

FreydCategories- ForCAP

**Freyd categories - Formal (co)kernels for
additive categories**

2020.09.21

21 September 2020

Sebastian Posur

Martin Bies

Sebastian Posur

Email: sebastian.posur@uni-siegen.de

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

Address: Department Mathematik

Universität Siegen

Walter-Flex-Straße 3

57068 Siegen

Germany

Martin Bies

Email: martin.bies@alumni.uni-heidelberg.de

Homepage: <https://www.ulb.ac.be/sciences/ptm/pmif/people.html>

Address: Physique Théorique et Mathématique

Université Libre de Bruxelles

Campus Plaine - CP 231

Building NO - Level 6 - Office O.6.111

1050 Brussels

Belgium

Contents

1	Basic operations	5
1.1	Weak kernel	5
1.2	Weak cokernel	7
1.3	Weak bi-fiber product	9
1.4	Biased weak fiber product	13
1.5	Weak bi-pushout	16
1.6	Biased weak pushout	19
1.7	Abelian constructions	21
2	Additive closure	24
2.1	GAP Categories	24
2.2	Constructors	24
2.3	Attributes	26
2.4	Operators	27
3	Example on additive closure	29
3.1	Using matrix data structures	29
4	Adelman category	31
4.1	GAP Categories	31
4.2	Constructors	32
4.3	Attributes and Properties	33
5	Category of rows	35
5.1	GAP Categories	35
6	Example on category of rows	36
6.1	Constructors of objects	36
6.2	Constructors of morphisms	36
6.3	A few categorical constructions for category of rows	37
6.4	Simplifications	42
7	Category of columns	45
7.1	GAP Categories	45

8	Example on category of columns	46
8.1	Constructors of objects	46
8.2	Constructors of morphisms	46
8.3	A few categorical constructions for category of columns	47
9	Category of graded rows and category of graded columns	52
9.1	Constructors	52
9.2	Attributes	53
9.3	GAP Categories	54
9.4	Tools to simplify code	55
10	Cokernel image closure	58
11	Freyd category	59
11.1	Internal Hom-Embedding	59
11.2	Convenient methods for tensor products of freyd objects and morphisms	59
12	Examples and Tests	60
12.1	Adelman 5 lemma	60
12.2	Adelman category basics for category of rows	61
12.3	Adelman category basics for for additive closure of algebroids	62
12.4	Adelman category basics for category of columns	63
12.5	Adelman category basics	64
12.6	Adelman snake lemma	65
12.7	Basics based on category of rows	66
12.8	Basics of additive closure	69
12.9	Basics based on category of columns	70
12.10	Cokernel image closure in category of rows	72
12.11	Cokernel image closure in category of columns	74
12.12	Grade filtration	75
12.13	Groups as categories	77
12.14	Homomorphisms between f.p. functors based on category of rows	77
12.15	Homomorphisms between f.p. functors based on category of columns	78
12.16	Linear closure of categories	78
12.17	Matrices over $\mathbb{Z}P\ K$	79
12.18	Matrices over $\mathbb{Z}G$	79
12.19	Prosets	80
12.20	Quiver rows basic	81
12.21	Quiver rows over the integers	83
12.22	Category of relations	84
12.23	Rings as Ab-categories	85
12.24	Snake lemma first proof	86
12.25	Snake lemma second proof	87
12.26	Subobject lattice	88
12.27	Adelman category theorem	88

13 Linear closure of a category	89
13.1 Functors	89
14 Examples on graded rows and columns	90
14.1 Freyd category of graded rows	90
14.2 Freyd category of graded columns	97
14.3 Constructors of objects and reduction of degree lists	104
14.4 Constructors of morphisms	105
14.5 The GAP categories	106
14.6 A few categorical constructions for graded rows	106
14.7 A few categorical constructions for graded columns	114
14.8 Additional examples on monoidal structure for graded rows	122
14.9 Additional examples on monoidal structure for graded columns	122
14.10 Examples to test Tools methods in graded rows/cols	123
15 Category of rows and columns over a field	125
15.1 Abelian operations for rows	125
16 Example on tensor products in Freyd categories	128
16.1 Tensor products for categories of rows	128
16.2 Tensor products for categories of columns	129
17 The CAP category of graded module presentations for CAP by use of Freyd categories	130
17.1 CAP categories	130
17.2 The GAP categories for graded module presentations for CAP	130
17.3 The GAP categories for graded module presentation morphisms for CAP	131
Index	132

Chapter 1

Basic operations

1.1 Weak kernel

For a given morphism $\alpha : A \rightarrow B$, a weak kernel of α consists of three parts:

- an object K ,
- a morphism $\iota : K \rightarrow A$ such that $\alpha \circ \iota \sim_{K,B} 0$,
- a dependent function u mapping each morphism $\tau : T \rightarrow A$ satisfying $\alpha \circ \tau \sim_{T,B} 0$ to a morphism $u(\tau) : T \rightarrow K$ such that $\iota \circ u(\tau) \sim_{T,A} \tau$.

The triple (K, ι, u) is called a *weak kernel* of α . We denote the object K of such a triple by $\text{WeakKernelObject}(\alpha)$. We say that the morphism $u(\tau)$ is induced by the *universal property of the weak kernel*.



1.1.1 WeakKernelObject (for IsCapCategoryMorphism)

▷ `WeakKernelObject(alpha)`

(attribute)

Returns: an object

The argument is a morphism α . The output is the weak kernel K of α .

1.1.2 WeakKernelEmbedding (for IsCapCategoryMorphism)

▷ `WeakKernelEmbedding(alpha)`

(attribute)

Returns: a morphism in $\text{Hom}(\text{WeakKernelObject}(\alpha), A)$

The argument is a morphism $\alpha : A \rightarrow B$. The output is the weak kernel embedding $\iota : \text{WeakKernelObject}(\alpha) \rightarrow A$.

1.1.3 WeakKernelEmbeddingWithGivenWeakKernelObject (for IsCapCategoryMorphism, IsCapCategoryObject)

▷ `WeakKernelEmbeddingWithGivenWeakKernelObject(alpha, K)` (operation)

Returns: a morphism in $\text{Hom}(K, A)$

The arguments are a morphism $\alpha : A \rightarrow B$ and an object $K = \text{WeakKernelObject}(\alpha)$. The output is the weak kernel embedding $\iota : K \rightarrow A$.

1.1.4 WeakKernelLift (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `WeakKernelLift(alpha, tau)` (operation)

Returns: a morphism in $\text{Hom}(T, \text{WeakKernelObject}(\alpha))$

The arguments are a morphism $\alpha : A \rightarrow B$ and a test morphism $\tau : T \rightarrow A$ satisfying $\alpha \circ \tau \sim_{T,B} 0$. The output is the morphism $u(\tau) : T \rightarrow \text{WeakKernelObject}(\alpha)$ given by the universal property of the weak kernel.

1.1.5 WeakKernelLiftWithGivenWeakKernelObject (for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject)

▷ `WeakKernelLiftWithGivenWeakKernelObject(alpha, tau, K)` (operation)

Returns: a morphism in $\text{Hom}(T, K)$

The arguments are a morphism $\alpha : A \rightarrow B$, a test morphism $\tau : T \rightarrow A$ satisfying $\alpha \circ \tau \sim_{T,B} 0$, and an object $K = \text{WeakKernelObject}(\alpha)$. The output is the morphism $u(\tau) : T \rightarrow K$ given by the universal property of the weak kernel.

1.1.6 AddWeakKernelObject (for IsCapCategory, IsFunction)

▷ `AddWeakKernelObject(C, F)` (operation)

Returns: nothing

The arguments are a category C and a function F . This operation adds the given function F to the category for the basic operation `WeakKernelObject`. $F : \alpha \mapsto \text{WeakKernelObject}(\alpha)$.

1.1.7 AddWeakKernelEmbedding (for IsCapCategory, IsFunction)

▷ `AddWeakKernelEmbedding(C, F)` (operation)

Returns: nothing

The arguments are a category C and a function F . This operation adds the given function F to the category for the basic operation `WeakKernelEmbedding`. $F : \alpha \mapsto \iota$.

1.1.8 AddWeakKernelEmbeddingWithGivenWeakKernelObject (for IsCapCategory, IsFunction)

▷ `AddWeakKernelEmbeddingWithGivenWeakKernelObject(C, F)` (operation)

Returns: nothing

The arguments are a category C and a function F . This operation adds the given function F to the category for the basic operation `WeakKernelEmbeddingWithGivenWeakKernelObject`. $F : (\alpha, K) \mapsto \iota$.

1.1.9 AddWeakKernelLift (for IsCapCategory, IsFunction)

▷ `AddWeakKernelLift(C, F)`

(operation)

Returns: nothing

The arguments are a category C and a function F . This operation adds the given function F to the category for the basic operation `WeakKernelLift`. $F : (\alpha, \tau) \mapsto u(\tau)$.

1.1.10 AddWeakKernelLiftWithGivenWeakKernelObject (for IsCapCategory, IsFunction)

▷ `AddWeakKernelLiftWithGivenWeakKernelObject(C, F)`

(operation)

Returns: nothing

The arguments are a category C and a function F . This operation adds the given function F to the category for the basic operation `WeakKernelLiftWithGivenWeakKernelObject`. $F : (\alpha, \tau, K) \mapsto u$.

1.2 Weak cokernel

For a given morphism $\alpha : A \rightarrow B$, a weak cokernel of α consists of three parts:

- an object K ,
- a morphism $\varepsilon : B \rightarrow K$ such that $\varepsilon \circ \alpha \sim_{A,K} 0$,
- a dependent function u mapping each $\tau : B \rightarrow T$ satisfying $\tau \circ \alpha \sim_{A,T} 0$ to a morphism $u(\tau) : K \rightarrow T$ such that $u(\tau) \circ \varepsilon \sim_{B,T} \tau$.

The triple (K, ε, u) is called a *weak cokernel* of α . We denote the object K of such a triple by `WeakCokernelObject(α)`. We say that the morphism $u(\tau)$ is induced by the *universal property of the weak cokernel*.



1.2.1 WeakCokernelObject (for IsCapCategoryMorphism)

▷ `WeakCokernelObject(alpha)`

(attribute)

Returns: an object

The argument is a morphism $\alpha : A \rightarrow B$. The output is the weak cokernel K of α .

1.2.2 WeakCokernelProjection (for IsCapCategoryMorphism)

▷ `WeakCokernelProjection(alpha)` (attribute)

Returns: a morphism in $\text{Hom}(B, \text{WeakCokernelObject}(\alpha))$

The argument is a morphism $\alpha : A \rightarrow B$. The output is the weak cokernel projection $\varepsilon : B \rightarrow \text{WeakCokernelObject}(\alpha)$.

1.2.3 WeakCokernelProjectionWithGivenWeakCokernelObject (for IsCapCategoryMorphism, IsCapCategoryObject)

▷ `WeakCokernelProjectionWithGivenWeakCokernelObject(alpha, K)` (operation)

Returns: a morphism in $\text{Hom}(B, K)$

The arguments are a morphism $\alpha : A \rightarrow B$ and an object $K = \text{WeakCokernelObject}(\alpha)$. The output is the weak cokernel projection $\varepsilon : B \rightarrow \text{WeakCokernelObject}(\alpha)$.

1.2.4 WeakCokernelColift (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `WeakCokernelColift(alpha, tau)` (operation)

Returns: a morphism in $\text{Hom}(\text{WeakCokernelObject}(\alpha), T)$

The arguments are a morphism $\alpha : A \rightarrow B$ and a test morphism $\tau : B \rightarrow T$ satisfying $\tau \circ \alpha \sim_{A,T} 0$. The output is the morphism $u(\tau) : \text{WeakCokernelObject}(\alpha) \rightarrow T$ given by the universal property of the weak cokernel.

1.2.5 WeakCokernelColiftWithGivenWeakCokernelObject (for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject)

▷ `WeakCokernelColiftWithGivenWeakCokernelObject(alpha, tau, K)` (operation)

Returns: a morphism in $\text{Hom}(K, T)$

The arguments are a morphism $\alpha : A \rightarrow B$, a test morphism $\tau : B \rightarrow T$ satisfying $\tau \circ \alpha \sim_{A,T} 0$, and an object $K = \text{WeakCokernelObject}(\alpha)$. The output is the morphism $u(\tau) : K \rightarrow T$ given by the universal property of the weak cokernel.

1.2.6 AddWeakCokernelObject (for IsCapCategory, IsFunction)

▷ `AddWeakCokernelObject(C, F)` (operation)

Returns: nothing

The arguments are a category C and a function F . This operation adds the given function F to the category for the basic operation `WeakCokernelObject`. $F : \alpha \mapsto K$.

1.2.7 AddWeakCokernelProjection (for IsCapCategory, IsFunction)

▷ `AddWeakCokernelProjection(C, F)` (operation)

Returns: nothing

The arguments are a category C and a function F . This operation adds the given function F to the category for the basic operation `WeakCokernelProjection`. $F : \alpha \mapsto \varepsilon$.

1.2.8 AddWeakCokernelProjectionWithGivenWeakCokernelObject (for IsCapCategory, IsFunction)

▷ AddWeakCokernelProjectionWithGivenWeakCokernelObject(C, F) (operation)

Returns: nothing

The arguments are a category C and a function F . This operation adds the given function F to the category for the basic operation WeakCokernelProjectionWithGivenWeakCokernelObject. $F : (\alpha, K) \mapsto \varepsilon$.

1.2.9 AddWeakCokernelColift (for IsCapCategory, IsFunction)

▷ AddWeakCokernelColift(C, F) (operation)

Returns: nothing

The arguments are a category C and a function F . This operation adds the given function F to the category for the basic operation WeakCokernelColift. $F : (\alpha, \tau) \mapsto u(\tau)$.

1.2.10 AddWeakCokernelColiftWithGivenWeakCokernelObject (for IsCapCategory, IsFunction)

▷ AddWeakCokernelColiftWithGivenWeakCokernelObject(C, F) (operation)

Returns: nothing

The arguments are a category C and a function F . This operation adds the given function F to the category for the basic operation WeakCokernelColiftWithGivenWeakCokernelObject. $F : (\alpha, \tau, K) \mapsto u(\tau)$.

1.3 Weak bi-fiber product

For a given pair of morphisms $(\alpha : A \rightarrow B, \beta : C \rightarrow B)$, a weak bi-fiber product of (α, β) consists of three parts:

- an object P ,
- morphisms $\pi_1 : P \rightarrow A, \pi_2 : P \rightarrow B$ such that $\alpha \circ \pi_1 \sim_{P,B} \beta \circ \pi_2$,
- a dependent function u mapping each pair $\tau = (\tau_1, \tau_2)$ of morphisms $\tau_1 : T \rightarrow A, \tau_2 : T \rightarrow C$ with the property $\alpha \circ \tau_1 \sim_{T,B} \beta \circ \tau_2$ to a morphism $u(\tau) : T \rightarrow P$ such that $\pi_1 \circ u(\tau) \sim_{A,T} \tau_1$ and $\pi_2 \circ u(\tau) \sim_{C,T} \tau_2$.

The quadrupel (P, π_1, π_2, u) is called a *weak bi-fiber product* of (α, β) . We denote the object P of such a quadrupel by $\text{WeakBiFiberProduct}(\alpha, \beta)$. We say that the morphism $u(\tau)$ is induced by the *universal property of the weak bi-fiber product*.



1.3.1 WeakBiFiberProduct (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ WeakBiFiberProduct(alpha, beta) (operation)

Returns: an object

The arguments are two morphisms $\alpha : A \rightarrow B$, $\beta : C \rightarrow B$. The output is the weak bi-fiber product P of α and β .

1.3.2 ProjectionInFirstFactorOfWeakBiFiberProduct (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ ProjectionInFirstFactorOfWeakBiFiberProduct(alpha, beta) (operation)

Returns: a morphism in $\text{Hom}(P, A)$

The arguments are two morphisms $\alpha : A \rightarrow B$, $\beta : C \rightarrow B$. The output is the first weak bi-fiber product projection $\pi_1 : P \rightarrow A$.

1.3.3 ProjectionInFirstFactorOfWeakBiFiberProductWithGivenWeakBiFiberProduct (for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject)

▷ ProjectionInFirstFactorOfWeakBiFiberProductWithGivenWeakBiFiberProduct(alpha, beta, P) (operation)

Returns: a morphism in $\text{Hom}(P, A)$

The arguments are two morphisms $\alpha : A \rightarrow B$, $\beta : C \rightarrow B$ and an object $P = \text{WeakBiFiberProduct}(\alpha, \beta)$. The output is the first weak bi-fiber product projection $\pi_1 : P \rightarrow A$.

1.3.4 ProjectionInSecondFactorOfWeakBiFiberProduct (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ ProjectionInSecondFactorOfWeakBiFiberProduct(alpha, beta) (operation)

Returns: a morphism in $\text{Hom}(P, C)$

The arguments are two morphisms $\alpha : A \rightarrow B$, $\beta : C \rightarrow B$. The output is the second weak bi-fiber product projection $\pi_2 : P \rightarrow C$.

