homalg

A GAP4 Meta-Package for Homological Algebra

Version 2009.11.09

November 2009

Mohamed Barakat

(*this manual is still under construction*)

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

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

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

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

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

Copyright

Acknowledgements

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

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

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

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

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

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

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

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

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

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

- Daniel Robertz

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

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

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

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

Contents

Chapter 1

Introduction

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

1.1.1 Philosophy

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

1.1.2 homalg provides ...

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

- rings
- ring maps
- matrices
- modules (generators, relations)
- maps
- filtrations
- complexes (of modules and of complexes)
- chain maps
- bicomplexes
- bigraded (differential) objects
- spectral sequences

- functors

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

- resolution of modules
- computation of subfactor modules
- applying functors (like `Ext`, `Tor`, ...) to modules, maps, complexes and chain maps
- derivation and composition of functors
- horse shoe resolution of short exact sequences of modules
- connecting homomorphisms and long exact sequences
- Cartan-Eilenberg resolution of complexes
- hyper (co)homology
- spectral sequences of bicomplexes
- the Grothendieck spectral sequences associated to two composable functors
- test if a module is torsion-free, reflexive, projective, stably free, free, pure
- determine the rank, codimension, 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.

1.1.3 homalg delegates ...

The package homalg *delegates all* matrix operations as it treats matrices and their rings as *black boxes*. homalg comes with a single predefined class of rings and a single predefined class of matrices over these rings – the so-called internal matrices (→ 6.1.2) over so-called internal rings (→ 4.1.4). An internal matrix (resp. ring) is simply a wrapper containing a GAP-builtin matrix (resp. ring). homalg allows other packages to define further classes or extend existing classes of rings and matrices *together* with their operations. For example:

- The homalg subpackage ResidueClassRingForHomalg (→ Appendix F) defines the classes of residue class rings, residue class ring elements, and matrices over residue class rings. Such a matrix is defined by a matrix over the ambient ring which is nevertheless interpreted modulo the ring relations, i.e. modulo the generators of the defining ideal.
- The package GaussForHomalg extends the class of internal matrices enabling it to wrap sparse matrices provided by the package Gauss. GaussForHomalg delegates the essential part of the matrix creation and all matrix operations to Gauss.

- The package HomalgToCAS (→ Appendix H) defines the classes of so-called external rings and matrices and the package RingsForHomalg delegates the essential part of the matrix creation and all matrix operations to external computer algebra systems like Singular, Macaulay2, Sage, Macaulay2, MAGMA, Maple, ... The package homalg accesses external matrices via pointers. The pointer of an external matrix is simply its name in the external system. HomalgToCAS chooses these names.

- The package LocalizeRingForHomalg defines the classes of local(ized) rings, local ring elements, and local matrices. A homalg local matrix contains a homalg matrix as a numerator and an element of the global ring as a denominator.

The matrix operations are divided into two classes called "Tools" and "Basic". The "Tools" operations include addition, subtraction, multiplication, extracting certain rows or columns, stacking, and augmenting matrices (→ Appendix D). The "Basic" operations include the two basic operations in linear algebra needed to solve an inhomogeneous linear system $XA = B$ with coefficients in a not necessarily commutative ring R (→ Appendix C):

- Effectively reducing B modulo A, i.e. effectively deciding if a row (or a set of rows) B lies in the R-span of the rows of the matrix A.

- Computing an R-generating set of row syzygies (=R-relations among the rows) of A, i.e. computing an R-generating set of the left kernel of A. This generating set is then given as the rows of a matrix Y and $YA = 0$.

The first operation is nothing but deciding the solvability of the inhomogeneous system $XA = B$ and if solvable to compute a particular solution X, while the second is to compute an R-generating set for the homogeneous solution space, i.e. the solution space of the homogeneous system $YA = 0$ (→ 1.1.4). The above is of course also valid for the column convention.

1.1.4 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 (→ 1.1.5).

1.1.5 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. homalg 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 $\mathtt{Hom}(M,N)$ of two (even) finitely generated modules over a *non*commutative 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].

1.1.6 The black box concept

Now we address the following concerns: Wouldn't the idea of using algorithms like the Gröbnerbasis algorithm(s) as a black box (→ 1.1.3) contradict the following facts?

- It is known that an efficient Gröbnerbasis algorithm depends on the ring R under consideration. For example the implementation of the algorithm depends on the ground ring (or field) k.
- Often enough highly specialized implementations are used to address specific types of linear systems of equations (occuring in specific homological problems) in order to increase the speed or reduce the space needed for the computations.

The following should clarify the above concerns.

- Since each ring comes with its own black box, the first point is automatically resolved.
- Allow the black box coming with each ring to contain the different available implementations and make them accessible to homalg via standarized names, independent of the computer algebra system used to perform computations.

See also 3.1.

1.1.7 homalg's dictionaries (technical)

homalg 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 homalg 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 homalg 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, homalg 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 homalg. 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) homalg no longer distinguishes between internal and external.

1.1.8 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 (→ Appendix A.2). 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.

1.1.9 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 (→ Appendix A.2.4): 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.

1.1.10 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 homalg's design. homalg 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 maps, 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.

1.1.11 Building upon the homalg package

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

- add constructors (sheaves, schemes, simplicial sets, ...)

- add methods for basic operation (Yoneda products, Massey products, Steenrod operations, ...)

- add methods to compute sheaf cohomology, local cohomology, Hochschild (co)homology, cyclic (co)homology...

- provide algorithms for holonomic D-modules based on the restriction algorithm: localization, computing tensor products, `Hom`, `Ext`, de Rham cohomology, ...
- support change of rings, Lyndon/Hochschild-Serre spectral sequence, base change spectral sequences, ...
- support perturbation techniques, Serre and Eilenberg-Moore spectral sequence of simplicial spaces of infinite type, ...
- ...

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

1.1.12 Frequently asked questions

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

  A: Yes.

- Q: Can homalg 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 homalg.

- 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 homalg 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 homalg. For this purpose, developers can use the unified communication interface provideb by HomalgToCAS. This is the interface used by homalg.

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 (→ 1.1.2) with its constructors, properties, attributes, and operations.

Chapter 2

Installation of the homalg Package

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

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

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

before its functions become available.

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

Mohamed Barakat

Chapter 3

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 (→ 1.1.4).

The second and more important reason is to underline the fact the all effective homological constructions that are relevant for homalg 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 homalg is designed to use rings and matrices over these rings together with all their operations as a black box. In other words: Because homalg works for $\mathbb{Z}$, it works by its design for all other computable rings (see also 1.1.6).

3.2 gap> ExamplesForHomalg();

To quickly create a ring for use with homalg 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 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 (→ Appendix F). 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 $Hom(Hom(-,\mathbb{Z}/2^7\mathbb{Z}),\mathbb{Z}/2^4\mathbb{Z})$ (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( );
<A homalg internal ring>
gap> Display( ZZ );
Z
gap> R := ZZ / 2^8;
<A homalg residue class ring>
gap> Display( R );
Z/( 256 )
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 generato\
r>
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 ]| >
```

```
-------------------------
[ [  24 ] ]

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 ]| >
-------------------------
[ [  6 ] ]
```

```
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 ]| >
--------------------------
[ [  6 ] ]

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 ]| >
--------------------------
[ [  6 ] ]
```

```
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 ]| >
--------------------------
[ [  6 ] ]

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

Rings

4.1 Rings: Category and Representations

4.1.1 IsHomalgRing

◊ `IsHomalgRing(R)` (Category)

Returns: `true` or `false`

The GAP category of homalg rings.

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

4.1.2 IsPreHomalgRing

◊ `IsPreHomalgRing(R)` (Category)

Returns: `true` or `false`

The GAP category of pre homalg rings.

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

These are rings with an incomplete `homalgTable`. They provide flexibility for developers to support a wider class of rings, as was necessary for the development of the LocalizeRingForHomalg package. They are not suited for direct usage.

4.1.3 IsHomalgRingElement

◊ `IsHomalgRingElement(r)` (Category)

Returns: `true` or `false`

The GAP category of elements of homalg rings which are not GAP4 built-in.

4.1.4 IsHomalgInternalRingRep

◊ `IsHomalgInternalRingRep(R)` (Representation)

Returns: `true` or `false`

The internal representation of homalg rings.

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

4.2 Rings: Constructors

This section describes how to construct rings for use with homalg, which exploit the GAP4-built-in abilities to perform the necessary ring operations. By this we also mean necessary matrix operations over such rings. For the purposes of homalg only the ring of integers is properly supported in GAP4. The GAP4 extension packages Gauss and GaussForHomalg extend these built-in abilities to operations with sparse matrices over the ring $\mathbb{Z}/p^n$ for p prime and n positive.

If a ring R is supported in homalg any of its residue class rings R/I is supported as well, provided the ideal I of relations admits a finite set of generators as a left resp. right ideal (→ \/ (4.2.3)). This is immediate for commutative noetherian rings.

4.2.1 HomalgRingOfIntegers (constructor for the integers)

◊ `HomalgRingOfIntegers()` (function)

Returns: a homalg ring

◊ `HomalgRingOfIntegers(c)` (function)

Returns: a homalg ring

The no-argument form returns the ring of integers $\mathbb{Z}$ for homalg.

The one-argument form accepts an integer c and returns the ring $\mathbb{Z}/c$ for homalg:

- $c = 0$ defaults to $\mathbb{Z}$
- if c is a prime power then the package GaussForHomalg is loaded (if it fails to load an error is issued)
- otherwise, the residue class ring constructor / (→ \/ (4.2.3)) is invoked

The operation `SetRingProperties` is automatically invoked to set the ring properties.

If for some reason you don't want to use the GaussForHomalg package (maybe because you didn't install it), then use

`HomalgRingOfIntegers( ) / c;`

but note that the computations will then be considerably slower.

4.2.2 HomalgFieldOfRationals (constructor for the field of rationals)

◊ `HomalgFieldOfRationals()` (function)

Returns: a homalg ring

The package GaussForHomalg is loaded and the field of rationals $\mathbb{Q}$ is returned. If GaussForHomalg fails to load an error is issued.

The operation `SetRingProperties` is automatically invoked to set the ring properties.

4.2.3 \/ (constructor for residue class rings)

◊ `\/(R, ring_rel)` (operation)

Returns: a homalg ring

This is the homalg constructor for residue class rings `R` $/I$, where `R` is a homalg ring and $I =$ `ring_rel` is the ideal of relations generated by `ring_rel`. `ring_rel` might be:

- a set of left resp. right relations on one generator

- a list of ring elements of *R*
- a ring element of *R*

For noncommutative rings: In the first case the left resp. right set of relations should generate the ideal of relations *I* as left resp. right ideal generators, and their involutions should generate *I* as right resp. left ideal generators. If *ring_rel* is not a set of relations, a *left* set of relations is constructed.

The operation SetRingProperties is automatically invoked to set the ring properties.

Example

```
gap> ZZ := HomalgRingOfIntegers( );
<A homalg internal ring>
gap> Display( ZZ );
Z
gap> Z256 := ZZ / 2^8;
<A homalg residue class ring>
gap> Display( Z256 );
Z/( 256 )
gap> Z2 := Z256 / 6;
<A homalg residue class ring>
gap> Display( Z2 );
Z/( 2 )
```

4.3 Rings: Properties

The following properties are declared for homalg rings. Note that (apart from so-called true and immediate methods (→ E.1)) there are no methods installed for ring properties. This means that if the value of the ring property Prop is not set for a homalg ring *R*, then

Prop(*R*);

will cause an error. One can use the usual GAP4 mechanism to check if the value of the property is set or not

HasProp(*R*);

If you discover that a specific property Prop is missing for a certain homalg ring *R* you can it add using the usual GAP4 mechanism

SetProp(*R*, true);

or

SetProp(*R*, false);

Be very cautious with setting "missing" properties to homalg objects: If the value you set is mathematically wrong homalg will probably draw wrong conclusions and might return wrong results.

4.3.1 ContainsAField

◊ ContainsAField(*R*) (property)

Returns: true or false

R is a ring for homalg.

4.3.2 IsRationalsForHomalg

◊ IsRationalsForHomalg(*R*) (property)

Returns: true or false

R is a ring for homalg.

4.3.3 IsFieldForHomalg

◊ `IsFieldForHomalg(R)` (property)

Returns: `true` or `false`

R is a ring for homalg.

4.3.4 IsDivisionRingForHomalg

◊ `IsDivisionRingForHomalg(R)` (property)

Returns: `true` or `false`

R is a ring for homalg.

4.3.5 IsIntegersForHomalg

◊ `IsIntegersForHomalg(R)` (property)

Returns: `true` or `false`

R is a ring for homalg.

4.3.6 IsResidueClassRingOfTheIntegers

◊ `IsResidueClassRingOfTheIntegers(R)` (property)

Returns: `true` or `false`

R is a ring for homalg.

4.3.7 IsBezoutRing

◊ `IsBezoutRing(R)` (property)

Returns: `true` or `false`

R is a ring for homalg.

4.3.8 IsIntegrallyClosedDomain

◊ `IsIntegrallyClosedDomain(R)` (property)

Returns: `true` or `false`

R is a ring for homalg.

4.3.9 IsUniqueFactorizationDomain

◊ `IsUniqueFactorizationDomain(R)` (property)

Returns: `true` or `false`

R is a ring for homalg.

4.3.10 IsKaplanskyHermite

◊ `IsKaplanskyHermite(R)` (property)

Returns: `true` or `false`

R is a ring for homalg.

4.3.11 IsDedekindDomain

◊ IsDedekindDomain(*R*) (property)

Returns: true or false

R is a ring for homalg.

4.3.12 IsDiscreteValuationRing

◊ IsDiscreteValuationRing(*R*) (property)

Returns: true or false

R is a ring for homalg.

4.3.13 IsFreePolynomialRing

◊ IsFreePolynomialRing(*R*) (property)

Returns: true or false

R is a ring for homalg.

4.3.14 IsWeylRing

◊ IsWeylRing(*R*) (property)

Returns: true or false

R is a ring for homalg.

4.3.15 IsGlobalDimensionFinite

◊ IsGlobalDimensionFinite(*R*) (property)

Returns: true or false

R is a ring for homalg.

4.3.16 IsLeftGlobalDimensionFinite

◊ IsLeftGlobalDimensionFinite(*R*) (property)

Returns: true or false

R is a ring for homalg.

4.3.17 IsRightGlobalDimensionFinite

◊ IsRightGlobalDimensionFinite(*R*) (property)

Returns: true or false

R is a ring for homalg.

4.3.18 HasInvariantBasisProperty

◊ HasInvariantBasisProperty(*R*) (property)

Returns: true or false

R is a ring for homalg.

4.3.19 HasLeftInvariantBasisProperty

◊ HasLeftInvariantBasisProperty(*R*) (property)

Returns: true or false

R is a ring for homalg.

4.3.20 HasRightInvariantBasisProperty

◊ HasRightInvariantBasisProperty(*R*) (property)

Returns: true or false

R is a ring for homalg.

4.3.21 IsLocalRing

◊ IsLocalRing(*R*) (property)

Returns: true or false

R is a ring for homalg.

4.3.22 IsSemiLocalRing

◊ IsSemiLocalRing(*R*) (property)

Returns: true or false

R is a ring for homalg.

4.3.23 IsIntegralDomain

◊ IsIntegralDomain(*R*) (property)

Returns: true or false

R is a ring for homalg.

4.3.24 IsHereditary

◊ IsHereditary(*R*) (property)

Returns: true or false

R is a ring for homalg.

4.3.25 IsLeftHereditary

◊ IsLeftHereditary(*R*) (property)

Returns: true or false

R is a ring for homalg.

4.3.26 IsRightHereditary

◊ IsRightHereditary(*R*) (property)

Returns: true or false

R is a ring for homalg.

4.3.27 IsHermite

◊ IsHermite(*R*) (property)

Returns: true or false

R is a ring for homalg.

4.3.28 IsLeftHermite

◊ IsLeftHermite(*R*) (property)

Returns: true or false

R is a ring for homalg.

4.3.29 IsRightHermite

◊ IsRightHermite(*R*) (property)

Returns: true or false

R is a ring for homalg.

4.3.30 IsNoetherian

◊ IsNoetherian(*R*) (property)

Returns: true or false

R is a ring for homalg.

4.3.31 IsLeftNoetherian

◊ IsLeftNoetherian(*R*) (property)

Returns: true or false

R is a ring for homalg.

4.3.32 IsRightNoetherian

◊ IsRightNoetherian(*R*) (property)

Returns: true or false

R is a ring for homalg.

4.3.33 IsArtinian (for rings)

◊ IsArtinian(*R*) (property)

Returns: true or false

R is a ring for homalg.

4.3.34 IsLeftArtinian

◊ IsLeftArtinian(*R*) (property)

Returns: true or false

R is a ring for homalg.

4.3.35 IsRightArtinian

◊ IsRightArtinian(*R*) (property)

Returns: true or false

R is a ring for homalg.

4.3.36 IsOreDomain

◊ IsOreDomain(*R*) (property)

Returns: true or false

R is a ring for homalg.

4.3.37 IsLeftOreDomain

◊ IsLeftOreDomain(*R*) (property)

Returns: true or false

R is a ring for homalg.

4.3.38 IsRightOreDomain

◊ IsRightOreDomain(*R*) (property)

Returns: true or false

R is a ring for homalg.

4.3.39 IsPrincipalIdealRing

◊ IsPrincipalIdealRing(*R*) (property)

Returns: true or false

R is a ring for homalg.

4.3.40 IsLeftPrincipalIdealRing

◊ IsLeftPrincipalIdealRing(*R*) (property)

Returns: true or false

R is a ring for homalg.

4.3.41 IsRightPrincipalIdealRing

◊ IsRightPrincipalIdealRing(*R*) (property)

Returns: true or false

R is a ring for homalg.

4.3.42 IsRegular

◊ IsRegular(*R*) (property)

Returns: true or false

R is a ring for homalg.

4.3.43 IsFiniteFreePresentationRing

◊ IsFiniteFreePresentationRing(*R*) (property)

Returns: true or false

R is a ring for homalg.

4.3.44 IsLeftFiniteFreePresentationRing

◊ IsLeftFiniteFreePresentationRing(*R*) (property)

Returns: true or false

R is a ring for homalg.

4.3.45 IsRightFiniteFreePresentationRing

◊ IsRightFiniteFreePresentationRing(*R*) (property)

Returns: true or false

R is a ring for homalg.

4.3.46 IsSimpleRing

◊ IsSimpleRing(*R*) (property)

Returns: true or false

R is a ring for homalg.

4.3.47 IsSemiSimpleRing

◊ IsSemiSimpleRing(*R*) (property)

Returns: true or false

R is a ring for homalg.

4.3.48 BasisAlgorithmRespectsPrincipalIdeals

◊ BasisAlgorithmRespectsPrincipalIdeals(*R*) (property)

Returns: true or false

R is a ring for homalg.

4.3.49 IsMinusOne

◊ IsMinusOne(*r*) (property)

Returns: true or false

r is a homalg ring element.

4.4 Rings: Attributes

4.4.1 Zero (for homalg ring elements)

◊ Zero(*r*) (attribute)

Returns: the zero of the homalg ring element *r*

4.4.2 One (for homalg ring elements)

◊ One(*r*) (attribute)

Returns: the one of the homalg ring element *r*

4.4.3 homalgTable

◊ homalgTable(*R*) (attribute)

Returns: a homalg table

The homalg table of *R* is a ring dictionary, i.e. the translator between homalg and the (specific implementation of the) ring.

Every homalg ring has a homalg table.

4.4.4 RingElementConstructor

◊ RingElementConstructor(*R*) (attribute)

Returns: a function

The constructor of ring elements in the homalg ring *R*.

4.4.5 TypeOfHomalgMatrix

◊ TypeOfHomalgMatrix(*R*) (attribute)

Returns: a type

The GAP4-type of homalg matrices over the homalg ring *R*.

4.4.6 ConstructorForHomalgMatrices

◊ ConstructorForHomalgMatrices(*R*) (attribute)

Returns: a type

The constructor for homalg matrices over the homalg ring *R*.

4.4.7 Zero (for homalg rings)

◊ Zero(*R*) (attribute)

Returns: the zero of the homalg ring *R*

4.4.8 One (for homalg rings)

◊ One(*R*) (attribute)

Returns: the one of the homalg ring *R*

4.4.9 MinusOne

◊ MinusOne(*R*) (attribute)

Returns: the minus one of the homalg ring *R*

4.4.10 IndeterminatesOfPolynomialRing

◊ IndeterminatesOfPolynomialRing(*R*) (attribute)

Returns: a list of homalg ring elements

The list of indeterminates of the homalg polynomial ring *R*.

4.4.11 IndeterminateCoordinatesOfRingOfDerivations

◊ IndeterminateCoordinatesOfRingOfDerivations(*R*) (attribute)

Returns: a list of homalg ring elements

The list of indeterminate coordinates of the homalg Weyl ring *R*.

4.4.12 IndeterminateDerivationsOfRingOfDerivations

◊ IndeterminateDerivationsOfRingOfDerivations(*R*) (attribute)

Returns: a list of homalg ring elements

The list of indeterminate derivations of the homalg Weyl ring *R*.

4.4.13 IndeterminateAntiCommutingVariablesOfExteriorRing

◊ IndeterminateAntiCommutingVariablesOfExteriorRing(*R*) (attribute)

Returns: a list of homalg ring elements

The list of anti-commuting indeterminates of the homalg exterior ring *R*.

4.4.14 IndeterminatesOfExteriorRing

◊ IndeterminatesOfExteriorRing(*R*) (attribute)

Returns: a list of homalg ring elements

The list of all indeterminates (commuting and anti-commuting) of the homalg exterior ring *R*.

4.4.15 WeightsOfIndeterminates

◊ WeightsOfIndeterminates(*R*) (attribute)

Returns: a list or listlist of integers

The list of degrees of the indeterminates of the homalg ring *R*.

4.4.16 MatrixOfWeightsOfIndeterminates

◊ MatrixOfWeightsOfIndeterminates(*R*) (attribute)

Returns: a homalg matrix

A homalg matrix where the list (or listlist) of degrees of the indeterminates of the homalg ring *R* is stored.

4.4.17 CoefficientsRing

◊ CoefficientsRing(*R*) (attribute)

Returns: a homalg ring

The ring of coefficients of the homalg ring *R*.

4.4.18 KrullDimension

◊ KrullDimension(*R*) (attribute)

Returns: a non-negative integer

The Krull dimension of the commutative homalg ring *R*.

4.4.19 LeftGlobalDimension

◊ LeftGlobalDimension(*R*) (attribute)

Returns: a non-negative integer

The left global dimension of the homalg ring *R*.

4.4.20 RightGlobalDimension

◊ RightGlobalDimension(*R*) (attribute)

Returns: a non-negative integer

The right global dimension of the homalg ring *R*.

4.4.21 GlobalDimension

◊ GlobalDimension(*R*) (attribute)

Returns: a non-negative integer

The global dimension of the homalg ring *R*. The global dimension is defined, only if the left and right global dimensions coincide.

4.4.22 GeneralLinearRank

◊ GeneralLinearRank(*R*) (attribute)

Returns: a non-negative integer

The general linear rank of the homalg ring *R* ([MR01], 11.1.14).

4.4.23 ElementaryRank

◊ ElementaryRank(*R*) (attribute)

Returns: a non-negative integer

The elementary rank of the homalg ring *R* ([MR01], 11.3.10).

4.4.24 StableRank

◊ StableRank(*R*) (attribute)

Returns: a non-negative integer

The stable rank of the homalg ring *R* ([MR01], 11.3.4).

4.5 Rings: Operations and Functions

Chapter 5

Ring Maps

A homalg ring map is a data structure for maps between finitely generated rings. homalg more or less provides the basic declarations and installs the generic methods for ring maps, but it is up to other high level packages to install methods applicable to specific rings. For example, the package Sheaves provides methods for ring maps of (finitely generated) affine rings.

5.1 Ring Maps: Category and Representations

5.1.1 IsHomalgRingMap

◊ `IsHomalgRingMap(phi)` (Category)

Returns: `true` or `false`

The GAP category of ring maps.

5.1.2 IsHomalgRingSelfMap

◊ `IsHomalgRingSelfMap(phi)` (Category)

Returns: `true` or `false`

The GAP category of ring self-maps.

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

5.1.3 IsHomalgRingMapRep

◊ `IsHomalgRingMapRep(phi)` (Representation)

Returns: `true` or `false`

The GAP representation of finitley presented homalg modules.

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

5.2 Ring Maps: Constructors

5.2.1 RingMap (constructor for ring maps)

◊ `RingMap(images, S, T)` (operation)

Returns: a homalg ring map

This constructor returns a ring map (homomorphism) of finitely generated rings/algebras. It is represented by the images `images` of the set of generators of the source homalg ring `S` in terms of the generators of the target ring `T` (→ 4.2). Unless the source ring is free *and* given on free ring/algebra generators the returned map will cautiously be indicated using parenthesis: "homomorphism". To verify if the result is indeed a well defined map use `IsMorphism` (5.3.1). If source and target are identical objects, and only then, the ring map is created as a selfmap.

5.3 Ring Maps: Properties

5.3.1 IsMorphism (for ring maps)

◊ `IsMorphism(phi)` (property)

Returns: `true` or `false`

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

5.3.2 IsIdentityMorphism (for ring maps)

◊ `IsIdentityMorphism(phi)` (property)

Returns: `true` or `false`

Check if the homalg ring map `phi` is the identity morphism.

5.3.3 IsMonomorphism (for ring maps)

◊ `IsMonomorphism(phi)` (property)

Returns: `true` or `false`

Check if the homalg ring map `phi` is a monomorphism.

5.3.4 IsEpimorphism (for ring maps)

◊ `IsEpimorphism(phi)` (property)

Returns: `true` or `false`

Check if the homalg ring map `phi` is an epimorphism.

5.3.5 IsIsomorphism (for ring maps)

◊ `IsIsomorphism(phi)` (property)

Returns: `true` or `false`

Check if the homalg ring map `phi` is an isomorphism.

5.3.6 IsAutomorphism (for ring maps)

◊ `IsAutomorphism(phi)` (property)

Returns: `true` or `false`

Check if the homalg ring map `phi` is an automorphism.

5.4 Ring Maps: Attributes

5.4.1 Source (for ring maps)

◊ `Source(phi)` (attribute)

Returns: a homalg ring

The source of the homalg ring map *phi*.

5.4.2 Range (for ring maps)

◊ `Range(phi)` (attribute)

Returns: a homalg ring

The target (range) of the homalg ring map *phi*.

5.4.3 DegreeOfMorphism (for ring maps)

◊ `DegreeOfMorphism(phi)` (attribute)

Returns: an integer

The degree of the morphism *phi* of graded rings.

(no method installed)

5.4.4 CoordinateRingOfGraph (for ring maps)

◊ `CoordinateRingOfGraph(phi)` (attribute)

Returns: a homalg ring

The coordinate ring of the graph of the ring map *phi*.

5.4.5 KernelSubmodule (for ring maps)

◊ `KernelSubmodule(phi)` (attribute)

Returns: a homalg submodule

The kernel ideal of the ring map *phi* (as a submodule).

5.4.6 KernelEmb (for ring maps)

◊ `KernelEmb(phi)` (attribute)

Returns: a homalg map

The embedding of the kernel ideal `Kernel`(*phi*) into the `Source`(*phi*), both viewed as modules over the ring R :=`Source`(*phi*) (cf. `Kernel` (5.5.1)).

