a different ring, 26

 transfer a module over a different ring (right), 26

AffineDegree, 32

ByASmallerPresentation

 for modules, 33

CanBeUsedToDecideZeroEffectively, 19

CayleyDeterminant, 67

CoefficientsOfNumeratorOfHilbert-
 PoincareSeries, 32

CoefficientsOfUnreducedNumeratorOf-
 HilbertPoincareSeries, 31

Cokernel, 42

DataOfHilbertFunction, 32

DefectOfExactness, 46

ElementaryDivisors, 31

Ext, 55

ExteriorAlgebra, 31

ExteriorPower, 65

ExteriorPowerBaseModule, 66

ExteriorPowerElementDual, 66

ExteriorPowerExponent, 65

FittingIdeal, 31

functor_Cokernel, 42

Functor_Ext, 54

Functor_Hom, 46

Functor_HomHom, 62

functor_ImageObject, 43

Functor_LHomHom, 62

Functor_LTensorProduct, 59

Functor_RHom, 56

Functor_TensorProduct, 51

Functor_Tor, 55

Gcd_UsingCayleyDeterminant, 67

HilbertFunction, 32

HilbertPoincareSeries, 32

Hom, 47

HomalgFreeLeftModule

 constructor for free left modules, 25

HomalgFreeRightModule

 constructor for free right modules, 25

HomalgIdentityMap

 constructor for identity maps, 38

HomalgMap

 constructor for maps, 36

 constructor for maps between free modules, 36

HomalgRing, 39

 for module elements, 41

 for modules, 33

HomalgZeroLeftModule

 constructor for zero left modules, 25

HomalgZeroMap

 constructor for zero maps, 38

HomalgZeroRightModule

 constructor for zero right modules, 25

ImageObject, 44

IndexOfRegularity, 32

IsCyclic, 29

IsElementOfAModuleGivenByAMorphismRep, 40

IsElementOfIntegers, 40

IsExteriorPower, 65

IsExteriorPowerElement, 66

IsFinitelyPresentedModuleOr-
 SubmoduleRep, 23

IsFinitelyPresentedModuleRep, 23

- IsFinitelyPresentedSubmoduleRep, 23
- IsGeneratorsOfFinitelyGenerated-
ModuleRep, 21
- IsGeneratorsOfModuleRep, 21
- IsHolonomic, 29
- IsHomalgElement, 40
- IsHomalgGenerators, 20
- IsHomalgGeneratorsOfLeftModule, 20
- IsHomalgGeneratorsOfRightModule, 20
- IsHomalgMap, 35
- IsHomalgModule, 22
- IsHomalgRelations, 18
- IsHomalgRelationsOfLeftModule, 18
- IsHomalgRelationsOfRightModule, 18
- IsHomalgSelfMap, 35
- IsInjectivePresentation, 19
- IsMapOfFinitelyGeneratedModulesRep, 36
- IsPrimeIdeal, 30
- IsPrimeModule
for modules, 30
- IsReduced
for generators, 21
for modules, 29
- IsRelationsOfFinitelyPresented-
ModuleRep, 19
- IsSymmetricPower, 64
- Kernel
for maps, 45
for ring maps, 17
- KernelEmb
for ring maps, 17
- KernelSubobject
for ring maps, 17
- KoszulCocomplex, 66
- LargestMinimalNumberOfLocalGenerators,
31
- LeftPresentation
constructor for left modules, 24
- LeftSubmodule
constructor for left submodules, 28
- LTensorProduct, 60
- ModuleOfKaehlerDifferentials, 30
- NonFlatLocus, 31
- NumeratorOfHilbertPoincareSeries, 32
- PreInverse, 39
- PrimaryDecomposition, 30
- ProcedureToReadjustGenerators, 21
- RadicalDecomposition, 30
- RadicalSubobject, 31
- ResidueClassRing, 30
- RHom, 57
- RightPresentation
constructor for right modules, 24
- RightSubmodule
constructor for right submodules, 29
- SingleValueOfExteriorPowerElement, 66
- Subobject
constructor for submodules using a list of
ring elements, 28
constructor for submodules using matrices,
28
- SubobjectQuotient
for submodules, 34
- SymmetricAlgebra, 31
- SymmetricPower, 64
- SymmetricPowerBaseModule, 64
- SymmetricPowerExponent, 64
- TensorProduct, 51
- Tor, 56
- UnreducedNumeratorOfHilbertPoincare-
Series, 32
- Wedge
for elements of exterior powers of free mod-
ules, 66