1.3.5 ProjectionInSecondFactorOfWeakBiFiberProductWithGivenWeakBiFiberProduct (for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject)

▷ `ProjectionInSecondFactorOfWeakBiFiberProductWithGivenWeakBiFiberProduct(alpha, beta, P)` (operation)

Returns: a morphism in $\text{Hom}(P, C)$

The arguments are two morphisms $\alpha : A \rightarrow B$, $\beta : C \rightarrow B$ and an object $P = \text{WeakBiFiberProduct}(\alpha, \beta)$. The output is the second weak bi-fiber product projection $\pi_2 : P \rightarrow C$.

1.3.6 UniversalMorphismIntoWeakBiFiberProduct (for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `UniversalMorphismIntoWeakBiFiberProduct(alpha, beta, tau_1, tau_2)` (operation)

Returns: a morphism in $\text{Hom}(T, P)$

The arguments are four morphisms $\alpha : A \rightarrow B$, $\beta : C \rightarrow B$, $\tau_1 : T \rightarrow A$, $\tau_2 : T \rightarrow C$. The output is the morphism $u(\tau)$ induced by the universal property of the weak bi-fiber product P of α and β .

1.3.7 UniversalMorphismIntoWeakBiFiberProductWithGivenWeakBiFiberProduct (for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject)

▷ `UniversalMorphismIntoWeakBiFiberProductWithGivenWeakBiFiberProduct(alpha, beta, tau_1, tau_2, P)` (operation)

Returns: a morphism in $\text{Hom}(T, P)$

The arguments are four morphisms $\alpha : A \rightarrow B$, $\beta : C \rightarrow B$, $\tau_1 : T \rightarrow A$, $\tau_2 : T \rightarrow C$ and an object $P = \text{WeakBiFiberProduct}(\alpha, \beta)$. The output is the morphism $u(\tau)$ induced by the universal property of the weak bi-fiber product P .

1.3.8 WeakBiFiberProductMorphismToDirectSum (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `WeakBiFiberProductMorphismToDirectSum(alpha, beta)` (operation)

Returns: a morphism in $\text{Hom}(P, A \oplus C)$

The arguments are two morphisms $\alpha : A \rightarrow B$, $\beta : C \rightarrow B$. The output is the morphism $P \rightarrow A \oplus C$ obtained from the two weak bi-fiber product projections π_1 and π_2 and the universal property of the direct sum.

1.3.9 AddWeakBiFiberProduct (for IsCapCategory, IsFunction)

▷ `AddWeakBiFiberProduct(C, F)` (operation)

Returns: nothing

The arguments are a category C and a function F . This operation adds the given function F to the category for the basic operation `WeakBiFiberProduct`. $F : (\alpha, \beta) \mapsto P$

1.3.10 AddProjectionInFirstFactorOfWeakBiFiberProduct (for IsCapCategory, IsFunction)

▷ AddProjectionInFirstFactorOfWeakBiFiberProduct(C, F) (operation)

Returns: nothing

The arguments are a category C and a function F . This operation adds the given function F to the category for the basic operation ProjectionInFirstFactorOfWeakBiFiberProduct. $F : (\alpha, \beta) \mapsto \pi_1$

1.3.11 AddProjectionInSecondFactorOfWeakBiFiberProduct (for IsCapCategory, IsFunction)

▷ AddProjectionInSecondFactorOfWeakBiFiberProduct(C, F) (operation)

Returns: nothing

The arguments are a category C and a function F . This operation adds the given function F to the category for the basic operation ProjectionInSecondFactorOfWeakBiFiberProduct. $F : (\alpha, \beta) \mapsto \pi_2$

1.3.12 AddProjectionInFirstFactorOfWeakBiFiberProductWithGivenWeakBiFiberProduct (for IsCapCategory, IsFunction)

▷ AddProjectionInFirstFactorOfWeakBiFiberProductWithGivenWeakBiFiberProduct(C, F) (operation)

Returns: nothing

The arguments are a category C and a function F . This operation adds the given function F to the category for the basic operation ProjectionInFirstFactorOfWeakBiFiberProductWithGivenWeakBiFiberProduct.

$F : (\alpha, \beta, P) \mapsto \pi_1$

1.3.13 AddProjectionInSecondFactorOfWeakBiFiberProductWithGivenWeakBiFiberProduct (for IsCapCategory, IsFunction)

▷ AddProjectionInSecondFactorOfWeakBiFiberProductWithGivenWeakBiFiberProduct(C, F) (operation)

Returns: nothing

The arguments are a category C and a function F . This operation adds the given function F to the category for the basic operation ProjectionInSecondFactorOfWeakBiFiberProductWithGivenWeakBiFiberProduct.

$F : (\alpha, \beta, P) \mapsto \pi_2$

1.3.14 AddUniversalMorphismIntoWeakBiFiberProduct (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismIntoWeakBiFiberProduct(C, F) (operation)

Returns: nothing

The arguments are a category C and a function F . This operation adds the given function F to the category for the basic operation UniversalMorphismIntoWeakBiFiberProduct. $F : (\alpha, \beta, \tau_1, \tau_2) \mapsto u(\tau)$

1.3.15 AddUniversalMorphismIntoWeakBiFiberProductWithGivenWeakBiFiberProduct (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismIntoWeakBiFiberProductWithGivenWeakBiFiberProduct(C, F)
(operation)

Returns: nothing

The arguments are a category C and a function F . This operation adds the given function F to the category for the basic operation UniversalMorphismIntoWeakBiFiberProductWithGivenWeakBiFiberProduct. $F : (\alpha, \beta, \tau_1, \tau_2, P) \mapsto u(\tau)$

1.3.16 AddWeakBiFiberProductMorphismToDirectSum (for IsCapCategory, IsFunction)

▷ AddWeakBiFiberProductMorphismToDirectSum(C, F)
(operation)

Returns: nothing

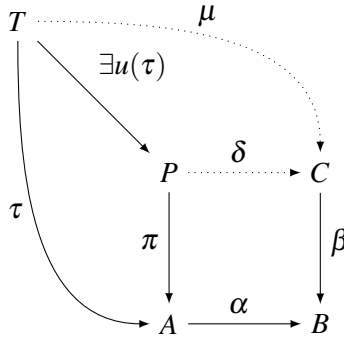
The arguments are a category C and a function F . This operation adds the given function F to the category for the basic operation WeakBiFiberProductMorphismToDirectSum. $F : (\alpha, \beta) \mapsto \text{WeakBiFiberProductMorphismToDirectSum}(\alpha, \beta)$

1.4 Biased weak fiber product

For a given pair of morphisms $(\alpha : A \rightarrow B, \beta : C \rightarrow B)$, a biased weak fiber product of (α, β) consists of three parts:

- an object P ,
- a morphism $\pi : P \rightarrow A$ such that there exists a morphism $\delta : P \rightarrow C$ such that $\beta \circ \delta \sim_{P,B} \alpha \circ \pi$,
- a dependent function u mapping each $\tau : T \rightarrow A$, which admits a morphism $\mu : T \rightarrow C$ with $\beta \circ \mu \sim_{T,B} \alpha \circ \tau$, to a morphism $u(\tau) : T \rightarrow P$ such that $\pi \circ u(\tau) \sim_{T,A} \tau$.

The triple (P, π, u) is called a *biased weak fiber product* of (α, β) . We denote the object P of such a triple by $\text{BiasedWeakFiberProduct}(\alpha, \beta)$. We say that the morphism $u(\tau)$ is induced by the *universal property of the biased weak fiber product*.



1.4.1 BiasedWeakFiberProduct (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `BiasedWeakFiberProduct(alpha, beta)` (operation)

Returns: an object

The arguments are two morphisms $\alpha : A \rightarrow B$, $\beta : C \rightarrow B$. The output is the biased weak fiber product P of α and β .

1.4.2 ProjectionOfBiasedWeakFiberProduct (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `ProjectionOfBiasedWeakFiberProduct(alpha, beta)` (operation)

Returns: a morphism in $\text{Hom}(P, A)$

The arguments are two morphisms $\alpha : A \rightarrow B$, $\beta : C \rightarrow B$. The output is the biased weak fiber product projection $\pi : P \rightarrow A$.

1.4.3 ProjectionOfBiasedWeakFiberProductWithGivenBiasedWeakFiberProduct (for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject)

▷ `ProjectionOfBiasedWeakFiberProductWithGivenBiasedWeakFiberProduct(alpha, beta, P)` (operation)

Returns: a morphism in $\text{Hom}(P, A)$

The arguments are two morphisms $\alpha : A \rightarrow B$, $\beta : C \rightarrow B$, and an object $P = \text{BiasedWeakFiberProduct}(\alpha, \beta)$. The output is the biased weak fiber product projection $\pi : P \rightarrow A$.

1.4.4 UniversalMorphismIntoBiasedWeakFiberProduct (for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `UniversalMorphismIntoBiasedWeakFiberProduct(alpha, beta, tau)` (operation)

Returns: a morphism in $\text{Hom}(T, P)$

The arguments are three morphisms $\alpha : A \rightarrow B$, $\beta : C \rightarrow B$, $\tau : T \rightarrow A$. The output is the morphism $u(\tau)$ induced by the universal property of the biased weak fiber product P of α and β .

1.4.5 UniversalMorphismIntoBiasedWeakFiberProductWithGivenBiasedWeakFiberProduct (for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject)

▷ `UniversalMorphismIntoBiasedWeakFiberProductWithGivenBiasedWeakFiberProduct(alpha, beta, tau, P)` (operation)

Returns: a morphism in $\text{Hom}(T, P)$

The arguments are three morphisms $\alpha : A \rightarrow B$, $\beta : C \rightarrow B$, $\tau : T \rightarrow A$ and an object $P = \text{BiasedWeakFiberProduct}(\alpha, \beta)$. The output is the morphism $u(\tau)$ induced by the universal property of the biased weak fiber product P of α and β .

1.4.6 AddBiasedWeakFiberProduct (for IsCapCategory, IsFunction)

▷ AddBiasedWeakFiberProduct(C, F) (operation)

Returns: nothing

The arguments are a category C and a function F . This operation adds the given function F to the category for the basic operation BiasedWeakFiberProduct. $F : (\alpha, \beta) \mapsto P$

1.4.7 AddProjectionOfBiasedWeakFiberProduct (for IsCapCategory, IsFunction)

▷ AddProjectionOfBiasedWeakFiberProduct(C, F) (operation)

Returns: nothing

The arguments are a category C and a function F . This operation adds the given function F to the category for the basic operation ProjectionOfBiasedWeakFiberProduct. $F : (\alpha, \beta) \mapsto \pi$

1.4.8 AddProjectionOfBiasedWeakFiberProductWithGivenBiasedWeakFiberProduct (for IsCapCategory, IsFunction)

▷ AddProjectionOfBiasedWeakFiberProductWithGivenBiasedWeakFiberProduct(C, F) (operation)

Returns: nothing

The arguments are a category C and a function F . This operation adds the given function F to the category for the basic operation ProjectionOfBiasedWeakFiberProductWithGivenBiasedWeakFiberProduct. $F : (\alpha, \beta, P) \mapsto \pi$

1.4.9 AddUniversalMorphismIntoBiasedWeakFiberProduct (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismIntoBiasedWeakFiberProduct(C, F) (operation)

Returns: nothing

The arguments are a category C and a function F . This operation adds the given function F to the category for the basic operation UniversalMorphismIntoBiasedWeakFiberProduct. $F : (\alpha, \beta, \tau) \mapsto u(\tau)$

1.4.10 AddUniversalMorphismIntoBiasedWeakFiberProductWithGivenBiasedWeakFiberProduct (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismIntoBiasedWeakFiberProductWithGivenBiasedWeakFiberProduct(C, F) (operation)

Returns: nothing

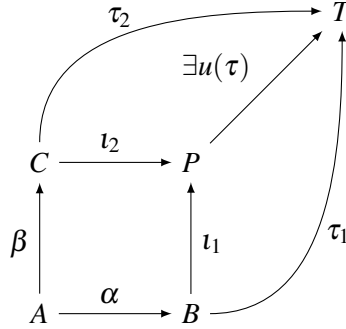
The arguments are a category C and a function F . This operation adds the given function F to the category for the basic operation UniversalMorphismIntoBiasedWeakFiberProductWithGivenBiasedWeakFiberProduct. $F : (\alpha, \beta, \tau, P) \mapsto u(\tau)$

1.5 Weak bi-pushout

For a given pair of morphisms $(\alpha : A \rightarrow B, \beta : A \rightarrow C)$, a weak bi-pushout of (α, β) consists of three parts:

- an object P ,
- morphisms $\iota_1 : B \rightarrow P$, $\iota_2 : C \rightarrow P$ such that $\iota_1 \circ \alpha \sim_{A,P} \iota_2 \circ \beta$,
- a dependent function u mapping each pair $\tau = (\tau_1, \tau_2)$ of morphisms $\tau_1 : B \rightarrow T$, $\tau_2 : C \rightarrow T$ with the property $\tau_1 \circ \alpha \sim_{A,T} \tau_2 \circ \beta$ to a morphism $u(\tau) : P \rightarrow T$ such that $u(\tau) \circ \iota_1 \sim_{B,T} \tau_1$ and $u(\tau) \circ \iota_2 \sim_{C,T} \tau_2$.

The quadrupel (P, ι_1, ι_2, u) is called a *weak bi-pushout* of (α, β) . We denote the object P of such a quadrupel by $\text{WeakBiPushout}(\alpha, \beta)$. We say that the morphism $u(\tau)$ is induced by the *universal property of the weak bi-pushout*.



1.5.1 WeakBiPushout (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `WeakBiPushout(alpha, beta)` (operation)

Returns: an object

The arguments are two morphisms $\alpha : A \rightarrow B$, $\beta : A \rightarrow C$. The output is the weak bi-pushout P of α and β .

1.5.2 InjectionOfFirstCofactorOfWeakBiPushout (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `InjectionOfFirstCofactorOfWeakBiPushout(alpha, beta)` (operation)

Returns: a morphism in $\text{Hom}(B, P)$

The arguments are two morphisms $\alpha : A \rightarrow B$, $\beta : A \rightarrow C$. The output is the first weak bi-pushout injection $\iota_1 : B \rightarrow P$.

1.5.3 InjectionOfSecondCofactorOfWeakBiPushout (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `InjectionOfSecondCofactorOfWeakBiPushout(alpha, beta)` (operation)

Returns: a morphism in $\text{Hom}(C, P)$

The arguments are two morphisms $\alpha : A \rightarrow B$, $\beta : A \rightarrow C$. The output is the second weak bi-pushout injection $\iota_2 : C \rightarrow P$.

1.5.4 InjectionOfFirstCofactorOfWeakBiPushoutWithGivenWeakBiPushout (for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject)

▷ `InjectionOfFirstCofactorOfWeakBiPushoutWithGivenWeakBiPushout(alpha, beta, P)` (operation)

Returns: a morphism in $\text{Hom}(B, P)$

The arguments are two morphisms $\alpha : A \rightarrow B$, $\beta : A \rightarrow C$ and an object $P = \text{WeakBiPushout}(\alpha, \beta)$. The output is the first weak bi-pushout injection $t_1 : B \rightarrow P$.

1.5.5 InjectionOfSecondCofactorOfWeakBiPushoutWithGivenWeakBiPushout (for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject)

▷ `InjectionOfSecondCofactorOfWeakBiPushoutWithGivenWeakBiPushout(alpha, beta, P)` (operation)

Returns: a morphism in $\text{Hom}(C, P)$

The arguments are two morphisms $\alpha : A \rightarrow B$, $\beta : A \rightarrow C$ and an object $P = \text{WeakBiPushout}(\alpha, \beta)$. The output is the second weak bi-pushout injection $t_2 : C \rightarrow P$.

1.5.6 UniversalMorphismFromWeakBiPushout (for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `UniversalMorphismFromWeakBiPushout(alpha, beta, tau_1, tau_2)` (operation)

Returns: a morphism in $\text{Hom}(P, T)$

The arguments are four morphisms $\alpha : A \rightarrow B$, $\beta : A \rightarrow C$, $\tau_1 : B \rightarrow T$, $\tau_2 : C \rightarrow T$. The output is the morphism $u(\tau)$ induced by the universal property of the weak bi-pushout P of α and β .

1.5.7 UniversalMorphismFromWeakBiPushoutWithGivenWeakBiPushout (for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject)

▷ `UniversalMorphismFromWeakBiPushoutWithGivenWeakBiPushout(alpha, beta, tau_1, tau_2, P)` (operation)

Returns: a morphism in $\text{Hom}(P, T)$

The arguments are four morphisms $\alpha : A \rightarrow B$, $\beta : A \rightarrow C$, $\tau_1 : B \rightarrow T$, $\tau_2 : C \rightarrow T$, and an object $P = \text{WeakBiPushout}(\alpha, \beta)$. The output is the morphism $u(\tau)$ induced by the universal property of the weak bi-pushout P of α and β .

1.5.8 DirectSumMorphismToWeakBiPushout (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `DirectSumMorphismToWeakBiPushout(alpha, beta)` (operation)

Returns: a morphism in $\text{Hom}(B \oplus C, P)$

The arguments are two morphisms $\alpha : A \rightarrow B$, $\beta : C \rightarrow B$. The output is the morphism $B \oplus C \rightarrow P$ obtained from the two weak bi-fiber product injections t_1 and t_2 and the universal property of the direct sum.