5.5 Ring Maps: Operations and Functions

5.5.1 Kernel (for ring maps)

◊ `Kernel(phi)` (method)

Returns: a homalg module

The kernel ideal of the ring map *phi* as an abstract module.

Chapter 6

Matrices

6.1 Matrices: Category and Representations

6.1.1 IsHomalgMatrix

◊ `IsHomalgMatrix(A)` (Category)

Returns: `true` or `false`

The GAP category of homalg matrices.

6.1.2 IsHomalgInternalMatrixRep

◊ `IsHomalgInternalMatrixRep(A)` (Representation)

Returns: `true` or `false`

The internal representation of homalg matrices.

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

6.2 Matrices: Constructors

6.2.1 HomalgInitialMatrix (constructor for initial matrices filled with zeros)

◊ `HomalgInitialMatrix(m, n, R)` (function)

Returns: a homalg matrix

A mutable unevaluated initial $m \times n$ homalg matrix filled with zeros over the homalg ring R. This construction is useful in case one wants to define a matrix by assigning its nonzero entries. Avoid asking about properties or attributes of the matrix until you finish filling it, since already computed values of properties and attributes will be cached and not recomputed unless the values are explicitly reset (⟶ `ResetFilterObj` (**Prg Tutorial: ResetFilterObj**)).

Example

```
gap> ZZ := HomalgRingOfIntegers( );
<A homalg internal ring>
gap> z := HomalgInitialMatrix( 2, 3, ZZ );
<An initial homalg internal 2 by 3 matrix>
gap> HasIsZero( z );
false
gap> IsZero( z );
true
```

```
gap> z;
<A homalg internal 2 by 3 zero matrix>
```

Example

```
gap> n := HomalgInitialMatrix( 2, 3, ZZ );
<An initial homalg internal 2 by 3 matrix>
gap> SetEntryOfHomalgMatrix( n, 1, 1, "1" );
gap> SetEntryOfHomalgMatrix( n, 2, 3, "1" );
gap> ResetFilterObj( n, IsMutableMatrix );
gap> Display( n );
[ [  1,  0,  0 ],
  [  0,  0,  1 ] ]
gap> IsZero( n );
false
gap> n;
<A non-zero homalg internal 2 by 3 matrix>
```

6.2.2 HomalgInitialIdentityMatrix (constructor for initial quadratic matrices with ones on the diagonal)

◊ HomalgInitialIdentityMatrix(*m*, *R*) (function)

Returns: a homalg matrix

A mutable unevaluated initial $m \times m$ homalg quadratic matrix with ones on the diagonal over the homalg ring R. This construction is useful in case one wants to define an elementary matrix by assigning its off-diagonal nonzero entries. Avoid asking about properties or attributes of the matrix until you finish filling it, since already computed values of properties and attributes will be cached and not recomputed unless the values are explicitly reset (→ ResetFilterObj (**Prg Tutorial: ResetFilterObj**)).

Example

```
gap> ZZ := HomalgRingOfIntegers( );
<A homalg internal ring>
gap> id := HomalgInitialIdentityMatrix( 3, ZZ );
<An initial identity homalg internal 3 by 3 matrix>
gap> HasIsIdentityMatrix( id );
false
gap> IsIdentityMatrix( id );
true
gap> id;
<A homalg internal 3 by 3 identity matrix>
```

Example

```
gap> e := HomalgInitialIdentityMatrix( 3, ZZ );
<An initial identity homalg internal 3 by 3 matrix>
gap> SetEntryOfHomalgMatrix( e, 1, 2, "1" );
gap> SetEntryOfHomalgMatrix( e, 2, 1, "-1" );
gap> ResetFilterObj( e, IsMutableMatrix );
gap> Display( e );
[ [   1,   1,   0 ],
  [  -1,   1,   0 ],
  [   0,   0,   1 ] ]
gap> IsIdentityMatrix( e );
false
```

```
gap> e;
<A homalg internal 3 by 3 matrix>
```

6.2.3 HomalgZeroMatrix (constructor for zero matrices)

◊ HomalgZeroMatrix(*m, n, R*) (function)

Returns: a homalg matrix

An immutable unevaluated $m \times n$ homalg zero matrix over the homalg ring R.

Example

```
gap> ZZ := HomalgRingOfIntegers( );
<A homalg internal ring>
gap> z := HomalgZeroMatrix( 2, 3, ZZ );
<An unevaluated homalg internal 2 by 3 zero matrix>
gap> Display( z );
[ [  0,  0,  0 ],
  [  0,  0,  0 ] ]
gap> z;
<A homalg internal 2 by 3 zero matrix>
```

6.2.4 HomalgIdentityMatrix (constructor for identity matrices)

◊ HomalgIdentityMatrix(*m, R*) (function)

Returns: a homalg matrix

An immutable unevaluated $m \times m$ homalg identity matrix over the homalg ring R.

Example

```
gap> ZZ := HomalgRingOfIntegers( );
<A homalg internal ring>
gap> id := HomalgIdentityMatrix( 3, ZZ );
<An unevaluated homalg internal 3 by 3 identity matrix>
gap> Display( id );
[ [  1,  0,  0 ],
  [  0,  1,  0 ],
  [  0,  0,  1 ] ]
gap> id;
<A homalg internal 3 by 3 identity matrix>
```

6.2.5 HomalgVoidMatrix (constructor for void matrices)

◊ HomalgVoidMatrix(*[m,][n,]R*) (function)

Returns: a homalg matrix

A void $m \times n$ homalg matrix.

6.2.6 HomalgMatrix (constructor for matrices using a listlist)

◊ HomalgMatrix(*llist, R*) (function)

◊ HomalgMatrix(*list, m, n, R*) (function)

◊ HomalgMatrix(*str_llist, R*) (function)

◊ HomalgMatrix(*str_list, m, n, R*) (function)

Returns: a homalg matrix

An immutable evaluated $m \times n$ homalg matrix over the homalg ring R.

Example

```
gap> ZZ := HomalgRingOfIntegers( );
<A homalg internal ring>
gap> m := HomalgMatrix( [ [ 1, 2, 3 ], [ 4, 5, 6 ] ], ZZ );
<A homalg internal 2 by 3 matrix>
gap> Display( m );
[ [  1,  2,  3 ],
  [  4,  5,  6 ] ]
```

Example

```
gap> m := HomalgMatrix( [ [ 1, 2, 3 ], [ 4, 5, 6 ] ], 2, 3, ZZ );
<A homalg internal 2 by 3 matrix>
gap> Display( m );
[ [  1,  2,  3 ],
  [  4,  5,  6 ] ]
```

Example

```
gap> m := HomalgMatrix( [ 1, 2, 3,   4, 5, 6 ], 2, 3, ZZ );
<A homalg internal 2 by 3 matrix>
gap> Display( m );
[ [  1,  2,  3 ],
  [  4,  5,  6 ] ]
```

Example

```
gap> m := HomalgMatrix( "[ [ 1, 2, 3 ], [ 4, 5, 6 ] ]", ZZ );
<A homalg internal 2 by 3 matrix>
gap> Display( m );
[ [  1,  2,  3 ],
  [  4,  5,  6 ] ]
```

Example

```
gap> m := HomalgMatrix( "[ [ 1, 2, 3 ], [ 4, 5, 6 ] ]", 2, 3, ZZ );
<A homalg internal 2 by 3 matrix>
gap> Display( m );
[ [  1,  2,  3 ],
  [  4,  5,  6 ] ]
```

It is nevertheless recommended to use the following form to create homalg matrices. This form can also be used to define external matrices. Since whitespaces (→ **Reference: Whitespaces**) are ignored, they can be used as optical delimiters:

Example

```
gap> m := HomalgMatrix( "[ 1, 2, 3,   4, 5, 6 ]", 2, 3, ZZ );
<A homalg internal 2 by 3 matrix>
gap> Display( m );
[ [  1,  2,  3 ],
  [  4,  5,  6 ] ]
```

One can split the input string over several lines using the backslash character '\' to end each line

Example

```
gap> m := HomalgMatrix( "[ \
> 1, 2, 3, \
> 4, 5, 6  \
> ]", 2, 3, ZZ );
```

```
<A homalg internal 2 by 3 matrix>
gap> Display( m );
[ [  1,  2,  3 ],
  [  4,  5,  6 ] ]
```

6.2.7 HomalgDiagonalMatrix (constructor for diagonal matrices)

◊ HomalgDiagonalMatrix(*diag*, *R*) (function)

Returns: a homalg matrix

An immutable unevaluated diagonal homalg matrix over the homalg ring *R*. The diagonal consists of the entries of the list *diag*.

Example

```
gap> ZZ := HomalgRingOfIntegers( );
<A homalg internal ring>
gap> d := HomalgDiagonalMatrix( [ 1, 2, 3 ], ZZ );
<An unevaluated diagonal homalg internal 3 by 3 matrix>
gap> Display( d );
[ [  1,  0,  0 ],
  [  0,  2,  0 ],
  [  0,  0,  3 ] ]
gap> d;
<A diagonal homalg internal 3 by 3 matrix>
```

6.2.8 * (copy a matrix over a different ring)

◊ *(*R*, *mat*) (operation)

◊ *(*mat*, *R*) (operation)

Returns: a homalg matrix

An immutable evaluated homalg matrix over the homalg ring *R* having the same entries as the matrix *mat*. Syntax: *R* * *mat* or *mat* * *R*

Example

```
gap> ZZ := HomalgRingOfIntegers( );
<A homalg internal ring>
gap> Z4 := ZZ / 4;
<A homalg residue class ring>
gap> Display( Z4 );
Z/( 4 )
gap> d := HomalgDiagonalMatrix( [ 2 .. 4 ], ZZ );
<An unevaluated diagonal homalg internal 3 by 3 matrix>
gap> d2 := Z4 * d; ## or d2 := d * Z4;
<A homalg residue class 3 by 3 matrix>
gap> Display( d2 );
[ [  2,  0,  0 ],
  [  0,  3,  0 ],
  [  0,  0,  4 ] ]

modulo [ 4 ]
gap> d;
<A diagonal homalg internal 3 by 3 matrix>
gap> ZeroRows( d );
[  ]
```

```
gap> ZeroRows( d2 );
[ 3 ]
gap> d;
<A non-zero diagonal homalg internal 3 by 3 matrix>
gap> d2;
<A non-zero homalg residue class 3 by 3 matrix>
```

6.3 Matrices: Properties

6.3.1 IsZero (for matrices)

◊ IsZero(*A*) (property)

Returns: true or false

Check if the homalg matrix *A* is a zero matrix, taking possible ring relations into account.
(for the installed standard method see IsZeroMatrix (D.1.16))

Example

```
gap> ZZ := HomalgRingOfIntegers( );
<A homalg internal ring>
gap> A := HomalgMatrix( "[ 2 ]", ZZ );
<A homalg internal 1 by 1 matrix>
gap> Z2 := ZZ / 2;
<A homalg residue class ring>
gap> A := Z2 * A;
<A homalg residue class 1 by 1 matrix>
gap> Display( A );
[ [  2 ] ]

modulo [ 2 ]
gap> IsZero( A );
true
```

6.3.2 IsIdentityMatrix

◊ IsIdentityMatrix(*A*) (property)

Returns: true or false

Check if the homalg matrix *A* is an identity matrix, taking possible ring relations into account.
(for the installed standard method see IsIdentityMatrix (D.2.2))

6.3.3 IsPermutationMatrix

◊ IsPermutationMatrix(*A*) (property)

Returns: true or false

A is a homalg matrix.

6.3.4 IsSpecialSubidentityMatrix

◊ IsSpecialSubidentityMatrix(*A*) (property)

Returns: true or false

A is a homalg matrix.

6.3.5 IsSubidentityMatrix

◊ IsSubidentityMatrix(A) (property)
Returns: true or false
A is a homalg matrix.

6.3.6 IsLeftRegularMatrix

◊ IsLeftRegularMatrix(A) (property)
Returns: true or false
A is a homalg matrix.

6.3.7 IsRightRegularMatrix

◊ IsRightRegularMatrix(A) (property)
Returns: true or false
A is a homalg matrix.

6.3.8 IsInvertibleMatrix

◊ IsInvertibleMatrix(A) (property)
Returns: true or false
A is a homalg matrix.

6.3.9 IsLeftInvertibleMatrix

◊ IsLeftInvertibleMatrix(A) (property)
Returns: true or false
A is a homalg matrix.

6.3.10 IsRightInvertibleMatrix

◊ IsRightInvertibleMatrix(A) (property)
Returns: true or false
A is a homalg matrix.

6.3.11 IsEmptyMatrix

◊ IsEmptyMatrix(A) (property)
Returns: true or false
A is a homalg matrix.

6.3.12 IsDiagonalMatrix

◊ IsDiagonalMatrix(A) (property)
Returns: true or false
Check if the homalg matrix A is an identity matrix, taking possible ring relations into account.
(for the installed standard method see IsDiagonalMatrix (D.2.3))

6.3.13 IsScalarlMatrix

◊ IsScalarlMatrix(A) (property)

Returns: true or false

A is a homalg matrix.

6.3.14 IsUpperTriangularMatrix

◊ IsUpperTriangularMatrix(A) (property)

Returns: true or false

A is a homalg matrix.

6.3.15 IsLowerTriangularMatrix

◊ IsLowerTriangularMatrix(A) (property)

Returns: true or false

A is a homalg matrix.

6.3.16 IsStrictUpperTriangularMatrix

◊ IsStrictUpperTriangularMatrix(A) (property)

Returns: true or false

A is a homalg matrix.

6.3.17 IsStrictLowerTriangularMatrix

◊ IsStrictLowerTriangularMatrix(A) (property)

Returns: true or false

A is a homalg matrix.

6.3.18 IsUpperStairCaseMatrix

◊ IsUpperStairCaseMatrix(A) (property)

Returns: true or false

A is a homalg matrix.

6.3.19 IsLowerStairCaseMatrix

◊ IsLowerStairCaseMatrix(A) (property)

Returns: true or false

A is a homalg matrix.

6.3.20 IsTriangularMatrix

◊ IsTriangularMatrix(A) (property)

Returns: true or false

A is a homalg matrix.

6.3.21 IsBasisOfRowsMatrix

◊ IsBasisOfRowsMatrix(A) (property)
Returns: true or false
A is a homalg matrix.

6.3.22 IsBasisOfColumnsMatrix

◊ IsBasisOfColumnsMatrix(A) (property)
Returns: true or false
A is a homalg matrix.

6.3.23 IsReducedBasisOfRowsMatrix

◊ IsReducedBasisOfRowsMatrix(A) (property)
Returns: true or false
A is a homalg matrix.

6.3.24 IsReducedBasisOfColumnsMatrix

◊ IsReducedBasisOfColumnsMatrix(A) (property)
Returns: true or false
A is a homalg matrix.

6.3.25 IsMutableMatrix

◊ IsMutableMatrix(A) (filter)
Returns: true or false
A is a homalg matrix.

6.3.26 IsInitialMatrix

◊ IsInitialMatrix(A) (property)
Returns: true or false
A is a homalg matrix.

6.3.27 IsInitialIdentityMatrix

◊ IsInitialIdentityMatrix(A) (property)
Returns: true or false
A is a homalg matrix.

6.3.28 IsVoidMatrix

◊ IsVoidMatrix(A) (property)
Returns: true or false
A is a homalg matrix.

6.4 Matrices: Attributes

6.4.1 NrRows

◊ NrRows(*A*) (attribute)

Returns: a nonnegative integer
The number of rows of the matrix *A*.
(for the installed standard method see NrRows (D.1.17))

6.4.2 NrColumns

◊ NrColumns(*A*) (attribute)

Returns: a nonnegative integer
The number of columns of the matrix *A*.
(for the installed standard method see NrColumns (D.1.18))

6.4.3 DeterminantMat

◊ DeterminantMat(*A*) (attribute)

Returns: a ring element
The determinant of the quadratic matrix *A*.
You can invoke it with Determinant(*A*).
(for the installed standard method see Determinant (D.1.19))

6.4.4 ZeroRows

◊ ZeroRows(*A*) (attribute)

Returns: a (possibly empty) list of positive integers
The list of zero rows of the matrix *A*.
(for the installed standard method see ZeroRows (D.2.4))

6.4.5 ZeroColumns

◊ ZeroColumns(*A*) (attribute)

Returns: a (possibly empty) list of positive integers
The list of zero columns of the matrix *A*.
(for the installed standard method see ZeroColumns (D.2.5))

6.4.6 NonZeroRows

◊ NonZeroRows(*A*) (attribute)

Returns: a (possibly empty) list of positive integers
The list of nonzero rows of the matrix *A*.

6.4.7 NonZeroColumns

◊ NonZeroColumns(*A*) (attribute)

Returns: a (possibly empty) list of positive integers
The list of nonzero columns of the matrix *A*.

6.4.8 PositionOfFirstNonZeroEntryPerRow

◊ PositionOfFirstNonZeroEntryPerRow(*A*) (attribute)

Returns: a list of nonnegative integers

The list of positions of the first nonzero entry per row of the matrix *A*, else zero.

6.4.9 PositionOfFirstNonZeroEntryPerColumn

◊ PositionOfFirstNonZeroEntryPerColumn(*A*) (attribute)

Returns: a list of nonnegative integers

The list of positions of the first nonzero entry per column of the matrix *A*, else zero.

6.4.10 DegreesOfEntries

◊ DegreesOfEntries(*A*) (attribute)

Returns: a listlist of degrees/multi-degrees

The matrix of degrees of the matrix *A*.

(for the installed standard method see DegreesOfEntries (D.2.9))

6.4.11 RowRankOfMatrix

◊ RowRankOfMatrix(*A*) (attribute)

Returns: a nonnegative integer

The row rank of the matrix *A*.

6.4.12 ColumnRankOfMatrix

◊ ColumnRankOfMatrix(*A*) (attribute)

Returns: a nonnegative integer

The column rank of the matrix *A*.

6.4.13 LeftInverse (for matrices)

◊ LeftInverse(*M*) (method)

Returns: a homalg matrix

A (lazy evaluated) left inverse *C* of the matrix *M*. If no left inverse exists then Eval(*C*) = false. (→ RightDivide (6.5.40))

(for the installed standard method see Eval (E.4.16))

6.4.14 RightInverse (for matrices)

◊ RightInverse(*M*) (attribute)

Returns: a homalg matrix

A (lazy evaluated) right inverse *C* of the matrix *M*. If no right inverse exists then Eval(*C*) = false. (→ LeftDivide (6.5.41))

(for the installed standard method see Eval (E.4.17))

6.5 Matrices: Operations and Functions

6.5.1 HomalgRing (for matrices)

◊ HomalgRing(*mat*) (operation)

Returns: a homalg ring

The homalg ring of the homalg matrix *mat*.

Example

```
gap> ZZ := HomalgRingOfIntegers( );
<A homalg internal ring>
gap> d := HomalgDiagonalMatrix( [ 2 .. 4 ], ZZ );
<An unevaluated diagonal homalg internal 3 by 3 matrix>
gap> R := HomalgRing( d );
<A homalg internal ring>
gap> IsIdenticalObj( R, ZZ );
true
```

6.5.2 Involution (for matrices)

◊ Involution(*M*) (method)

Returns: a homalg matrix

The twisted transpose of the homalg matrix *M*.

(for the installed standard method see Eval (E.4.5))

6.5.3 CertainRows (for matrices)

◊ CertainRows(*M, plist*) (method)

Returns: a homalg matrix

The matrix of which the i-th row is the k-th row of the homalg matrix *M*, where $k = plist[i]$.

(→ Eval (E.4.6))

6.5.4 CertainColumns (for matrices)

◊ CertainColumns(*M, plist*) (method)

Returns: a homalg matrix

The matrix of which the j-th column is the l-th column of the homalg matrix *M*, where $l = plist[i]$.

(for the installed standard method see Eval (E.4.7))

6.5.5 UnionOfRows (for matrices)

◊ UnionOfRows(*A, B*) (method)

Returns: a homalg matrix

Stack the two homalg matrices *A* and *B*.

(for the installed standard method see Eval (E.4.8))

6.5.6 UnionOfColumns (for matrices)

◊ `UnionOfColumns(A, B)` (method)

Returns: a homalg matrix

Augment the two homalg matrices `A` and `B`.

(for the installed standard method see `Eval` (E.4.9))

6.5.7 DiagMat (for matrices)

◊ `DiagMat(list)` (method)

Returns: a homalg matrix

Build the block diagonal matrix out of the homalg matrices listed in `list`. An error is issued if `list` is empty or if one of the arguments is not a homalg matrix.

(for the installed standard method see `Eval` (E.4.10))

6.5.8 KroneckerMat (for matrices)

◊ `KroneckerMat(A, B)` (method)

Returns: a homalg matrix

The Kronecker (or tensor) product of the two homalg matrices `A` and `B`.

(for the installed standard method see `Eval` (E.4.11))

6.5.9 * (for ring elements and matrices)

◊ `\*(a, A)` (method)

Returns: a homalg matrix

The product of the ring element `a` with the homalg matrix `A` (enter: `a * A;`).

(for the installed standard method see `Eval` (E.4.12))

6.5.10 \+ (for matrices)

◊ `\+(A, B)` (method)

Returns: a homalg matrix

The sum of the two homalg matrices `A` and `B` (enter: `A + B;`).

(for the installed standard method see `Eval` (E.4.13))

6.5.11 \- (for matrices)

◊ `\-(A, B)` (method)

Returns: a homalg matrix

The difference of the two homalg matrices `A` and `B` (enter: `A - B;`).

(for the installed standard method see `Eval` (E.4.14))

6.5.12 * (for composable matrices)

◊ `\*(A, B)` (method)

Returns: a homalg matrix

The matrix product of the two homalg matrices `A` and `B` (enter: `A * B;`).

(for the installed standard method see `Eval` (E.4.15))

6.5.13 \= (for matrices)

◊ \=(*A*, *B*) (operation)

Returns: `true` or `false`

Check if the homalg matrices *A* and *B* are equal (enter: *A* = *B*;), taking possible ring relations into account.

(for the installed standard method see `AreEqualMatrices` (D.2.1))

Example

```
gap> ZZ := HomalgRingOfIntegers( );
<A homalg internal ring>
gap> A := HomalgMatrix( "[ 1 ]", ZZ );
<A homalg internal 1 by 1 matrix>
gap> B := HomalgMatrix( "[ 3 ]", ZZ );
<A homalg internal 1 by 1 matrix>
gap> Z2 := ZZ / 2;
<A homalg residue class ring>
gap> A := Z2 * A;
<A homalg residue class 1 by 1 matrix>
gap> B := Z2 * B;
<A homalg residue class 1 by 1 matrix>
gap> Display( A );
[ [  1 ] ]

modulo [ 2 ]
gap> Display( B );
[ [  3 ] ]

modulo [ 2 ]
gap> A = B;
true
```

6.5.14 GetColumnIndependentUnitPositions (for matrices)

◊ GetColumnIndependentUnitPositions(*A*, *poslist*) (operation)

Returns: a (possibly empty) list of pairs of positive integers

The list of column independet unit position of the matrix *A*. We say that a unit $A[i,k]$ is column independet from the unit $A[l,j]$ if $i > l$ and $A[l,k] = 0$. The rows are scanned from top to bottom and within each row the columns are scanned from right to left searching for new units, column independent from the preceding ones. If $A[i,k]$ is a new column independent unit then $[i,k]$ is added to the output list. If *A* has no units the empty list is returned.

(for the installed standard method see `GetColumnIndependentUnitPositions` (D.2.6))

6.5.15 GetRowIndependentUnitPositions (for matrices)

◊ GetRowIndependentUnitPositions(*A*, *poslist*) (operation)

Returns: a (possibly empty) list of pairs of positive integers

The list of row independet unit position of the matrix *A*. We say that a unit $A[k,j]$ is row independet from the unit $A[i,l]$ if $j > l$ and $A[k,l] = 0$. The columns are scanned from left to right and within each column the rows are scanned from bottom to top searching for new units, row independent from the

preceding ones. If $A[k,j]$ is a new row independent unit then $[j,k]$ (yes $[j,k]$) is added to the output list. If A has no units the empty list is returned.

(for the installed standard method see GetRowIndependentUnitPositions (D.2.7))

6.5.16 GetUnitPosition (for matrices)

◊ GetUnitPosition(*A, poslist*) (operation)

Returns: a (possibly empty) list of pairs of positive integers

The position $[i,j]$ of the first unit $A[i,j]$ in the matrix A, where the rows are scanned from top to bottom and within each row the columns are scanned from left to right. If $A[i,j]$ is the first occurrence of a unit then the position pair $[i,j]$ is returned. Otherwise fail is returned.