1.5.9 AddWeakBiPushout (for IsCapCategory, IsFunction)

▷ AddWeakBiPushout(C , F) (operation)

Returns: nothing

The arguments are a category C and a function F . This operation adds the given function F to the category for the basic operation WeakBiPushout. $F : (\alpha, \beta) \mapsto P$

1.5.10 AddInjectionOfFirstCofactorOfWeakBiPushout (for IsCapCategory, IsFunction)

▷ AddInjectionOfFirstCofactorOfWeakBiPushout(C , F) (operation)

Returns: nothing

The arguments are a category C and a function F . This operation adds the given function F to the category for the basic operation InjectionOfFirstCofactorOfWeakBiPushout. $F : (\alpha, \beta) \mapsto \iota_1$

1.5.11 AddInjectionOfSecondCofactorOfWeakBiPushout (for IsCapCategory, IsFunction)

▷ AddInjectionOfSecondCofactorOfWeakBiPushout(C , F) (operation)

Returns: nothing

The arguments are a category C and a function F . This operation adds the given function F to the category for the basic operation InjectionOfSecondCofactorOfWeakBiPushout. $F : (\alpha, \beta) \mapsto \iota_2$

1.5.12 AddInjectionOfFirstCofactorOfWeakBiPushoutWithGivenWeakBiPushout (for IsCapCategory, IsFunction)

▷ AddInjectionOfFirstCofactorOfWeakBiPushoutWithGivenWeakBiPushout(C , F) (operation)

Returns: nothing

The arguments are a category C and a function F . This operation adds the given function F to the category for the basic operation InjectionOfFirstCofactorOfWeakBiPushoutWithGivenWeakBiPushout. $F : (\alpha, \beta, P) \mapsto \iota_1$

1.5.13 AddInjectionOfSecondCofactorOfWeakBiPushoutWithGivenWeakBiPushout (for IsCapCategory, IsFunction)

▷ AddInjectionOfSecondCofactorOfWeakBiPushoutWithGivenWeakBiPushout(C , F) (operation)

Returns: nothing

The arguments are a category C and a function F . This operation adds the given function F to the category for the basic operation InjectionOfSecondCofactorOfWeakBiPushoutWithGivenWeakBiPushout. $F : (\alpha, \beta, P) \mapsto \iota_2$

1.5.14 AddUniversalMorphismFromWeakBiPushout (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismFromWeakBiPushout(C , F) (operation)

Returns: nothing

The arguments are a category C and a function F . This operation adds the given function F to the category for the basic operation `UniversalMorphismFromWeakBiPushout`. $F : (\alpha, \beta, \tau_1, \tau_2) \mapsto u(\tau)$

1.5.15 AddUniversalMorphismFromWeakBiPushoutWithGivenWeakBiPushout (for IsCapCategory, IsFunction)

▷ `AddUniversalMorphismFromWeakBiPushoutWithGivenWeakBiPushout(C, F)` (operation)

Returns: nothing

The arguments are a category C and a function F . This operation adds the given function F to the category for the basic operation `UniversalMorphismFromWeakBiPushoutWithGivenWeakBiPushout`. $F : (\alpha, \beta, \tau_1, \tau_2, P) \mapsto u(\tau)$

1.5.16 AddDirectSumMorphismToWeakBiPushout (for IsCapCategory, IsFunction)

▷ `AddDirectSumMorphismToWeakBiPushout(C, F)` (operation)

Returns: nothing

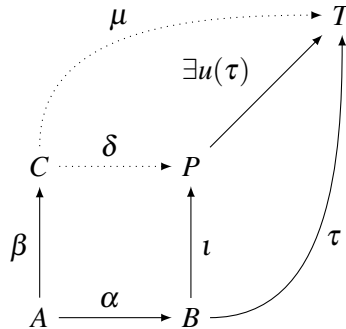
The arguments are a category C and a function F . This operation adds the given function F to the category for the basic operation `DirectSumMorphismToWeakBiPushout`. $F : (\alpha, \beta) \mapsto \text{DirectSumMorphismToWeakBiPushout}(\alpha, \beta)$

1.6 Biased weak pushout

For a given pair of morphisms $(\alpha : A \rightarrow B, \beta : A \rightarrow C)$, a biased weak pushout of (α, β) consists of three parts:

- an object P ,
- a morphism $\iota : B \rightarrow P$ such that there exists a morphism $\delta : C \rightarrow P$ such that $\delta \circ \beta \sim_{A,P} \iota \circ \alpha$,
- a dependent function u mapping each $\tau : B \rightarrow T$, which admits a morphism $\mu : C \rightarrow T$ with $\mu \circ \beta \sim_{B,T} \tau \circ \alpha$, to a morphism $u(\tau) : P \rightarrow T$ such that $u(\tau) \circ \iota \sim_{A,T} \tau$.

The triple (P, ι, u) is called a *biased weak pushout* of (α, β) . We denote the object P of such a triple by `BiasedWeakPushout` (α, β) . We say that the morphism $u(\tau)$ is induced by the *universal property of the biased weak pushout*.



1.6.1 BiasedWeakPushout (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `BiasedWeakPushout(alpha, beta)` (operation)

Returns: an object

The arguments are two morphisms $\alpha : A \rightarrow B$, $\beta : A \rightarrow C$. The output is the biased weak pushout P of α and β .

1.6.2 InjectionOfBiasedWeakPushout (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `InjectionOfBiasedWeakPushout(alpha, beta)` (operation)

Returns: a morphism in $\text{Hom}(B, P)$

The arguments are two morphisms $\alpha : A \rightarrow B$, $\beta : A \rightarrow C$. The output is the biased weak pushout injection $\iota : B \rightarrow P$.

1.6.3 InjectionOfBiasedWeakPushoutWithGivenBiasedWeakPushout (for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject)

▷ `InjectionOfBiasedWeakPushoutWithGivenBiasedWeakPushout(alpha, beta, P)` (operation)

Returns: a morphism in $\text{Hom}(B, P)$

The arguments are two morphisms $\alpha : A \rightarrow B$, $\beta : A \rightarrow C$ and an object $P = \text{BiasedWeakPushout}(\alpha, \beta)$. The output is the biased weak pushout injection $\iota : B \rightarrow P$.

1.6.4 UniversalMorphismFromBiasedWeakPushout (for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `UniversalMorphismFromBiasedWeakPushout(alpha, beta, tau)` (operation)

Returns: a morphism in $\text{Hom}(P, T)$

The arguments are three morphisms $\alpha : A \rightarrow B$, $\beta : A \rightarrow C$, $\tau : B \rightarrow T$. The output is the morphism $u(\tau)$ induced by the universal property of the biased weak pushout P of α and β .

1.6.5 UniversalMorphismFromBiasedWeakPushoutWithGivenBiasedWeakPushout (for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject)

▷ `UniversalMorphismFromBiasedWeakPushoutWithGivenBiasedWeakPushout(alpha, beta, tau, P)` (operation)

Returns: a morphism in $\text{Hom}(P, T)$

The arguments are three morphisms $\alpha : A \rightarrow B$, $\beta : A \rightarrow C$, $\tau : B \rightarrow T$ and an object $P = \text{BiasedWeakPushout}(\alpha, \beta)$. The output is the morphism $u(\tau)$ induced by the universal property of the biased weak pushout P of α and β .

1.6.6 AddBiasedWeakPushout (for IsCapCategory, IsFunction)

▷ `AddBiasedWeakPushout(C, F)` (operation)

Returns: nothing

The arguments are a category C and a function F . This operation adds the given function F to the category for the basic operation `BiasedWeakPushout`. $F : (\alpha, \beta) \mapsto P$

1.6.7 AddInjectionOfBiasedWeakPushout (for IsCapCategory, IsFunction)

▷ `AddInjectionOfBiasedWeakPushout(C, F)` (operation)

Returns: nothing

The arguments are a category C and a function F . This operation adds the given function F to the category for the basic operation `InjectionOfBiasedWeakPushout`. $F : (\alpha, \beta) \mapsto \iota$

1.6.8 AddInjectionOfBiasedWeakPushoutWithGivenBiasedWeakPushout (for IsCapCategory, IsFunction)

▷ `AddInjectionOfBiasedWeakPushoutWithGivenBiasedWeakPushout(C, F)` (operation)

Returns: nothing

The arguments are a category C and a function F . This operation adds the given function F to the category for the basic operation `InjectionOfBiasedWeakPushoutWithGivenBiasedWeakPushout`. $F : (\alpha, \beta, P) \mapsto \iota$

1.6.9 AddUniversalMorphismFromBiasedWeakPushout (for IsCapCategory, IsFunction)

▷ `AddUniversalMorphismFromBiasedWeakPushout(C, F)` (operation)

Returns: nothing

The arguments are a category C and a function F . This operation adds the given function F to the category for the basic operation `UniversalMorphismFromBiasedWeakPushout`. $F : (\alpha, \beta, \tau) \mapsto u(\tau)$

1.6.10 AddUniversalMorphismFromBiasedWeakPushoutWithGivenBiasedWeakPushout (for IsCapCategory, IsFunction)

▷ `AddUniversalMorphismFromBiasedWeakPushoutWithGivenBiasedWeakPushout(C, F)` (operation)

Returns: nothing

The arguments are a category C and a function F . This operation adds the given function F to the category for the basic operation `UniversalMorphismFromBiasedWeakPushoutWithGivenBiasedWeakPushout`. $F : (\alpha, \beta, \tau, P) \mapsto u(\tau)$

1.7 Abelian constructions

1.7.1 SomeProjectiveObjectForKernelObject (for IsCapCategoryMorphism)

▷ `SomeProjectiveObjectForKernelObject(alpha)` (attribute)

Returns: an object

The argument is a morphism α . The output is the source of `EpimorphismFromSomeProjectiveObjectForKernelObject` applied to α .

1.7.2 EpimorphismFromSomeProjectiveObjectForKernelObject (for IsCapCategory-Morphism)

▷ `EpimorphismFromSomeProjectiveObjectForKernelObject(alpha)` (attribute)

Returns: a morphism in $\text{Hom}(P, \text{KernelObject}(\alpha))$

The argument is a morphism α . The output is an epimorphism $\pi : P \rightarrow \text{KernelObject}(\alpha)$ with P a projective object.

1.7.3 EpimorphismFromSomeProjectiveObjectForKernelObjectWithGivenSomeProjectiveObjectForKernelObject (for IsCapCategoryMorphism, IsCapCategoryObject)

▷ `EpimorphismFromSomeProjectiveObjectForKernelObjectWithGivenSomeProjectiveObjectForKernelObject(alpha, P)` (operation)

Returns: a morphism in $\text{Hom}(P, \text{KernelObject}(\alpha))$

The arguments are a morphism α and an object $P = \text{SomeProjectiveObjectForKernelObject}(\alpha)$. The output is an epimorphism $\pi : P \rightarrow \text{KernelObject}(\alpha)$.

1.7.4 SomeInjectiveObjectForCokernelObject (for IsCapCategoryMorphism)

▷ `SomeInjectiveObjectForCokernelObject(alpha)` (attribute)

Returns: an object

The argument is a morphism α . The output is the range of `MonomorphismToSomeInjectiveObjectForCokernelObject` applied to α .

1.7.5 MonomorphismToSomeInjectiveObjectForCokernelObject (for IsCapCategoryMorphism)

▷ `MonomorphismToSomeInjectiveObjectForCokernelObject(alpha)` (attribute)

Returns: a morphism in $\text{Hom}(\text{CokernelObject}(\alpha), I)$

The argument is a morphism α . The output is a monomorphism $\iota : \text{CokernelObject}(\alpha) \rightarrow I$ with I an injective object.

1.7.6 MonomorphismToSomeInjectiveObjectForCokernelObjectWithGivenSomeInjectiveObjectForCokernelObject (for IsCapCategoryMorphism, IsCapCategoryObject)

▷ `MonomorphismToSomeInjectiveObjectForCokernelObjectWithGivenSomeInjectiveObjectForCokernelObject(alpha, I)` (operation)

Returns: a morphism in $\text{Hom}(\text{CokernelObject}(\alpha), I)$

The arguments are a morphism α and an object $I = \text{SomeInjectiveObjectForCokernelObject}(\alpha)$. The output is a monomorphism $\iota : \text{CokernelObject}(\alpha) \rightarrow I$.

1.7.7 AddSomeProjectiveObjectForKernelObject (for IsCapCategory, IsFunction)

▷ `AddSomeProjectiveObjectForKernelObject(C, F)` (operation)

Returns: nothing

The arguments are a category C and a function F . This operation adds the given function F to the category for the basic operation `SomeProjectiveObjectForKernelObject`. $F : \alpha \mapsto P$.

1.7.8 AddSomeInjectiveObjectForCokernelObject (for IsCapCategory, IsFunction)

▷ AddSomeInjectiveObjectForCokernelObject(C, F) (operation)

Returns: nothing

The arguments are a category C and a function F . This operation adds the given function F to the category for the basic operation SomeInjectiveObjectForCokernelObject. $F : \alpha \mapsto I$.

1.7.9 AddEpimorphismFromSomeProjectiveObjectForKernelObject (for IsCapCategory, IsFunction)

▷ AddEpimorphismFromSomeProjectiveObjectForKernelObject(C, F) (operation)

Returns: nothing

The arguments are a category C and a function F . This operation adds the given function F to the category for the basic operation EpimorphismFromSomeProjectiveObjectForKernelObject. $F : \alpha \mapsto \pi$.

1.7.10 AddMonomorphismToSomeInjectiveObjectForCokernelObject (for IsCapCategory, IsFunction)

▷ AddMonomorphismToSomeInjectiveObjectForCokernelObject(C, F) (operation)

Returns: nothing

The arguments are a category C and a function F . This operation adds the given function F to the category for the basic operation MonomorphismToSomeInjectiveObjectForCokernelObject. $F : \alpha \mapsto \iota$.

1.7.11 AddEpimorphismFromSomeProjectiveObjectForKernelObjectWithGivenSomeProjectiveObject (for IsCapCategory, IsFunction)

▷ AddEpimorphismFromSomeProjectiveObjectForKernelObjectWithGivenSomeProjectiveObjectForKernelObject(C, F) (operation)

Returns: nothing

The arguments are a category C and a function F . This operation adds the given function F to the category for the basic operation EpimorphismFromSomeProjectiveObjectForKernelObjectWithGivenSomeProjectiveObjectForKernelObject. $F : (\alpha, P) \mapsto \pi$.

1.7.12 AddMonomorphismToSomeInjectiveObjectForCokernelObjectWithGivenSomeInjectiveObject (for IsCapCategory, IsFunction)

▷ AddMonomorphismToSomeInjectiveObjectForCokernelObjectWithGivenSomeInjectiveObjectForCokernelObject(C, F) (operation)

Returns: nothing

The arguments are a category C and a function F . This operation adds the given function F to the category for the basic operation MonomorphismToSomeInjectiveObjectForCokernelObjectWithGivenSomeInjectiveObjectForCokernelObject. $F : (\alpha, I) \mapsto \iota$.

Chapter 2

Additive closure

2.1 GAP Categories

2.1.1 IsAdditiveClosureCategory (for IsCapCategory)

- ▷ `IsAdditiveClosureCategory(object)` (filter)
Returns: true or false
The GAP category of additive closures of Ab-categories.

2.1.2 IsAdditiveClosureObject (for IsCapCategoryObject)

- ▷ `IsAdditiveClosureObject(object)` (filter)
Returns: true or false
The GAP category of objects in additive closures of Ab-categories.

2.1.3 IsAdditiveClosureMorphism (for IsCapCategoryMorphism)

- ▷ `IsAdditiveClosureMorphism(object)` (filter)
Returns: true or false
The GAP category of morphisms in additive closures of Ab-categories.

2.2 Constructors

2.2.1 AdditiveClosure (for IsCapCategory)

- ▷ `AdditiveClosure(C)` (attribute)
Returns: the category C^\oplus
The argument is an Ab-category C . The output is its additive closure C^\oplus . By default, lists of lists are used as the underlying data structure for morphisms. This can be changed via the two options `matrix_element_as_morphism` and `list_list_as_matrix`, see `AdditiveClosureMorphism` (2.2.4) for details.

2.2.2 AdditiveClosureObject (for IsList, IsAdditiveClosureCategory)

- ▷ `AdditiveClosureObject(L, C^\oplus\oplus)` (operation)
Returns: an object in C^\oplus

The argument is a list of objects $L = [A_1, \dots, A_n]$ in an Ab-category C . The output is the formal direct sum $A_1 \oplus \dots \oplus A_n$ in the additive closure C^\oplus .

2.2.3 AsAdditiveClosureObject (for IsCapCategoryObject)

▷ `AsAdditiveClosureObject(A)` (attribute)
Returns: an object in C^\oplus

The argument is an object A in an Ab-category C . The output is the image of A under the inclusion functor $\iota : C \rightarrow C^\oplus$.