(for the installed standard method see GetUnitPosition (D.2.8))

6.5.17 BasisOfRowModule (for matrices)

◊ BasisOfRowModule(*M*) (operation)

Returns: a homalg matrix

Let R be the ring over which M is defined (R :=HomalgRing(M)) and S be the row span of M, i.e. the R-submodule of the free module $R^{(1\times NrColumns(M))}$ spanned by the rows of M. A solution to the "submodule membership problem" is an algorithm which can decide if an element m in $R^{(1\times NrColumns(M))}$ is contained in S or not. And exactly like the Gaussian (resp. Hermite) normal form when R is a field (resp. principal ideal ring), the row span of the resulting matrix B coincides with the row span S of M, and computing B is typically the first step of such an algorithm. (→ Appendix B)

6.5.18 BasisOfColumnModule (for matrices)

◊ BasisOfColumnModule(*M*) (operation)

Returns: a homalg matrix

Let R be the ring over which M is defined (R :=HomalgRing(M)) and S be the column span of M, i.e. the R-submodule of the free module $R^{(NrRows(M)\times 1)}$ spanned by the columns of M. A solution to the "submodule membership problem" is an algorithm which can decide if an element m in $R^{(NrRows(M)\times 1)}$ is contained in S or not. And exactly like the Gaussian (resp. Hermite) normal form when R is a field (resp. principal ideal ring), the column span of the resulting matrix B coincides with the column span S of M, and computing B is typically the first step of such an algorithm. (→ Appendix B)

6.5.19 DecideZeroRows (for pairs of matrices)

◊ DecideZeroRows(*A, B*) (operation)

Returns: a homalg matrix

Let A and B be matrices having the same number of columns and defined over the same ring R (:=HomalgRing(A)) and S be the row span of B, i.e. the R-submodule of the free module $R^{(1\times NrColumns(B))}$ spanned by the rows of B. The result is a matrix C having the same shape as A, for which the i-th row C^i is equivalent to the i-th row A^i of A modulo S, i.e. $C^i - A^i$ is an element of the row span S of B. Moreover, the row C^i is zero, if and only if the row A^i is an element of S. So DecideZeroRows decides which rows of A are zero modulo the rows of B. (→ Appendix B)

6.5.20 DecideZeroColumns (for pairs of matrices)

◊ DecideZeroColumns(A, B) (operation)

Returns: a homalg matrix

Let A and B be matrices having the same number of rows and defined over the same ring R (:=HomalgRing(A)) and S be the column span of B, i.e. the R-submodule of the free module $R^{(NrRows(B)\times 1)}$ spanned by the columns of B. The result is a matrix C having the same shape as A, for which the i-th column C_i is equivalent to the i-th column A_i of A modulo S, i.e. $C_i - A_i$ is an element of the column span S of B. Moreover, the column C_i is zero, if and only if the column A_i is an element of S. So DecideZeroColumns decides which columns of A are zero modulo the columns of B. (→ Appendix B)

6.5.21 SyzygiesGeneratorsOfRows (for matrices)

◊ SyzygiesGeneratorsOfRows(M) (operation)

Returns: a homalg matrix

Let R be the ring over which M is defined (R :=HomalgRing(M)). The matrix of row syzygies SyzygiesGeneratorsOfRows(M) is a matrix whose rows span the left kernel of M, i.e. the R-submodule of the free module $R^{(1\times NrRows(M))}$ consisting of all rows X satisfying $XM = 0$. (→ Appendix B)

6.5.22 SyzygiesGeneratorsOfColumns (for matrices)

◊ SyzygiesGeneratorsOfColumns(M) (operation)

Returns: a homalg matrix

Let R be the ring over which M is defined (R :=HomalgRing(M)). The matrix of column syzygies SyzygiesGeneratorsOfColumns(M) is a matrix whose columns span the right kernel of M, i.e. the R-submodule of the free module $R^{(NrColumns(M)\times 1)}$ consisting of all columns X satisfying $MX = 0$. (→ Appendix B)

6.5.23 SyzygiesGeneratorsOfRows (for pairs of matrices)

◊ SyzygiesGeneratorsOfRows(M, M2) (operation)

Returns: a homalg matrix

Let R be the ring over which M is defined (R :=HomalgRing(M)). The matrix of *relative* row syzygies SyzygiesGeneratorsOfRows(M, M2) is a matrix whose rows span the left kernel of M modulo M2, i.e. the R-submodule of the free module $R^{(1\times NrRows(M))}$ consisting of all rows X satisfying $XM + YM2 = 0$ for some row $Y \in R^{(1\times NrRows(M2))}$. (→ Appendix B)

6.5.24 SyzygiesGeneratorsOfColumns (for pairs of matrices)

◊ SyzygiesGeneratorsOfColumns(M, M2) (operation)

Returns: a homalg matrix

Let R be the ring over which M is defined (R :=HomalgRing(M)). The matrix of *relative* column syzygies SyzygiesGeneratorsOfColumns(M, M2) is a matrix whose columns span the right kernel of M modulo M2, i.e. the R-submodule of the free module $R^{(NrColumns(M)\times 1)}$ consisting of all columns X satisfying $MX + M2Y = 0$ for some column $Y \in R^{(NrColumns(M2)\times 1)}$. (→ Appendix B)

6.5.25 ReducedBasisOfRowModule (for matrices)

◊ ReducedBasisOfRowModule(*M*) (operation)

Returns: a homalg matrix

Like BasisOfRowModule(*M*) but where the matrix SyzygiesGeneratorsOfRows(ReducedBasisOfRowModule(*M*)) contains no units. This can easily be achieved starting from *B* :=BasisOfRowModule(*M*) (and using GetColumnIndependentUnitPositions (6.5.14) applied to the matrix of row syzygies of *B*, etc). (→ Appendix B)

6.5.26 ReducedBasisOfColumnModule (for matrices)

◊ ReducedBasisOfColumnModule(*M*) (operation)

Returns: a homalg matrix

Like BasisOfColumnModule(*M*) but where the matrix SyzygiesGeneratorsOfColumns(ReducedBasisOfColumnModule(*M*)) contains no units. This can easily be achieved starting from *B* :=BasisOfColumnModule(*M*) (and using GetRowIndependentUnitPositions (6.5.15) applied to the matrix of column syzygies of *B*, etc.). (→ Appendix B)

6.5.27 ReducedSyzygiesGeneratorsOfRows (for matrices)

◊ ReducedSyzygiesGeneratorsOfRows(*M*) (operation)

Returns: a homalg matrix

Like SyzygiesGeneratorsOfRows(*M*) but where the matrix SyzygiesGeneratorsOfRows(ReducedSyzygiesGeneratorsOfRows(*M*)) contains no units. This can easily be achieved starting from *C* :=SyzygiesGeneratorsOfRows(*M*) (and using GetColumnIndependentUnitPositions (6.5.14) applied to the matrix of row syzygies of *C*, etc.). (→ Appendix B)

6.5.28 ReducedSyzygiesGeneratorsOfColumns (for matrices)

◊ ReducedSyzygiesGeneratorsOfColumns(*M*) (operation)

Returns: a homalg matrix

Like SyzygiesGeneratorsOfColumns(*M*) but where the matrix SyzygiesGeneratorsOfColumns(ReducedSyzygiesGeneratorsOfColumns(*M*)) contains no units. This can easily be achieved starting from *C* :=SyzygiesGeneratorsOfColumns(*M*) (and using GetRowIndependentUnitPositions (6.5.15) applied to the matrix of column syzygies of *C*, etc.). (→ Appendix B)

6.5.29 BasisOfRowsCoeff (for matrices)

◊ BasisOfRowsCoeff(*M*, *T*) (operation)

Returns: a homalg matrix

Returns *B* :=BasisOfRowModule(*M*) and assigns the *void* matrix *T* (→ HomalgVoidMatrix (6.2.5)) such that $B = TM$. (→ Appendix B)

6.5.30 BasisOfColumnsCoeff (for matrices)

◊ BasisOfColumnsCoeff(*M*, *T*) (operation)

Returns: a homalg matrix

Returns B :=BasisOfRowModule(M) and assigns the *void* matrix T (→ HomalgVoidMatrix (6.2.5)) such that $B = MT$. (→ Appendix B)

6.5.31 DecideZeroRowsEffectively (for pairs of matrices)

◊ DecideZeroRowsEffectively(A, B, T) (operation)

Returns: a homalg matrix

Returns M :=DecideZeroRows(A, B) and assigns the *void* matrix T (→ HomalgVoidMatrix (6.2.5)) such that $M = A + TB$. (→ Appendix B)

6.5.32 DecideZeroColumnsEffectively (for pairs of matrices)

◊ DecideZeroColumnsEffectively(A, B, T) (operation)

Returns: a homalg matrix

Returns M :=DecideZeroColumns(A, B) and assigns the *void* matrix T (→ HomalgVoidMatrix (6.2.5)) such that $M = A + BT$. (→ Appendix B)

6.5.33 BasisOfRows (for matrices)

◊ BasisOfRows(M) (operation)

◊ BasisOfRows(M, T) (operation)

Returns: a homalg matrix

With one argument it is a synonym of BasisOfRowModule (6.5.17). with two arguments it is a synonym of BasisOfRowsCoeff (6.5.29).

6.5.34 BasisOfColumns (for matrices)

◊ BasisOfColumns(M) (operation)

◊ BasisOfColumns(M, T) (operation)

Returns: a homalg matrix

With one argument it is a synonym of BasisOfColumnModule (6.5.18). with two arguments it is a synonym of BasisOfColumnsCoeff (6.5.30).

6.5.35 DecideZero (for matrices and relations)

◊ DecideZero(mat, rel) (operation)

Returns: a homalg matrix

Code

```
InstallMethod( DecideZero,
        "for sets of relations of homalg modules",
        [ IsHomalgMatrix, IsHomalgRelations ],

  function( mat, rel )
    local rel_mat;

    rel_mat := MatrixOfRelations( BasisOfModule( rel ) );

    if IsHomalgRelationsOfLeftModule( rel ) then
        return DecideZeroRows( mat, rel_mat );
```

```
    else
        return DecideZeroColumns( mat, rel_mat );
    fi;

end );
```

6.5.36 SyzygiesOfRows (for matrices)

◊ SyzygiesOfRows(*M*) (operation)
◊ SyzygiesOfRows(*M*, *M2*) (operation)

Returns: a homalg matrix

With one argument it is a synonym of SyzygiesGeneratorsOfRows (6.5.21). with two arguments it is a synonym of SyzygiesGeneratorsOfRows (6.5.23).

6.5.37 SyzygiesOfColumns (for matrices)

◊ SyzygiesOfColumns(*M*) (operation)
◊ SyzygiesOfColumns(*M*, *M2*) (operation)

Returns: a homalg matrix

With one argument it is a synonym of SyzygiesGeneratorsOfColumns (6.5.22). with two arguments it is a synonym of SyzygiesGeneratorsOfColumns (6.5.24).

6.5.38 ReducedSyzygiesOfRows (for matrices)

◊ ReducedSyzygiesOfRows(*M*) (operation)
◊ ReducedSyzygiesOfRows(*M*, *M2*) (operation)

Returns: a homalg matrix

With one argument it is a synonym of ReducedSyzygiesGeneratorsOfRows (6.5.27). With two arguments it calls ReducedBasisOfRowModule(SyzygiesGeneratorsOfRows(*M*, *M2*)). (→ ReducedBasisOfRowModule (6.5.25) and SyzygiesGeneratorsOfRows (6.5.23))

6.5.39 ReducedSyzygiesOfColumns (for matrices)

◊ ReducedSyzygiesOfColumns(*M*) (operation)
◊ ReducedSyzygiesOfColumns(*M*, *M2*) (operation)

Returns: a homalg matrix

With one argument it is a synonym of ReducedSyzygiesGeneratorsOfColumns (6.5.28). With two arguments it calls ReducedBasisOfColumnModule(SyzygiesGeneratorsOfColumns(*M*, *M2*)). (→ ReducedBasisOfColumnModule (6.5.26) and SyzygiesGeneratorsOfColumns (6.5.24))

6.5.40 RightDivide (for pairs of matrices)

◊ RightDivide(*B*, *A*) (operation)

Returns: a homalg matrix or false

Let *B* and *A* be matrices having the same number of columns and defined over the same ring. The matrix RightDivide(*B*, *A*) is a particular solution of the inhomogeneous (one sided) linear system of equations $XA = B$ in case it is solvable. Otherwise false is returned. The name RightDivide

suggests "$X = BA^{-1}$". This generalizes LeftInverse (6.4.13) for which B becomes the identity matrix. (→ SyzygiesGeneratorsOfRows (6.5.21))

6.5.41 LeftDivide (for pairs of matrices)

◊ LeftDivide(A, B) (operation)

Returns: a homalg matrix or false

Let A and B be matrices having the same number of rows and defined over the same ring. The matrix LeftDivide(A, B) is a particular solution of the inhomogeneous (one sided) linear system of equations $AX = B$ in case it is solvable. Otherwise false is returned. The name LeftDivide suggests "$X = A^{-1}B$". This generalizes RightInverse (6.4.14) for which B becomes the identity matrix. (→ SyzygiesGeneratorsOfColumns (6.5.22))

6.5.42 RightDivide (for triples of matrices)

◊ RightDivide(B, A, L) (operation)
◊ RightDivide(B, A, L) (operation)

Returns: a homalg matrix or false

Let B, A and L be matrices having the same number of columns and defined over the same ring (L might also be a set of relations for a left module with NrGenerators(L) = NrColumns(B)). The matrix RightDivide(B, A, L) is a particular solution of the inhomogeneous (one sided) linear system of equations $XA + YL = B$ in case it is solvable (for some Y which is forgotten). Otherwise false is returned. The name RightDivide suggests "$X = BA^{-1}$ modulo L". (Cf. [BR08, Subsection 3.1.1])

Code

```
InstallMethod( RightDivide,
        "for homalg matrices",
        [ IsHomalgMatrix, IsHomalgMatrix, IsHomalgRelationsOfLeftModule ],

  function( B, A, L )        ## CAUTION: Do not use lazy evaluation here!!!
    local R, BL, ZA, AL, CA, IAL, ZB, CB, NF, X;

    R := HomalgRing( B );

    BL := BasisOfModule( L );

    ## first reduce A modulo L
    ZA := DecideZero( A, BL );

    AL := UnionOfRows( ZA, MatrixOfRelations( BL ) );

    ## CA * AL = IAL
    CA := HomalgVoidMatrix( R );
    IAL := BasisOfRows( AL, CA );

    ## also reduce B modulo L
    ZB := DecideZero( B, BL );

    ## knowing this will avoid computations
    IsIdentityMatrix( IAL );
```

```
    ## IsSpecialSubidentityMatrix( IAL );        ## does not increase performance

    ## NF = ZB + CB * IAL
    CB := HomalgVoidMatrix( R );
    NF := DecideZeroRowsEffectively( ZB, IAL, CB );

    ## NF <> 0
    if not IsZero( NF ) then
        return false;
    fi;

    ## CD = -CB * CA => CD * A = B
    X := -CB * CertainColumns( CA, [ 1 .. NrRows( A ) ] );

    ## check assertion
    Assert( 3, IsZero( DecideZero( X * A - B, L ) ) );

    return X;

    ## technical: -CB * CA := (-CB) * CA and COLEM should take over
    ## since CB := -matrix

end );


InstallMethod( RightDivide,
        "for homalg matrices",
        [ IsHomalgMatrix, IsHomalgMatrix, IsHomalgMatrix ],

  function( B, A, L )

    return RightDivide( B, A, HomalgRelationsForLeftModule( L ) );

end );
```

6.5.43 LeftDivide (for triples of matrices)

◇ LeftDivide(*A*, *B*, *L*) (operation)
◇ LeftDivide(*A*, *B*, *L*) (operation)

Returns: a homalg matrix or false

Let A, B and L be matrices having the same number of columns and defined over the same ring (L might also be a set of relations for a right module with NrGenerators(L) = NrRows(B)). The matrix LeftDivide(A, B, L) is a particular solution of the inhomogeneous (one sided) linear system of equations $AX + LY = B$ in case it is solvable (for some Y which is forgotten). Otherwise false is returned. The name LeftDivide suggests "$X = A^{-1}B$ modulo L". (Cf. [BR08, Subsection 3.1.1])

Code

```
InstallMethod( LeftDivide,
        "for homalg matrices",
        [ IsHomalgMatrix, IsHomalgMatrix, IsHomalgRelationsOfRightModule ],
```

```
  function( A, B, L )        ## CAUTION: Do not use lazy evaluation here!!!
    local R, BL, ZA, AL, CA, IAL, ZB, CB, NF, X;

    R := HomalgRing( B );

    BL := BasisOfModule( L );

    ## first reduce A modulo L
    ZA := DecideZero( A, BL );

    AL := UnionOfColumns( ZA, MatrixOfRelations( BL ) );

    ## AL * CA = IAL
    CA := HomalgVoidMatrix( R );
    IAL := BasisOfColumns( AL, CA );

    ## also reduce B modulo L
    ZB := DecideZero( B, BL );

    ## knowing this will avoid computations
    IsIdentityMatrix( IAL );

    ## IsSpecialSubidentityMatrix( IAL );        ## does not increase performance

    ## NF = ZB + IAL * CB
    CB := HomalgVoidMatrix( R );
    NF := DecideZeroColumnsEffectively( ZB, IAL, CB );

    ## NF <> 0
    if not IsZero( NF ) then
        return false;
    fi;

    ## CD = CA * -CB => A * CD = B
    X := CertainRows( CA, [ 1 .. NrColumns( A ) ] ) * -CB;

    ## check assertion
    Assert( 3, IsZero( DecideZero( A * X - B, L ) ) );

    return X;

    ## technical: CA * -CB := CA * (-CB) and COLEM should take over since
    ## CB := -matrix

end );


InstallMethod( LeftDivide,
        "for homalg matrices",
        [ IsHomalgMatrix, IsHomalgMatrix, IsHomalgMatrix ],

  function( A, B, L )
```

```
    return LeftDivide( A, B, HomalgRelationsForRightModule( L ) );

end );
```

6.5.44 GenerateSameRowModule (for pairs of matrices)

◊ GenerateSameRowModule(*M*, *N*) (operation)

Returns: true or false

Check if the row span of *M* and of *N* are identical or not (→ RightDivide (6.5.40)).

6.5.45 GenerateSameColumnModule (for pairs of matrices)

◊ GenerateSameColumnModule(*M*, *N*) (operation)

Returns: true or false

Check if the column span of *M* and of *N* are identical or not (→ LeftDivide (6.5.41)).

Chapter 7

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 (→ Chapter 8)). The different sets of relations are linked with so-called transition matrices (→ Chapter 9).

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

7.1 Relations: Categories and Representations

7.1.1 IsHomalgRelations

◊ IsHomalgRelations(*rel*) (Category)

Returns: true or false

The GAP category of homalg relations.

7.1.2 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.)

7.1.3 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.)

7.1.4 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 (7.1.1))

7.2 Relations: Constructors

7.3 Relations: Properties

7.3.1 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)

7.3.2 IsInjectivePresentation

◊ IsInjectivePresentation(*rel*) (property)

Returns: true or false

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

7.4 Relations: Attributes

7.4.1 FreeResolution

◊ FreeResolution(*rel*) (attribute)

Returns: a homalg complex

The computed (part of a) free resolution of the module presented by homalg set of relations *rel*.

7.5 Relations: Operations and Functions

Chapter 8

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 (→ Chapter 7)). The different sets of generators are linked with so-called transition matrices (→ Chapter 9).

8.1 Generators: Categories and Representations

8.1.1 IsHomalgGenerators

◊ IsHomalgGenerators(*rel*) (Category)

Returns: true or false

The GAP category of homalg generators.

8.1.2 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.)

8.1.3 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.)

8.1.4 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 category IsHomalgGenerators (8.1.1))

8.2 Generators: Constructors

8.3 Generators: Properties

8.3.1 IsReduced

◊ IsReduced(*gen*) (property)

Returns: true or false

Check if the homalg set of generators *gen* is marked reduced.
(no method installed)

8.4 Generators: Attributes

8.4.1 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).

8.5 Generators: Operations and Functions

Chapter 9

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.

9.1 Modules: Category and Representations

9.1.1 IsHomalgModule

◊ `IsHomalgModule(M)` (Category)

Returns: `true` or `false`

The GAP category of homalg modules.

(It is a subcategory of the GAP categories `IsHomalgRingOrModule` and `IsHomalgObject`.)

9.1.2 IsFinitelyPresentedModuleOrSubmoduleRep

◊ `IsFinitelyPresentedModuleOrSubmoduleRep(M)` (Representation)

Returns: `true` or `false`

The GAP representation of finitley presented homalg modules.

(It is a representation of the GAP category `IsHomalgModule` (9.1.1).)

9.1.3 IsFinitelyPresentedModuleRep

◊ IsFinitelyPresentedModuleRep(*M*) (Representation)

Returns: true or false

The GAP representation of finitley presented homalg modules.

(It is a representation of the GAP category IsHomalgModule (9.1.1), which is a subrepresentation of the GAP representations IsFinitelyPresentedModuleOrSubmoduleRep, IsFinitelyPresentedObjectRep, and IsHomalgRingOrFinitelyPresentedModuleRep.)

9.1.4 IsFinitelyPresentedSubmoduleRep

◊ IsFinitelyPresentedSubmoduleRep(*M*) (Representation)

Returns: true or false

The GAP representation of finitley generated homalg submodules.

(It is a representation of the GAP category IsHomalgModule (9.1.1), IsFinitelyPresentedModuleOrSubmoduleRep, IsFinitelyPresentedSubobjectRep, and IsHomalgRingOrFinitelyPresentedModuleRep.)

9.2 Modules: Constructors

9.2.1 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 homalg internal 2 by 3 matrix>
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)
```

9.2.2 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 homalg internal 2 by 3 matrix>
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 >
```

9.2.3 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>
```

9.2.4 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>
```

9.2.5 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>
```

9.2.6 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>
```

9.2.7 * (transfer a module over a different ring)

◊ *(*R*, *M*) (operation)

◊ *(*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 / (→ \/ (4.2.3)).
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( );
<A homalg internal ring>
gap> Z4 := ZZ / 4;
<A homalg residue class ring>
gap> Display( Z4 );
Z/( 4 )
gap> M := HomalgDiagonalMatrix( [ 2 .. 4 ], ZZ );
<An unevaluated diagonal homalg internal 3 by 3 matrix>
gap> M := LeftPresentation( M );
<A 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( );;
gap> M := HomalgMatrix( "[ \
> 2, 3, 4, \
> 5, 6, 7  \
> ]", 2, 3, ZZ );
<A homalg internal 2 by 3 matrix>
gap> M := LeftPresentation( M );
<A non-torsion left module presented by 2 relations for 3 generators>
gap> Z4 := ZZ / 4;;
gap> Display( Z4 );
Z/( 4 )
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> Hom( d, Z4 );
<A cocomplex containing 2 morphisms of right 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 >
--------------------------
```

9.2.8 Subobject (constructor for submodules using maps)

◊ `Subobject(phi)` (operation)

Returns: a homalg submodule

A synonym of `ImageSubmodule` (10.4.8).

9.2.9 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*.

9.2.10 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*.

9.2.11 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, NrColumns( gen ) * R );

end );
```

Example

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

9.2.12 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 * NrRows( gen ) );

end );
```

Example

```
gap> Z4 := HomalgRingOfIntegers( ) / 4;
<A homalg residue class ring>
gap> J := HomalgMatrix( "[ 2 ]", 1, 1, Z4 );
<A homalg residue class 1 by 1 matrix>
gap> J := RightSubmodule( J );
<A principal (right) ideal given by a cyclic generator>
gap> IsFree( J );
false
gap> J;
```

```
<A principal reflexive non-projective (right) ideal given by a cyclic generato\
r>
```

9.3 Modules: Properties

9.3.1 IsFree

◊ IsFree(*M*) (property)

Returns: true or false

Check if the homalg module *M* is free.

9.3.2 IsStablyFree

◊ IsStablyFree(*M*) (property)

Returns: true or false

Check if the homalg module *M* is stably free.

9.3.3 IsProjective

◊ IsProjective(*M*) (property)

Returns: true or false

Check if the homalg module *M* is projective.

9.3.4 FiniteFreeResolutionExists

◊ FiniteFreeResolutionExists(*M*) (property)

Returns: true or false

Check if the homalg module *M* allows a finite free resolution.

(no method installed)

9.3.5 IsReflexive

◊ IsReflexive(*M*) (property)

Returns: true or false

Check if the homalg module *M* is reflexive.

9.3.6 IsTorsionFree

◊ IsTorsionFree(*M*) (property)

Returns: true or false

Check if the homalg module *M* is torsion-free.

9.3.7 IsArtinian (for modules)

◊ IsArtinian(*M*) (property)

Returns: true or false

Check if the homalg module *M* is artinian.

9.3.8 IsCyclic

◊ IsCyclic(*M*) (property)

Returns: true or false

Check if the homalg module *M* is cyclic.

9.3.9 IsTorsion

◊ IsTorsion(*M*) (property)

Returns: true or false

Check if the homalg module *M* is torsion.

9.3.10 IsHolonomic

◊ IsHolonomic(*M*) (property)

Returns: true or false

Check if the homalg module *M* is holonomic.

9.3.11 IsPure

◊ IsPure(*M*) (property)

Returns: true or false

Check if the homalg module *M* is pure.

9.3.12 HasConstantRank

◊ HasConstantRank(*M*) (property)

Returns: true or false

Check if the homalg module *M* has constant rank.

(no method installed)

9.3.13 ConstructedAsAnIdeal

◊ ConstructedAsAnIdeal(*J*) (property)

Returns: true or false

Check if the homalg submodule *J* was constructed as an ideal.

(no method installed)

9.3.14 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)

9.4 Modules: Attributes

9.4.1 TheZeroMorphism

◊ TheZeroMorphism(*M*) (attribute)

Returns: a homalg map

The zero endomorphism of the homalg module *M*.

9.4.2 TheIdentityMorphism

◊ TheIdentityMorphism(*M*) (attribute)

Returns: a homalg map

The identity automorphism of the homalg module *M*.

9.4.3 FullSubmodule

◊ FullSubmodule(*M*) (attribute)

Returns: a homalg submodule

The homalg module *M* as a submodule of itself.

9.4.4 EmbeddingInSuperObject

◊ EmbeddingInSuperObject(*N*) (attribute)

Returns: a homalg map

In case *N* was defined as a submodule of some module *L* the embedding of *N* in *L* is returned.

9.4.5 FactorObject

◊ FactorObject(*N*) (attribute)

Returns: a homalg module

In case *N* was defined as a submodule of some module *L* the factor module L/N is returned.

9.4.6 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.

9.4.7 UnderlyingSubobject

◊ UnderlyingSubobject(*M*) (attribute)

Returns: a homalg submodule

In case *M* was defined as the module underlying a submodule *L* then *L* is returned.

(no method installed)

9.4.8 NatTrIdToHomHom_R (for maps)

◊ NatTrIdToHomHom_R(*M*) (attribute)

Returns: a homalg map

The natural evaluation map from the homalg module *M* to its double dual HomHom(*M*) (cf. Functor_HomHom (16.4.21)).

9.4.9 PrimaryDecomposition

◊ PrimaryDecomposition(*J*) (attribute)

Returns: a list

The primary decomposition of the ideal *J*. The ring has to be commutative..
(no method installed)

9.4.10 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)

9.4.11 RankOfModule

◊ RankOfModule(*M*) (attribute)

Returns: a nonnegative integer

The projective rank of the homalg module *M*.

9.4.12 ProjectiveDimension

◊ ProjectiveDimension(*M*) (attribute)

Returns: a nonnegative integer

The projective dimension of the homalg module *M*.

9.4.13 DegreeOfTorsionFreeness

◊ DegreeOfTorsionFreeness(*M*) (attribute)

Returns: a nonnegative integer of infinity

Auslander's degree of torsion-freeness of the homalg module *M*. It is set to infinity only for $M = 0$.

9.4.14 Codim

◊ Codim(*M*) (attribute)

Returns: a nonnegative integer of infinity

The codimension of the homalg module *M*. It is set to infinity only for $M = 0$.

9.4.15 PurityFiltration

◊ PurityFiltration(*M*) (attribute)

Returns: a homalg filtration

The purity filtration of the homalg module *M*.

9.4.16 CodegreeOfPurity

◊ CodegreeOfPurity(*M*) (attribute)

Returns: a list of nonnegative integers

The codegree of purity of the homalg module *M*.

9.4.17 BettiDiagram (for modules)

◊ BettiDiagram(*M*) (attribute)

Returns: a homalg diagram

The Betti diagram of the homalg graded module *M*.

9.4.18 CastelnuovoMumfordRegularity

◊ CastelnuovoMumfordRegularity(*M*) (attribute)

Returns: a non-negative integer

The Castelnuovo-Mumford regularity of the homalg graded module *M*.

9.5 Modules: Operations and Functions

9.5.1 HomalgRing (for modules)

◊ HomalgRing(*M*) (operation)

Returns: a homalg ring

The homalg ring of the homalg module *M*.

Example

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

9.5.2 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 homalg internal 2 by 3 matrix>
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 3 sets of generators of a homalg module>
gap> SetsOfRelations( M );
<A set containing 3 sets of relations of a homalg module>
gap> M;
<A rank 1 left module presented by 1 relation for 2 generators>
gap> SetPositionOfTheDefaultSetOfRelations( M, 1 );
gap> M;
<A rank 1 left module presented by 2 relations for 3 generators>
```

9.5.3 UnderlyingObject (for submodules)

◊ UnderlyingObject(*M*) (operation)

Returns: a homalg module

In case *M* was defined as a submodule of some module *L* the module underlying the submodule *M* is returned.

9.5.4 SuperObject (for submodules)

◊ SuperObject(*M*) (operation)

Returns: a homalg module

In case *M* was defined as a submodule of some module *L* the super module *L* is returned.

9.5.5 * (constructor for ideal multiples)

◊ *(*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*.

9.5.6 SubmoduleQuotient (for submodules)

◊ SubmoduleQuotient(*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.