2.2.4 AdditiveClosureMorphism (for IsAdditiveClosureObject, IsObject, IsAdditiveClosureObject)

▷ `AdditiveClosureMorphism(A, M, B)` (operation)
Returns: a morphism in $\text{Hom}_{C^\oplus}(A, B)$

The arguments are formal direct sums $A = A_1 \oplus \dots \oplus A_m$, $B = B_1 \oplus \dots \oplus B_n$ in some additive category C^\oplus and an $m \times n$ matrix (see below) $M := (\alpha_{ij} : A_i \rightarrow B_j)_{ij}$ for $i = 1, \dots, m, j = 1, \dots, n$. The output is the formal morphism between A and B that is defined by M .

If $m \neq 0 \neq n$, M has to provide access to its elements via the operation `[,]`. In case that the elements of M first have to be wrapped to actually obtain morphisms in C , you can provide the function `matrix_element_as_morphism` (default: `IdFunc`) as an option to `AdditiveClosure` (2.2.1) which will internally be automatically applied to the elements of M . In this case you also have to provide the function `list_list_as_matrix` (default: `ReturnFirst`) as an option to `AdditiveClosure` (2.2.1): It gets passed a list of list of morphisms α_{ij} as well as m and n as above and has to return the corresponding matrix M . If `IsMatrixObj(M)`, then `NrRows(M)` resp. `NrCols(M)` must be m resp. n .

The default values of `matrix_element_as_morphism` and `list_list_as_matrix` allow to use lists of lists as the data structure of M .

2.2.5 AdditiveClosureMorphismListList (for IsAdditiveClosureObject, IsList, IsAdditiveClosureObject)

▷ `AdditiveClosureMorphismListList(A, L, B)` (operation)
Returns: a morphism in $\text{Hom}_{C^\oplus}(A, B)$

Input and return value are the same as for `AdditiveClosureMorphism` except that the matrix M can be given as a list (of lists) L to which `list_list_as_matrix` will be applied automatically.

2.2.6 AsAdditiveClosureMorphism (for IsCapCategoryMorphism)

▷ `AsAdditiveClosureMorphism(alpha)` (attribute)
Returns: a morphism in C^\oplus

The argument is a morphism α in an Ab-category C . The output is the image of α under the inclusion functor $\iota : C \rightarrow C^\oplus$.

2.2.7 InclusionFunctorInAdditiveClosure (for IsCapCategory)

▷ `InclusionFunctorInAdditiveClosure(C)` (attribute)
Returns: a functor $C \rightarrow C^\oplus$

The argument is an Ab-category C . The output is the inclusion functor $\iota : C \rightarrow C^\oplus$.

2.2.8 ExtendFunctorToAdditiveClosures (for IsCapFunctor)

▷ `ExtendFunctorToAdditiveClosures(F)` (attribute)

Returns: a functor $C^\oplus \rightarrow D^\oplus$

The argument is a functor $F : C \rightarrow D$, and the output is the extension functor $F^\oplus : C^\oplus \rightarrow D^\oplus$.

2.2.9 ExtendFunctorWithAdditiveRangeToFunctorFromAdditiveClosureOfSource (for IsCapFunctor)

▷ `ExtendFunctorWithAdditiveRangeToFunctorFromAdditiveClosureOfSource(F)` (attribute)

Returns: a functor $C^\oplus \rightarrow D$

The argument is a functor $F : C \rightarrow D$, where D is an additive category. The output is the extension functor $F^\oplus : C^\oplus \rightarrow D$.

2.2.10 ExtendFunctorToAdditiveClosureOfSource (for IsCapFunctor)

▷ `ExtendFunctorToAdditiveClosureOfSource(F)` (attribute)

Returns: a functor $C^\oplus \rightarrow D^\oplus$ or $C^\oplus \rightarrow D$

The argument is a functor $F : C \rightarrow D$. If D is not known to be an additive category, then return `ExtendFunctorToAdditiveClosures(F)`, otherwise return `ExtendFunctorWithAdditiveRangeToFunctorFromAdditiveClosureOfSource(F)`.

2.2.11 ExtendNaturalTransformationToAdditiveClosureOfSource (for IsCapNaturalTransformation)

▷ `ExtendNaturalTransformationToAdditiveClosureOfSource(eta)` (attribute)

Returns: a natural transformation from F^\oplus to G^\oplus

The argument is a natural transformation $\eta : (F : C \rightarrow D) \Rightarrow (G : C \rightarrow D)$ where D is an additive category. The output is the extension natural transformation $\eta^\oplus : (F^\oplus : C^\oplus \rightarrow D) \rightarrow (G^\oplus : C^\oplus \rightarrow D)$.

2.3 Attributes

2.3.1 UnderlyingCategory (for IsAdditiveClosureCategory)

▷ `UnderlyingCategory(A)` (attribute)

Returns: the category C

The argument is some additive closure category $A := C^\oplus$. The output is C .

2.3.2 ObjectList (for IsAdditiveClosureObject)

▷ `ObjectList(A)` (attribute)

Returns: a list of the objects in C

The argument is a formal direct sum $A := A_1 \oplus \dots \oplus A_m$ in some additive closure category C^\oplus . The output is the list $[A_1, \dots, A_m]$.

2.3.3 MorphismMatrix (for IsAdditiveClosureMorphism)

▷ `MorphismMatrix(alpha)` (attribute)

Returns: a list of lists the morphisms in C

The argument is a morphism $\alpha : A \rightarrow B$ between formal direct sums in some additive closure category C^\oplus . The output is the defining matrix of α .

2.3.4 NrRows (for IsAdditiveClosureMorphism)

▷ `NrRows(alpha)` (attribute)

Returns: a non-negative integer

The argument is a morphism $\alpha : A \rightarrow B$ between formal direct sums. The output is the number of summands of the the source.

2.3.5 NrCols (for IsAdditiveClosureMorphism)

▷ `NrCols(alpha)` (attribute)

Returns: a non-negative integer

The argument is a morphism $\alpha : A \rightarrow B$ between formal direct sums. The output is the number of summands of the the range.

2.4 Operators

2.4.1 $\backslash[\backslash]$ (for IsAdditiveClosureObject, IsInt)

▷ $\backslash[\backslash](A, i)$ (operation)

Returns: an object in C

The arguments are a formal direct sum A in some additive category C^\oplus and an integers i . The output is the i 'th entry in `ObjectList(A)`.

2.4.2 $[$ (for IsAdditiveClosureMorphism, IsInt, IsInt)

▷ $[(alpha, i, j)$ (operation)

Returns: a morphism C

The arguments are a morphism $\alpha : A \rightarrow B$ between formal direct sums in some additive category C^\oplus and two integers i, j . The output is the (i, j) 'th entry in `MorphismMatrix(alpha)`.

2.4.3 $\backslash/$ (for IsList, IsAdditiveClosureCategory)

▷ $\backslash/(arg1, arg2)$ (operation)

The input is either a list of objects or list of lists of morphisms. The method delegates to either `AdditiveClosureObject` or `AdditiveClosureMorphism`.

2.4.4 $\backslash/$ (for IsCapCategoryCell, IsAdditiveClosureCategory)

▷ $\backslash/(arg1, arg2)$ (operation)

This is a convenience method for `AsAdditiveClosureObject` and `AsAdditiveClosureMorphism`.

Chapter 3

Example on additive closure

3.1 Using matrix data structures

Example

```
gap> QQ := HomalgFieldOfRationalsInSingular();;
gap> R := QQ * "x,y,z";;
gap> matrix_element_as_morphism :=
>   r -> RingAsCategoryMorphism( RingAsCategory( R ), r );;
gap> list_list_as_matrix := {listlist, m, n} ->
>   HomalgMatrix(
>     List( listlist,
>       row -> List( row,
>         entry -> UnderlyingRingElement( entry )
>       )
>     ), m, n, R );;
gap> CR := RingAsCategory( R );;
gap> CRplus := AdditiveClosure( CR :
>   matrix_element_as_morphism := matrix_element_as_morphism,
>   list_list_as_matrix := list_list_as_matrix
> );;
gap> M := HomalgMatrix( "[x^2,4*y]", 1, 2, R );;
gap> N := HomalgMatrix( "[[1,3*x], [2*y^3,5*y]]", 2, 2, R );;
gap> P := M * N;;
gap> o := AsAdditiveClosureObject( RingAsCategoryUniqueObject( CR ) );;
gap> A := o;;
gap> B := DirectSum( o, o );;
gap> alpha := AdditiveClosureMorphism( A, M, B );;
gap> IsWellDefined( alpha );
true
gap> beta := AdditiveClosureMorphism( B, N, B );;
gap> IsWellDefined( beta );
true
gap> gamma := PreCompose( alpha, beta );;
gap> IsWellDefined( gamma );
true
gap> MorphismMatrix( gamma ) = P;
true
gap> # E and EE are both occupied by GAP
> EEE := KoszulDualRing( R );;
```

```

gap> matrix_element_as_morphism :=
>   r -> RingAsCategoryMorphism( RingAsCategory( EEE ), r );;
gap> list_list_as_matrix := {listlist, m, n} ->
>   HomalgMatrix(
>     List( listlist,
>       row -> List( row,
>         entry -> UnderlyingRingElement( entry )
>       )
>     ), m, n, EEE );;
gap> CEEE := RingAsCategory( EEE );;
gap> CEEEplus := AdditiveClosure( CEEE :
>   matrix_element_as_morphism := matrix_element_as_morphism,
>   list_list_as_matrix := list_list_as_matrix
> );;
gap> M := HomalgMatrix( "[[e0*e1,3*e0]]", 1, 2, EEE );;
gap> N := HomalgMatrix( "[[1,e0*e2], [2*e0*e1*e2,5*e2]]", 2, 2, EEE );;
gap> P := M * N;;
gap> o := AsAdditiveClosureObject( RingAsCategoryUniqueObject( CEEE ) );;
gap> A := o;;
gap> B := DirectSum( o, o );;
gap> alpha := AdditiveClosureMorphism( A, M, B );;
gap> IsWellDefined( alpha );
true
gap> beta := AdditiveClosureMorphism( B, N, B );;
gap> IsWellDefined( beta );
true
gap> gamma := PreCompose( alpha, beta );;
gap> IsWellDefined( gamma );
true
gap> MorphismMatrix( gamma ) = P;
true

```

Chapter 4

Adelman category

Let A be an additive category. The Adelman category of A is the free abelian category induced by A . An object x of the Adelman category of A consists of a composable pair $(\rho : a \rightarrow b, \gamma : b \rightarrow c)$ in A . We call ρ the *relation morphism*, and γ the *corelation morphism* of x .

Given two objects $x = (\rho : a \rightarrow b, \gamma : b \rightarrow c)$ and $y = (\rho' : a' \rightarrow b', \gamma' : b' \rightarrow c')$, a morphism α from x to y in the Adelman category of A consists of a morphism $\beta : b \rightarrow b'$, called the *morphism datum*, that has to fit into some commutative diagram of the form

$$\begin{array}{ccccc}
 a & \xrightarrow{\rho} & b & \xrightarrow{\gamma} & c \\
 \omega \downarrow & & \beta \downarrow & & \downarrow \psi \\
 a' & \xrightarrow{\rho'} & b' & \xrightarrow{\gamma'} & c'
 \end{array}$$

Any such morphism ω is called a *relation witness*, any such morphism ψ is called a *corelation witness*. Two morphisms between x and y with morphism data β and β' are congruent iff there exists $\sigma_1 : b \rightarrow a'$ and $\sigma_2 : c \rightarrow b'$ such that $\beta - \beta' = \sigma_1 \cdot \rho' + \gamma \cdot \sigma_2$. We call any such pair (σ_1, σ_2) a *witness pair* for β, β' being congruent.

4.1 GAP Categories

4.1.1 IsAdelmanCategoryObject (for IsCapCategoryObject)

▷ `IsAdelmanCategoryObject(a)` (filter)

Returns: true or false

The GAP category of objects of an Adelman category. Every object of an Adelman category lies in this GAP category.

4.1.2 IsAdelmanCategoryMorphism (for IsCapCategoryMorphism)

▷ `IsAdelmanCategoryMorphism(alpha)` (filter)

Returns: true or false

The GAP category of morphisms of an Adelman category. Every morphism of an Adelman category lies in this GAP category.

4.1.3 IsAdelmanCategory (for IsCapCategory)

▷ IsAdelmanCategory(*C*) (filter)

Returns: true or false

The GAP category of Adelman categories. Every CAP category which was created as an Adelman category lies in this GAP category.

4.2 Constructors

4.2.1 AdelmanCategory (for IsCapCategory)

▷ AdelmanCategory(*A*) (attribute)

Returns: a category

The argument is an additive CAP category *A*. The output is the Adelman category of *A*.

4.2.2 AdelmanCategoryObject (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ AdelmanCategoryObject(*alpha*, *beta*) (operation)

Returns: an object

The arguments are two morphisms $\alpha : a \rightarrow b$, $\beta : b \rightarrow c$ of the same additive category *A*. The output is an object in the Adelman category of *A* whose relation morphism is α and whose corelation morphism is β .

4.2.3 AdelmanCategoryMorphism (for IsAdelmanCategoryObject, IsCapCategoryMorphism, IsAdelmanCategoryObject)

▷ AdelmanCategoryMorphism(*x*, *alpha*, *y*) (operation)

Returns: a morphism in $\text{Hom}(x, y)$

Let *A* be an additive category. The arguments are an object *x* in the Adelman category of *A*, a morphism $\alpha : a \rightarrow b$ of *A*, and an object *y* in the Adelman category of *A*. The output is a morphism in the Adelman category of *A* whose morphism datum is given by α .

4.2.4 AsAdelmanCategoryObject (for IsCapCategoryObject)

▷ AsAdelmanCategoryObject(*a*) (attribute)

Returns: an object

The argument is an object *a* of an additive category *A*. The output is an object in the Adelman category of *A* whose relation morphism is $0 \rightarrow a$ and whose corelation morphism is $a \rightarrow 0$.

4.2.5 AsAdelmanCategoryMorphism (for IsCapCategoryMorphism)

▷ AsAdelmanCategoryMorphism(*alpha*) (attribute)

Returns: a morphism in $\text{Hom}(x, y)$

The argument is a morphism $\alpha : a \rightarrow b$ of an additive category *A*. The output is a morphism in the Adelman category of *A* whose source *x* is AsAdelmanCategoryObject(*a*), whose range *y* is AsAdelmanCategoryObject(*b*), and whose morphism datum is α .

4.2.6 $\backslash/$ (for IsCapCategoryObject, IsAdelmanCategory)

▷ $\backslash/(a, C)$ (operation)

Returns: an object

This is a convenience method. The first argument is an object a which either lies in an additive category A (which was not created as a Freyd category) or in a Freyd category F of an underlying additive category A . The second argument is an Adelman category C of A . If a lies in A this method returns `AsAdelmanCategoryObject(a)`. If a lies in F , this method return an object in C whose relation morphism is the same as the relation morphism of a , and whose corelation morphism is 0.

4.2.7 $\backslash/$ (for IsCapCategoryMorphism, IsAdelmanCategory)

▷ $\backslash/(\alpha, C)$ (operation)

Returns: a morphism in $\text{Hom}(x, y)$

This is a convenience method. The first argument is a morphism α which lies in an additive category A . The second argument is an Adelman category C of A . This method returns `AsAdelmanCategoryMorphism(alpha)`. We set $x = \text{AsAdelmanCategoryObject}(\text{Source}(\alpha))$ and $y = \text{AsAdelmanCategoryObject}(\text{Range}(\alpha))$.

4.3 Attributes and Properties

4.3.1 UnderlyingCategory (for IsAdelmanCategory)

▷ `UnderlyingCategory(C)` (attribute)

Returns: a category

The argument is an Adelman category C . The output is its underlying category A with which it was constructed.

4.3.2 RelationMorphism (for IsAdelmanCategoryObject)

▷ `RelationMorphism(x)` (attribute)

Returns: a morphism in $\text{Hom}(a, b)$

The argument is an object x in an Adelman category. The output is its relation morphism $\rho : a \rightarrow b$.

4.3.3 CorelationMorphism (for IsAdelmanCategoryObject)

▷ `CorelationMorphism(x)` (attribute)

Returns: a morphism in $\text{Hom}(b, c)$

The argument is an object x in an Adelman category. The output is its corelation morphism $\gamma : b \rightarrow c$.

4.3.4 MorphismDatum (for IsAdelmanCategoryMorphism)

▷ `MorphismDatum(alpha)` (attribute)

Returns: a morphism in $\text{Hom}(b, b')$

The argument is a morphism α in an Adelman category. The output is its morphism datum $\beta : b \rightarrow b'$.

4.3.5 RelationWitness (for IsAdelmanCategoryMorphism)

▷ `RelationWitness(alpha)` (attribute)

Returns: a morphism in $\text{Hom}(a, a')$

The argument is a morphism α in an Adelman category. The output is its relation witness $\omega : a \rightarrow a'$.

4.3.6 CorelationWitness (for IsAdelmanCategoryMorphism)

▷ `CorelationWitness(alpha)` (attribute)

Returns: a morphism in $\text{Hom}(c, c')$

The argument is a morphism α in an Adelman category. The output is its corelation witness $\psi : c \rightarrow c'$.

4.3.7 WitnessPairForBeingCongruentToZero (for IsAdelmanCategoryMorphism)

▷ `WitnessPairForBeingCongruentToZero(alpha)` (attribute)

Returns: a list of morphisms or `fail`

The argument is a morphism α in an Adelman category. If α is congruent to zero, the output is a witness pair. If α is not congruent to zero, the output is `fail`.

4.3.8 IsSequenceAsAdelmanCategoryObject (for IsAdelmanCategoryObject)

▷ `IsSequenceAsAdelmanCategoryObject(x)` (property)

Returns: a boolean

The argument is an object x in an Adelman category. The output is `true` if the composition of its relation morphism and its corelation morphism yields zero. Otherwise, the output is `false`.

Chapter 5

Category of rows

5.1 GAP Categories

5.1.1 IsCategoryOfRowsObject (for IsCapCategoryObject)

▷ `IsCategoryOfRowsObject(object)`

(filter)

Returns: true or false

The GAP category of objects in the category of rows over a ring R .

Chapter 6

Example on category of rows

6.1 Constructors of objects

Example

```
gap> S := HomalgRingOfIntegers();
Z
gap> rows := CategoryOfRows( S );
Rows( Z )
gap> obj1 := CategoryOfRowsObject( 2, rows );
<A row module over Z of rank 2>
gap> obj2 := CategoryOfRowsObject( 8, rows );
<A row module over Z of rank 8>
```

6.2 Constructors of morphisms

Example

```
gap> obj3 := CategoryOfRowsObject( 1, rows );
<A row module over Z of rank 1>
gap> IsWellDefined( obj1 );
true
gap> obj4 := CategoryOfRowsObject( 2, rows );
<A row module over Z of rank 2>
gap> mor := CategoryOfRowsMorphism( obj3, HomalgMatrix( [[1,2]], S ), obj4 );
<A morphism in Rows( Z )>
gap> IsWellDefined( mor );
true
```

Example

```
gap> Display( Source( mor ) );
A row module over Z of rank 1
gap> Display( Range( mor ) );
A row module over Z of rank 2
gap> Display( UnderlyingMatrix( mor ) );
[ [ 1, 2 ] ]
```

6.3 A few categorical constructions for category of rows

Example

```
gap> ZeroObject( rows );
<A row module over Z of rank 0>
gap> obj5 := CategoryOfRowsObject( 2, rows );
<A row module over Z of rank 2>
```

Example

```
gap> Display( ZeroMorphism( ZeroObject( rows ), obj5 ) );
A zero, split monomorphism in Rows( Z )

Source:
A row module over Z of rank 0

Matrix:
(an empty 0 x 2 matrix)

Range:
A row module over Z of rank 2
```

Example

```
gap> obj6 := CategoryOfRowsObject( 1, rows );
<A row module over Z of rank 1>
```

Example

```
gap> Display( IdentityMorphism( obj6 ) );
An identity morphism in Rows( Z )

Source:
A row module over Z of rank 1

Matrix:
[ [ 1 ] ]

Range:
A row module over Z of rank 1
```

Example

```
gap> directSum := DirectSum( [ obj5, obj6 ] );
<A row module over Z of rank 3>
```

Example

```
gap> Display( directSum );
A row module over Z of rank 3
```

Example

```
gap> i1 := InjectionOfCofactorOfDirectSum( [ obj5, obj6 ], 1 );
<A morphism in Rows( Z )>
```

Example

```
gap> Display( i1 );
A morphism in Rows( Z )

Source:
A row module over Z of rank 2
```

Matrix:
 $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$

Range:
 A row module over \mathbb{Z} of rank 3

Example

```
gap> i2 := InjectionOfCofactorOfDirectSum( [ obj5, obj6 ], 2 );
<A morphism in Rows( Z )>
```

Example

```
gap> Display( i2 );
A morphism in Rows( Z )
```

Source:
 A row module over \mathbb{Z} of rank 1

Matrix:
 $\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$

Range:
 A row module over \mathbb{Z} of rank 3

Example

```
gap> proj1 := ProjectionInFactorOfDirectSum( [ obj5, obj6 ], 1 );
<A morphism in Rows( Z )>
```

Example

```
gap> Display( proj1 );
A morphism in Rows( Z )
```

Source:
 A row module over \mathbb{Z} of rank 3

Matrix:
 $\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$

Range:
 A row module over \mathbb{Z} of rank 2

Example

```
gap> proj2 := ProjectionInFactorOfDirectSum( [ obj5, obj6 ], 2 );
<A morphism in Rows( Z )>
```

Example

```
gap> Display( proj2 );
A morphism in Rows( Z )
```

Source:
 A row module over \mathbb{Z} of rank 3

Matrix:
 $\begin{bmatrix} 0 \end{bmatrix}$,

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

Range:

A row module over Z of rank 1

Example

```
gap> k := WeakKernelEmbedding( proj1 );
<A morphism in Rows( Z )>
```

Example

```
gap> Display( k );
A morphism in Rows( Z )

Source:
A row module over Z of rank 1

Matrix:
[ [ 0, 0, 1 ] ]

Range:
A row module over Z of rank 3
```

Example

```
gap> ck := WeakCokernelProjection( k );
<A morphism in Rows( Z )>
```

Example

```
gap> Display( ck );
A morphism in Rows( Z )

Source:
A row module over Z of rank 3

Matrix:
[ [ 0, -1 ],
  [ 1, 0 ],
  [ 0, 0 ] ]

Range:
A row module over Z of rank 2
```

Example

```
gap> IsMonomorphism( k );
true
gap> IsEpimorphism( k );
false
gap> IsMonomorphism( ck );
false
gap> IsEpimorphism( ck );
true
gap> mor1 := CategoryOfRowsMorphism( obj5, HomalgMatrix( [[ 1 ], [ 2 ]], S ), obj6 );
<A morphism in Rows( Z )>
```

Example

```
gap> Display( mor1 );
A morphism in Rows( Z )
```


Source:
A row module over \mathbb{Z} of rank 2

Matrix:

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

Range:
A row module over \mathbb{Z} of rank 1

Example

```
gap> mor2 := IdentityMorphism( obj6 );
<An identity morphism in Rows( Z )>
```

Example

```
gap> Display( mor2 );
An identity morphism in Rows( Z )
```

Source:
A row module over \mathbb{Z} of rank 1

Matrix:

$$\begin{bmatrix} 1 \end{bmatrix}$$

Range:
A row module over \mathbb{Z} of rank 1

Example

```
gap> lift := Lift( mor1, mor2 );
<A morphism in Rows( Z )>
```

Example

```
gap> Display( lift );
A morphism in Rows( Z )
```

Source:
A row module over \mathbb{Z} of rank 2

Matrix:

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

Range:
A row module over \mathbb{Z} of rank 1

Example

```
gap> source := CategoryOfRowsObject( 1, rows );
<A row module over Z of rank 1>
gap> range := CategoryOfRowsObject( 2, rows );
<A row module over Z of rank 2>
gap> mor := CategoryOfRowsMorphism( source, HomalgMatrix( [[ 2, 3 ]], S ), range );
<A morphism in Rows( Z )>
gap> colift := Colift( mor2, mor );
<A morphism in Rows( Z )>
```

Example

```
gap> Display( colift );
A morphism in Rows( Z )

Source:
A row module over Z of rank 1

Matrix:
[ [ 2, 3 ] ]

Range:
A row module over Z of rank 2
```

Example

```
gap> fp := WeakBiFiberProduct( mor1, mor2 );
<A row module over Z of rank 2>
gap> fp_proj := ProjectionOfBiasedWeakFiberProduct( mor1, mor2 );
<A morphism in Rows( Z )>
```

Example

```
gap> Display( fp_proj );
A morphism in Rows( Z )

Source:
A row module over Z of rank 2

Matrix:
[ [ -2, 1 ],
  [ -1, 0 ] ]

Range:
A row module over Z of rank 2
```

Example

```
gap> po := WeakBiPushout( mor, mor2 );
<A row module over Z of rank 2>
gap> inj_push := InjectionOfBiasedWeakPushout( mor, mor2 );
<A morphism in Rows( Z )>
```

Example

```
gap> Display( inj_push );
A morphism in Rows( Z )

Source:
A row module over Z of rank 2

Matrix:
[ [ -3, 1 ],
  [ 2, -1 ] ]

Range:
A row module over Z of rank 2
```

6.4 Simplifications

Example

```
gap> R := HomalgRingOfIntegers();
gap> rows := CategoryOfRows( R );
gap> M := HomalgMatrix( [ [ 2, 2, 2 ], [ 3, 3, 3 ] ], 2, 3, R );
gap> alpha := AsCategoryOfRowsMorphism( M, rows );
gap> pi := PreCompose( [
>   SimplifySourceAndRange_IsoFromInputSource( alpha, infinity ),
>   SimplifySourceAndRange( alpha, infinity ),
>   SimplifySourceAndRange_IsoToInputRange( alpha, infinity ) ] );
gap> IsCongruentForMorphisms( pi, alpha );
true
gap> IsOne(
>   PreCompose( SimplifySourceAndRange_IsoFromInputSource( alpha, infinity ), SimplifySourceAndRange_IsoToInputRange( alpha, infinity ) );
> );
true
gap> IsOne(
>   PreCompose( SimplifySourceAndRange_IsoFromInputRange( alpha, infinity ), SimplifySourceAndRange_IsoToInputSource( alpha, infinity ) );
> );
true
gap> pi2 := PreCompose(
>   SimplifySource_IsoFromInputObject( alpha, infinity ),
>   SimplifySource( alpha, infinity )
> );
gap> IsCongruentForMorphisms( pi2, alpha );
true
gap> IsOne( PreCompose( SimplifySource_IsoFromInputObject( alpha, infinity ), SimplifySource_IsoToInputObject( alpha, infinity ) );
true
gap> pi3 := PreCompose(
>   SimplifyRange( alpha, infinity ),
>   SimplifyRange_IsoToInputObject( alpha, infinity )
> );
gap> IsCongruentForMorphisms( pi3, alpha );
true
gap> IsOne( PreCompose( SimplifyRange_IsoFromInputObject( alpha, infinity ), SimplifyRange_IsoToInputObject( alpha, infinity ) );
true
```

Example

```
gap> R := HomalgRingOfIntegers();
gap> cols := CategoryOfColumns( R );
gap> M := HomalgMatrix( [ [ 2, 2, 2 ], [ 3, 3, 3 ] ], 2, 3, R );
gap> alpha := AsCategoryOfColumnsMorphism( M, cols );
gap> pi := PreCompose( [
>   SimplifySourceAndRange_IsoFromInputSource( alpha, infinity ),
>   SimplifySourceAndRange( alpha, infinity ),
>   SimplifySourceAndRange_IsoToInputRange( alpha, infinity ) ] );
gap> IsCongruentForMorphisms( pi, alpha );
true
gap> IsOne(
>   PreCompose( SimplifySourceAndRange_IsoFromInputSource( alpha, infinity ), SimplifySourceAndRange_IsoToInputRange( alpha, infinity ) );
> );
true
gap> IsOne(
```

```

>   PreCompose( SimplifySourceAndRange_IsoFromInputRange( alpha, infinity ), SimplifySourceAndRange_IsoFromInputRange_IsoToInputObject( alpha, infinity ) );
> );
true
gap> pi2 := PreCompose(
>   SimplifySource_IsoFromInputObject( alpha, infinity ),
>   SimplifySource( alpha, infinity )
> );;;
gap> IsCongruentForMorphisms( pi2, alpha );
true
gap> IsOne( PreCompose( SimplifySource_IsoFromInputObject( alpha, infinity ), SimplifySource_IsoFromInputObject_IsoToInputObject( alpha, infinity ) );
true
gap> pi3 := PreCompose(
>   SimplifyRange( alpha, infinity ),
>   SimplifyRange_IsoToInputObject( alpha, infinity )
> );;;
gap> IsCongruentForMorphisms( pi3, alpha );
true
gap> IsOne( PreCompose( SimplifyRange_IsoFromInputObject( alpha, infinity ), SimplifyRange_IsoToInputObject_IsoToInputObject( alpha, infinity ) );
true

```

Example

```

gap> Qxyz := HomalgFieldOfRationalsInDefaultCAS( ) * "x,y,z";;
gap> A3 := RingOfDerivations( Qxyz, "Dx,Dy,Dz" );;
gap> M1 := HomalgMatrix( "[ \
> Dx \
> ]", 1, 1, A3 );;
gap> M2 := HomalgMatrix( "[ \
> Dx, \
> Dy \
> ]", 2, 1, A3 );;
gap> M3 := HomalgMatrix( "[ \
> Dx, \
> Dy, \
> Dz \
> ]", 3, 1, A3 );;
gap> M := DiagMat( [ M1, M2, M3 ] );;
gap> M := ShallowCopy( M );;
gap> SetIsMutableMatrix( M, true );;
gap> M[ 1, 2 ] := "1";;
gap> M[ 2, 3 ] := "1";;
gap> M[ 3, 3 ] := "1";;
gap> MakeImmutable( M );;
gap> tau1 := HomalgMatrix( "[ \
> 1, Dx, Dz, \
> 0, 0, 1, \
> 0, 1, Dy \
> ]", 3, 3, A3 );;
gap> tau2 := HomalgMatrix( "[ \
> 0, 1, Dz+x*y, \
> 0, 0, 1, \
> 1, Dz, x-y \
> ]", 3, 3, A3 );;
gap> tau3 := HomalgMatrix( "[ \

```

```

> 1, 0, 0, \
> 1, 1, 0, \
> 0, -1, 1 \
> ]", 3, 3, A3 );;
gap> tau := tau1 * tau2 * tau3;;
gap> M := M * tau;;
gap> rows := CategoryOfRows( A3 );;
gap> alpha := AsCategoryOfRowsMorphism( M, rows );;
gap> Mrows := FreydCategoryObject( alpha );;
gap> Srows := SimplifyObject( Mrows, infinity );;
gap> RankOfObject( Source( RelationMorphism( Srows ) ) );
4
gap> RankOfObject( Range( RelationMorphism( Srows ) ) );
2
gap> IsIsomorphism( SimplifyObject_IsoFromInputObject( Mrows, infinity ) );
true
gap> IsIsomorphism( SimplifyObject_IsoToInputObject( Mrows, infinity ) );
true

```

Computing the grade filtration:

Example

```

gap> mu1 := GradeFiltrationNthMonomorphism( Mrows, 1 );;
gap> IsZero( mu1 );
false
gap> IsMonomorphism( mu1 );
true
gap> mu2 := GradeFiltrationNthMonomorphism( Mrows, 2 );;
gap> IsZero( mu2 );
false
gap> IsMonomorphism( mu2 );
true
gap> mu3 := GradeFiltrationNthMonomorphism( Mrows, 3 );;
gap> IsZero( mu3 );
false
gap> IsMonomorphism( mu3 );
true
gap> mu4 := GradeFiltrationNthMonomorphism( Mrows, 4 );;
gap> IsZero( mu4 );
true

```

Example

```

gap> cols := CategoryOfColumns( A3 );;
gap> alpha := AsCategoryOfColumnsMorphism( M, cols );;
gap> Mcols := FreydCategoryObject( alpha );;
gap> Scols := SimplifyObject( Mcols, infinity );;
gap> RankOfObject( Source( RelationMorphism( Scols ) ) );
1
gap> RankOfObject( Range( RelationMorphism( Scols ) ) );
4
gap> IsIsomorphism( SimplifyObject_IsoFromInputObject( Mcols, infinity ) );
true
gap> IsIsomorphism( SimplifyObject_IsoToInputObject( Mcols, infinity ) );
true

```

Chapter 7

Category of columns

7.1 GAP Categories

7.1.1 IsCategoryOfColumnsObject (for IsCapCategoryObject)

▷ `IsCategoryOfColumnsObject(object)`

(filter)

Returns: true or false

The GAP category of objects in the category of columns over a ring R .