9.5.7 Saturate (for ideals)

◊ Saturate(*K*, *J*) (operation)

Returns: a homalg ideal

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

Example

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

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

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

A (left) ideal generated by the entry of the above matrix
gap> Jm := SubmoduleQuotient( J, m );
<A principal (left) ideal given by a cyclic generator>
gap> Display( Jm );
[ [  -12 ] ]

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

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

Code

```
InstallMethod( Saturate,
        "for homalg submodules",
```

```
        [ IsFinitelyPresentedSubmoduleRep, IsFinitelyPresentedSubmoduleRep ],

  function( K, J )
    local quotient_last, quotient;

    quotient_last := SubmoduleQuotient( K, J );

    quotient := SubmoduleQuotient( quotient_last, J );

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

    return quotient_last;

end );


InstallMethod( \-,        ## a geometrically motivated definition
        "for homalg submodules",
        [ IsFinitelyPresentedSubmoduleRep, IsFinitelyPresentedSubmoduleRep ],

  function( K, J )

    return Saturate( K, J );

end );
```

Chapter 10

Maps

A homalg map is a data structures for maps (module homomorphisms) between finitely generated modules. Each map in homalg knows its source (→ `Source` (10.4.1)) and its target (→ `Range` (10.4.2)). 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 (→ Chapter 9), 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.

10.1 Maps: Categories and Representations

10.1.1 IsHomalgMap

◊ `IsHomalgMap(`*phi*`)` (Category)

Returns: `true` or `false`

The GAP category of homalg maps.

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

10.1.2 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`.)

10.1.3 IsMapOfFinitelyGeneratedModulesRep

◊ `IsMapOfFinitelyGeneratedModulesRep(`*phi*`)` (Representation)

Returns: `true` or `false`

The GAP representation of maps between finitley generated homalg modules.

(It is a representation of the GAP category `IsHomalgChainMap` (12.1.1), which is a subrepresentation of the GAP representation `IsMorphismOfFinitelyGeneratedModulesRep`.)

10.2 Maps: Constructors

10.2.1 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* (→ 9.2). 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 (10.3.1). 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 homalg internal 2 by 3 matrix>
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 homalg internal 2 by 4 matrix>
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 homalg internal 3 by 4 matrix>
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 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, "l" );
<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>
```

10.2.2 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 homalg internal 2 by 3 matrix>
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 homalg internal 2 by 4 matrix>
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>
```

10.2.3 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 homalg internal 2 by 3 matrix>
gap> M := LeftPresentation( M );
<A non-torsion left module presented by 2 relations for 3 generators>
gap> HomalgIdentityMap( M );
<The identity morphism of a left module>
```

10.3 Maps: Properties

10.3.1 IsMorphism (for maps)

◊ IsMorphism(*phi*) (property)

Returns: `true` or `false`

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

10.3.2 IsGeneralizedMorphism (for maps)

◊ IsGeneralizedMorphism(*phi*) (property)

Returns: `true` or `false`

Check if *phi* is a generalized morphism.

10.3.3 IsGeneralizedEpimorphism (for maps)

◊ IsGeneralizedEpimorphism(*phi*) (property)

Returns: `true` or `false`

Check if *phi* is a generalized epimorphism.

10.3.4 IsGeneralizedMonomorphism (for maps)

◊ IsGeneralizedMonomorphism(*phi*) (property)

Returns: true or false

Check if *phi* is a generalized monomorphism.

10.3.5 IsGeneralizedIsomorphism (for maps)

◊ IsGeneralizedIsomorphism(*phi*) (property)

Returns: true or false

Check if *phi* is a generalized isomorphism.

10.3.6 IsIdentityMorphism (for maps)

◊ IsIdentityMorphism(*phi*) (property)

Returns: true or false

Check if the homalg map *phi* is the identity morphism.

10.3.7 IsMonomorphism (for maps)

◊ IsMonomorphism(*phi*) (property)

Returns: true or false

Check if the homalg map *phi* is a monomorphism.

10.3.8 IsEpimorphism (for maps)

◊ IsEpimorphism(*phi*) (property)

Returns: true or false

Check if the homalg map *phi* is an epimorphism.

10.3.9 IsSplitMonomorphism (for maps)

◊ IsSplitMonomorphism(*phi*) (property)

Returns: true or false

Check if the homalg map *phi* is a split monomorphism.

10.3.10 IsSplitEpimorphism (for maps)

◊ IsSplitEpimorphism(*phi*) (property)

Returns: true or false

Check if the homalg map *phi* is a split epimorphism.

10.3.11 IsIsomorphism (for maps)

◊ IsIsomorphism(*phi*) (property)

Returns: true or false

Check if the homalg map *phi* is an isomorphism.

10.3.12 IsAutomorphism (for maps)

◊ IsAutomorphism(*phi*) (property)

Returns: true or false

Check if the homalg map *phi* is an automorphism.

10.4 Maps: Attributes

10.4.1 Source (for maps)

◊ Source(*phi*) (attribute)

Returns: a homalg module

The source of the homalg map *phi*.

10.4.2 Range (for maps)

◊ Range(*phi*) (attribute)

Returns: a homalg module

The target (range) of the homalg map *phi*.

10.4.3 DegreeOfMorphism (for maps)

◊ DegreeOfMorphism(*phi*) (attribute)

Returns: an integer

The degree of the morphism *phi* of graded modules.

(no method installed)

10.4.4 CokernelEpi (for maps)

◊ CokernelEpi(*phi*) (attribute)

Returns: a homalg map

The natural epimorphism from the Range(*phi*) onto the Cokernel(*phi*) (cf. Cokernel (16.4.2)).

10.4.5 CokernelNaturalGeneralizedIsomorphism (for maps)

◊ CokernelNaturalGeneralizedIsomorphism(*phi*) (attribute)

Returns: a homalg map

The natural generalized isomorphism from the Cokernel(*phi*) onto the Range(*phi*) (cf. Cokernel (16.4.2)).

10.4.6 KernelSubmodule (for maps)

◊ KernelSubmodule(*phi*) (attribute)

Returns: a homalg submodule

This constructor returns the finitely generated kernel of the homalg map *phi* as a submodule of the homalg module Source(*phi*) with generators given by the syzygies of *phi*.

10.4.7 KernelEmb (for maps)

◊ KernelEmb(*phi*) (attribute)

Returns: a homalg map

The natural embedding of the Kernel(*phi*) into the Source(*phi*) (cf. Kernel (16.4.6)).

10.4.8 ImageSubmodule (for maps)

◊ ImageSubmodule(*phi*) (attribute)

Returns: a homalg submodule

This constructor returns the finitely generated image of the homalg map *phi* as a submodule of the homalg module Range(*phi*) with generators given by *phi* applied to the generators of its source module.

10.4.9 ImageModuleEmb (for maps)

◊ ImageModuleEmb(*phi*) (attribute)

Returns: a homalg map

The natural embedding of the ImageModule(*phi*) into the Range(*phi*) (cf. ImageModule (16.4.4)).

10.4.10 ImageModuleEpi (for maps)

◊ ImageModuleEpi(*phi*) (attribute)

Returns: a homalg map

The natural epimorphism from the Source(*phi*) onto the ImageModule(*phi*) (cf. ImageModule (16.4.4)).

10.4.11 MorphismAidMap (for maps)

◊ MorphismAidMap(*phi*) (attribute)

Returns: a homalg map

The morphism aid map of a true generalized map.
(no method installed)

10.5 Maps: Operations and Functions

10.5.1 HomalgRing (for maps)

◊ HomalgRing(*phi*) (operation)

Returns: a homalg ring

The homalg ring of the homalg map *phi*.

Example

```
gap> ZZ := HomalgRingOfIntegers( );
<A homalg internal ring>
gap> phi := HomalgIdentityMap( 2 * ZZ );
<The identity morphism of a left module>
gap> R := HomalgRing( phi );
<A homalg internal ring>
```

```
gap> IsIdenticalObj( R, ZZ );
true
```

10.5.2 ByASmallerPresentation (for maps)

◊ ByASmallerPresentation(*phi*) (method)

Returns: a homalg map

See ByASmallerPresentation (9.5.2) on modules.

Code

```
InstallMethod( ByASmallerPresentation,
        "for homalg maps",
        [ IsMapOfFinitelyGeneratedModulesRep ],

  function( phi )

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

    return phi;

end );
```

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

Example

```
gap> ZZ := HomalgRingOfIntegers( );;
gap> M := HomalgMatrix( "[ 2, 3, 4,   5, 6, 7 ]", 2, 3, ZZ );
<A homalg internal 2 by 3 matrix>
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 homalg internal 2 by 4 matrix>
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 homalg internal 3 by 4 matrix>
gap> phi := HomalgMap( mat, M, N );
<A "homomorphism" of left modules>
gap> IsMorphism( phi );
true
gap> phi;
<A homomorphism of left modules>
gap> Display( phi );
[ [   1,   0,  -2,  -4 ],
  [   0,   1,   4,   7 ],
  [   1,   0,  -2,  -4 ] ]

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

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

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

10.5.3 PreInverse (for maps)

◊ PreInverse(*phi*) (operation)

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

Compute a pre-inverse of the morphism *phi* in case one exists. For a pre-inverse to exist *phi* 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 $R :=$HomalgRing(*phi*). For *non*commutative rings a heuristic method is installed. If it finds a pre-inverse it returns it, otherwise it returns `fail` (→ 1.1.5). The operation `PreInverse` is used to install a method for the property `IsSplitEpimorphism` (10.3.10).

`PreInverse` checks if it can decide the projectivity of Range(*phi*). To decide the projectivity of a module M over a *commutative* ring you can use

IsSplitEpimorphism(FreeHullEpi(M));

Of course you can use IsProjective(M) which triggers other methods.

Chapter 11

Complexes

11.1 Complexes: Category and Representations

11.1.1 IsHomalgComplex

◊ IsHomalgComplex(*C*) (Category)

Returns: true or false

The GAP category of homalg (co)complexes.

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

11.1.2 IsComplexOfFinitelyPresentedObjectsRep

◊ IsComplexOfFinitelyPresentedObjectsRep(*C*) (Representation)

Returns: true or false

The GAP representation of complexes of finitley generated homalg modules.

(It is a representation of the GAP category IsHomalgComplex (11.1.1), which is a subrepresentation of the GAP representation IsFinitelyPresentedObjectRep.)

11.1.3 IsCocomplexOfFinitelyPresentedObjectsRep

◊ IsCocomplexOfFinitelyPresentedObjectsRep(*C*) (Representation)

Returns: true or false

The GAP representation of cocomplexes of finitley generated homalg modules.

(It is a representation of the GAP category IsHomalgComplex (11.1.1), which is a subrepresentation of the GAP representation IsFinitelyPresentedObjectRep.)

11.2 Complexes: Constructors

11.2.1 HomalgComplex (constructor for complexes given a module)

◊ HomalgComplex(*M*[, *d*]) (function)
◊ HomalgComplex(*phi*[, *d*]) (function)
◊ HomalgComplex(*C*[, *d*]) (function)
◊ HomalgComplex(*cm*[, *d*]) (function)

Returns: a homalg complex

The first syntax creates a complex (i.e. chain complex) with the single homalg module *M* (→ 9.2) at (homological) degree *d*.

The second syntax creates a complex with the single homalg map *phi* (→ HomalgMap (10.2.1)), its source placed at (homological) degree *d* (and its target at *d*−1).

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

The fourth syntax creates a complex with the single homalg (co)chain map *cm* (→ HomalgChainMap (12.2.1)), its source placed at (homological) degree *d* (and its target at *d*−1).

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

To add a map (resp. (co)chain map) to a complex use Add (11.5.1).

Example

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

The first possibility:

Example

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

The second possibility:

Example

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

11.2.2 HomalgCocomplex (constructor for cocomplexes given a module)

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

Returns: a homalg complex

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

The second syntax creates a cocomplex with the single homalg map *phi* (→ HomalgMap (10.2.1)), its source placed at (cohomological) degree *d* (and its target at *d*+1).

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

The fourth syntax creates a cocomplex with the single homalg (co)chain map *cm* (→ HomalgChainMap (12.2.1)), its source placed at (cohomological) degree *d* (and its target at *d*+1).

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

To add a map (resp. (co)chain map) to a cocomplex use Add (11.5.1).

Example

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

The first possibility:

Example

```
<A homomorphism of right modules>
gap> C := HomalgCocomplex( M );
<A non-zero graded cohomology object consisting of a single right module at de\
gree 0>
gap> Add( C, phi );
gap> C;
<A cocomplex containing a single morphism of right modules at degrees
[ 0 .. 1 ]>
```

The second possibility:

Example

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

11.3 Complexes: Properties

11.3.1 IsSequence

◊ IsSequence(*C*) (property)

Returns: true or false

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

11.3.2 IsComplex

◊ IsComplex(*C*) (property)

Returns: true or false

Check if *C* is complex.

11.3.3 IsAcyclic

◊ IsAcyclic(*C*) (property)

Returns: true or false

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

11.3.4 IsRightAcyclic

◊ IsRightAcyclic(*C*) (property)

Returns: true or false

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

11.3.5 IsLeftAcyclic

◊ IsLeftAcyclic(*C*) (property)

Returns: true or false

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

11.3.6 IsGradedObject

◊ IsGradedObject(*C*) (property)

Returns: true or false

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

11.3.7 IsExactSequence

◊ IsExactSequence(*C*) (property)

Returns: true or false

Check if the homalg complex *C* is exact.

11.3.8 IsShortExactSequence

◊ IsShortExactSequence(C) (property)

Returns: true or false

Check if the homalg complex C is a short exact sequence.

11.3.9 IsSplitShortExactSequence

◊ IsSplitShortExactSequence(C) (property)

Returns: true or false

Check if the homalg complex C is a split short exact sequence.

11.3.10 IsTriangle

◊ IsTriangle(C) (property)

Returns: true or false

Set to true if the homalg complex C is a triangle.

11.3.11 IsExactTriangle

◊ IsExactTriangle(C) (property)

Returns: true or false

Check if the homalg complex C is an exact triangle.

11.4 Complexes: Attributes

11.4.1 BettiDiagram (for complexes)

◊ BettiDiagram(C) (attribute)

Returns: a homalg diagram

The Betti diagram of the homalg complex C of graded modules.

11.5 Complexes: Operations and Functions

11.5.1 Add (to complexes given a map)

◊ Add(C, phi) (operation)
◊ Add(C, mat) (operation)

Returns: a homalg complex

In the first syntax the map phi is added to the (co)chain complex C (→ 11.2) as the new *highest* degree morphism and the altered argument C is returned. In case C is a chain complex, the highest degree module in C and the target of phi must be *identical*. In case C is a *co*chain complex, the highest degree module in C and the source of phi must be *identical*.

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

case C is a *co*chain complex, the highest degree left (resp. right) module C^d in C is declared as the source of *psi*, while its target is taken to be a free left (resp. right) module of rank equal to NrColumns(*mat*) (resp. NrRows(*mat*)). For this NrRows(*mat*) (resp. Columns(*mat*)) must coincide with the NrGenerators(C^d).

Example

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

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

the map is currently represented by the above 2 x 2 matrix
------------v------------
at homology degree: 0
Z^(1 x 2)
--------------------------
gap> IsComplex( C );
true
gap> IsAcyclic( C );
true
gap> IsExactSequence( C );
false
gap> C;
<A non-zero acyclic complex containing 2 morphisms of left modules at degrees
[ 0 .. 2 ]>
```

11.5.2 ByASmallerPresentation (for complexes)

◊ ByASmallerPresentation(*C*) (method)

Returns: a homalg complex

See ByASmallerPresentation (9.5.2) on modules.

Code

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

  function( C )

    List( ObjectsOfComplex( C ), ByASmallerPresentation );

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

    IsZero( C );

    return C;

end );
```

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

Example

```
gap> ZZ := HomalgRingOfIntegers( );;
gap> M := HomalgMatrix( "[ 2, 3, 4,   5, 6, 7 ]", 2, 3, ZZ );
<A homalg internal 2 by 3 matrix>
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 homalg internal 2 by 4 matrix>
gap> N := LeftPresentation( N );
<A non-torsion left module presented by 2 relations for 4 generators>
gap> mat := HomalgMatrix( "[ \
> 0, 3, 6, 9, \
> 0, 2, 4, 6, \
> 0, 3, 6, 9  \
> ]", 3, 4, ZZ );
<A homalg internal 3 by 4 matrix>
gap> phi := HomalgMap( mat, M, N );
<A "homomorphism" of left modules>
gap> IsMorphism( phi );
true
gap> phi;
<A homomorphism of left modules>
gap> C := HomalgComplex( phi );
<A non-zero acyclic complex containing a single morphism of left modules at de\
grees [ 0 .. 1 ]>
gap> Display( C );
-------------------------
at homology degree: 1
[ [  2,  3,  4 ],
  [  5,  6,  7 ] ]

Cokernel of the map
```

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

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

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

Cokernel of the map

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

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

And now:

Example

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

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

Chapter 12

Chain Maps

12.1 ChainMaps: Categories and Representations

12.1.1 IsHomalgChainMap

◊ IsHomalgChainMap(*cm*) (Category)

Returns: true or false

The GAP category of homalg (co)chain maps.

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

12.1.2 IsHomalgChainSelfMap

◊ IsHomalgChainSelfMap(*cm*) (Category)

Returns: true or false

The GAP category of homalg (co)chain self-maps.

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

12.1.3 IsChainMapOfFinitelyPresentedObjectsRep

◊ IsChainMapOfFinitelyPresentedObjectsRep(*c*) (Representation)

Returns: true or false

The GAP representation of chain maps of finitley generated homalg modules.

(It is a representation of the GAP category IsHomalgChainMap (12.1.1), which is a subrepresentation of the GAP representation IsMorphismOfFinitelyGeneratedModulesRep.)

12.1.4 IsCochainMapOfFinitelyPresentedObjectsRep

◊ IsCochainMapOfFinitelyPresentedObjectsRep(*c*) (Representation)

Returns: true or false

The GAP representation of cochain maps of finitley generated homalg modules.

(It is a representation of the GAP category IsHomalgChainMap (12.1.1), which is a subrepresentation of the GAP representation IsMorphismOfFinitelyGeneratedModulesRep.)

12.2 Chain Maps: Constructors

12.2.1 HomalgChainMap (constructor for chain maps given a map)

◊ HomalgChainMap(*phi[, C][, D][, d]*) (function)

Returns: a homalg chain map

The constructor creates a (co)chain map given a source homalg (co)chain complex *C*, a target homalg (co)chain complex *D* (→ 9.2), and a homalg map *phi* (→ 10.2)at (co)homological degree *d*. The returned (co)chain map will cautiously be indicated using parenthesis: "chain map". To verify if the result is indeed a (co)chain map use IsMorphism (12.3.1). If source and target are identical objects, and only then, the (co)chain map is created as a (co)chain selfmap.

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

Example

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

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

12.3 Chain Maps: Properties

12.3.1 IsMorphism (for chain maps)

◊ IsMorphism(*cm*) (property)

Returns: true or false

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

12.3.2 IsGeneralizedMorphism (for chain maps)

◊ IsGeneralizedMorphism(*cm*) (property)

Returns: true or false

Check if *cm* is a generalized morphism.

12.3.3 IsGeneralizedEpimorphism (for chain maps)

◊ IsGeneralizedEpimorphism(*cm*) (property)

Returns: true or false

Check if *cm* is a generalized epimorphism.

12.3.4 IsGeneralizedMonomorphism (for chain maps)

◊ IsGeneralizedMonomorphism(*cm*) (property)

Returns: true or false

Check if *cm* is a generalized monomorphism.

12.3.5 IsGeneralizedIsomorphism (for chain maps)

◊ IsGeneralizedIsomorphism(*cm*) (property)

Returns: true or false

Check if *cm* is a generalized isomorphism.

12.3.6 IsIdentityMorphism (for chain maps)

◊ IsIdentityMorphism(*cm*) (property)

Returns: true or false

Check if the homalg chain map *cm* is the identity chain map.

12.3.7 IsMonomorphism (for chain maps)

◊ IsMonomorphism(*cm*) (property)

Returns: true or false

Check if the homalg chain map *cm* is a monomorphism.

12.3.8 IsEpimorphism (for chain maps)

◊ IsEpimorphism(*cm*) (property)

Returns: true or false

Check if the homalg chain map *cm* is an epimorphism.

12.3.9 IsSplitMonomorphism (for chain maps)

◊ IsSplitMonomorphism(*cm*) (property)

Returns: true or false

Check if the homalg chain map *cm* is a split monomorphism.

12.3.10 IsSplitEpimorphism (for chain maps)

◊ IsSplitEpimorphism(*cm*) (property)

Returns: true or false

Check if the homalg chain map *cm* is a split epimorphism.

12.3.11 IsIsomorphism (for chain maps)

◊ IsIsomorphism(*cm*) (property)

Returns: true or false

Check if the homalg chain map *cm* is an isomorphism.

12.3.12 IsAutomorphism (for chain maps)

◊ IsAutomorphism(*cm*) (property)

Returns: true or false

Check if the homalg chain map *cm* is an automorphism.

12.3.13 IsGradedMorphism (for chain maps)

◊ IsGradedMorphism(*cm*) (property)

Returns: true or false

Check if the source and target complex of the homalg chain map *cm* are graded objects, i.e. if all their morphisms vanish.

12.3.14 IsQuasiIsomorphism (for chain maps)

◊ IsQuasiIsomorphism(*cm*) (property)

Returns: true or false

Check if the homalg chain map *cm* is a quasi-isomorphism.

12.4 Chain Maps: Attributes

12.4.1 Source (for chain maps)

◊ `Source(`*cm*`)` (attribute)

Returns: a homalg complex

The source of the homalg chain map *cm*.

12.4.2 Range (for chain maps)

◊ `Range(`*cm*`)` (attribute)

Returns: a homalg complex

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

12.5 Chain Maps: Operations and Functions

12.5.1 ByASmallerPresentation (for chain maps)

◊ `ByASmallerPresentation(`*cm*`)` (method)

Returns: a homalg complex

See `ByASmallerPresentation` (11.5.2) on complexes.

Code

```
InstallMethod( ByASmallerPresentation,
        "for homalg chain maps",
        [ IsHomalgChainMap ],

  function( cm )

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

    List( MorphismsOfChainMap( cm ), DecideZero );

    return cm;

end );
```

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

Chapter 13

Bicomplexes

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

13.1 Bicomplexes: Category and Representations

13.1.1 IsHomalgBicomplex

◊ `IsHomalgBicomplex(`*BC*`)` (Category)

Returns: `true` or `false`

The GAP category of homalg bi(co)complexes.

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

13.1.2 IsBicomplexOfFinitelyPresentedObjectsRep

◊ `IsBicomplexOfFinitelyPresentedObjectsRep(`*BC*`)` (Representation)

Returns: `true` or `false`

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

(It is a representation of the GAP category `IsHomalgBicomplex` (13.1.1), which is a subrepresentation of the GAP representation `IsFinitelyPresentedObjectRep`.)

13.1.3 IsBicocomplexOfFinitelyPresentedObjectsRep

◊ `IsBicocomplexOfFinitelyPresentedObjectsRep(`*BC*`)` (Representation)

Returns: `true` or `false`

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

(It is a representation of the GAP category `IsHomalgBicomplex` (13.1.1), which is a subrepresentation of the GAP representation `IsFinitelyPresentedObjectRep`.)

13.2 Bicomplexes: Constructors

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

◊ HomalgBicomplex(*C*) (function)

Returns: a homalg bicomplex

This constructor creates a bicomplex (homological bicomplex) given a homalg complex of (co)complexes *C* (→ HomalgComplex (11.2.1)), resp. creates a bicocomplex (cohomological bicomplex) given a homalg cocomplex of (co)complexes *C* (→ HomalgCocomplex (11.2.2)). Using the usual sign-trick a complex of complexes gives rise to a bicomplex and vice versa.

Example

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

13.3 Bicomplexes: Properties

13.3.1 IsBisequence

◊ IsBisequence(*BC*) (property)

Returns: true or false

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

13.3.2 IsBicomplex

◊ IsBicomplex(*BC*) (property)

Returns: true or false

Check if *BC* is bicomplex.

13.3.3 IsTransposedWRTTheAssociatedComplex

◊ IsTransposedWRTTheAssociatedComplex(*BC*) (property)

Returns: true or false

Check if *BC* is transposed with respect to the associated complex of complexes.
(no method installed).

13.4 Bicomplexes: Attributes

13.4.1 TotalComplex

◊ TotalComplex(*BC*) (attribute)

Returns: a homalg (co)complex

The associated total complex.

13.4.2 SpectralSequence (for bicomplexes)

◊ SpectralSequence(*BC*) (attribute)

Returns: a homalg (co)homological spectral sequence

The associated spectral sequence.

13.5 Bicomplexes: Operations and Functions

13.5.1 UnderlyingComplex

◊ UnderlyingComplex(*BC*) (function)

Returns: a homalg complex

The (co)complex of (co)complexes underlying the (co)homological bicomplex *BC*.

13.5.2 ByASmallerPresentation (for bicomplexes)

◊ ByASmallerPresentation(*B*) (method)

Returns: a homalg bicomplex

See ByASmallerPresentation (11.5.2) on complexes.

Code

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

  function( B )

    ByASmallerPresentation( UnderlyingComplex( B ) );

    IsZero( B );

    return B;

end );
```

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

Chapter 14

Bigraded Objects

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

14.1 BigradedObjects: Categories and Representations

14.1.1 IsHomalgBigradedObject

◊ `IsHomalgBigradedObject(Er)` (Category)

Returns: `true` or `false`

The GAP category of homalg bigraded objects.

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

14.1.2 IsHomalgBigradedObjectAssociatedToAnExactCouple

◊ `IsHomalgBigradedObjectAssociatedToAnExactCouple(Er)` (Category)

Returns: `true` or `false`

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

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

14.1.3 IsHomalgBigradedObjectAssociatedToAFilteredComplex

◊ `IsHomalgBigradedObjectAssociatedToAFilteredComplex(Er)` (Category)

Returns: `true` or `false`

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

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

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

14.1.4 IsHomalgBigradedObjectAssociatedToABicomplex

◊ `IsHomalgBigradedObjectAssociatedToABicomplex(Er)` (Category)

Returns: `true` or `false`

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

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

14.1.5 IsBigradedObjectOfFinitelyPresentedObjectsRep

◊ IsBigradedObjectOfFinitelyPresentedObjectsRep(*Er*) (Representation)

Returns: `true` or `false`

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

(It is a representation of the GAP category `IsHomalgBigradedObject` (14.1.1), which is a subrepresentation of the GAP representation `IsFinitelyPresentedObjectRep`.)

14.2 Bigraded Objects: Constructors

14.2.1 HomalgBigradedObject (constructor for bigraded objects given a bicomplex)

◊ HomalgBigradedObject(*B*) (operation)

Returns: a homalg bigraded object

This constructor creates a homological (resp. cohomological) bigraded object given a homological (resp. cohomological) homalg bicomplex *B* (→ `HomalgBicomplex` (13.2.1)). This is nothing but the level zero sheet (without differential) of the spectral sequence associated to the bicomplex *B*. So it is the double array of homalg objects (i.e. modules or complexes) in *B* forgetting the morphisms.

Example

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

 * *
 * *
```

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

◊ AsDifferentialObject(*Er*) (method)

Returns: a homalg bigraded object

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

For an example see `DefectOfExactness` (14.2.3) below.

14.2.3 DefectOfExactness (for homalg differential bigraded objects)

◇ DefectOfExactness(*Er*) (method)

Returns: a homalg bigraded object

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

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

The differential of the resulting level $r+1$ object can a posteriori be computed using AsDifferentialObject (14.2.2). The objects in the result are subquotients of the objects in *Er*. An object in *Er* (at a spot (p,q)) is called *stable* if no passage to a true subquotient occurs at any higher level. Of course, a zero object (at a spot (p,q)) is always stable.