Chapter 8

Example on category of columns

8.1 Constructors of objects

Example

```
gap> S := HomalgRingOfIntegers();
Z
gap> cols := CategoryOfColumns( S );
Columns( Z )
gap> obj1 := CategoryOfColumnsObject( 2, cols );
<A column module over Z of rank 2>
gap> obj2 := CategoryOfColumnsObject( 8, cols );
<A column module over Z of rank 8>
```

8.2 Constructors of morphisms

Example

```
gap> obj3 := CategoryOfColumnsObject( 1, cols );
<A column module over Z of rank 1>
gap> IsWellDefined( obj1 );
true
gap> obj4 := CategoryOfColumnsObject( 2, cols );
<A column module over Z of rank 2>
gap> mor := CategoryOfColumnsMorphism( obj3, HomalgMatrix( [[1],[2]], S ), obj4 );
<A morphism in Columns( Z )>
gap> IsWellDefined( mor );
true
```

Example

```
gap> Display( Source( mor ) );
A column module over Z of rank 1
gap> Display( Range( mor ) );
A column module over Z of rank 2
gap> Display( UnderlyingMatrix( mor ) );
[ [ 1 ],
  [ 2 ] ]
```

8.3 A few categorical constructions for category of columns

Example

```
gap> ZeroObject( cols );
<A column module over Z of rank 0>
gap> obj5 := CategoryOfColumnsObject( 2, cols );
<A column module over Z of rank 2>
```

Example

```
gap> Display( ZeroMorphism( ZeroObject( cols ), obj5 ) );
A zero, split monomorphism in Columns( Z )

Source:
A column module over Z of rank 0

Matrix:
(an empty 2 x 0 matrix)

Range:
A column module over Z of rank 2
```

Example

```
gap> obj6 := CategoryOfColumnsObject( 1, cols );
<A column module over Z of rank 1>
```

Example

```
gap> Display( IdentityMorphism( obj6 ) );
An identity morphism in Columns( Z )

Source:
A column module over Z of rank 1

Matrix:
[ [ 1 ] ]

Range:
A column module over Z of rank 1
```

Example

```
gap> directSum := DirectSum( [ obj5, obj6 ] );
<A column module over Z of rank 3>
```

Example

```
gap> Display( directSum );
A column module over Z of rank 3
```

Example

```
gap> i1 := InjectionOfCofactorOfDirectSum( [ obj5, obj6 ], 1 );
<A morphism in Columns( Z )>
```

Example

```
gap> Display( i1 );
A morphism in Columns( Z )

Source:
A column module over Z of rank 2
```


Matrix:
 $\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$

Range:
 A column module over \mathbb{Z} of rank 3

Example

```
gap> i2 := InjectionOfCofactorOfDirectSum( [ obj5, obj6 ], 2 );
<A morphism in Columns( Z )>
```

Example

```
gap> Display( i2 );
A morphism in Columns( Z )
```

Source:
 A column module over \mathbb{Z} of rank 1

Matrix:
 $\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$

Range:
 A column module over \mathbb{Z} of rank 3

Example

```
gap> proj1 := ProjectionInFactorOfDirectSum( [ obj5, obj6 ], 1 );
<A morphism in Columns( Z )>
```

Example

```
gap> Display( proj1 );
A morphism in Columns( Z )
```

Source:
 A column module over \mathbb{Z} of rank 3

Matrix:
 $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$

Range:
 A column module over \mathbb{Z} of rank 2

Example

```
gap> proj2 := ProjectionInFactorOfDirectSum( [ obj5, obj6 ], 2 );
<A morphism in Columns( Z )>
```

Example

```
gap> Display( proj2 );
A morphism in Columns( Z )
```

Source:
 A column module over \mathbb{Z} of rank 3

Matrix:
 $\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$

Range:
 A column module over \mathbb{Z} of rank 1

Example

```
gap> k := WeakKernelEmbedding( proj1 );
<A morphism in Columns( Z )>
```

Example

```
gap> Display( k );
A morphism in Columns( Z )

Source:
A column module over Z of rank 1

Matrix:
 $\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$ 

Range:
A column module over Z of rank 3
```

Example

```
gap> ck := WeakCokernelProjection( k );
<A morphism in Columns( Z )>
```

Example

```
gap> Display( ck );
A morphism in Columns( Z )

Source:
A column module over Z of rank 3

Matrix:
 $\begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix}$ 

Range:
A column module over Z of rank 2
```

Example

```
gap> IsMonomorphism( k );
true
gap> IsEpimorphism( k );
false
gap> IsMonomorphism( ck );
false
gap> IsEpimorphism( ck );
true
gap> mor1 := CategoryOfColumnsMorphism( obj5, HomalgMatrix( [[ 1, 2 ]], S ), obj6 );
<A morphism in Columns( Z )>
```

Example

```
gap> Display( mor1 );
A morphism in Columns( Z )

Source:
A column module over Z of rank 2

Matrix:
[ [ 1, 2 ] ]

Range:
A column module over Z of rank 1
```

Example

```
gap> mor2 := IdentityMorphism( obj6 );
<An identity morphism in Columns( Z )>
```

Example

```
gap> Display( mor2 );
An identity morphism in Columns( Z )

Source:
A column module over Z of rank 1

Matrix:
[ [ 1 ] ]

Range:
A column module over Z of rank 1
```

Example

```
gap> lift := Lift( mor1, mor2 );
<A morphism in Columns( Z )>
```

Example

```
gap> Display( lift );
A morphism in Columns( Z )

Source:
A column module over Z of rank 2

Matrix:
[ [ 1, 2 ] ]

Range:
A column module over Z of rank 1
```

Example

```
gap> source := CategoryOfColumnsObject( 1, cols );
<A column module over Z of rank 1>
gap> range := CategoryOfColumnsObject( 2, cols );
<A column module over Z of rank 2>
gap> mor := CategoryOfColumnsMorphism( source, HomalgMatrix( [[ 2 ], [ 3 ]], S ), range );
<A morphism in Columns( Z )>
gap> colift := Colift( mor2, mor );
<A morphism in Columns( Z )>
```

Example

```
gap> Display( colift );
A morphism in Columns( Z )

Source:
A column module over Z of rank 1

Matrix:
[ [ 2 ],
  [ 3 ] ]

Range:
A column module over Z of rank 2
```

Example

```
gap> fp := WeakBiFiberProduct( mor1, mor2 );
<A column module over Z of rank 2>
gap> fp_proj := ProjectionOfBiasedWeakFiberProduct( mor1, mor2 );
<A morphism in Columns( Z )>
```

Example

```
gap> Display( fp_proj );
A morphism in Columns( Z )

Source:
A column module over Z of rank 2

Matrix:
[ [ -2, -1 ],
  [ 1, 0 ] ]

Range:
A column module over Z of rank 2
```

Example

```
gap> po := WeakBiPushout( mor, mor2 );
<A column module over Z of rank 2>
gap> inj_push := InjectionOfBiasedWeakPushout( mor, mor2 );
<A morphism in Columns( Z )>
```

Example

```
gap> Display( inj_push );
A morphism in Columns( Z )

Source:
A column module over Z of rank 2

Matrix:
[ [ -3, 2 ],
  [ 1, -1 ] ]

Range:
A column module over Z of rank 2
```

Chapter 9

Category of graded rows and category of graded columns

9.1 Constructors

9.1.1 CategoryOfGradedColumns (for IsHomalgGradedRing)

▷ `CategoryOfGradedColumns(R)` (attribute)

Returns: a category

The argument is a homalg graded ring R . The output is the category of graded columns over R .

9.1.2 CategoryOfGradedRows (for IsHomalgGradedRing)

▷ `CategoryOfGradedRows(R)` (attribute)

Returns: a category

The argument is a homalg graded ring R . The output is the category of graded rows over R .

9.1.3 GradedRow (for IsList, IsHomalgGradedRing)

▷ `GradedRow($degree_list$, R)` (operation)

Returns: an object

The arguments are a list of degrees and a homalg graded ring R . The list of degrees must be of the form $[[d_1, n_1], [d_2, n_2], \dots]$ where d_i are degrees, i.e. elements in the degree group of R and the n_i are non-negative integers. Currently there are two formats that are supported to enter the degrees. Either one can enter them as lists of integers, say $d_1 = [1, 1, 0, 2]$, or they can be entered as `Homalg_Module_Elements` of the degree group of R . In either case, the result is the graded row associated to the degrees d_i and their multiplicities n_i .

9.1.4 GradedRow (for IsList, IsHomalgGradedRing, IsBool)

▷ `GradedRow($degree_list$, R)` (operation)

Returns: an object

As 'GradedRow', but the boolean (= third argument) allows to switch off checks on the input data. If this boolean is set to true, then the input checks are performed and otherwise they are not. Calling this constructor with 'false' is therefore suited for high performance applications.

9.1.5 GradedColumn (for IsList, IsHomalgGradedRing)

▷ `GradedColumn(degree_list, R)` (operation)

Returns: an object

The arguments are a list of degrees and a homalg graded ring R . The list of degrees must be of the form $[[d_1, n_1], [d_2, n_2], \dots]$ where d_i are degrees, i.e. elements in the degree group of R and the n_i are non-negative integers. Currently there are two formats that are supported to enter the degrees. Either one can enter them as lists of integers, say $d_1 = [1, 1, 0, 2]$, or they can be entered as `Homalg_Module_Elements` of the degree group of R . In either case, the result is the graded column associated to the degrees d_i and their multiplicities n_i .

9.1.6 GradedColumn (for IsList, IsHomalgGradedRing, IsBool)

▷ `GradedColumn(degree_list, R)` (operation)

Returns: an object

As 'GradedColumn', but the boolean (= third argument) allows to switch off checks on the input data. If this boolean is set to true, then the input checks are performed and otherwise they are not. Calling this constructor with 'false' is therefore suited for high performance applications.

9.1.7 GradedRowOrColumnMorphism (for IsGradedRowOrColumn, IsHomalgMatrix, IsGradedRowOrColumn)

▷ `GradedRowOrColumnMorphism(S, M, T)` (operation)

Returns: a morphism in $\text{Hom}(S, T)$

The arguments are an object S in the category of graded rows or columns over a homalg graded ring R , a homalg matrix M over R and another graded row or column T over R . The output is the morphism $S \rightarrow T$ in the category of graded rows and columns over R , whose underlying matrix is given by M .

9.1.8 GradedRowOrColumnMorphism (for IsGradedRowOrColumn, IsHomalgMatrix, IsGradedRowOrColumn, IsBool)

▷ `GradedRowOrColumnMorphism(S, M, T)` (operation)

Returns: a morphism in $\text{Hom}(S, T)$

As 'GradedRowOrColumnMorphism', but carries a fourth input parameter. If this boolean is set to false, then no checks on the input are performed. That option is therefore better suited for high performance applications.

9.2 Attributes

9.2.1 UnderlyingHomalgGradedRing (for IsGradedRowOrColumn)

▷ `UnderlyingHomalgGradedRing(A)` (attribute)

Returns: a homalg graded ring

The argument is a graded row or column A over a homalg graded ring R . The output is then the graded ring R .

9.2.2 DegreeList (for IsGradedRowOrColumn)

▷ `DegreeList(A)` (attribute)

Returns: a list

The argument is a graded row or column A over a homalg graded ring R . The output is the `degree_list` of this object. To handle `degree_lists` most easily, `degree_lists` are reduced whenever an object is added to the category. E.g. the input `degree_list [[d_1 , 1], [d_1 , 1]]` will be turned into `[[d_1 , 2]]`.

9.2.3 RankOfObject (for IsGradedRowOrColumn)

▷ `RankOfObject(A)` (attribute)

Returns: an integer

The argument is a graded row or column over a homalg graded ring R . The output is the rank of this module.

9.2.4 UnderlyingHomalgGradedRing (for IsGradedRowOrColumnMorphism)

▷ `UnderlyingHomalgGradedRing(alpha)` (attribute)

Returns: a homalg graded ring

The argument is a morphism α in the category of graded rows or columns over a homalg graded ring R . The output is the homalg graded ring R .

9.2.5 UnderlyingHomalgMatrix (for IsGradedRowOrColumnMorphism)

▷ `UnderlyingHomalgMatrix(alpha)` (attribute)

Returns: a matrix over a homalg graded ring

The argument is a morphism α in the category of graded rows or columns over a homalg graded ring R . The output is the underlying homalg matrix over R .

9.3 GAP Categories

9.3.1 IsGradedRowOrColumn (for IsCapCategoryObject)

▷ `IsGradedRowOrColumn(object)` (filter)

Returns: true or false

The GAP category of graded rows and columns over a graded ring R .

9.3.2 IsGradedRow (for IsGradedRowOrColumn)

▷ `IsGradedRow(object)` (filter)

Returns: true or false

The GAP category of graded rows over a graded ring R .

9.3.3 IsGradedColumn (for IsGradedRowOrColumn)

▷ `IsGradedColumn(object)` (filter)

Returns: true or false

The GAP category of graded columns over a graded ring R .

9.3.4 IsGradedRowOrColumnMorphism (for IsCapCategoryMorphism)

▷ IsGradedRowOrColumnMorphism(*object*) (filter)

Returns: true or false

The GAP category of morphisms of graded rows and columns over a graded ring R .

9.3.5 IsGradedRowMorphism (for IsGradedRowOrColumnMorphism)

▷ IsGradedRowMorphism(*object*) (filter)

Returns: true or false

The GAP category of morphisms of graded rows over a graded ring R .

9.3.6 IsGradedColumnMorphism (for IsGradedRowOrColumnMorphism)

▷ IsGradedColumnMorphism(*object*) (filter)

Returns: true or false

The GAP category of morphisms of graded columns over a graded ring R .

9.4 Tools to simplify code

9.4.1 DeduceMapFromMatrixAndRangeForGradedRows (for IsHomalgMatrix, IsGradedRow)

▷ DeduceMapFromMatrixAndRangeForGradedRows(m , R) (operation)

Returns: a morphism

The argument is a homalg_matrix m and a graded row R . We then consider the module map induced from m with range R . This operation then deduces the source of this map and returns the map in the category of graded rows if the degrees of the source are uniquely determined.

9.4.2 DeduceSomeMapFromMatrixAndRangeForGradedRows (for IsHomalgMatrix, IsGradedRow)

▷ DeduceSomeMapFromMatrixAndRangeForGradedRows(m , R) (operation)

Returns: a morphism

The argument is a homalg_matrix m and a graded row R . This operation deduces the source of some map with matrix m and range R and returns the map in the category of graded rows.

9.4.3 DeduceMapFromMatrixAndSourceForGradedRows (for IsHomalgMatrix, IsGradedRow)

▷ DeduceMapFromMatrixAndSourceForGradedRows(m , S) (operation)

Returns: a morphism

The argument is a homalg_matrix m and a graded row S . We then consider the module map induced from m with source S . This operation then deduces the range of this map and returns the map in the category of graded rows if the degrees of the range are uniquely determined.

9.4.4 DeduceSomeMapFromMatrixAndSourceForGradedRows (for IsHomalgMatrix, IsGradedRow)

▷ `DeduceSomeMapFromMatrixAndSourceForGradedRows(m , S)` (operation)

Returns: a morphism

The argument is a `homalg_matrix` m and a graded row S . This operation deduces the range of some map with matrix m and source S and returns the map in the category of graded rows.

9.4.5 DeduceMapFromMatrixAndRangeForGradedCols (for IsHomalgMatrix, IsGradedColumn)

▷ `DeduceMapFromMatrixAndRangeForGradedCols(m , R)` (operation)

Returns: a morphism

The argument is a `homalg_matrix` m and a graded column R . We then consider the module map induced from m with range R . This operation then deduces the source of this map and returns the map in the category of graded columns if the degrees of the source are uniquely determined.

9.4.6 DeduceSomeMapFromMatrixAndRangeForGradedCols (for IsHomalgMatrix, IsGradedColumn)

▷ `DeduceSomeMapFromMatrixAndRangeForGradedCols(m , R)` (operation)

Returns: a morphism

The argument is a `homalg_matrix` m and a graded column R . This operation deduces the source of some map with matrix m and range R and returns the map in the category of graded columns.

9.4.7 DeduceMapFromMatrixAndSourceForGradedCols (for IsHomalgMatrix, IsGradedColumn)

▷ `DeduceMapFromMatrixAndSourceForGradedCols(m , S)` (operation)

Returns: a morphism

The argument is a `homalg_matrix` m and a graded column S . We then consider the module map induced from m with source S . This operation then deduces the range of this map and returns the map in the category of graded columns if the degrees of the range are uniquely determined.

9.4.8 DeduceSomeMapFromMatrixAndSourceForGradedCols (for IsHomalgMatrix, IsGradedColumn)

▷ `DeduceSomeMapFromMatrixAndSourceForGradedCols(m , S)` (operation)

Returns: a morphism

The argument is a `homalg_matrix` m and a graded column S . This operation deduces the range of some map with matrix m and source S and returns the map in the category of graded columns.

9.4.9 UnzipDegreeList (for IsGradedRowOrColumn)

▷ `UnzipDegreeList(S)` (operation)

Returns: a list

Given a graded row or column S , the degrees are stored in compact form. For example, the degrees `[1, 1, 1, 1]` is stored internally as `[1, 4]`. The second argument is thus the multiplicity with which

three degree 1 appears. Still, it can be useful at times to also go in the opposite direction, i.e. to take the compact form [#! 1, 4] and turn it into [1, 1, 1, 1]. This is performed by this operation and the obtained extended degree #! list is returned.

Chapter 10

Cokernel image closure

Chapter 11

Freyd category

11.1 Internal Hom-Embedding

11.1.1 INTERNAL_HOM_EMBEDDING (for IsFreydCategoryObject, IsFreydCategoryObject)

▷ INTERNAL_HOM_EMBEDDING(*objects*, *a*, *b*) (operation)

Returns: a (mono)morphism

The arguments are two objects *a* and *b* of a Freyd category. Assume that the relation morphism for *a* is $\alpha: R_A \rightarrow A$, then we have the exact sequence $0 \rightarrow \underline{\text{Hom}}(a, b) \rightarrow \underline{\text{Hom}}(A, b) \rightarrow \underline{\text{Hom}}(R_A, b)$. The embedding of $\underline{\text{Hom}}(a, b)$ into $\underline{\text{Hom}}(A, b)$ is the internal Hom-embedding. This method returns this very map.