Example

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

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

Example

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

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

```
gap> Display( I_E1 );
Level 1:

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

 s .
 . .
```

Legend:

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

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

Example

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

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

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

```
<A bigraded object containing left modules at bidegrees [ -1 .. 0 ]x
[ 0 .. 1 ]>
gap> Display( II_E2 );
Level 2:

 s .
 . s
```

14.3 Bigraded Objects: Properties

14.3.1 IsEndowedWithDifferential

◊ IsEndowedWithDifferential(*Er*) (property)

Returns: true or false

Check if *Er* is a differential bigraded object.

(no method installed)

14.3.2 IsStableSheet

◊ IsStableSheet(*Er*) (property)

Returns: true or false

Check if *Er* is stable.

(no method installed)

14.4 Bigraded Objects: Operations and Functions

14.4.1 ByASmallerPresentation (for bigraded objects)

◊ ByASmallerPresentation(*Er*) (method)

Returns: a homalg bigraded object

See ByASmallerPresentation (9.5.2) on modules.

Code

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

  function( Er )

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

    return Er;

end );
```

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

Chapter 15

Spectral Sequences

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

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

15.1 SpectralSequences: Categorie and Representations

15.1.1 IsHomalgSpectralSequence

◊ `IsHomalgSpectralSequence(`*E*`)` (Category)

Returns: `true` or `false`

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

15.1.2 IsHomalgSpectralSequenceAssociatedToAnExactCouple

◊ `IsHomalgSpectralSequenceAssociatedToAnExactCouple(`*E*`)` (Category)

Returns: `true` or `false`

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

15.1.3 IsHomalgSpectralSequenceAssociatedToAFilteredComplex

◊ `IsHomalgSpectralSequenceAssociatedToAFilteredComplex(`*E*`)` (Category)

Returns: `true` or `false`

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

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

15.1.4 IsHomalgSpectralSequenceAssociatedToABicomplex

◊ IsHomalgSpectralSequenceAssociatedToABicomplex(*E*) (Category)

Returns: true or false

The GAP category of homalg associated to a bicomplex.

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

15.1.5 IsSpectralSequenceOfFinitelyPresentedObjectsRep

◊ IsSpectralSequenceOfFinitelyPresentedObjectsRep(*E*) (Representation)

Returns: true or false

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

(It is a representation of the GAP category IsHomalgSpectralSequence (15.1.1), which is a subrepresentation of the GAP representation IsFinitelyPresentedObjectRep.)

15.1.6 IsSpectralCosequenceOfFinitelyPresentedObjectsRep

◊ IsSpectralCosequenceOfFinitelyPresentedObjectsRep(*E*) (Representation)

Returns: true or false

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

(It is a representation of the GAP category IsHomalgSpectralSequence (15.1.1), which is a subrepresentation of the GAP representation IsFinitelyPresentedObjectRep.)

15.2 Spectral Sequences: Constructors

15.2.1 HomalgSpectralSequence (constructor for spectral sequences given a bicomplex)

◊ HomalgSpectralSequence(*r*, *B*, *a*) (operation)
◊ HomalgSpectralSequence(*r*, *B*) (operation)
◊ HomalgSpectralSequence(*B*, *a*) (operation)
◊ HomalgSpectralSequence(*B*) (operation)

Returns: a homalg spectral sequence

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

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

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

In the second syntax a is set to -1.

In the third syntax r is set to -1.

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

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

Example

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

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

Example

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

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

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

 s s
 . .
```

Legend:

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

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

Example

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

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

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

 s s
 . s
```

15.3 Spectral Sequences: Attributes

15.3.1 GeneralizedEmbeddingsInTotalObjects

◊ GeneralizedEmbeddingsInTotalObjects(*E*) (attribute)

Returns: a record containing homalg maps

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

15.3.2 GeneralizedEmbeddingsInTotalDefects

◊ GeneralizedEmbeddingsInTotalDefects(*E*) (attribute)

Returns: a record containing homalg maps

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

15.4 Spectral Sequences: Operations and Functions

15.4.1 ByASmallerPresentation (for spectral sequences)

◊ ByASmallerPresentation(*E*) (method)

Returns: a homalg spectral sequence

See ByASmallerPresentation (14.4.1) on bigraded object.

Code

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

  function( E )

    ByASmallerPresentation( HighestLevelSheetInSpectralSequence( E ) );

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

    return E;

end );
```

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

Chapter 16

Functors

Functors and their natural transformations form the heart of the homalg package. Usually, a functor is realized in computer algebra systems as a procedure which can be applied to a certain type of objects. In [BR08] it was explained how to implement a functor of abelian categories – by itself – as an object which can be further manipulated (composed, derived, ...). So in addition to the constructor `CreateHomalgFunctor` (16.2.1) which is used to create functors from scratch, homalg provides further easy-to-use constructors to create new functors out of existing ones:

- `InsertObjectInMultiFunctor` (16.2.2)
- `RightSatelliteOfCofunctor` (16.2.3)
- `LeftSatelliteOfFunctor` (16.2.4)
- `RightDerivedCofunctor` (16.2.5)
- `LeftDerivedFunctor` (16.2.6)
- `ComposeFunctors` (16.2.7)

In homalg each functor is implemented as a GAP4 object.

So-called installers (→ `InstallFunctor` (16.7.2) and `InstallDeltaFunctor` (16.7.3)) take such a functor object and create operations in order to apply the functor on objects, morphisms, complexes (of objects or again of complexes), and chain maps. The installer `InstallDeltaFunctor` (16.7.3) creates additional operations for δ-functors in order to compute connecting homomorphisms, exact triangles, and associated long exact sequences by starting with a short exact sequence.

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

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

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

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

any functor which returns an object over R will first apply ByASmallerPresentation (9.5.2) to its result before returning it.

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

16.1 Functors: Category and Representations

16.1.1 IsHomalgFunctor

◊ IsHomalgFunctor(*F*) (Category)

Returns: true or false

The GAP category of homalg (multi-)functors.

16.1.2 IsHomalgFunctorRep

◊ IsHomalgFunctorRep(*E*) (Representation)

Returns: true or false

The GAP representation of homalg (multi-)functors.

(It is a representation of the GAP category IsHomalgFunctor (16.1.1).)

16.2 Functors: Constructors

16.2.1 CreateHomalgFunctor (constructor for functors)

◊ CreateHomalgFunctor(*list1, list2, ...*) (function)

Returns: a homalg functor

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

The constructor does *not* invoke InstallFunctor (16.7.2). This has to be done manually!

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

◊ InsertObjectInMultiFunctor(*F, p, obj, H*) (operation)

Returns: a homalg functor

Given a homalg multi-functor *F* with multiplicity m and a string *H* return the functor Functor_*H* := $F(...,obj,...)$, where *obj* is inserted at the p-th position. Of course *obj* must be an object (e.g. ring, module, ...) that can be inserted at this particular position. The string *H* becomes the name of the returned functor (→ NameOfFunctor (16.7.1)). The variable Functor_*H* will automatically be assigned if free, otherwise a warning is issued.

The constructor automatically invokes InstallFunctor (16.7.2) which installs several necessary operations under the name *H*.

Example

```
gap> ZZ := HomalgRingOfIntegers( );;
gap> ZZ * 1;
<The free right module of rank 1 on a free generator>
gap> InsertObjectInMultiFunctor( Functor_Hom, 2, ZZ * 1, "Hom_ZZ" );
<The functor Hom_ZZ>
gap> Functor_Hom_ZZ;        ## got automatically defined
<The functor Hom_ZZ>
gap> Hom_ZZ;                ## got automatically defined
<Operation "Hom_ZZ">
```

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

◊ RightSatelliteOfCofunctor(*F[, p][, H]*) (operation)

Returns: a homalg functor

Given a homalg (multi-)functor *F* and a string *H* return the right satellite of *F* with respect to its *p*-th argument. *F* is assumed contravariant in its *p*-th argument. The string *H* becomes the name of the returned functor (→ NameOfFunctor (16.7.1)). The variable Functor_*H* will automatically be assigned if free, otherwise a warning is issued.

If *p* is not specified it is assumed 1. If the string *H* is not specified the letter 'S' is added to the left of the name of *F* (→ NameOfFunctor (16.7.1)).

The constructor automatically invokes InstallFunctor (16.7.2) which installs several necessary operations under the name *H*.

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

Code

```
RightSatelliteOfCofunctor( Functor_Hom, "Ext" );
```

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

◊ LeftSatelliteOfFunctor(*F[, p][, H]*) (operation)

Returns: a homalg functor

Given a homalg (multi-)functor *F* and a string *H* return the left satellite of *F* with respect to its *p*-th argument. *F* is assumed covariant in its *p*-th argument. The string *H* becomes the name of the returned functor (→ NameOfFunctor (16.7.1)). The variable Functor_*H* will automatically be assigned if free, otherwise a warning is issued.

If *p* is not specified it is assumed 1. If the string *H* is not specified the string "S_" is added to the left of the name of *F* (→ NameOfFunctor (16.7.1)).

The constructor automatically invokes InstallFunctor (16.7.2) which installs several necessary operations under the name *H*.

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

Code

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

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

◊ RightDerivedCofunctor(*F[, p][, H]*) (operation)

Returns: a homalg functor

Given a homalg (multi-)functor *F* and a string *H* return the right derived functor of *F* with respect to its *p*-th argument. *F* is assumed contravariant in its *p*-th argument. The string *H* becomes the name of the returned functor (→ NameOfFunctor (16.7.1)). The variable Functor_*H* will automatically be assigned if free, otherwise a warning is issued.

If *p* is not specified it is assumed 1. If the string *H* is not specified the letter 'R' is added to the left of the name of *F* (→ NameOfFunctor (16.7.1)).

The constructor automatically invokes InstallFunctor (16.7.2) and InstallDeltaFunctor (16.7.3) which install several necessary operations under the name *H*.

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

Code

```
RightDerivedCofunctor( Functor_Hom );
```

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

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

Returns: a homalg functor

Given a homalg (multi-)functor *F* and a string *H* return the left derived functor of *F* with respect to its *p*-th argument. *F* is assumed covariant in its *p*-th argument. The string *H* becomes the name of the returned functor (→ NameOfFunctor (16.7.1)). The variable Functor_*H* will automatically be assigned if free, otherwise a warning is issued.

If *p* is not specified it is assumed 1. If the string *H* is not specified the letter "S_" is added to the left of the name of *F* (→ NameOfFunctor (16.7.1)).

The constructor automatically invokes InstallFunctor (16.7.2) and InstallDeltaFunctor (16.7.3) which install several necessary operations under the name *H*.

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

Code

```
LeftDerivedFunctor( Functor_TensorProduct );
```

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

Code

```
LeftDerivedFunctor( Functor_HomHom );
```

16.2.7 ComposeFunctors (constructor for functors given two functors)

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

Returns: a homalg functor

Given two homalg (multi-)functors *F* and *G* and a string *H* return the composed functor Functor_*H* := $F(\ldots,G(\ldots),\ldots)$, where *G* is inserted at the *p*-th position. Of course *G* must be a functor that can

be inserted at this particular position. The string *H* becomes the name of the returned functor (→ NameOfFunctor (16.7.1)). The variable Functor_*H* will automatically be assigned if free, otherwise a warning is issued.

If *p* is not specified it is assumed 1. If the string *H* is not specified the names of *F* and *G* are concatenated in this order (→ NameOfFunctor (16.7.1)).

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

The constructor automatically invokes InstallFunctor (16.7.2) which installs several necessary operations under the name *H*.

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

Code

```
Functor_Hom * Functor_Hom;
```

Check this:

Example

```
gap> Functor_Hom * Functor_TensorProduct;
<The functor HomTensorProduct>
gap> Functor_HomTensorProduct;        ## got automatically defined
<The functor HomTensorProduct>
gap> HomTensorProduct;                ## got automatically defined
<Operation "HomTensorProduct">
```

16.3 Functors: Attributes

16.3.1 Genesis

◊ Genesis(*F*) (attribute)

Returns: a list

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

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

- CreateHomalgFunctor:

Example

```
gap> Functor_Hom;
<The functor Hom>
gap> Genesis( Functor_Hom );
[ "CreateHomalgFunctor", [ "name", "Hom" ], [ "number_of_arguments", 2 ],
  [ "1", [ [ "contravariant", "right adjoint", "distinguished" ] ] ],
  [ "2", [ [ "covariant", "left exact" ] ] ],
  [ "OnObjects", function( M, N ) ... end ],
  [ "OnMorphisms", function( M_or_mor, N_or_mor ) ... end ] ]
```

- InsertObjectInMultiFunctor:

Example

```
gap> ZZ := HomalgRingOfIntegers( );;
gap> LeftDualizingFunctor( ZZ, "ZZ_Hom" );
<The functor ZZ_Hom>
gap> Functor_ZZ_Hom;        ## got automatically defined
<The functor ZZ_Hom>
```

```
gap> ZZ_Hom;                ## got automatically defined
<Operation "ZZ_Hom">
gap> Genesis( Functor_ZZ_Hom );
[ "InsertObjectInMultiFunctor", <The functor Hom>, 2,
  <The free left module of rank 1 on a free generator> ]
gap> 1 * ZZ;
<The free left module of rank 1 on a free generator>
```

- LeftDerivedFunctor:

Example

```
gap> Functor_TensorProduct;
<The functor TensorProduct>
gap> Genesis( Functor_LTensorProduct );
[ "LeftDerivedFunctor", <The functor TensorProduct>, 1 ]
```

- RightDerivedCofunctor:

Example

```
gap> Genesis( Functor_RHom );
[ "RightDerivedCofunctor", <The functor Hom>, 1 ]
```

- LeftSatelliteOfFunctor:

Example

```
gap> Genesis( Functor_Tor );
[ "LeftSatelliteOfFunctor", <The functor TensorProduct>, 1 ]
```

- RightSatelliteOfCofunctor:

Example

```
gap> Genesis( Functor_Ext );
[ "RightSatelliteOfCofunctor", <The functor Hom>, 1 ]
```

- ComposeFunctors:

Example

```
gap> Genesis( Functor_HomHom );
[ "ComposeFunctors", [ <The functor Hom>, <The functor Hom> ], 1 ]
gap> ValueGlobal( "ComposeFunctors" );
<Operation "ComposeFunctors">
```

16.4 Basic Functors

16.4.1 functor_Cokernel

◊ functor_Cokernel (global variable)

The functor that associates to a map its cokernel.

Code

```
InstallValue( functor_Cokernel,
        CreateHomalgFunctor(
                [ "name", "Cokernel" ],
                [ "natural_transformation", "CokernelEpi" ],
                [ "special", true ],
```

```
                [ "number_of_arguments", 1 ],
                [ "1", [ [ "covariant" ],
                        [ IsMapOfFinitelyGeneratedModulesRep,
                          [ IsHomalgChainMap, IsImageSquare ] ] ] ],
                [ "OnObjects", _Functor_Cokernel_OnObjects ]
                )
        );
```

16.4.2 Cokernel

◊ `Cokernel(`*phi*`)` (operation)

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

Example

```
gap> ZZ := HomalgRingOfIntegers( );;
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 );
<An 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
```

16.4.3 functor_ImageModule

◊ functor_ImageModule (global variable)

The functor that associates to a map its image.

Code

```
InstallValue( functor_ImageModule,
        CreateHomalgFunctor(
                [ "name", "ImageModule" ],
                [ "natural_transformation", "ImageModuleEmb" ],
                [ "number_of_arguments", 1 ],
                [ "1", [ [ "covariant" ],
                        [ IsMapOfFinitelyGeneratedModulesRep ] ] ],
                [ "OnObjects", _Functor_ImageModule_OnObjects ]
                )
        );
```

16.4.4 ImageModule

◊ ImageModule(*phi*) (operation)

The following example also makes use of the natural transformations ImageModuleEpi and ImageModuleEmb.

Example

```
gap> ZZ := HomalgRingOfIntegers( );;
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 := ImageModule( 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 := ImageModuleEpi( phi );
<A split epimorphism of left modules>
gap> epsilon := ImageModuleEmb( phi );
<A monomorphism of left modules>
gap> phi = pi * epsilon;
true
```

16.4.5 functor_Kernel

◊ functor_Kernel (global variable)

The functor that associates to a map its kernel.

Code

```
InstallValue( functor_Kernel,
        CreateHomalgFunctor(
                [ "name", "Kernel" ],
                [ "natural_transformation", "KernelEmb" ],
                [ "special", true ],
                [ "number_of_arguments", 1 ],
                [ "1", [ [ "covariant" ],
                        [ IsMapOfFinitelyGeneratedModulesRep,
                          [ IsHomalgChainMap, IsKernelSquare ] ] ] ],
                [ "OnObjects", _Functor_Kernel_OnObjects ]
                )
        );
```

16.4.6 Kernel (for maps)

◊ Kernel(*phi*) (operation)

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

Example

```
gap> ZZ := HomalgRingOfIntegers( );;
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 generato\
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,  2,  4 ] ]

the map is currently represented by the above 1 x 3 matrix
gap> DefectOfExactness( iota, phi );
<A zero left module>
gap> ByASmallerPresentation( iota );
<A monomorphism of left modules>
gap> Display( iota );
[ [  2,  0 ] ]

the map is currently represented by the above 1 x 2 matrix
gap> PostInverse( iota );
false
```

16.4.7 functor_DefectOfExactness

◊ functor_DefectOfExactness (global variable)

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

Code

```
InstallValue( functor_DefectOfExactness,
        CreateHomalgFunctor(
                [ "name", "DefectOfExactness" ],
                [ "special", true ],
                [ "number_of_arguments", 1 ],
                [ "1", [ [ "covariant" ],
                        [ IsHomalgComplex and IsATwoSequence,
                          [ IsHomalgChainMap, IsLambekPairOfSquares ] ] ] ],
                [ "OnObjects", _Functor_DefectOfExactness_OnObjects ]
                )
        );
```

16.4.8 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( );;
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 >
```

16.4.9 Functor_Hom

◊ Functor_Hom (global variable)

The bifunctor Hom.

Code

```
InstallValue( Functor_Hom,
        CreateHomalgFunctor(
                [ "name", "Hom" ],
                [ "number_of_arguments", 2 ],
                [ "1", [ [ "contravariant", "right adjoint", "distinguished" ] ] ],
                [ "2", [ [ "covariant", "left exact" ] ] ],
                [ "OnObjects", _Functor_Hom_OnObjects ],
                [ "OnMorphisms", _Functor_Hom_OnMorphisms ]
                )
        );
```

16.4.10 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 map.

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

Example

```
gap> ZZ := HomalgRingOfIntegers( );;
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 homomorphism of right modules>
gap> Display( psi );
[ [   1,   1,   0,   1 ],
  [   2,   2,   0,   0 ],
  [   0,   0,   6,  10 ] ]

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

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

[ [   0,   1,   3 ],
  [   0,   0,  -2 ],
  [   0,   1,   3 ] ]

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 ] ]

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

[ [   0,   0,  -3 ],
  [   0,   0,   7 ],
  [   0,   0,  -5 ],
  [   0,   0,   1 ] ]

[ [   0,   1,  -3 ],
  [   0,   0,  12 ],
  [   0,   0,  -9 ],
  [   0,   2,   6 ] ]
```

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 ] ]

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

[ [   0,   0,   0 ],
```

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

[ [   0,   9,  18 ],
  [   0,   0,  -9 ],
  [   0,   2,   6 ] ]
```

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

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 );
<An acyclic cocomplex containing a single morphism of right complexes at degre\
es [ 0 .. 1 ]>
gap> BMM := HomalgBicomplex( hMM );
<A bicocomplex 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
1 relation>
   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 non-torsion 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)
```

16.4.11 Functor_TensorProduct

◊ Functor_TensorProduct (global variable)

The tensor product bifunctor.

Code

```
InstallValue( Functor_TensorProduct,
        CreateHomalgFunctor(
                [ "name", "TensorProduct" ],
                [ "number_of_arguments", 2 ],
                [ "1", [ [ "covariant", "left adjoint", "distinguished" ] ] ],
                [ "2", [ [ "covariant", "left adjoint" ] ] ],
                [ "OnObjects", _Functor_TensorProduct_OnObjects ],
                [ "OnMorphisms", _Functor_TensorProduct_OnMorphisms ]
                )
        );
```

16.4.12 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 map.

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

Example

```
gap> ZZ := HomalgRingOfIntegers( );;
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> L := Hom( ZZ, M );
<A 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;        ## the display method found out further information about the module 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 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 non-torsion right module on 4 generators satisfying 3 relations>
gap> IsIdenticalObj( ML, M * L );        ## the caching at work
true
gap> NL := Range( psi );
<A non-torsion 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 );
<An 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 degre\
es [ 0 .. 1 ]>
gap> FCE := Hom( CE, L );
<An acyclic complex containing a single morphism of left cocomplexes at degree\
s [ 0 .. 1 ]>
gap> BC := HomalgBicomplex( FCE );
<A 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 non-torsion 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 10 relations for 6 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 left module presented by 2 relations for 2 generators>
of
<A non-torsion 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)
```

16.4.13 Functor_Ext

◊ Functor_Ext (global variable)

The bifunctor Ext.

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

Code

```
RightSatelliteOfCofunctor( Functor_Hom, "Ext" );
```

16.4.14 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 map. 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( );;
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 := TorsionSubmodule( M );
<A cyclic torsion left module presented by yet unknown relations for a cyclic \
generator>
gap> iota := TorsionSubmoduleEmb( M );
<A monomorphism of left modules>
gap> psi := Ext( 1, iota, N );
<A homomorphism of right modules>
gap> ByASmallerPresentation( psi );
<A homomorphism of right modules>
gap> Display( psi );
[ [  2 ] ]

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

16.4.15 Functor_Tor

◊ Functor_Tor (global variable)

The bifunctor Tor.

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

Code

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

16.4.16 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 map. 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( );;
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 := TorsionSubmodule( M );
<A cyclic torsion left module presented by yet unknown relations for a cyclic \
generator>
gap> iota := TorsionSubmoduleEmb( M );
<A monomorphism of left modules>
gap> psi := Tor( 1, iota, N );
<A homomorphism of left modules>
gap> ByASmallerPresentation( psi );
<A 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 cyclic torsion left module presented by 1 relation for a cyclic generator>
gap> IsIdenticalObj( torNN, Tor( 1, N, N ) );        ## the caching at work
true
gap> torMN := Range( psi );
<A cyclic torsion left module presented by 1 relation for a cyclic generator>
gap> IsIdenticalObj( torMN, Tor( 1, M, N ) );        ## the caching at work
true
gap> Display( torNN );
Z/< 3 >
gap> Display( torMN );
Z/< 3 >
```

16.4.17 Functor_RHom

◊ `Functor_RHom` (global variable)

The bifunctor `RHom`.

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

Code

```
RightDerivedCofunctor( Functor_Hom );
```

16.4.18 RHom

◊ `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 map. The string `str` may take different values:

- If `str`="a" then $R^iHom(o1,o2)$ 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 $R^iHom(o1,o2)$ 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( );;
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> 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 left module presented by yet unknown relations for a cyclic generato\
r>
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,  0 ] ]

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 ]>
```

16.4.19 Functor_LTensorProduct

◊ Functor_LTensorProduct (global variable)

The bifunctor LTensorProduct.

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

Code

```
LeftDerivedFunctor( Functor_TensorProduct );
```

16.4.20 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 map. The string *str* may take different values:

- If *str*="a" then $L_i TensorProduct(o1,o2)$ 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 $L_i TensorProduct(o1,o2)$ 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( );;
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> 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 left module presented by yet unknown relations for a cyclic generato\
r>
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 >
-------------------------
[ [  1,  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,  1 ] ]

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 ]>
```

16.4.21 Functor_HomHom

◊ Functor_HomHom (global variable)

The bifunctor HomHom.

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

Code

```
Functor_Hom * Functor_Hom;
```

16.4.22 Functor_LHomHom

◊ Functor_LHomHom (global variable)

The bifunctor LHomHom.

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

Code

```
LeftDerivedFunctor( Functor_HomHom );
```

16.5 Tool Functors

16.6 Other Functors

16.7 Functors: Operations and Functions

16.7.1 NameOfFunctor

◊ NameOfFunctor(*F*) (operation)

Returns: a string

The name of the homalg functor *F*.

Example

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

16.7.2 InstallFunctor

◊ InstallFunctor(*F*) (operation)

Install several methods for GAP operations that get declared under the name of the homalg (multi-)functor *F* (→ NameOfFunctor (16.7.1)). These methods are used to apply the functor to objects, morphisms, (co)complexes of objects, and (co)chain maps. The objects in the (co)complexes might again be (co)complexes.

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

Code

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

  function( Functor )

    InstallFunctorOnObjects( Functor );

    if IsSpecialFunctor( Functor ) then

        InstallSpecialFunctorOnMorphisms( Functor );

    else

        InstallFunctorOnMorphisms( Functor );

        InstallFunctorOnComplexes( Functor );

        InstallFunctorOnChainMaps( Functor );

    fi;

end );
```

The method does not return anything.

16.7.3 InstallDeltaFunctor

◊ InstallDeltaFunctor(*F*) (operation)

In case *F* is a δ-functor in the sense of Grothendieck the procedure installs several operations under the name of the homalg (multi-)functor *F* (→ NameOfFunctor (16.7.1)) allowing one to compute connecting homomorphisms, exact triangles, and associated long exact sequences. The input of these operations is a short exact sequence.

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

Code

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

```
  function( Functor )
    local number_of_arguments;

    number_of_arguments := MultiplicityOfFunctor( Functor );

    if number_of_arguments = 1 then

        HelperToInstallUnivariateDeltaFunctor( Functor );

    elif number_of_arguments = 2 then

        HelperToInstallFirstArgumentOfBivariateDeltaFunctor( Functor );
        HelperToInstallSecondArgumentOfBivariateDeltaFunctor( Functor );

    elif number_of_arguments = 3 then

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

    fi;

end );
```

The method does not return anything.

Chapter 17

Examples

17.1 ExtExt

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

Example

```
gap> ZZ := HomalgRingOfIntegers( );;
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 homalg internal 5 by 4 matrix>
gap> M := LeftPresentation( imat );
<A left module presented by 5 relations for 4 generators>
gap> N := Hom( ZZ, M );
<A right module on 4 generators satisfying yet unknown relations>
gap> F := InsertObjectInMultiFunctor( Functor_Hom, 2, N, "TensorN" );
<The functor TensorN>
gap> G := LeftDualizingFunctor( ZZ );;
gap> II_E := GrothendieckSpectralSequence( F, G, M );
<A stable homological spectral sequence with sheets at levels
[ 0 .. 2 ] each consisting of left modules at bidegrees [ -1 .. 0 ]x
[ 0 .. 1 ]>
gap> Display( II_E );
The associated transposed spectral sequence:

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

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

 * *
```

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

 s s
 . .