11.2 Convenient methods for tensor products of freyd objects and morphisms

11.2.1 * (for IsFreydCategoryObject, IsFreydCategoryObject)

▷ *(*arg1*, *arg2*) (operation)

11.2.2 \^ (for IsFreydCategoryObject, IsInt)

▷ \^(*arg1*, *arg2*) (operation)

11.2.3 * (for IsFreydCategoryMorphism, IsFreydCategoryMorphism)

▷ *(*arg1*, *arg2*) (operation)

11.2.4 \^ (for IsFreydCategoryMorphism, IsInt)

▷ \^(*arg1*, *arg2*) (operation)

Chapter 12

Examples and Tests

12.1 Adelman 5 lemma

Example

```
gap> quiver := RightQuiver( "Q(8)[a:1->2,b:2->3,c:3->4,d:3->5,e:4->6,f:5->6,g:6->7,h:7->8]" );;
gap> QQ := HomalgFieldOfRationals();;
gap> A := PathAlgebra( QQ, quiver );;
gap> B := QuotientOfPathAlgebra( A,
> [
>   A.ce - A.df,
>   A.abd,
>   A.egh,
>   A.bc,
>   A.fg,
>   A.ab #since d is supposed to be a mono
> ] );;
gap> QRowsB := QuiverRowsDescentToZDefinedByBasisPaths( B );;
gap> Adel := AdelmanCategory( QRowsB );;
gap> a := B.a/QRowsB/Adel;;
gap> b := B.b/QRowsB/Adel;;
gap> c := B.c/QRowsB/Adel;;
gap> d := B.d/QRowsB/Adel;;
gap> e := B.e/QRowsB/Adel;;
gap> f := B.f/QRowsB/Adel;;
gap> g := B.g/QRowsB/Adel;;
gap> h := B.h/QRowsB/Adel;;
gap> l := CokernelProjection( a );;
gap> k := CokernelColift( a, PreCompose( b, d ) );;
gap> i := KernelEmbedding( h );;
gap> j := KernelLift( h, PreCompose( e, g ) );;
gap> Kd := KernelObject( d );;
gap> Hbc := HomologyObject( b, c );;
gap> Hcj := HomologyObject( c, j );;
gap> Hkf := HomologyObject( k, f );;
gap> Hfg := HomologyObject( f, g );;
gap> L := [ Kd, Hbc, Hcj, Hkf, Hfg ];;
gap> K := KernelObject( e );;
gap> test_func := MembershipFunctionSerreSubcategoryGeneratedByObjects( L, Adel );;
Warning: the provided function returns either true or fail!
```

```

gap> C := FullSubcategoryByMembershipFunction( Adel, test_func );
gap> Serre := Adel/C;;
gap> K := K/Serre;;
gap> IsZero( K );
true

```

12.2 Adelman category basics for category of rows

Example

```

gap> R := HomalgRingOfIntegers();
gap> rows := CategoryOfRows( R );
gap> adelman := AdelmanCategory( rows );
gap> obj1 := CategoryOfRowsObject( 1, rows );
gap> obj2 := CategoryOfRowsObject( 2, rows );
gap> id := IdentityMorphism( obj2 );
gap> alpha := CategoryOfRowsMorphism( obj1, HomalgMatrix( [ [ 1, 2 ] ], 1, 2, R ), obj2 );
gap> beta := CategoryOfRowsMorphism( obj2, HomalgMatrix( [ [ 1, 2 ], [ 3, 4 ] ], 2, 2, R ), obj2 );
gap> gamma := CategoryOfRowsMorphism( obj2, HomalgMatrix( [ [ 1, 3 ], [ 3, 4 ] ], 2, 2, R ), obj2 );
gap> obj1_a := AsAdelmanCategoryObject( obj1 );
gap> obj2_a := AsAdelmanCategoryObject( obj2 );
gap> m := AsAdelmanCategoryMorphism( beta );
gap> n := AsAdelmanCategoryMorphism( gamma );
gap> IsWellDefined( m );
true
gap> IsCongruentForMorphisms( PreCompose( m, n ), PreCompose( n, m ) );
false
gap> IsCongruentForMorphisms( SubtractionForMorphisms( m, m ), ZeroMorphism( obj2_a, obj2_a ) );
true
gap> IsCongruentForMorphisms( ZeroObjectFunctorial( adelman ),
>                               PreCompose( UniversalMorphismFromZeroObject( obj1_a ), UniversalMorphismFromZeroObject( obj2_a ) );
>                               );
true
gap> d := [ obj1_a, obj2_a ];
gap> pi1 := ProjectionInFactorOfDirectSum( d, 1 );
gap> pi2 := ProjectionInFactorOfDirectSum( d, 2 );
gap> id := IdentityMorphism( DirectSum( d ) );
gap> iota1 := InjectionOfCofactorOfDirectSum( d, 1 );
gap> iota2 := InjectionOfCofactorOfDirectSum( d, 2 );
gap> IsCongruentForMorphisms( PreCompose( pi1, iota1 ) + PreCompose( pi2, iota2 ), id );
true
gap> IsCongruentForMorphisms( UniversalMorphismIntoDirectSum( d, [ pi1, pi2 ] ), id );
true
gap> IsCongruentForMorphisms( UniversalMorphismFromDirectSum( d, [ iota1, iota2 ] ), id );
true
gap> c := CokernelProjection( m );
gap> c2 := CokernelProjection( c );
gap> IsCongruentForMorphisms( c2, ZeroMorphism( Source( c2 ), Range( c2 ) ) );
true
gap> IsWellDefined( CokernelProjection( m ) );
true
gap> IsCongruentForMorphisms( CokernelColift( m, CokernelProjection( m ) ), IdentityMorphism( CokernelColift( m, CokernelProjection( m ) ) );
true

```

```

gap> k := KernelEmbedding( c );;
gap> IsZeroForMorphisms( PreCompose( k, c ) );
true
gap> IsCongruentForMorphisms( Kernellift( m, KernelEmbedding( m ) ), IdentityMorphism( KernelObj
true

```

12.3 Adelman category basics for additive closure of algebroids

Example

```

gap> quiver := RightQuiver( "Q(9)[a:1->2,b:2->3,c:1->4,d:2->5,e:3->6,f:4->5,g:5->6,h:4->7,i:5->8,
gap> kQ := PathAlgebra( HomalgFieldOfRationals(), quiver );;
gap> Aoid := Algebroid( kQ, [ kQ.ad - kQ.cf,
>                               kQ.dg - kQ.be,
>                               kQ.( "fi" ) - kQ.hk,
>                               kQ.gj - kQ.il,
>                               kQ.mk + kQ.bn - kQ.di ] );;
gap> mm := SetOfGeneratingMorphisms( Aoid );;
gap> CapCategorySwitchLogicOff( Aoid );;
gap> Acat := AdditiveClosure( Aoid );;
gap> a := AsAdditiveClosureMorphism( mm[1] );;
gap> b := AsAdditiveClosureMorphism( mm[2] );;
gap> c := AsAdditiveClosureMorphism( mm[3] );;
gap> d := AsAdditiveClosureMorphism( mm[4] );;
gap> e := AsAdditiveClosureMorphism( mm[5] );;
gap> f := AsAdditiveClosureMorphism( mm[6] );;
gap> g := AsAdditiveClosureMorphism( mm[7] );;
gap> h := AsAdditiveClosureMorphism( mm[8] );;
gap> i := AsAdditiveClosureMorphism( mm[9] );;
gap> j := AsAdditiveClosureMorphism( mm[10] );;
gap> k := AsAdditiveClosureMorphism( mm[11] );;
gap> l := AsAdditiveClosureMorphism( mm[12] );;
gap> m := AsAdditiveClosureMorphism( mm[13] );;
gap> n := AsAdditiveClosureMorphism( mm[14] );;
gap> Adel := AdelmanCategory( Acat );;
gap> A := AdelmanCategoryObject( a, b );;
gap> B := AdelmanCategoryObject( f, g );;
gap> alpha := AdelmanCategoryMorphism( A, d, B );;
gap> IsWellDefined( alpha );
true
gap> IsWellDefined( KernelEmbedding( alpha ) );
true
gap> IsWellDefined( CokernelProjection( alpha ) );
true
gap> T := AdelmanCategoryObject( k, l );;
gap> tau := AdelmanCategoryMorphism( B, i, T );;
gap> IsZeroForMorphisms( PreCompose( alpha, tau ) );
true
gap> colift := CokernelColift( alpha, tau );;
gap> IsWellDefined( colift );
true
gap> IsCongruentForMorphisms( PreCompose( CokernelProjection( alpha ), colift ), tau );
true

```

```

gap> lift := KernelLift( tau, alpha );;
gap> IsWellDefined( lift );
true
gap> IsCongruentForMorphisms( PreCompose( lift, KernelEmbedding( tau ) ), alpha );
true
gap> IsCongruentForMorphisms( ColiftAlongEpimorphism( CokernelProjection( alpha ), tau ), colift );
true
gap> IsCongruentForMorphisms( LiftAlongMonomorphism( KernelEmbedding( tau ), alpha ), lift );
true

```

12.4 Adelman category basics for category of columns

Example

```

gap> R := HomalgRingOfIntegers();;
gap> cols := CategoryOfColumns( R );;
gap> adelman := AdelmanCategory( cols );;
gap> obj1 := CategoryOfColumnsObject( 1, cols );;
gap> obj2 := CategoryOfColumnsObject( 2, cols );;
gap> id := IdentityMorphism( obj2 );;
gap> alpha := CategoryOfColumnsMorphism( obj1, HomalgMatrix( [ [ 1 ], [ 2 ] ], 1, 2, R ), obj2 );;
gap> beta := CategoryOfColumnsMorphism( obj2, HomalgMatrix( [ [ 1, 3 ], [ 2, 4 ] ], 2, 2, R ), obj2 );;
gap> gamma := CategoryOfColumnsMorphism( obj2, HomalgMatrix( [ [ 1, 3 ], [ 3, 4 ] ], 2, 2, R ), obj2 );;
gap> obj1_a := AsAdelmanCategoryObject( obj1 );;
gap> obj2_a := AsAdelmanCategoryObject( obj2 );;
gap> m := AsAdelmanCategoryMorphism( beta );;
gap> n := AsAdelmanCategoryMorphism( gamma );;
gap> IsWellDefined( m );
true
gap> IsCongruentForMorphisms( PreCompose( m, n ), PreCompose( n, m ) );
false
gap> IsCongruentForMorphisms( SubtractionForMorphisms( m, m ), ZeroMorphism( obj2_a, obj2_a ) );
true
gap> IsCongruentForMorphisms( ZeroObjectFunctorial( adelman ),
>                               PreCompose( UniversalMorphismFromZeroObject( obj1_a ), UniversalMorphismFromZeroObject( obj2_a ) );
>                               );
true
gap> d := [ obj1_a, obj2_a ];;
gap> pi1 := ProjectionInFactorOfDirectSum( d, 1 );;
gap> pi2 := ProjectionInFactorOfDirectSum( d, 2 );;
gap> id := IdentityMorphism( DirectSum( d ) );;
gap> iota1 := InjectionOfCofactorOfDirectSum( d, 1 );;
gap> iota2 := InjectionOfCofactorOfDirectSum( d, 2 );;
gap> IsCongruentForMorphisms( PreCompose( pi1, iota1 ) + PreCompose( pi2, iota2 ), id );
true
gap> IsCongruentForMorphisms( UniversalMorphismIntoDirectSum( d, [ pi1, pi2 ] ), id );
true
gap> IsCongruentForMorphisms( UniversalMorphismFromDirectSum( d, [ iota1, iota2 ] ), id );
true
gap> c := CokernelProjection( m );;
gap> c2 := CokernelProjection( c );;
gap> IsCongruentForMorphisms( c2, ZeroMorphism( Source( c2 ), Range( c2 ) ) );
true

```



```

gap> IsWellDefined( CokernelProjection( m ) );
true
gap> IsCongruentForMorphisms( CokernelColift( m, CokernelProjection( m ) ), IdentityMorphism( CokernelObj ) );
true
gap> k := KernelEmbedding( c );
gap> IsZeroForMorphisms( PreCompose( k, c ) );
true
gap> IsCongruentForMorphisms( KernelLift( m, KernelEmbedding( m ) ), IdentityMorphism( KernelObj ) );
true

```

12.5 Adelman category basics

Example

```

gap> quiver := RightQuiver( "Q(3)[a:1->2,b:1->2,c:2->3]" );
gap> kQ := PathAlgebra( HomalgFieldOfRationals(), quiver );
gap> Aoid := Algebroid( kQ );
gap> SetIsProjective( DistinguishedObjectOfHomomorphismStructure( Aoid ), true );
gap> mm := SetOfGeneratingMorphisms( Aoid );
gap> CapCategorySwitchLogicOff( Aoid );
gap> Acat := AdditiveClosure( Aoid );
gap> a := AsAdditiveClosureMorphism( mm[1] );
gap> b := AsAdditiveClosureMorphism( mm[2] );
gap> c := AsAdditiveClosureMorphism( mm[3] );
gap> a := AsAdelmanCategoryMorphism( a );
gap> b := AsAdelmanCategoryMorphism( b );
gap> c := AsAdelmanCategoryMorphism( c );
gap> A := Source( a );
gap> B := Range( a );
gap> C := Range( c );
gap> HomomorphismStructureOnObjects( A, C );
gap> HomomorphismStructureOnMorphisms( IdentityMorphism( A ), c );
gap> mor := InterpretMorphismAsMorphismFromDistinguishedObjectToHomomorphismStructure( a );
gap> int := InterpretMorphismFromDistinguishedObjectToHomomorphismStructureAsMorphism( A, B, mor );
gap> IsCongruentForMorphisms( int, a );
true

```

Example

```

gap> R := HomalgRingOfIntegers();
gap> RowsR := CategoryOfRows( R );
gap> one := AsCategoryOfRowsMorphism( HomalgMatrix( [ [ 1 ] ], 1, 1, R ), RowsR );
gap> two := AsCategoryOfRowsMorphism( HomalgMatrix( [ [ 2 ] ], 1, 1, R ), RowsR );
gap> four := AsCategoryOfRowsMorphism( HomalgMatrix( [ [ 4 ] ], 1, 1, R ), RowsR );
gap> source := AdelmanCategoryObject( two, two );
gap> range := AdelmanCategoryObject( two, four );
gap> mor := AdelmanCategoryMorphism( source, one, range );
gap> IsZero( mor );
false
gap> emb := EmbeddingFunctorIntoFreydCategory( RowsR );
gap> ind := AdelmanCategoryFunctorInducedByUniversalProperty( emb );
gap> IsZero( ApplyFunctor( ind, mor ) );
true
gap> M := FreydCategoryObject( AsCategoryOfRowsMorphism( HomalgMatrix( [ [ 2, 2 ], [ 4, 4, 6 ] ], 2, 2, R ), RowsR ), RowsR );
gap> as_tensor := EmbeddingFunctorOfFreydCategoryIntoAdelmanCategory( RowsR );

```

```

gap> Mt := ApplyFunctor( as_tensor, M );;
gap> lsat := LeftSatelliteAsEndofunctorOfAdelmanCategory( RowsR );;
gap> rsat := RightSatelliteAsEndofunctorOfAdelmanCategory( RowsR );;
gap> torsion := ApplyFunctor( ind, ( ApplyFunctor( rsat, ApplyFunctor( lsat, Mt ) ) ) );;
gap> unit := UnitOfSatelliteAdjunctionOfAdelmanCategory( RowsR );;
gap> IsZero( ApplyNaturalTransformation( unit, Mt ) );
true
gap> counit := CounitOfSatelliteAdjunctionOfAdelmanCategory( RowsR );;
gap> t := ApplyNaturalTransformation( counit, Mt );;

```

12.6 Adelman snake lemma

Example

```

gap> DeactivateDefaultCaching();
gap> SwitchGeneralizedMorphismStandard( "span" );;
gap> snake_quiver := RightQuiver( "Q(6)[a:1->2,b:2->3,c:3->4]" );;
gap> kQ := PathAlgebra( HomalgFieldOfRationals(), snake_quiver );;
gap> Aoid := Algebroid( kQ, [ kQ.abc ] );;
gap> CapCategorySwitchLogicOff( Aoid );;
gap> m := SetOfGeneratingMorphisms( Aoid );;
gap> a := m[1];;
gap> b := m[2];;
gap> c := m[3];;
gap> add := AdditiveClosure( Aoid );;
gap> DisableInputSanityChecks( add );;
gap> adelman := AdelmanCategory( add );;
gap> a := AsAdditiveClosureMorphism( a );;
gap> b := AsAdditiveClosureMorphism( b );;
gap> c := AsAdditiveClosureMorphism( c );;
gap> aa := AsAdelmanCategoryMorphism( a );;
gap> bb := AsAdelmanCategoryMorphism( b );;
gap> cc := AsAdelmanCategoryMorphism( c );;
gap> dd := CokernelProjection( aa );;
gap> ee := CokernelColift( aa, PreCompose( bb, cc ) );;
gap> ff := KernelEmbedding( ee );;
gap> gg := KernelEmbedding( cc );;
gap> hh := KernelLift( cc, PreCompose( aa, bb ) );;
gap> ii := CokernelProjection( hh );;
gap> fff := AsGeneralizedMorphism( ff );;
gap> ddd := AsGeneralizedMorphism( dd );;
gap> bbb := AsGeneralizedMorphism( bb );;
gap> ggg := AsGeneralizedMorphism( gg );;
gap> iii := AsGeneralizedMorphism( ii );;
gap> p := PreCompose( [ fff, PseudoInverse( ddd ), bbb, PseudoInverse( ggg ), iii ] );;
gap> IsHonest( p );
true
gap> jj := KernelObjectFunctorial( bb, dd, ee );;
gap> pp := HonestRepresentative( p );;
gap> comp := PreCompose( jj, pp );;
gap> IsZero( comp );
true