Now the spectral sequence of the bicomplex:

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

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

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

 s s
 . s
gap> filt := FiltrationBySpectralSequence( II_E, 0 );
<An ascending filtration with degrees [ -1 .. 0 ] and graded parts:
   0:        <A non-torsion left module presented by 3 relations for 4 generators>
  -1:        <A non-zero left module presented by 33 relations for 8 generators>
of
<A non-zero left module presented by 27 relations for 19 generators>>
gap> ByASmallerPresentation( filt );
<An ascending filtration with degrees [ -1 .. 0 ] and graded parts:
   0:        <A non-torsion left module presented by 2 relations for 3 generators>
  -1:        <A non-zero left module presented by 6 relations for 6 generators>
of
<A non-torsion left module presented by 8 relations for 9 generators>>
gap> m := IsomorphismOfFiltration( filt );
<An isomorphism of left modules>
```

17.2 Purity

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

Example

```
gap> ZZ := HomalgRingOfIntegers( );;
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 homalg internal 5 by 4 matrix>
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 5 relations for 4 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 );
<An 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 >
```

17.3 TorExt-Grothendieck

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

Example

```
gap> ZZ := HomalgRingOfIntegers( );;
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 homalg internal 5 by 4 matrix>
gap> M := LeftPresentation( imat );
<A left module presented by 5 relations for 4 generators>
gap> F := InsertObjectInMultiFunctor( Functor_TensorProduct, 2, M, "TensorM" );
<The functor TensorM>
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 8 relations for 8 generators>
   0:        <A non-torsion left module presented by 3 relations for 4 generators>
of
<A left module presented by yet unknown relations for 29 generators>>
gap> ByASmallerPresentation( filt );
<A descending filtration with degrees [ -1 .. 0 ] and graded parts:
  -1:        <A non-zero left module presented by 4 relations for 4 generators>
   0:        <A non-torsion left module presented by 2 relations for 3 generators>
of
<A non-torsion left module presented by 6 relations for 7 generators>>
gap> m := IsomorphismOfFiltration( filt );
```

```
<An isomorphism of left modules>
```

17.4 TorExt

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

Example

```
gap> ZZ := HomalgRingOfIntegers( );;
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 homalg internal 5 by 4 matrix>
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 );
<An acyclic cocomplex containing a single morphism of right modules at degrees\
 [ 0 .. 1 ]>
gap> FGP := GP * P;
<A non-zero acyclic cocomplex containing a single morphism of left complexes a\
t degrees [ 0 .. 1 ]>
gap> BC := HomalgBicomplex( FGP );
<A non-zero bicocomplex containing left modules at bidegrees [ 0 .. 1 ]x
[ -1 .. 0 ]>
gap> p_degrees := ObjectDegreesOfBicomplex( BC )[1];
[ 0, 1 ]
gap> II_E := SecondSpectralSequenceWithFiltration( BC, p_degrees );
<A stable cohomological spectral sequence with sheets at levels
[ 0 .. 2 ] each consisting of left modules at bidegrees [ -1 .. 0 ]x
[ 0 .. 1 ]>
gap> Display( II_E );
The associated transposed spectral sequence:

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

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

 * *
 . .
---------
```

```
Level 2:

 s s
 . .

Now the spectral sequence of the bicomplex:

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

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

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

 s s
 . s
gap> filt := FiltrationBySpectralSequence( II_E, 0 );
<A descending filtration with degrees [ -1 .. 0 ] and graded parts:
  -1:        <A non-zero left module presented by 10 relations for 10 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 left module presented by 4 relations for 4 generators>
   0:        <A rank 1 left module presented by 2 relations for 3 generators>
of
<A non-torsion left module presented by 6 relations for 7 generators>>
gap> m := IsomorphismOfFiltration( filt );
<An isomorphism of left modules>
```

Appendix A

Development

A.1 Why was homalg discontinued in Maple?

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

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

A.2 Why GAP4?

A.2.1 GAP is free and open software

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

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

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

I had the pleasure of being one of his students.

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

A.2.2 GAP has an area of expertise

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

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

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

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

A.2.3 GAP4 can communicate

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

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

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

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

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

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

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

A.2.5 GAP4 packages are easily extendible

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

A.3 Why not Sage?

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

A.4 How does homalg compare to Sage?

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

A.4.1 They differ in objectives and scale

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

A.4.2 They differ in the programming language

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

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

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

external computer algebra systems in the background and tries to understand the internals of the objects residing in them. An object in the external computer algebra system is wrapped by an object in Sage and supporting this external object involves understanding its details in the external system. homalg follows a different strategy: The only external objects homalg needs (beside rings) are non-empty matrices. And being zero or not is basically the only thing homalg wants to know about a matrix after knowing its dimension. I myself was stunned by this insight, which culminated in *the principle of least communication* (→ 1.1.10).

In particular, Sage can make use of all of homalg, but for in order to make full use, Sage needs to understand the internals of the homalg objects. On the contrary, homalg can only make limited use of Sage (or of virtually any computer algebra system that supports rings in a sufficient way (→ 1.1.4)), but without the need to delve into the inner life of the Sage objects.

Appendix B

The Mathematical Idea behind homalg

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 constructive one needs to find apropriate 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 apropriate 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. $AX = B$ (→ 1.1.5).

The two appendices C and D provide the list of matrix operations needed by homalg to perform homological computations. Subsection 1.1.3 explains how these matrix operations can be delegated to external systems.

Appendix C

The Basic Matrix Operations

These are the operations used to solve one-sided (in)homogeneous linear systems $XA = B$ resp. $AX = B$ ($\rightarrow$ Appendix B).

C.1 Main

- `BasisOfRowModule` (6.5.17)
- `BasisOfColumnModule` (6.5.18)
- `DecideZeroRows` (6.5.19)
- `DecideZeroColumns` (6.5.20)
- `SyzygiesGeneratorsOfRows` (6.5.21)
- `SyzygiesGeneratorsOfColumns` (6.5.22)

C.2 Effective

- `BasisOfRowsCoeff` (6.5.29)
- `BasisOfColumnsCoeff` (6.5.30)
- `DecideZeroRowsEffectively` (6.5.31)
- `DecideZeroColumnsEffectively` (6.5.32)

C.3 Relative

- `SyzygiesGeneratorsOfRows` (6.5.23)
- `SyzygiesGeneratorsOfColumns` (6.5.24)

C.4 Reduced

- ReducedBasisOfRowModule (6.5.25)
- ReducedBasisOfColumnModule (6.5.26)
- ReducedSyzygiesGeneratorsOfRows (6.5.27)
- ReducedSyzygiesGeneratorsOfColumns (6.5.28)

Appendix D

The Matrix Tool Operations

The functions listed below are components of the homalgTable object stored in the ring. They are only indirectly accessible through standard methods that invoke them.

D.1 The Tool Operations *without* a Fallback Method

There are matrix methods for which homalg needs a homalgTable entry for non-internal rings, as it cannot provide a suitable fallback. Below is the list of these homalgTable entries.

D.1.1 ZeroMatrix (homalgTable entry for initial matrices)

◊ ZeroMatrix(*C*) (function)

Returns: the Eval value of a homalg matrix *C*

Let *R* := HomalgRing(*C*) and *RP* := homalgTable(*R*). If the homalgTable component *RP*!.ZeroMatrix is bound then the method Eval (E.4.1) resets the filter IsInitialMatrix and returns *RP*!.ZeroMatrix(*C*).

D.1.2 IdentityMatrix (homalgTable entry for initial identity matrices)

◊ IdentityMatrix(*C*) (function)

Returns: the Eval value of a homalg matrix *C*

Let *R* := HomalgRing(*C*) and *RP* := homalgTable(*R*). If the homalgTable component *RP*!.IdentityMatrix is bound then the method Eval (E.4.2) resets the filter IsInitialIdentityMatrix and returns *RP*!.IdentityMatrix(*C*).

D.1.3 ZeroMatrix (homalgTable entry)

◊ ZeroMatrix(*C*) (function)

Returns: the Eval value of a homalg matrix *C*

Let *R* := HomalgRing(*C*) and *RP* := homalgTable(*R*). If the homalgTable component *RP*!.ZeroMatrix is bound then the method Eval (E.4.3) returns *RP*!.ZeroMatrix(*C*).

D.1.4 IdentityMatrix (homalgTable entry)

◊ IdentityMatrix(*C*) (function)

Returns: the Eval value of a homalg matrix *C*

Let *R* := HomalgRing(*C*) and *RP* := homalgTable(*R*). If the homalgTable component *RP*!.IdentityMatrix is bound then the method Eval (E.4.4) returns *RP*!.IdentityMatrix(*C*).

D.1.5 Involution (homalgTable entry)

◊ Involution(*M*) (function)

Returns: the Eval value of a homalg matrix *C*

Let *R* := HomalgRing(*C*) and *RP* := homalgTable(*R*). If the homalgTable component *RP*!.Involution is bound then the method Eval (E.4.5) returns *RP*!.Involution applied to the content of the attribute EvalInvolution(*C*) = *M*.

D.1.6 CertainRows (homalgTable entry)

◊ CertainRows(*M*, *plist*) (function)

Returns: the Eval value of a homalg matrix *C*

Let *R* := HomalgRing(*C*) and *RP* := homalgTable(*R*). If the homalgTable component *RP*!.CertainRows is bound then the method Eval (E.4.6) returns *RP*!.CertainRows applied to the content of the attribute EvalCertainRows(*C*) = [*M*, *plist*].

D.1.7 CertainColumns (homalgTable entry)

◊ CertainColumns(*M*, *plist*) (function)

Returns: the Eval value of a homalg matrix *C*

Let *R* := HomalgRing(*C*) and *RP* := homalgTable(*R*). If the homalgTable component *RP*!.CertainColumns is bound then the method Eval (E.4.7) returns *RP*!.CertainColumns applied to the content of the attribute EvalCertainColumns(*C*) = [*M*, *plist*].

D.1.8 UnionOfRows (homalgTable entry)

◊ UnionOfRows(*A*, *B*) (function)

Returns: the Eval value of a homalg matrix *C*

Let *R* := HomalgRing(*C*) and *RP* := homalgTable(*R*). If the homalgTable component *RP*!.UnionOfRows is bound then the method Eval (E.4.8) returns *RP*!.UnionOfRows applied to the content of the attribute EvalUnionOfRows(*C*) = [*A*, *B*].

D.1.9 UnionOfColumns (homalgTable entry)

◊ UnionOfColumns(*A*, *B*) (function)

Returns: the Eval value of a homalg matrix *C*

Let *R* := HomalgRing(*C*) and *RP* := homalgTable(*R*). If the homalgTable component *RP*!.UnionOfColumns is bound then the method Eval (E.4.9) returns *RP*!.UnionOfColumns applied to the content of the attribute EvalUnionOfColumns(*C*) = [*A*, *B*].

D.1.10 DiagMat (homalgTable entry)

◊ DiagMat(e) (function)

Returns: the Eval value of a homalg matrix C

Let R := HomalgRing(C) and RP := homalgTable(R). If the homalgTable component RP!.DiagMat is bound then the method Eval (E.4.10) returns RP!.DiagMat applied to the content of the attribute EvalDiagMat(C) = e.

D.1.11 KroneckerMat (homalgTable entry)

◊ KroneckerMat(A, B) (function)

Returns: the Eval value of a homalg matrix C

Let R := HomalgRing(C) and RP := homalgTable(R). If the homalgTable component RP!.KroneckerMat is bound then the method Eval (E.4.11) returns RP!.KroneckerMat applied to the content of the attribute EvalKroneckerMat(C) = [A, B].

D.1.12 MulMat (homalgTable entry)

◊ MulMat(a, A) (function)

Returns: the Eval value of a homalg matrix C

Let R := HomalgRing(C) and RP := homalgTable(R). If the homalgTable component RP!.MulMat is bound then the method Eval (E.4.12) returns RP!.MulMat applied to the content of the attribute EvalMulMat(C) = [a, A].

D.1.13 AddMat (homalgTable entry)

◊ AddMat(A, B) (function)

Returns: the Eval value of a homalg matrix C

Let R := HomalgRing(C) and RP := homalgTable(R). If the homalgTable component RP!.AddMat is bound then the method Eval (E.4.13) returns RP!.AddMat applied to the content of the attribute EvalAddMat(C) = [A, B].

D.1.14 SubMat (homalgTable entry)

◊ SubMat(A, B) (function)

Returns: the Eval value of a homalg matrix C

Let R := HomalgRing(C) and RP := homalgTable(R). If the homalgTable component RP!.SubMat is bound then the method Eval (E.4.14) returns RP!.SubMat applied to the content of the attribute EvalSubMat(C) = [A, B].

D.1.15 Compose (homalgTable entry)

◊ Compose(A, B) (function)

Returns: the Eval value of a homalg matrix C

Let R := HomalgRing(C) and RP := homalgTable(R). If the homalgTable component RP!.Compose is bound then the method Eval (E.4.15) returns RP!.Compose applied to the content of the attribute EvalCompose(C) = [A, B].

D.1.16 IsZeroMatrix (homalgTable entry)

◊ IsZeroMatrix(*M*) (function)

Returns: true or false

Let *R* := HomalgRing(*M*) and *RP* := homalgTable(*R*). If the homalgTable component *RP*!.IsZeroMatrix is bound then the standard method for the property IsZero (6.3.1) shown below returns *RP*!.IsZeroMatrix(*M*).

Code

```
InstallMethod( IsZero,
        "for homalg matrices",
        [ IsHomalgMatrix ],

  function( M )
    local R, RP;

    R := HomalgRing( M );

    RP := homalgTable( R );

    if IsBound(RP!.IsZeroMatrix) then
        ## CAUTION: the external system must be able
        ## to check zero modulo possible ring relations!

        return RP!.IsZeroMatrix( M ); ## with this, \= can fall back to IsZero
    fi;

    #=====# the fallback method #=====#

    ## from the GAP4 documentation: ?Zero
    ## 'ZeroSameMutability( <obj> )' is equivalent to '0 * <obj>'.

    return M = 0 * M; ## hence, by default, IsZero falls back to \= (see below)

end );
```

D.1.17 NrRows (homalgTable entry)

◊ NrRows(*C*) (function)

Returns: a nonnegative integer

Let *R* := HomalgRing(*C*) and *RP* := homalgTable(*R*). If the homalgTable component *RP*!.NrRows is bound then the standard method for the attribute NrRows (6.4.1) shown below returns *RP*!.NrRows(*C*).

Code

```
InstallMethod( NrRows,
        "for homalg matrices",
        [ IsHomalgMatrix ],

  function( C )
    local R, RP;

    R := HomalgRing( C );
```

```
    RP := homalgTable( R );

    if IsBound(RP!.NrRows) then
        return RP!.NrRows( C );
    fi;

    if not IsHomalgInternalMatrixRep( C ) then
        Error( "could not find a procedure called NrRows in the ",
               "homalgTable to apply to a non-internal matrix\n" );
    fi;

    #=====# can only work for homalg internal matrices #=====#

    return Length( Eval( C )!.matrix );

end );
```

D.1.18 NrColumns (homalgTable entry)

◇ NrColumns(*C*) (function)

Returns: a nonnegative integer

Let $R :=$ HomalgRing(C) and $RP :=$ homalgTable(R). If the homalgTable component RP!.NrColumns is bound then the standard method for the attribute NrColumns (6.4.2) shown below returns RP!.NrColumns(C).

Code

```
InstallMethod( NrColumns,
        "for homalg matrices",
        [ IsHomalgMatrix ],

  function( C )
    local R, RP;

    R := HomalgRing( C );

    RP := homalgTable( R );

    if IsBound(RP!.NrColumns) then
        return RP!.NrColumns( C );
    fi;

    if not IsHomalgInternalMatrixRep( C ) then
        Error( "could not find a procedure called NrColumns in the ",
               "homalgTable to apply to a non-internal matrix\n" );
    fi;

    #=====# can only work for homalg internal matrices #=====#

    return Length( Eval( C )!.matrix[ 1 ] );

end );
```

D.1.19 Determinant (homalgTable entry)

◊ Determinant(*C*) (function)

Returns: a ring element

Let $R :=$ HomalgRing(C) and $RP :=$ homalgTable(R). If the homalgTable component RP!.Determinant is bound then the standard method for the attribute DeterminantMat (6.4.3) shown below returns RP!.Determinant(C).

Code

```
InstallMethod( DeterminantMat,
        "for homalg matrices",
        [ IsHomalgMatrix ],

  function( C )
    local R, RP;

    R := HomalgRing( C );

    RP := homalgTable( R );

    if NrRows( C ) <> NrColumns( C ) then
        Error( "the matrix is not quadratic\n" );
    fi;

    if IsBound(RP!.Determinant) then
        return RingElementConstructor( R )( RP!.Determinant( C ), R );
    fi;

    if not IsHomalgInternalMatrixRep( C ) then
        Error( "could not find a procedure called Determinant in the ",
               "homalgTable to apply to a non-internal matrix\n" );
    fi;

    #=====# can only work for homalg internal matrices #=====#

    return Determinant( Eval( C )!.matrix );

end );


InstallMethod( Determinant,
        "for homalg matrices",
        [ IsHomalgMatrix ],

  function( C )

    return DeterminantMat( C );

end );
```

D.2 The Tool Operations with a Fallback Method

These are the methods for which it is recommended for performance reasons to have a homalgTable entry for non-internal rings. homalg only provides a generic fallback method.

D.2.1 AreEqualMatrices (homalgTable entry)

◊ AreEqualMatrices(*M1*, *M2*) (function)

Returns: true or false

Let $R :=$ HomalgRing(*M1*) and $RP :=$ homalgTable(R). If the homalgTable component *RP*!.AreEqualMatrices is bound then the standard method for the operation \= (6.5.13) shown below returns *RP*!.AreEqualMatrices(*M1*,*M2*).

Code

```
InstallMethod( \=,
        "for homalg comparable matrices",
        [ IsHomalgMatrix, IsHomalgMatrix ],

  function( M1, M2 )
    local R, RP;

    R := HomalgRing( M1 );

    RP := homalgTable( R );

    if IsBound(RP!.AreEqualMatrices) then
        ## CAUTION: the external system must be able to check equality
        ## modulo possible ring relations (known to the external system)!
        return RP!.AreEqualMatrices( M1, M2 );
    elif IsBound(RP!.Equal) then
        ## CAUTION: the external system must be able to check equality
        ## modulo possible ring relations (known to the external system)!
        return RP!.Equal( M1, M2 );
    elif IsBound(RP!.IsZeroMatrix) then   ## ensuring this avoids infinite loops
        return IsZero( M1 - M2 );
    fi;

    TryNextMethod( );

end );
```

D.2.2 IsIdentityMatrix (homalgTable entry)

◊ IsIdentityMatrix(*M*) (function)

Returns: true or false

Let $R :=$ HomalgRing(M) and $RP :=$ homalgTable(R). If the homalgTable component *RP*!.IsIdentityMatrix is bound then the standard method for the property IsIdentityMatrix (6.3.2) shown below returns *RP*!.IsIdentityMatrix(M).

Code

```
InstallMethod( IsIdentityMatrix,
        "for homalg matrices",
        [ IsHomalgMatrix ],
```

```
  function( M )
    local R, RP;

    R := HomalgRing( M );

    RP := homalgTable( R );

    if IsBound(RP!.IsIdentityMatrix) then
        return RP!.IsIdentityMatrix( M );
    fi;

    #=====# the fallback method #=====#

    return M = HomalgIdentityMatrix( NrRows( M ), HomalgRing( M ) );

end );
```

D.2.3 IsDiagonalMatrix (homalgTable entry)

◊ IsDiagonalMatrix(*M*) (function)

Returns: true or false

Let *R* := HomalgRing(*M*) and *RP* := homalgTable(*R*). If the homalgTable component *RP*!.IsDiagonalMatrix is bound then the standard method for the property IsDiagonalMatrix (6.3.12) shown below returns *RP*!.IsDiagonalMatrix(*M*).

Code

```
InstallMethod( IsDiagonalMatrix,
        "for homalg matrices",
        [ IsHomalgMatrix ],

  function( M )
    local R, RP, diag;

    R := HomalgRing( M );

    RP := homalgTable( R );

    if IsBound(RP!.IsDiagonalMatrix) then
        return RP!.IsDiagonalMatrix( M );
    fi;

    #=====# the fallback method #=====#

    diag := DiagonalEntries( M );

    return M = HomalgDiagonalMatrix( diag, NrRows( M ), NrColumns( M ), R );

end );
```

D.2.4 ZeroRows (homalgTable entry)

◊ ZeroRows(*C*) (function)

Returns: a (possibly empty) list of positive integers

Let *R* := HomalgRing(*C*) and *RP* := homalgTable(*R*). If the homalgTable component *RP*!.ZeroRows is bound then the standard method of the attribute ZeroRows (6.4.4) shown below returns *RP*!.ZeroRows(*C*).

Code

```
InstallMethod( ZeroRows,
        "for homalg matrices",
        [ IsHomalgMatrix ],

  function( C )
    local R, RP, z;

    R := HomalgRing( C );

    RP := homalgTable( R );

    if IsBound(RP!.ZeroRows) then
        return RP!.ZeroRows( C );
    fi;

    #=====# the fallback method #=====#

    z := HomalgZeroMatrix( 1, NrColumns( C ), R );

    return Filtered( [ 1 .. NrRows( C ) ], a -> CertainRows( C, [ a ] ) = z );

end );
```

D.2.5 ZeroColumns (homalgTable entry)

◊ ZeroColumns(*C*) (function)

Returns: a (possibly empty) list of positive integers

Let *R* := HomalgRing(*C*) and *RP* := homalgTable(*R*). If the homalgTable component *RP*!.ZeroColumns is bound then the standard method of the attribute ZeroColumns (6.4.5) shown below returns *RP*!.ZeroColumns(*C*).

Code

```
InstallMethod( ZeroColumns,
        "for homalg matrices",
        [ IsHomalgMatrix ],

  function( C )
    local R, RP, z;

    R := HomalgRing( C );

    RP := homalgTable( R );

    if IsBound(RP!.ZeroColumns) then
```

```
        return RP!.ZeroColumns( C );
    fi;

    #=====# the fallback method #=====#

    z := HomalgZeroMatrix( NrRows( C ), 1, R );

    return Filtered( [ 1 .. NrColumns( C ) ], a -> CertainColumns( C, [ a ] ) = z );

end );
```

D.2.6 GetColumnIndependentUnitPositions (homalgTable entry)

◊ GetColumnIndependentUnitPositions(*M*, *poslist*) (function)

Returns: a (possibly empty) list of pairs of positive integers

Let *R* := HomalgRing(*M*) and *RP* := homalgTable(*R*). If the homalgTable component *RP*!.GetColumnIndependentUnitPositions is bound then the standard method of the operation GetColumnIndependentUnitPositions (6.5.14) returns *RP*!.GetColumnIndependentUnitPositions(*M*,*poslist*).

Code

```
InstallMethod( GetColumnIndependentUnitPositions,
        "for homalg matrices",
        [ IsHomalgMatrix, IsHomogeneousList ],

  function( M, poslist )
    local R, RP, rest, pos, i, j, k;

    R := HomalgRing( M );

    RP := homalgTable( R );

    if IsBound(RP!.GetColumnIndependentUnitPositions) then
        pos := RP!.GetColumnIndependentUnitPositions( M, poslist );
        if pos <> [ ] then
            SetIsZero( M, false );
        fi;
        return pos;
    fi;

    #=====# the fallback method #=====#

    rest := [ 1 .. NrColumns( M ) ];

    pos := [ ];

    for i in [ 1 .. NrRows( M ) ] do
        for k in Reversed( rest ) do
            if not [ i, k ] in poslist and
               IsUnit( R, GetEntryOfHomalgMatrix( M, i, k ) ) then
                Add( pos, [ i, k ] );
                rest := Filtered( rest,
```

```
                                a -> IsZero( GetEntryOfHomalgMatrix( M, i, a ) ) );
                break;
            fi;
        od;
    od;

    if pos <> [ ] then
        SetIsZero( M, false );
    fi;

    return pos;

end );
```

D.2.7 GetRowIndependentUnitPositions (homalgTable entry)

◊ GetRowIndependentUnitPositions(*M, poslist*) (function)

Returns: a (possibly empty) list of pairs of positive integers

Let *R* := HomalgRing(*M*) and *RP* := homalgTable(*R*). If the homalgTable component *RP*!.GetRowIndependentUnitPositions is bound then the standard method of the operation GetRowIndependentUnitPositions (6.5.15) returns *RP*!.GetRowIndependentUnitPositions(*M*,*poslist*).

Code

```
InstallMethod( GetRowIndependentUnitPositions,
        "for homalg matrices",
        [ IsHomalgMatrix, IsHomogeneousList ],

  function( M, poslist )
    local R, RP, rest, pos, j, i, k;

    R := HomalgRing( M );

    RP := homalgTable( R );

    if IsBound(RP!.GetRowIndependentUnitPositions) then
        pos := RP!.GetRowIndependentUnitPositions( M, poslist );
        if pos <> [ ] then
            SetIsZero( M, false );
        fi;
        return pos;
    fi;

    #=====# the fallback method #=====#

    rest := [ 1 .. NrRows( M ) ];

    pos := [ ];

    for j in [ 1 .. NrColumns( M ) ] do
        for k in Reversed( rest ) do
            if not [ j, k ] in poslist and
```

```
                IsUnit( R, GetEntryOfHomalgMatrix( M, k, j ) ) then
                 Add( pos, [ j, k ] );
                 rest := Filtered( rest,
                                 a -> IsZero( GetEntryOfHomalgMatrix( M, a, j ) ) );
                 break;
             fi;
         od;
     od;

     if pos <> [ ] then
         SetIsZero( M, false );
     fi;

     return pos;

 end );
```

D.2.8 GetUnitPosition (homalgTable entry)

◇ GetUnitPosition(*M, poslist*) (function)

Returns: a (possibly empty) list of pairs of positive integers

Let $R :=$ HomalgRing(M) and $RP :=$ homalgTable(R). If the homalgTable component RP!.GetUnitPosition is bound then the standard method of the operation GetUnitPosition (6.5.16) returns RP!.GetUnitPosition(M,*poslist*).

Code

```
 InstallMethod( GetUnitPosition,
         "for homalg matrices",
         [ IsHomalgMatrix, IsHomogeneousList ],

   function( M, poslist )
     local R, RP, pos, m, n, i, j;

     R := HomalgRing( M );

     RP := homalgTable( R );

     if IsBound(RP!.GetUnitPosition) then
         pos := RP!.GetUnitPosition( M, poslist );
         if IsList( pos ) and IsPosInt( pos[1] ) and IsPosInt( pos[2] ) then
             SetIsZero( M, false );
         fi;
         return pos;
     fi;

     #=====# the fallback method #=====#

     m := NrRows( M );
     n := NrColumns( M );

     for i in [ 1 .. m ] do
         for j in [ 1 .. n ] do
             if not [ i, j ] in poslist and not j in poslist and
```

```
                IsUnit( R, GetEntryOfHomalgMatrix( M, i, j ) ) then
                 SetIsZero( M, false );
                 return [ i, j ];
            fi;
        od;
    od;

    return fail;

end );
```

D.2.9 DegreesOfEntries (homalgTable entry)

◇ DegreesOfEntries(*C*) (function)

Returns: a listlist of degrees/multi-degrees

Let *R* := HomalgRing(*C*) and *RP* := homalgTable(*R*). If the homalgTable component *RP*!.DegreesOfEntries is bound then the standard method for the attribute DegreesOfEntries (6.4.10) shown below returns *RP*!.DegreesOfEntries(*C*).

Code

```
InstallMethod( DegreesOfEntries,
        "for homalg matrices",
        [ IsHomalgMatrix ],

  function( C )
    local R, RP, weights, e, c;

    if IsZero( C ) then
        return ListWithIdenticalEntries( NrRows( C ),
                       ListWithIdenticalEntries( NrColumns( C ), -1 ) );
    fi;

    R := HomalgRing( C );

    RP := homalgTable( R );

    if Set( WeightsOfIndeterminates( R ) ) <> [ 1 ] then

        weights := WeightsOfIndeterminates( R );

        if IsList( weights[1] ) then
            if IsBound(RP!.MultiWeightedDegreesOfEntries) then
                return RP!.MultiWeightedDegreesOfEntries( C, weights );
            fi;
        elif IsBound(RP!.WeightedDegreesOfEntries) then
            return RP!.WeightedDegreesOfEntries( C, weights );
        fi;

    elif IsBound(RP!.DegreesOfEntries) then
        return RP!.DegreesOfEntries( C );
    fi;

    #=====# the fallback method #=====#
```

```
    e := EntriesOfHomalgMatrix( C );

    e := List( e, DegreeMultivariatePolynomial );

    c := NrColumns( C );

    return List( [ 1 .. NrRows( C ) ], r -> e{[ ( r - 1 ) * c + 1 .. r * c ]} );

end );
```

Appendix E

Logic Subpackages

E.1 LIRNG: Logical Implications for Rings

E.2 LIMAP: Logical Implications for Ring Maps

E.3 LIMAT: Logical Implications for Matrices

E.4 COLEM: Clever Operations for Lazy Evaluated Matrices

Most of the matrix tool operations listed in Appendix D.1 which return a new matrix are lazy evaluated. The value of a homalg matrix is stored in the attribute `Eval`. Below is the list of the installed methods for the attribute `Eval`.

E.4.1 Eval (for matrices created with HomalgInitialMatrix)

◊ `Eval(`*C*`)` (method)

Returns: the `Eval` value of a homalg matrix *C*

In case the matrix *A* was created using `HomalgInitialMatrix` (6.2.1) then the filter `IsInitialMatrix` for *A* is set to true and the `homalgTable` function (→ `ZeroMatrix` (D.1.1)) will be used to set the attribute `Eval` and resets the filter `IsInitialMatrix`.

Code

```
InstallMethod( Eval,
        "for homalg matrices (IsInitialMatrix)",
        [ IsHomalgMatrix and IsInitialMatrix and
          HasNrRows and HasNrColumns ],

  function( C )
    local R, RP, z, zz;

    R := HomalgRing( C );

    RP := homalgTable( R );

    if IsBound( RP!.ZeroMatrix ) then
        ResetFilterObj( C, IsInitialMatrix );
        return RP!.ZeroMatrix( C );
```

```
    fi;

    if not IsHomalgInternalMatrixRep( C ) then
        Error( "could not find a procedure called ZeroMatrix in the ",
               "homalgTable to evaluate a non-internal initial matrix\n" );
    fi;

    #=====# can only work for homalg internal matrices #=====#

    z := Zero( HomalgRing( C ) );

    ResetFilterObj( C, IsInitialMatrix );

    zz := ListWithIdenticalEntries( NrColumns( C ), z );

    return homalgInternalMatrixHull(
                  List( [ 1 .. NrRows( C ) ], i -> ShallowCopy( zz ) ) );

end );
```

E.4.2 Eval (for matrices created with HomalgInitialIdentityMatrix)

◊ Eval(*C*) (method)

Returns: the Eval value of a homalg matrix *C*

In case the matrix *A* was created using HomalgInitialIdentityMatrix (6.2.2) then the filter IsInitialIdentityMatrix for *A* is set to true and the homalgTable function (→ IdentityMatrix (D.1.2)) will be used to set the attribute Eval and resets the filter IsInitialIdentityMatrix.

Code

```
InstallMethod( Eval,
        "for homalg matrices (IsInitialIdentityMatrix)",
        [ IsHomalgMatrix and IsInitialIdentityMatrix and
          HasNrRows and HasNrColumns ],

  function( C )
    local R, RP, o, z, zz, id;

    R := HomalgRing( C );

    RP := homalgTable( R );

    if IsBound( RP!.IdentityMatrix ) then
        ResetFilterObj( C, IsInitialIdentityMatrix );
        return RP!.IdentityMatrix( C );
    fi;

    if not IsHomalgInternalMatrixRep( C ) then
        Error( "could not find a procedure called IdentityMatrix in the ",
               "homalgTable to evaluate a non-internal initial identity matrix\n" );
    fi;

    #=====# can only work for homalg internal matrices #=====#
```

```
    z := Zero( HomalgRing( C ) );
    o := One( HomalgRing( C ) );

    ResetFilterObj( C, IsInitialIdentityMatrix );

    zz := ListWithIdenticalEntries( NrColumns( C ), z );

    id := List( [ 1 .. NrRows( C ) ],
                function(i)
                  local z;
                  z := ShallowCopy( zz ); z[i] := o; return z;
                end );

    return homalgInternalMatrixHull( id );

end );
```

E.4.3 Eval (for matrices created with HomalgZeroMatrix)

◊ Eval(*C*) (method)

Returns: the Eval value of a homalg matrix *C*

In case the matrix *A* was created using HomalgZeroMatrix (6.2.3) then the filter IsZeroMatrix for *A* is set to true and the homalgTable function (→ ZeroMatrix (D.1.3)) will be used to set the attribute Eval.

Code

```
InstallMethod( Eval,
        "for homalg matrices (IsZero)",
        [ IsHomalgMatrix and IsZero and HasNrRows and HasNrColumns ], 20,

  function( C )
    local R, RP, z;

    R := HomalgRing( C );

    RP := homalgTable( R );

    if ( NrRows( C ) = 0 or NrColumns( C ) = 0 ) and
       not ( IsBound( R!.SafeToEvaluateEmptyMatrices ) and
             R!.SafeToEvaluateEmptyMatrices = true ) then
        Info( InfoWarning, 1, "\033[01m\033[5;31;47m",
              "an empty matrix is about to get evaluated!",
              "\033[0m" );
    fi;

    if IsBound( RP!.ZeroMatrix ) then
        return RP!.ZeroMatrix( C );
    fi;

    if not IsHomalgInternalMatrixRep( C ) then
        Error( "could not find a procedure called ZeroMatrix in the ",
               "homalgTable to evaluate a non-internal zero matrix\n" );
    fi;
```

```
    #=====# can only work for homalg internal matrices #=====#

    z := Zero( HomalgRing( C ) );

    ## copying the rows saves memory;
    ## we assume that the entries are never modified!!!
    return homalgInternalMatrixHull(
                   ListWithIdenticalEntries( NrRows( C ),
                          ListWithIdenticalEntries( NrColumns( C ), z ) ) );

end );
```

E.4.4 Eval (for matrices created with HomalgIdentityMatrix)

◊ Eval(*C*) (method)

Returns: the Eval value of a homalg matrix *C*

In case the matrix *A* was created using HomalgIdentityMatrix (6.2.4) then the filter IsIdentityMatrix for *A* is set to true and the homalgTable function (→ IdentityMatrix (D.1.4)) will be used to set the attribute Eval.

Code

```
InstallMethod( Eval,
        "for homalg matrices (IsIdentityMatrix)",
        [ IsHomalgMatrix and IsIdentityMatrix and HasNrRows and HasNrColumns ], 10,

  function( C )
    local R, RP, o, z, zz, id;

    R := HomalgRing( C );

    RP := homalgTable( R );

    if IsBound( RP!.IdentityMatrix ) then
        return RP!.IdentityMatrix( C );
    fi;

    if not IsHomalgInternalMatrixRep( C ) then
        Error( "could not find a procedure called IdentityMatrix in the ",
               "homalgTable to evaluate a non-internal identity matrix\n" );
    fi;

    #=====# can only work for homalg internal matrices #=====#

    z := Zero( HomalgRing( C ) );
    o := One( HomalgRing( C ) );

    zz := ListWithIdenticalEntries( NrColumns( C ), z );

    id := List( [ 1 .. NrRows( C ) ],
                function(i)
                  local z;
                  z := ShallowCopy( zz ); z[i] := o; return z;
```

```
                end );

    return homalgInternalMatrixHull( id );

end );
```

E.4.5 Eval (for matrices created with Involution)

◇ Eval(*C*) (method)

Returns: the Eval value of a homalg matrix *C*

In case the matrix was created using Involution (6.5.2) then the filter HasEvalInvolution for *A* is set to true and the homalgTable function Involution (D.1.5) will be used to set the attribute Eval.

Code

```
InstallMethod( Eval,
        "for homalg matrices (HasEvalInvolution)",
        [ IsHomalgMatrix and HasEvalInvolution ],

  function( C )
    local R, RP, M;

    R := HomalgRing( C );

    RP := homalgTable( R );

    M :=  EvalInvolution( C );

    if IsBound(RP!.Involution) then
        return RP!.Involution( M );
    fi;

    if not IsHomalgInternalMatrixRep( C ) then
        Error( "could not find a procedure called Involution in the ",
               "homalgTable to apply to a non-internal matrix\n" );
    fi;

    #=====# can only work for homalg internal matrices #=====#

    return homalgInternalMatrixHull( TransposedMat( Eval( M )!.matrix ) );

end );
```

E.4.6 Eval (for matrices created with CertainRows)

◇ Eval(*C*) (method)

Returns: the Eval value of a homalg matrix *C*

In case the matrix was created using CertainRows (6.5.3) then the filter HasEvalCertainRows for *A* is set to true and the homalgTable function CertainRows (D.1.6) will be used to set the attribute Eval.

Code

```
InstallMethod( Eval,
        "for homalg matrices (HasEvalCertainRows)",
        [ IsHomalgMatrix and HasEvalCertainRows ],

  function( C )
    local R, RP, e, M, plist;

    R := HomalgRing( C );

    RP := homalgTable( R );

    e :=  EvalCertainRows( C );

    M := e[1];
    plist := e[2];

    if IsBound(RP!.CertainRows) then
        return RP!.CertainRows( M, plist );
    fi;

    if not IsHomalgInternalMatrixRep( C ) then
        Error( "could not find a procedure called CertainRows in the ",
               "homalgTable to apply to a non-internal matrix\n" );
    fi;

    #=====# can only work for homalg internal matrices #=====#

    return homalgInternalMatrixHull( Eval( M )!.matrix{ plist } );

end );
```

E.4.7 Eval (for matrices created with CertainColumns)

◊ Eval(*C*) (method)

Returns: the Eval value of a homalg matrix *C*

In case the matrix was created using CertainColumns (6.5.4) then the filter HasEvalCertainColumns for *A* is set to true and the homalgTable function CertainColumns (D.1.7) will be used to set the attribute Eval.

Code

```
InstallMethod( Eval,
        "for homalg matrices (HasEvalCertainColumns)",
        [ IsHomalgMatrix and HasEvalCertainColumns ],

  function( C )
    local R, RP, e, M, plist;

    R := HomalgRing( C );

    RP := homalgTable( R );

    e :=  EvalCertainColumns( C );
```

```
    M := e[1];
    plist := e[2];

    if IsBound(RP!.CertainColumns) then
        return RP!.CertainColumns( M, plist );
    fi;

    if not IsHomalgInternalMatrixRep( C ) then
        Error( "could not find a procedure called CertainColumns in the ",
               "homalgTable to apply to a non-internal matrix\n" );
    fi;

    #=====# can only work for homalg internal matrices #=====#

    return homalgInternalMatrixHull(
                  Eval( M )!.matrix{[ 1 .. NrRows( M ) ]}{plist} );

end );
```

E.4.8 Eval (for matrices created with UnionOfRows)

◊ Eval(*C*) (method)

Returns: the Eval value of a homalg matrix *C*

In case the matrix was created using UnionOfRows (6.5.5) then the filter HasEvalUnionOfRows for *A* is set to true and the homalgTable function UnionOfRows (D.1.8) will be used to set the attribute Eval.

Code

```
InstallMethod( Eval,
        "for homalg matrices (HasEvalUnionOfRows)",
        [ IsHomalgMatrix and HasEvalUnionOfRows ],

  function( C )
    local R, RP, e, A, B, U;

    R := HomalgRing( C );

    RP := homalgTable( R );

    e :=  EvalUnionOfRows( C );

    A := e[1];
    B := e[2];

    if IsBound(RP!.UnionOfRows) then
        return RP!.UnionOfRows( A, B );
    fi;

    if not IsHomalgInternalMatrixRep( C ) then
        Error( "could not find a procedure called UnionOfRows in the ",
               "homalgTable to apply to a non-internal matrix\n" );
    fi;
```

```
    #=====# can only work for homalg internal matrices #=====#

    U := ShallowCopy( Eval( A )!.matrix );

    U{ [ NrRows( A ) + 1 .. NrRows( A ) + NrRows( B ) ] } := Eval( B )!.matrix;

    return homalgInternalMatrixHull( U );

end );
```

E.4.9 Eval (for matrices created with UnionOfColumns)

◊ Eval(*C*) (method)

Returns: the Eval value of a homalg matrix *C*

In case the matrix was created using UnionOfColumns (6.5.6) then the filter HasEvalUnionOfColumns for *A* is set to true and the homalgTable function UnionOfColumns (D.1.9) will be used to set the attribute Eval.

Code

```
InstallMethod( Eval,
        "for homalg matrices (HasEvalUnionOfColumns)",
        [ IsHomalgMatrix and HasEvalUnionOfColumns ],

  function( C )
    local R, RP, e, A, B, U;

    R := HomalgRing( C );

    RP := homalgTable( R );

    e :=  EvalUnionOfColumns( C );

    A := e[1];
    B := e[2];

    if IsBound(RP!.UnionOfColumns) then
        return RP!.UnionOfColumns( A, B );
    fi;

    if not IsHomalgInternalMatrixRep( C ) then
        Error( "could not find a procedure called UnionOfColumns in the ",
               "homalgTable to apply to a non-internal matrix\n" );
    fi;

    #=====# can only work for homalg internal matrices #=====#

    U := List( Eval( A )!.matrix, ShallowCopy );

    U{ [ 1 .. NrRows( A ) ] }
      { [ NrColumns( A ) + 1 .. NrColumns( A ) + NrColumns( B ) ] }
      := Eval( B )!.matrix;
```

```
    return homalgInternalMatrixHull( U );

end );
```

E.4.10 Eval (for matrices created with DiagMat)

◊ Eval(*C*) (method)

Returns: the Eval value of a homalg matrix *C*

In case the matrix was created using DiagMat (6.5.7) then the filter HasEvalDiagMat for *A* is set to true and the homalgTable function DiagMat (D.1.10) will be used to set the attribute Eval.

Code

```
InstallMethod( Eval,
        "for homalg matrices (HasEvalDiagMat)",
        [ IsHomalgMatrix and HasEvalDiagMat ],

  function( C )
    local R, RP, e, z, m, n, diag, mat;

    R := HomalgRing( C );

    RP := homalgTable( R );

    e :=  EvalDiagMat( C );

    if IsBound(RP!.DiagMat) then
        return RP!.DiagMat( e );
    fi;

    if not IsHomalgInternalMatrixRep( C ) then
        Error( "could not find a procedure called DiagMat in the ",
               "homalgTable to apply to a non-internal matrix\n" );
    fi;

    #=====# can only work for homalg internal matrices #=====#

    z := Zero( R );

    m := Sum( List( e, NrRows ) );
    n := Sum( List( e, NrColumns ) );

    diag := List( [ 1 .. m ],  a -> List( [ 1 .. n ], b -> z ) );

    m := 0;
    n := 0;

    for mat in e do
        diag{ [ m + 1 .. m + NrRows( mat ) ] }{ [ n + 1 .. n + NrColumns( mat ) ] }
          := Eval( mat )!.matrix;

        m := m + NrRows( mat );
        n := n + NrColumns( mat );
    od;
```

```
    return homalgInternalMatrixHull( diag );

end );
```

E.4.11 Eval (for matrices created with KroneckerMat)

◊ Eval(*C*) (method)

Returns: the Eval value of a homalg matrix *C*

In case the matrix was created using KroneckerMat (6.5.8) then the filter HasEvalKroneckerMat for *A* is set to true and the homalgTable function KroneckerMat (D.1.11) will be used to set the attribute Eval.

Code

```
InstallMethod( Eval,
        "for homalg matrices (HasEvalKroneckerMat)",
        [ IsHomalgMatrix and HasEvalKroneckerMat ],

  function( C )
    local R, RP, A, B;

    R := HomalgRing( C );

    if HasIsCommutative( R ) and not IsCommutative( R ) then
        Info( InfoWarning, 1, "\033[01m\033[5;31;47m",
              "the Kronecker product is only defined for commutative rings!",
              "\033[0m" );
    fi;

    RP := homalgTable( R );

    A :=  EvalKroneckerMat( C )[1];
    B :=  EvalKroneckerMat( C )[2];

    if IsBound(RP!.KroneckerMat) then
        return RP!.KroneckerMat( A, B );
    fi;

    if not IsHomalgInternalMatrixRep( C ) then
        Error( "could not find a procedure called KroneckerMat in the ",
               "homalgTable to apply to a non-internal matrix\n" );
    fi;

    #=====# can only work for homalg internal matrices #=====#

    return homalgInternalMatrixHull(
                  KroneckerProduct( Eval( A )!.matrix, Eval( B )!.matrix ) );
    ## this was easy, thanks GAP :)

end );
```

E.4.12 Eval (for matrices created with MulMat)

◊ Eval(*C*) (method)

Returns: the Eval value of a homalg matrix *C*

In case the matrix was created using * (6.5.9) then the filter HasEvalMulMat for *A* is set to true and the homalgTable function MulMat (D.1.12) will be used to set the attribute Eval.

Code

```
InstallMethod( Eval,
        "for homalg matrices (HasEvalMulMat)",
        [ IsHomalgMatrix and HasEvalMulMat ],

  function( C )
    local R, RP, e, a, A;

    R := HomalgRing( C );

    RP := homalgTable( R );

    e :=  EvalMulMat( C );

    a := e[1];
    A := e[2];

    if IsBound(RP!.MulMat) then
        return RP!.MulMat( a, A );
    fi;

    if not IsHomalgInternalMatrixRep( C ) then
        Error( "could not find a procedure called MulMat in the ",
               "homalgTable to apply to a non-internal matrix\n" );
    fi;

    #=====# can only work for homalg internal matrices #=====#

    return a * Eval( A );

end );
```

E.4.13 Eval (for matrices created with AddMat)

◊ Eval(*C*) (method)

Returns: the Eval value of a homalg matrix *C*

In case the matrix was created using \+ (6.5.10) then the filter HasEvalAddMat for *A* is set to true and the homalgTable function AddMat (D.1.13) will be used to set the attribute Eval.

Code

```
InstallMethod( Eval,
        "for homalg matrices (HasEvalAddMat)",
        [ IsHomalgMatrix and HasEvalAddMat ],

  function( C )
    local R, RP, e, A, B;
```

```
    R := HomalgRing( C );

    RP := homalgTable( R );

    e :=  EvalAddMat( C );

    A := e[1];
    B := e[2];

    if IsBound(RP!.AddMat) then
        return RP!.AddMat( A, B );
    fi;

    if not IsHomalgInternalMatrixRep( C ) then
        Error( "could not find a procedure called AddMat in the ",
               "homalgTable to apply to a non-internal matrix\n" );
    fi;

    #=====# can only work for homalg internal matrices #=====#

    return Eval( A ) + Eval( B );

end );
```

E.4.14 Eval (for matrices created with SubMat)

◇ Eval(*C*) (method)

Returns: the Eval value of a homalg matrix *C*

In case the matrix was created using \- (6.5.11) then the filter HasEvalSubMat for *A* is set to true and the homalgTable function SubMat (D.1.14) will be used to set the attribute Eval.

Code

```
InstallMethod( Eval,
        "for homalg matrices (HasEvalSubMat)",
        [ IsHomalgMatrix and HasEvalSubMat ],

  function( C )
    local R, RP, e, A, B;

    R := HomalgRing( C );

    RP := homalgTable( R );

    e :=  EvalSubMat( C );

    A := e[1];
    B := e[2];

    if IsBound(RP!.SubMat) then
        return RP!.SubMat( A, B );
    fi;

    if not IsHomalgInternalMatrixRep( C ) then
```

```
        Error( "could not find a procedure called SubMat in the ",
               "homalgTable to apply to a non-internal matrix\n" );
    fi;

    #=====# can only work for homalg internal matrices #=====#

    return Eval( A ) - Eval( B );

end );
```

E.4.15 Eval (for matrices created with Compose)

◊ Eval(*C*) (method)

Returns: the Eval value of a homalg matrix *C*

In case the matrix was created using * (6.5.12) then the filter HasEvalCompose for *A* is set to true and the homalgTable function Compose (D.1.15) will be used to set the attribute Eval.

Code

```
InstallMethod( Eval,
        "for homalg matrices (HasEvalCompose)",
        [ IsHomalgMatrix and HasEvalCompose ],

  function( C )
    local R, RP, e, A, B;

    R := HomalgRing( C );

    RP := homalgTable( R );

    e :=  EvalCompose( C );

    A := e[1];
    B := e[2];

    if IsBound(RP!.Compose) then
        return RP!.Compose( A, B );
    fi;

    if not IsHomalgInternalMatrixRep( C ) then
        Error( "could not find a procedure called Compose in the ",
               "homalgTable to apply to a non-internal matrix\n" );
    fi;

    #=====# can only work for homalg internal matrices #=====#

    return Eval( A ) * Eval( B );

end );
```

E.4.16 Eval (for matrices created with LeftInverse)

◊ Eval(*A*) (method)

Returns: see below

In case the matrix was created using LeftInverse (6.4.13) then the filter HasEvalLeftInverse for *A* is set to true and the method listed below will be used to set the attribute Eval. (→ RightDivide (6.5.40))

Code

```
InstallMethod( Eval,
        "for homalg matrices",
        [ IsHomalgMatrix and HasEvalLeftInverse ],

  function( LI )
    local R, RI, Id, left_inv;

    R := HomalgRing( LI );

    RI := EvalLeftInverse( LI );

    Id := HomalgIdentityMatrix( NrColumns( RI ), R );

    left_inv := RightDivide( Id, RI );        ## ( cf. [BR, Subsection 3.1.3] )

    if IsBool( left_inv ) then
        return false;
    fi;

    ## CAUTION: for the following SetXXX RightDivide is assumed
    ## NOT to be lazy evaluated!!!

    SetIsLeftInvertibleMatrix( RI, true );

    if HasIsInvertibleMatrix( RI ) and IsInvertibleMatrix( RI ) then
        SetIsInvertibleMatrix( LI, true );
    else
        SetIsRightInvertibleMatrix( LI, true );
    fi;

    return Eval( left_inv );

end );
```

E.4.17 Eval (for matrices created with RightInverse)

◊ Eval(*A*) (method)

Returns: see below

In case the matrix was created using RightInverse (6.4.14) then the filter HasEvalRightInverse for *A* is set to true and the method listed below will be used to set the attribute Eval. (→ LeftDivide (6.5.41))

Code

```
InstallMethod( Eval,
        "for homalg matrices",
```

```
        [ IsHomalgMatrix and HasEvalRightInverse ],

  function( RI )
    local R, LI, Id, right_inv;

    R := HomalgRing( RI );

    LI := EvalRightInverse( RI );

    Id := HomalgIdentityMatrix( NrRows( LI ), R );

    right_inv := LeftDivide( LI, Id );        ## ( cf. [BR, Subsection 3.1.3] )

    if IsBool( right_inv ) then
        return false;
    fi;

    ## CAUTION: for the following SetXXX LeftDivide is assumed
    ## NOT to be lazy evaluated!!!

    SetIsRightInvertibleMatrix( LI, true );

    if HasIsInvertibleMatrix( LI ) and IsInvertibleMatrix( LI ) then
        SetIsInvertibleMatrix( RI, true );
    else
        SetIsLeftInvertibleMatrix( RI, true );
    fi;

    return Eval( right_inv );

end );
```

E.5 LIMOD: Logical Implications for Modules

E.6 LIMOR: Logical Implications for Morphisms

E.7 LICPX: Logical Implications for Complexes

Appendix F

The subpackage ResidueClassRingForHomalg as a sample ring package

For an example see 3.3.1.

F.1 The Mandatory Basic Operations

F.1.1 BasisOfRowModule (ResidueClassRing)

◊ BasisOfRowModule(*M*) (function)

Returns: a homalg matrix over the ambient ring

Code

```
BasisOfRowModule :=
  function( M )
    local Mrel;

    Mrel := UnionOfRows( M );

    Mrel := HomalgResidueClassMatrix(
                    BasisOfRowModule( Mrel ), HomalgRing( M ) );

    return GetRidOfObsoleteRows( Mrel );

  end,
```

F.1.2 BasisOfColumnModule (ResidueClassRing)

◊ BasisOfColumnModule(*M*) (function)

Returns: a homalg matrix over the ambient ring

Code

```
BasisOfColumnModule :=
  function( M )
    local Mrel;
```

```
    Mrel := UnionOfColumns( M );

    Mrel := HomalgResidueClassMatrix(
                    BasisOfColumnModule( Mrel ), HomalgRing( M ) );

    return GetRidOfObsoleteColumns( Mrel );

  end,
```

F.1.3 DecideZeroRows (ResidueClassRing)

◊ DecideZeroRows(*A*, *B*) (function)

Returns: a homalg matrix over the ambient ring

Code

```
DecideZeroRows :=
  function( A, B )
    local Brel;

    Brel := UnionOfRows( B );

    Brel := BasisOfRowModule( Brel );

    return HomalgResidueClassMatrix(
                  DecideZeroRows( Eval( A ), Brel ), HomalgRing( A ) );

  end,
```

F.1.4 DecideZeroColumns (ResidueClassRing)

◊ DecideZeroColumns(*A*, *B*) (function)

Returns: a homalg matrix over the ambient ring

Code

```
DecideZeroColumns :=
  function( A, B )
    local Brel;

    Brel := UnionOfColumns( B );

    Brel := BasisOfColumnModule( Brel );

    return HomalgResidueClassMatrix(
                  DecideZeroColumns( Eval( A ), Brel ), HomalgRing( A ) );

  end,
```

F.1.5 SyzygiesGeneratorsOfRows (ResidueClassRing)

◊ SyzygiesGeneratorsOfRows(*M*) (function)

Returns: a homalg matrix over the ambient ring

Code

```
SyzygiesGeneratorsOfRows :=
  function( M )
    local R, ring_rel, rel, S;

    R := HomalgRing( M );

    ring_rel := RingRelations( R );

    rel := MatrixOfRelations( ring_rel );

    if IsHomalgRelationsOfRightModule( ring_rel ) then
        rel := Involution( rel );
    fi;

    rel := DiagMat( ListWithIdenticalEntries( NrColumns( M ), rel ) );

    S := SyzygiesGeneratorsOfRows( Eval( M ), rel );

    S := HomalgResidueClassMatrix( S, R );

    S := GetRidOfObsoleteRows( S );

    if IsZero( S ) then

        SetIsLeftRegularMatrix( M, true );

    fi;

    return S;

  end,
```

F.1.6 SyzygiesGeneratorsOfColumns (ResidueClassRing)

◊ SyzygiesGeneratorsOfColumns(*M*) (function)

Returns: a homalg matrix over the ambient ring

Code

```
SyzygiesGeneratorsOfColumns :=
  function( M )
    local R, ring_rel, rel, S;

    R := HomalgRing( M );

    ring_rel := RingRelations( R );

    rel := MatrixOfRelations( ring_rel );

    if IsHomalgRelationsOfLeftModule( ring_rel ) then
        rel := Involution( rel );
    fi;
```

```
    rel := DiagMat( ListWithIdenticalEntries( NrRows( M ), rel ) );

    S := SyzygiesGeneratorsOfColumns( Eval( M ), rel );

    S := HomalgResidueClassMatrix( S, R );

    S := GetRidOfObsoleteColumns( S );

    if IsZero( S ) then

        SetIsRightRegularMatrix( M, true );

    fi;

    return S;

  end,
```

F.1.7 BasisOfRowsCoeff (ResidueClassRing)

◊ BasisOfRowsCoeff(*M*, *T*) (function)

Returns: a homalg matrix over the ambient ring

Code

```
BasisOfRowsCoeff :=
  function( M, T )
    local Mrel, TT, bas;

    Mrel := UnionOfRows( M );

    TT := HomalgVoidMatrix( HomalgRing( Mrel ) );

    bas := BasisOfRowsCoeff( Mrel, TT );

    SetEval( T, CertainColumns( TT, [ 1 .. NrRows( M ) ] ) );

    ResetFilterObj( T, IsVoidMatrix );

    ## FIXME: GetRidOfObsoleteRows and correct T
    return HomalgResidueClassMatrix( bas, HomalgRing( M ) );

  end,
```

F.1.8 BasisOfColumnsCoeff (ResidueClassRing)

◊ BasisOfColumnsCoeff(*M*, *T*) (function)

Returns: a homalg matrix over the ambient ring

Code

```
BasisOfColumnsCoeff :=
  function( M, T )
    local Mrel, TT, bas;
```

```
    Mrel := UnionOfColumns( M );

    TT := HomalgVoidMatrix( HomalgRing( Mrel ) );

    bas := BasisOfColumnsCoeff( Mrel, TT );

    SetEval( T, CertainRows( TT, [ 1 .. NrColumns( M ) ] ) );

    ResetFilterObj( T, IsVoidMatrix );

    ## FIXME: GetRidOfObsoleteColumns and correct T
    return HomalgResidueClassMatrix( bas, HomalgRing( M ) );

  end,
```

F.1.9 DecideZeroRowsEffectively (ResidueClassRing)

◊ DecideZeroRowsEffectively(*A, B, T*) (function)

Returns: a homalg matrix over the ambient ring

Code

```
DecideZeroRowsEffectively :=
  function( A, B, T )
    local Brel, TT, red;

    Brel := UnionOfRows( B );

    TT := HomalgVoidMatrix( HomalgRing( Brel ) );

    red := DecideZeroRowsEffectively( Eval( A ), Brel, TT );

    SetEval( T, CertainColumns( TT, [ 1 .. NrRows( B ) ] ) );

    ResetFilterObj( T, IsVoidMatrix );

    return HomalgResidueClassMatrix( red, HomalgRing( A ) );

  end,
```

F.1.10 DecideZeroColumnsEffectively (ResidueClassRing)

◊ DecideZeroColumnsEffectively(*A, B, T*) (function)

Returns: a homalg matrix over the ambient ring

Code

```
DecideZeroColumnsEffectively :=
  function( A, B, T )
    local Brel, TT, red;

    Brel := UnionOfColumns( B );

    TT := HomalgVoidMatrix( HomalgRing( Brel ) );
```

```
    red := DecideZeroColumnsEffectively( Eval( A ), Brel, TT );

    SetEval( T, CertainRows( TT, [ 1 .. NrColumns( B ) ] ) );

    ResetFilterObj( T, IsVoidMatrix );

    return HomalgResidueClassMatrix( red, HomalgRing( A ) );

  end,
```

F.1.11 RelativeSyzygiesGeneratorsOfRows (ResidueClassRing)

◊ RelativeSyzygiesGeneratorsOfRows(*M, M2*) (function)

Returns: a homalg matrix over the ambient ring

Code

```
RelativeSyzygiesGeneratorsOfRows :=
  function( M, M2 )
    local M2rel, S;

    M2rel := UnionOfRows( M2 );

    S := SyzygiesGeneratorsOfRows( Eval( M ), M2rel );

    S := HomalgResidueClassMatrix( S, HomalgRing( M ) );

    S := GetRidOfObsoleteRows( S );

    if IsZero( S ) then

        SetIsLeftRegularMatrix( M, true );

    fi;

    return S;

  end,
```

F.1.12 RelativeSyzygiesGeneratorsOfColumns (ResidueClassRing)

◊ RelativeSyzygiesGeneratorsOfColumns(*M, M2*) (function)

Returns: a homalg matrix over the ambient ring

Code

```
RelativeSyzygiesGeneratorsOfColumns :=
  function( M, M2 )
    local M2rel, S;

    M2rel := UnionOfColumns( M2 );

    S := SyzygiesGeneratorsOfColumns( Eval( M ), M2rel );

    S := HomalgResidueClassMatrix( S, HomalgRing( M ) );
```

```
    S := GetRidOfObsoleteColumns( S );

    if IsZero( S ) then

        SetIsRightRegularMatrix( M, true );

    fi;

    return S;

  end,
```

F.2 The Mandatory Tool Operations

Here we list those matrix operations for which homalg provides no fallback method.

F.2.1 ZeroMatrix (ResidueClassRing)

◊ ZeroMatrix() (function)

Returns: a homalg matrix over the ambient ring
(→ ZeroMatrix (D.1.1) and ZeroMatrix (D.1.3))

Code

```
ZeroMatrix := C -> HomalgZeroMatrix(
                      NrRows( C ), NrColumns( C ), AmbientRing( HomalgRing( C ) ) ),
```

F.2.2 IdentityMatrix (ResidueClassRing)

◊ IdentityMatrix() (function)

Returns: a homalg matrix over the ambient ring
(→ IdentityMatrix (D.1.2) and IdentityMatrix (D.1.4))

Code

```
IdentityMatrix := C -> HomalgIdentityMatrix(
        NrRows( C ), AmbientRing( HomalgRing( C ) ) ),
```

F.2.3 Involution (ResidueClassRing)

◊ Involution() (function)

Returns: a homalg matrix over the ambient ring
(→ Involution (D.1.5))

Code

```
Involution :=
  function( M )
    local N, R;

    N := Involution( Eval( M ) );

    R := HomalgRing( N );
```

```
    if not ( HasIsCommutative( R ) and IsCommutative( R ) and
             HasIsReducedModuloRingRelations( M ) and
             IsReducedModuloRingRelations( M ) ) then

        ## reduce the matrix N w.r.t. the ring relations
        N := DecideZero( N, HomalgRing( M ) );
    fi;

    return N;

  end,
```

F.2.4 CertainRows (ResidueClassRing)

◊ CertainRows() (function)

Returns: a homalg matrix over the ambient ring
(→ CertainRows (D.1.6))

Code

```
CertainRows :=
  function( M, plist )
    local N;

    N := CertainRows( Eval( M ), plist );

    if not ( HasIsReducedModuloRingRelations( M ) and
             IsReducedModuloRingRelations( M ) ) then

        ## reduce the matrix N w.r.t. the ring relations
        N := DecideZero( N, HomalgRing( M ) );
    fi;

    return N;

  end,
```

F.2.5 CertainColumns (ResidueClassRing)

◊ CertainColumns() (function)

Returns: a homalg matrix over the ambient ring
(→ CertainColumns (D.1.7))

Code

```
CertainColumns :=
  function( M, plist )
    local N;

    N := CertainColumns( Eval( M ), plist );

    if not ( HasIsReducedModuloRingRelations( M ) and
             IsReducedModuloRingRelations( M ) ) then

        ## reduce the matrix N w.r.t. the ring relations
```

```
        N := DecideZero( N, HomalgRing( M ) );
    fi;

    return N;

  end,
```

F.2.6 UnionOfRows (ResidueClassRing)

◊ UnionOfRows() (function)

Returns: a homalg matrix over the ambient ring
(→ UnionOfRows (D.1.8))

Code

```
UnionOfRows :=
  function( A, B )
    local N;

    N := UnionOfRows( Eval( A ), Eval( B ) );

    if not ForAll( [ A, B ], HasIsReducedModuloRingRelations and
              IsReducedModuloRingRelations ) then

        ## reduce the matrix N w.r.t. the ring relations
        N := DecideZero( N, HomalgRing( A ) );
    fi;

    return N;

  end,
```

F.2.7 UnionOfColumns (ResidueClassRing)

◊ UnionOfColumns() (function)

Returns: a homalg matrix over the ambient ring
(→ UnionOfColumns (D.1.9))

Code

```
UnionOfColumns :=
  function( A, B )
    local N;

    N := UnionOfColumns( Eval( A ), Eval( B ) );

    if not ForAll( [ A, B ], HasIsReducedModuloRingRelations and
              IsReducedModuloRingRelations ) then

        ## reduce the matrix N w.r.t. the ring relations
        N := DecideZero( N, HomalgRing( A ) );
    fi;

    return N;
```

```
  end,
```

F.2.8 DiagMat (ResidueClassRing)

◊ DiagMat ( ) (function)

Returns: a homalg matrix over the ambient ring
(→ DiagMat (D.1.10))

Code

```
DiagMat :=
  function( e )
    local N;

    N := DiagMat( List( e, Eval ) );

    if not ForAll( e, HasIsReducedModuloRingRelations and
               IsReducedModuloRingRelations ) then

        ## reduce the matrix N w.r.t. the ring relations
        N := DecideZero( N, HomalgRing( e[1] ) );
    fi;

    return N;

  end,
```

F.2.9 KroneckerMat (ResidueClassRing)

◊ KroneckerMat ( ) (function)

Returns: a homalg matrix over the ambient ring
(→ KroneckerMat (D.1.11))

Code

```
KroneckerMat :=
  function( A, B )
    local N;

    N := KroneckerMat( Eval( A ), Eval( B ) );

    if not ForAll( [ A, B ], HasIsReducedModuloRingRelations and
               IsReducedModuloRingRelations ) then

        ## reduce the matrix N w.r.t. the ring relations
        N := DecideZero( N, HomalgRing( A ) );
    fi;

    return N;

  end,
```

F.2.10 MulMat (ResidueClassRing)

◊ MulMat ( ) (function)

Returns: a homalg matrix over the ambient ring
(→ MulMat (D.1.12))

Code

```
MulMat :=
  function( a, A )

    return DecideZero( EvalRingElement( a ) * Eval( A ), HomalgRing( A ) );

  end,
```

F.2.11 AddMat (ResidueClassRing)

◊ AddMat ( ) (function)

Returns: a homalg matrix over the ambient ring
(→ AddMat (D.1.13))

Code

```
AddMat :=
  function( A, B )

    return DecideZero( Eval( A ) + Eval( B ), HomalgRing( A ) );

  end,
```

F.2.12 SubMat (ResidueClassRing)

◊ SubMat ( ) (function)

Returns: a homalg matrix over the ambient ring
(→ SubMat (D.1.14))

Code

```
SubMat :=
  function( A, B )

    return DecideZero( Eval( A ) - Eval( B ), HomalgRing( A ) );

  end,
```

F.2.13 Compose (ResidueClassRing)

◊ Compose ( ) (function)

Returns: a homalg matrix over the ambient ring
(→ Compose (D.1.15))

Code

```
Compose :=
  function( A, B )

    return DecideZero( Eval( A ) * Eval( B ), HomalgRing( A ) );

  end,
```

F.2.14 IsZeroMatrix (ResidueClassRing)

◊ IsZeroMatrix(*M*) (function)

Returns: true or false

(→ IsZeroMatrix (D.1.16))

Code

```
IsZeroMatrix := M -> IsZero( DecideZero( Eval( M ), HomalgRing( M ) ) ),
```

F.2.15 NrRows (ResidueClassRing)

◊ NrRows(*C*) (function)

Returns: a nonnegative integer

(→ NrRows (D.1.17))

Code

```
NrRows := C -> NrRows( Eval( C ) ),
```

F.2.16 NrColumns (ResidueClassRing)

◊ NrColumns(*C*) (function)

Returns: a nonnegative integer

(→ NrColumns (D.1.18))

Code

```
NrColumns := C -> NrColumns( Eval( C ) ),
```

F.2.17 Determinant (ResidueClassRing)

◊ Determinant(*C*) (function)

Returns: an element of ambient homalg ring

(→ Determinant (D.1.19))

Code

```
Determinant := C -> DecideZero( Determinant( Eval( C ) ), HomalgRing( C ) ),
```

F.3 Some of the Recommended Tool Operations

Here we list those matrix operations for which homalg does provide a fallback method. But specifying the below homalgTable functions increases the performance by replacing the fallback method.

F.3.1 AreEqualMatrices (ResidueClassRing)

◊ AreEqualMatrices(*A*, *B*) (function)

Returns: true or false

(→ AreEqualMatrices (D.2.1))

Code

```
AreEqualMatrices :=
  function( A, B )

    return IsZero( DecideZero( Eval( A ) - Eval( B ), HomalgRing( A ) ) );

  end,
```

F.3.2 IsIdentityMatrix (ResidueClassRing)

◊ IsIdentityMatrix(*M*) (function)

Returns: true or false

(→ IsIdentityMatrix (D.2.2))

Code

```
IsIdentityMatrix := M ->
          IsIdentityMatrix( DecideZero( Eval( M ), HomalgRing( M ) ) ),
```

F.3.3 IsDiagonalMatrix (ResidueClassRing)

◊ IsDiagonalMatrix(*M*) (function)

Returns: true or false

(→ IsDiagonalMatrix (D.2.3))

Code

```
IsDiagonalMatrix := M ->
          IsDiagonalMatrix( DecideZero( Eval( M ), HomalgRing( M ) ) ),
```

F.3.4 ZeroRows (ResidueClassRing)

◊ ZeroRows(*C*) (function)

Returns: a homalg matrix over the ambient ring

(→ ZeroRows (D.2.4))

Code

```
ZeroRows := C -> ZeroRows( DecideZero( Eval( C ), HomalgRing( C ) ) ),
```

F.3.5 ZeroColumns (ResidueClassRing)

◊ ZeroColumns(*C*) (function)

Returns: a homalg matrix over the ambient ring

(→ ZeroColumns (D.2.5))

Code

```
ZeroColumns := C -> ZeroColumns( DecideZero( Eval( C ), HomalgRing( C ) ) ),
```

Appendix G

Debugging homalg

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

G.1 Increase the assertion level

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

`SetAssertionLevel( level );`

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

level	description
0	no assertion checks whatsoever
1	"high"-level homological assertions are checked
2	"mid"-level homological assertions are checked
3	"low"-level homological assertions are checked
4	assertions about basic matrix operations are checked (→ Appendix C) (these are among the operations often delegated to external systems)

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

Below you can find the record of the available level-4 assertions, which is a GAP-component of every homalg ring. Each assertion can thus be overwritten by package developers or even ordinary users.

Code

```
asserts :=
  rec(
```

```
BasisOfRowsCoeff :=
  function( B, T, M ) return B = T * M; end,

BasisOfColumnsCoeff :=
  function( B, M, T ) return B = M * T; end,

DecideZeroRows_Effectively :=
  function( M, A, B ) return M = DecideZeroRows( A, B ); end,

DecideZeroColumns_Effectively :=
  function( M, A, B ) return M = DecideZeroColumns( A, B ); end,

DecideZeroRowsEffectively :=
  function( M, A, T, B ) return M = A + T * B; end,

DecideZeroColumnsEffectively :=
  function( M, A, B, T ) return M = A + B * T; end,

DecideZeroRowsWRTNonBasis :=
  function( B )
    local R;
    R := HomalgRing( B );
    if not ( HasIsBasisOfRowsMatrix( B ) and
             IsBasisOfRowsMatrix( B ) ) and
       IsBound( R!.DecideZeroWRTNonBasis ) then
        if R!.DecideZeroWRTNonBasis = "warn" then
            Info( InfoWarning, 1,
                  "about to reduce with respect to a matrix",
                  "with IsBasisOfRowsMatrix not set to true" );
        elif R!.DecideZeroWRTNonBasis = "error" then
            Error( "about to reduce with respect to a matrix",
                   "with IsBasisOfRowsMatrix not set to true\n" );
        fi;
    fi;
  end,

DecideZeroColumnsWRTNonBasis :=
  function( B )
    local R;
    R := HomalgRing( B );
    if not ( HasIsBasisOfColumnsMatrix( B ) and
             IsBasisOfColumnsMatrix( B ) ) and
       IsBound( R!.DecideZeroWRTNonBasis ) then
        if R!.DecideZeroWRTNonBasis = "warn" then
            Info( InfoWarning, 1,
                  "about to reduce with respect to a matrix",
                  "with IsBasisOfColumnsMatrix not set to true" );
        elif R!.DecideZeroWRTNonBasis = "error" then
            Error( "about to reduce with respect to a matrix",
                   "with IsBasisOfColumnsMatrix not set to true\n" );
        fi;
    fi;
  end,
```

```
        ReducedBasisOfRowModule :=
          function( M, B )
            return GenerateSameRowModule( B, BasisOfRowModule( M ) );
          end,

        ReducedBasisOfColumnModule :=
          function( M, B )
            return GenerateSameColumnModule( B, BasisOfColumnModule( M ) );
          end,

        ReducedSyzygiesGeneratorsOfRows :=
          function( M, S )
            return GenerateSameRowModule( S, SyzygiesGeneratorsOfRows( M ) );
          end,

        ReducedSyzygiesGeneratorsOfColumns :=
          function( M, S )
            return GenerateSameColumnModule( S, SyzygiesGeneratorsOfColumns( M ) );
          end,

      );
```

G.2 Use homalgMode

G.2.1 homalgMode

◊ homalgMode(*str[, str2]*) (method)

This function sets different modes which influence how much of the basic matrix operations and the logical matrix methods become visible (→ Appendices C, E). Handling the string *str* is *not* case-sensitive. If a second string *str2* is given, then homalgMode(*str2*) is invoked at the end. In case you let homalg delegate matrix operations to an external system the you might also want to check homalgIOMode in the HomalgToCAS package manual.

str	*str* (long form)	mode description
""	""	the default mode, i.e. the computation protocol won't be visible (homalgMode( ) is a short form for homalgMode(""))
"b"	"basic"	make the basic matrix operations visible + homalgMode("logic")
"d"	"debug"	same as "basic" but also makes Row/ColumnReducedEchelonForm visible
"l"	"logic"	make the logical methods in LIMAT and COLEM visible

All modes other than the "default"-mode only set their specific values and leave the other values untouched, which allows combining them to some extent. This also means that in order to get from

one mode to a new mode (without the aim to combine them) one needs to reset to the ”default”-mode first. This can be done using `homalgMode( ”” , new_mode );`

Code

```
InstallGlobalFunction( homalgMode,
  function( arg )
    local nargs, mode, s;

    nargs := Length( arg );

    if nargs = 0 or ( IsString( arg[1] ) and arg[1] = "" ) then
        mode := "default";
    elif IsString( arg[1] ) then        ## now we know, the string is not empty
        s := arg[1];
        if LowercaseString( s{[1]} ) = "b" then
            mode := "basic";
        elif LowercaseString( s{[1]} ) = "d" then
            mode := "debug";
        elif LowercaseString( s{[1]} ) = "l" then
            mode := "logic";
        else
            mode := "";
        fi;
    else
        Error( "the first argument must be a string\n" );
    fi;

    if mode = "default" then
        HOMALG.color_display := false;
        SetInfoLevel( InfoCOLEM, 1 );
        SetInfoLevel( InfoLIMAT, 1 );
        SetInfoLevel( InfoHomalgBasicOperations, 1 );
    elif mode = "basic" then
        SetInfoLevel( InfoHomalgBasicOperations, 3 );
        homalgMode( "logic" );
    elif mode = "debug" then
        SetInfoLevel( InfoHomalgBasicOperations, 4 );
        homalgMode( "logic" );
    elif mode = "logic" then
        HOMALG.color_display := true;
        SetInfoLevel( InfoCOLEM, 2 );
        SetInfoLevel( InfoLIMAT, 2 );
    fi;

    if nargs > 1 and IsString( arg[2] ) then
        homalgMode( arg[2] );
    fi;

end );
```

Appendix H

The Core Packages and the Idea behind their Splitting

I will try to explain the idea behind splitting the 5 *core packages*:

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

H.1 The 5=1+4 split

The following is an attempt to explain the 5=1+4 split.

H.1.1 Logically independent

The package homalg is logically independent from all other packages in the project. And among the 5 core packages it is the only package that has to do with mathematics. The remaining four packages are of technical nature. More precisely, homalg is a stand alone package, that offers abstract homological constructions for any computable ring. But since the ring of integers (at least up till now) is the only ring which for the purposes of homological algebra is *sufficiently supported* in GAP (→ 1.1.4), homalg can put the above mentioned abstract constructions into action only for the ring of integers and by generic (but of course non-efficient) methods for any of its residue class rings (Simon Görtzen's package Gauss adds the missing sufficient support for $\mathbb{Z}/p^n$ and $\mathbb{Q}$ to GAP and his other package GaussForHomalg makes this support visible to homalg).

H.1.2 Black boxes

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

the concern of homalg. Even the GAP representation for external rings, external ring elements, and external matrices are declared in the package HomalgToCAS and not in homalg.

H.1.3 Summing up

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

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

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

H.2.1 HomalgToCAS

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

H.2.2 IO_ForHomalg and Alternatives

The package IO_ForHomalg (which needs HomalgToCAS) allows GAP to communicate via I/O-streams with computer algebra systems that come with a terminal interface. IO_ForHomalg uses Max Neunhöffer's IO package, yet it is independent from the specific computer algebra system, as long as the latter provides a terminal interface. Splitting IO_ForHomalg from HomalgToCAS gives the freedom to replace the former by another package that lets GAP communicate with an external system using a different technology. So making IO_ForHomalg a package of its own makes it clear for developers of a new communication method which package of the homalg project has to be imitated/replaced. To be concrete, Thomas Bächler wrote a package called MapleForHomalg that enables GAP to communicate with Maple without the need for a terminal interface, as, for example, such an interface is missing from recent versions of Maple on MAC OS X starting from Maple 10.

H.2.3 RingsForHomalg

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

H.2.4 Your own RingsForHomalg

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

H.2.5 ExamplesForHomalg

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

H.2.6 Documentation

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

H.2.7 Crediting

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

H.2.8 Stability

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

Appendix I

Overview of the homalg Package Source Code

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

I.1 Rings, Ring Maps, Matrices, Relations, and Generators

Filename `.gd/.gi`	Content
`homalg`	definitions of the basic GAP4 categories and some tool functions (e.g. `homalgMode`)
`homalgTable`	dictionaries between homalg and the computing engines
`HomalgRing`	internal and external rings
`HomalgRingMap`	ring maps
`HomalgMatrix`	internal and external matrices
`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*

I.2 The Basic Objects

Filename `.gd/.gi`	Content
`HomalgModule`	modules and submodules allowing several presentations linked with transition matrices
`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
`HomalgFunctor`	constructors of (multi) functors of module categories (yet over the same ring), left derivation of covariant functors, right derivation of contravariant functors, left satellites of covariant functors, right satellites of contravariant functors, and composition of functors
`HomalgDiagram`	Betti diagrams

Table: *The homalg package files (continued)*

I.3 The Low Level Algorithms

In the following CAS or CASystem mean computer algebra systems.

Filename .gd/.gi	Content
Tools	the elementary matrix operations that can be overwritten using the homalgTable (and hence delegable even to other CASystems)
Service	the three operations: basis, reduction, and syzygies; they can also be overwritten using the homalgTable (and hence delegable even to other CASystems)
Basic	higher level operations for matrices (cannot be overwritten using the homalgTable)

Table: *The homalg package files (continued)*

I.4 The High Level Algorithms

Filename .gd/.gi	Content
Modules	subfactors, resolutions, syzygy modules, parameterizations, intersections, annihilators
Maps	resolutions, (co)kernel sequences
Complexes	(co)homology, horse shoe lemma, connecting homomorphisms, Cartan-Eilenberg resolution
ChainMaps	(co)homology
SpectralSequences	Grothendieck bicomplexes associated to two composable functors, spectral sequences of bicomplexes, Grothendieck spectral sequences
Filtrations	spectral filtrations, i.e. filtrations induced by spectral sequences of bicomplexes, purity filtration
ToolFunctors	composition, addition, substraction, stacking, augmentation, and post dividing maps
BasicFunctors	cokernel, image, kernel, tensor product, Hom, Ext, Tor, RHom, LTensorProduct, HomHom, LHomHom, BaseChange (preliminary)
OtherFunctors	torsion submodule, torsion free factor, direct sum, pullback, pushout, Auslander dual

Table: *The homalg package files (continued)*

I.5 Logical Implications for homalg Objects

Filename .gd/.gi	Content
LIRNG	logical implications for rings
LIMAP	logical implications for ring maps
LIMAT	logical implications for matrices
COLEM	clever operations for lazy evaluated matrices
LIMOD	logical implications for modules
LIMOR	logical implications for morphisms
LICPX	logical implications for complexes

Table: *The homalg package files (continued)*

I.6 The subpackage ResidueClassRingForHomalg

Filename .gd/.gi	Content
ResidueClassRingForHomalg	some global variables
ResidueClassRing	residue class rings, their elements, and matrices, together with their constructors and operations
ResidueClassRingTools	the elementary matrix operations for matrices over residue class rings
ResidueClassRingBasic	the three operations: basis, reduction, and syzygies for matrices over residue class rings

Table: *The homalg package files (continued)*

I.7 The homalgTable for GAP4 built-in rings

For the purposes of homalg, the ring of integers is, at least up till now, the only ring which is properly supported in GAP4. The GAP4 built-in cababilities for polynomial rings (also univariate) and group rings do not statisfy the minimum requirements of homalg. The GAP4 package Gauss enables GAP to fullfil the homalg requirements for prime fields, and $\mathbb{Z}/p^n$.

Filename .gi	Content
Integers	the homalgTable for the ring of integers

Table: *The homalg package files (continued)*

References

[Bar] M. Barakat. Spectral Filtrations via Generalized Morphisms. arxiv.org/abs/0904.0240. 153, 154, 156, 158

[BR06] M. Barakat and D. 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), Sevilla (Spain)*, pages 29–32, 2006. http://homalg.math.rwth-aachen.de/maple/homalg_eaca06.pdf. 26

[BR08] M. Barakat and D. Robertz. homalg – A Meta-Package for Homological Algebra. *J. Algebra Appl.*, 7(3):299–317, 2008. arXiv:math.AC/0701146. 21, 66, 67, 125

[CE99] H. Cartan and S. Eilenberg. *Homological algebra*. Princeton Landmarks in Mathematics. Princeton University Press, Princeton, NJ, 1999. With an appendix by David A. Buchsbaum, Reprint of the 1956 original. 19

[GM03] S. I. Gelfand and Y. I. Manin. *Methods of homological algebra*. Springer Monographs in Mathematics. Springer-Verlag, Berlin, 2. edition, 2003. 19

[HS97] P. J. Hilton and U. Stammbach. *A course in homological algebra*, volume 4 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, second edition, 1997. 19

[ML63] S. Mac Lane. *Homology*. Die Grundlehren der mathematischen Wissenschaften, Bd. 114. Academic Press Inc., Publishers, New York, 1963. 19

[MR01] J. C. McConnell and J. C. Robson. *Noncommutative Noetherian rings*, volume 30 of *Graduate Studies in Mathematics*. American Mathematical Society, Providence, RI, revised edition, 2001. With the cooperation of L. W. Small. 43

[Rot79] J. J. Rotman. *An introduction to homological algebra*, volume 85 of *Pure and Applied Mathematics*. Academic Press Inc. [Harcourt Brace Jovanovich Publishers], New York, 1979. 19, 120

[Wei94] C. A. Weibel. *An introduction to homological algebra*, volume 38 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, Cambridge, 1994. 19

Index