```

Example

```

gap> SwitchGeneralizedMorphismStandard( "cospan" );;
gap> snake_quiver := RightQuiver( "Q(6)[a:1->2,b:2->3,c:3->4]" );;
gap> QQ := HomalgFieldOfRationals();;
gap> A := PathAlgebra( QQ, snake_quiver );;
gap> A := QuotientOfPathAlgebra( A, [ A.abc ] );;
gap> QRowsA := QuiverRows( A );;
gap> SetIsProjective( DistinguishedObjectOfHomomorphismStructure( QRowsA ), true );;
gap> a := AsQuiverRowsMorphism( A.a, QRowsA );;
gap> b := AsQuiverRowsMorphism( A.b, QRowsA );;
gap> c := AsQuiverRowsMorphism( A.c, QRowsA );;
gap> aa := AsAdelmanCategoryMorphism( a );;
gap> bb := AsAdelmanCategoryMorphism( b );;
gap> cc := AsAdelmanCategoryMorphism( c );;
gap> dd := CokernelProjection( aa );;
gap> ee := CokernelColift( aa, PreCompose( bb, cc ) );;
gap> ff := KernelEmbedding( ee );;
gap> gg := KernelEmbedding( cc );;
gap> hh := KernelLift( cc, PreCompose( aa, bb ) );;
gap> ii := CokernelProjection( hh );;
gap> fff := AsGeneralizedMorphism( ff );;
gap> ddd := AsGeneralizedMorphism( dd );;
gap> bbb := AsGeneralizedMorphism( bb );;
gap> ggg := AsGeneralizedMorphism( gg );;
gap> iii := AsGeneralizedMorphism( ii );;
gap> p := PreCompose( [ fff, PseudoInverse( ddd ), bbb, PseudoInverse( ggg ), iii ] );;
gap> IsHonest( p );
true
gap> jj := KernelObjectFunctorial( bb, dd, ee );;
gap> kk := CokernelObjectFunctorial( hh, gg, bb );;
gap> pp := HonestRepresentative( p );;
gap> comp := PreCompose( jj, pp );;
gap> IsZero( comp );
true
gap> comp := PreCompose( pp, kk );;
gap> IsZero( comp );
true
gap> homology := function( alpha, beta ) return CokernelObject( LiftAlongMonomorphism( KernelEmbedding( alpha ), beta ) );;
gap> IsZero( homology( jj, pp ) );
true
gap> IsZero( homology( pp, kk ) );
true

```

12.7 Basics based on category of rows

Example

```

gap> R := HomalgRingOfIntegers();;
gap> cat := CategoryOfRows( R );;
gap> obj1 := CategoryOfRowsObject( 1, cat );;
gap> obj2 := CategoryOfRowsObject( 2, cat );;
gap> id := IdentityMorphism( obj2 );;
gap> alpha := CategoryOfRowsMorphism( obj1, HomalgMatrix( [ [ 1, 2 ] ], 1, 2, R ), obj2 );;

```

```

gap> beta := CategoryOfRowsMorphism( obj2, HomalgMatrix( [ [ 1, 2 ], [ 3, 4 ] ], 2, 2, R ), obj2
gap> comp := PreCompose( alpha, beta );;
gap> IsZero( comp );;
gap> zero := ZeroMorphism( obj1, obj2 );;
gap> IsZero( zero );;
gap> ZeroObject( cat );;
gap> UniversalMorphismIntoZeroObject( obj2 );;
gap> UniversalMorphismFromZeroObject( obj1 );;
gap> DirectSum( obj1, obj2 );;
gap> DirectSumFunctorial( [ alpha, beta, id ] );;
gap> ProjectionInFactorOfDirectSum( [ obj2, obj1, obj2 ], 3 );;
gap> UniversalMorphismIntoDirectSum( [ alpha, alpha, alpha ] );;
gap> InjectionOfCofactorOfDirectSum( [ obj2, obj2, obj1 ], 2 );;
gap> gamma := CategoryOfRowsMorphism( obj2, HomalgMatrix( [ [ 1, 1 ], [ 1, 1 ] ], 2, 2, R ), obj2
gap> IsColiftable( beta, gamma );
true
gap> IsColiftable( gamma, beta );
false
gap> ProjectionInFirstFactorOfWeakBiFiberProduct( gamma, gamma );;
gap> ProjectionInFirstFactorOfWeakBiFiberProduct( gamma, ZeroMorphism( Range( gamma ), Range( gamma
gap> lift_arg_1 := PreCompose( ProjectionInFirstFactorOfWeakBiFiberProduct( gamma, gamma + gamma
gap> lift_arg_2 := gamma + gamma;;
gap> IsLiftable( lift_arg_1, lift_arg_2 );;
gap> Lift( lift_arg_1, lift_arg_2 );;
gap> pi1 := ProjectionInFirstFactorOfWeakBiFiberProduct( alpha, beta );;
gap> pi2 := ProjectionInSecondFactorOfWeakBiFiberProduct( alpha, beta );;
gap> IsEqualForMorphisms( PreCompose( pi1, alpha ), PreCompose( pi2, beta ) );;
gap> inj1 := InjectionOfFirstCofactorOfWeakBiPushout( gamma + gamma, gamma );;
gap> inj2 := InjectionOfSecondCofactorOfWeakBiPushout( gamma + gamma, gamma );;
gap> IsEqualForMorphisms( PreCompose( gamma + gamma, inj1 ), PreCompose( gamma, inj2 ) );;
gap> WeakKernelLift( WeakCokernelProjection( gamma ), gamma );;
gap> pi1 := InjectionOfFirstCofactorOfWeakBiPushout( alpha, alpha );;
gap> pi2 := InjectionOfSecondCofactorOfWeakBiPushout( alpha, alpha );;
gap> UniversalMorphismFromWeakBiPushout( alpha, alpha, pi1, pi2 );;
gap> ## Freyd categories
> freyd := FreydCategory( cat );;
gap> IsAbelianCategory( freyd );;
gap> obj_gamma := FreydCategoryObject( gamma );;
gap> f := FreydCategoryMorphism( obj_gamma, gamma, obj_gamma );;
gap> witness := MorphismWitness( f );;
gap> g := FreydCategoryMorphism( obj_gamma, ZeroMorphism( obj2, obj2 ), obj_gamma );;
gap> IsCongruentForMorphisms( f, g );;
gap> c := PreCompose( f, f );;
gap> s := g + g;;
gap> a := CategoryOfRowsMorphism( obj1, HomalgMatrix( [ [ 2 ] ], 1, 1, R ), obj1 );;
gap> Z2 := FreydCategoryObject( a );;
gap> id := IdentityMorphism( Z2 );;
gap> z := id + id + id;;
gap> d := DirectSumFunctorial( [ z, z, z ] );;
gap> pr2 := ProjectionInFactorOfDirectSum( [ Z2, Z2, Z2 ], 2 );;
gap> pr3 := ProjectionInFactorOfDirectSum( [ Z2, Z2, Z2 ], 3 );;
gap> UniversalMorphismIntoDirectSum( [ pr3, pr2 ] );;

```

```

gap> inj1 := InjectionOfCofactorOfDirectSum( [ Z2, Z2, Z2 ], 1 );;
gap> inj2 := InjectionOfCofactorOfDirectSum( [ Z2, Z2, Z2 ], 2 );;
gap> UniversalMorphismFromDirectSum( [ inj2, inj1 ] );;
gap> ZFree := obj1/freyd;;
gap> id := IdentityMorphism( ZFree );;
gap> z := id + id;;
gap> CokernelProjection( z );;
gap> CokernelColift( z, CokernelProjection( z ) );;
gap> S := HomalgFieldOfRationalsInSingular() * "x,y,z";;
gap> Rows_S := CategoryOfRows( S );;
gap> S3 := CategoryOfRowsObject( 3, Rows_S );;
gap> S1 := CategoryOfRowsObject( 1, Rows_S );;
gap> mor := CategoryOfRowsMorphism( S3, HomalgMatrix( "[x,y,z]", 3, 1, S ), S1 );;
gap> biased_w := CategoryOfRowsMorphism( S3, HomalgMatrix( "[x,0,0,0,x,0,0,0,x]", 3, 3, S ), S3 );;
gap> biased_h := CategoryOfRowsMorphism( S3, HomalgMatrix( "[x*y, x*z, y^2]", 3, 3, S ), S3 );;
gap> BiasedWeakFiberProduct( biased_h, biased_w );;
gap> ProjectionOfBiasedWeakFiberProduct( biased_h, biased_w );;
gap> IsCongruentForMorphisms(
>   PreCompose( UniversalMorphismIntoBiasedWeakFiberProduct( biased_h, biased_w, biased_h ), Pro
>   biased_h
> );
true
gap> IsCongruentForMorphisms(
>   PreCompose( InjectionOfBiasedWeakPushout( biased_h, biased_w ), UniversalMorphismFromBiasedWe
>   biased_h
> );
true
gap> k := FreydCategoryObject( mor );;
gap> w := EpimorphismFromSomeProjectiveObjectForKernelObject( UniversalMorphismIntoZeroObject( k
gap> k := KernelEmbedding( w );;
gap> ColiftAlongEpimorphism( CokernelProjection( k ), CokernelProjection( k ) );;
gap> ## Homomorphism structures
> a := InterpretMorphismAsMorphismFromDistinguishedObjectToHomomorphismStructure( gamma );;
gap> IsCongruentForMorphisms( InterpretMorphismFromDistinguishedObjectToHomomorphismStructureAsMo
gap> a := ZeroObjectFunctorial( cat );;
gap> IsCongruentForMorphisms( InterpretMorphismFromDistinguishedObjectToHomomorphismStructureAsMo
gap> Z4 := FreydCategoryObject( AsCategoryOfRowsMorphism( HomalgMatrix( "[4]", 1, 1, R ), cat ) );
gap> Z3 := FreydCategoryObject( AsCategoryOfRowsMorphism( HomalgMatrix( "[3]", 1, 1, R ), cat ) );
gap> HomomorphismStructureOnObjects( Z4, Z2 );;
gap> HomomorphismStructureOnObjects( Z4, Z4 );;
gap> HomomorphismStructureOnObjects( Z2, Z4 );;
gap> HomomorphismStructureOnObjects( Z3, Z4 );;
gap> HomomorphismStructureOnMorphisms( IdentityMorphism( DirectSum( Z4, Z2, Z3 ) ), -IdentityMorp
gap> ## Lifts
> S2 := CategoryOfRowsObject( 2, Rows_S );;
gap> S4 := CategoryOfRowsObject( 4, Rows_S );;
gap> S1_freyd := AsFreydCategoryObject( S1 );;
gap> S2_freyd := AsFreydCategoryObject( S2 );;
gap> S3_freyd := AsFreydCategoryObject( S3 );;
gap> S4_freyd := AsFreydCategoryObject( S4 );;
gap> lift := FreydCategoryMorphism( S1_freyd, CategoryOfRowsMorphism( S1, HomalgMatrix( "[x]", 1,
gap> gamma := FreydCategoryMorphism( S1_freyd, CategoryOfRowsMorphism( S1, HomalgMatrix( "[y]", 1,

```

```

gap> alpha := PreCompose( lift, gamma );
gap> IsLiftable( alpha, gamma );
true
gap> Lift( alpha, gamma );
gap> IsColiftable( lift, alpha );
true
gap> IsCongruentForMorphisms( PreCompose( lift, Colift( lift, alpha ) ), alpha );
true
gap> lift := FreydCategoryMorphism( S2_freyd, CategoryOfRowsMorphism( S2, HomalgMatrix( "[x,y,z,x]" ) );
gap> gamma := FreydCategoryMorphism( S3_freyd, CategoryOfRowsMorphism( S3, HomalgMatrix( "[x,y,z,x]" ) );
gap> alpha := PreCompose( lift, gamma );
gap> Lift( alpha, gamma );
gap> Colift( lift, alpha );
gap> IsCongruentForMorphisms( PreCompose( lift, Colift( lift, alpha ) ), alpha );
gap> interpretation := InterpretMorphismAsMorphismFromDistinguishedObjectToHomomorphismStructure( lift, gamma );
gap> IsCongruentForMorphisms( gamma,
> InterpretMorphismFromDistinguishedObjectToHomomorphismStructureAsMorphism( Source( gamma ), Range( gamma ) );
gap> ## Opposite
> HomomorphismStructureOnObjects( Opposite( Z4 ), Opposite( Z2 ) );
gap> HomomorphismStructureOnObjects( Z2, Z4 );
gap> interpretation := InterpretMorphismAsMorphismFromDistinguishedObjectToHomomorphismStructure( lift, gamma );
gap> IsCongruentForMorphisms( Opposite( gamma ),
> InterpretMorphismFromDistinguishedObjectToHomomorphismStructureAsMorphism( Source( Opposite( gamma ) ), Range( Opposite( gamma ) );
true

```

12.8 Basics of additive closure

Example

```

gap> ## Algebroid
> snake_quiver := RightQuiver( "Q(6)[a:1->2,b:2->3,c:1->4,d:2->5,e:3->6,f:4->5,g:5->6]" );
gap> kQ := PathAlgebra( HomalgFieldOfRationalsInSingular(), snake_quiver );
gap> A := kQ / [ kQ.ad - kQ.cf, kQ.dg - kQ.be, kQ.ab, kQ.fg ];
gap> Aoid := Algebroid( kQ, [ kQ.ad - kQ.cf, kQ.dg - kQ.be, kQ.ab, kQ.fg ] );
gap> s := SetOfObjects( Aoid );
gap> m := SetOfGeneratingMorphisms( Aoid );
gap> interpretation := InterpretMorphismAsMorphismFromDistinguishedObjectToHomomorphismStructure( lift, gamma );
gap> InterpretMorphismFromDistinguishedObjectToHomomorphismStructureAsMorphism( Source( m[3] ), Range( m[3] ) );
gap> ## additive closure
> add := AdditiveClosure( Aoid );
gap> obj1 := AdditiveClosureObject( [ s[1], s[2] ], add );
gap> mor := AdditiveClosureMorphism( obj1, [ IdentityMorphism( s[1] ), ZeroMorphism( s[1], s[2] ) ] );
gap> IsWellDefined( mor );
gap> IsCongruentForMorphisms( PreCompose( mor, mor ), IdentityMorphism( obj1 ) );
gap> obj2 := AdditiveClosureObject( [ s[3], s[3] ], add );
gap> id := IdentityMorphism( obj2 );
gap> objs1 := AdditiveClosureObject( [ s[1] ], add );
gap> objs2 := AdditiveClosureObject( [ s[2] ], add );
gap> ids1 := IdentityMorphism( objs1 );
gap> ids2 := IdentityMorphism( objs2 );
gap> HomomorphismStructureOnMorphisms( DirectSumFunctorial( [ ids1, ids2 ] ), ids1 );
gap> interpretation := InterpretMorphismAsMorphismFromDistinguishedObjectToHomomorphismStructure( lift, gamma );
gap> IsCongruentForMorphisms(

```

```

> InterpretMorphismFromDistinguishedObjectToHomomorphismStructureAsMorphism( Source( mor ), Range( mor ) );
> mor );
gap> a := AsAdditiveClosureMorphism( m[1] );
gap> b := AsAdditiveClosureMorphism( m[2] );
gap> c := AsAdditiveClosureMorphism( m[3] );
gap> d := AsAdditiveClosureMorphism( m[4] );
gap> e := AsAdditiveClosureMorphism( m[5] );
gap> f := AsAdditiveClosureMorphism( m[6] );
gap> g := AsAdditiveClosureMorphism( m[7] );
gap> l := Lift( PreCompose( a, d ), f );
gap> IsCongruentForMorphisms( PreCompose( l, f ), PreCompose( a, d ) );
true
gap> l := Colift( c, PreCompose( a, d ) );
gap> IsCongruentForMorphisms( PreCompose( c, l ), PreCompose( a, d ) );
true

```

12.9 Basics based on category of columns

Example

```

gap> R := HomalgRingOfIntegers();
gap> cat := CategoryOfColumns( R );
gap> obj1 := CategoryOfColumnsObject( 1, cat );
gap> obj2 := CategoryOfColumnsObject( 2, cat );
gap> id := IdentityMorphism( obj2 );
gap> alpha := CategoryOfColumnsMorphism( obj1, HomalgMatrix( [ [ 1 ], [ 2 ] ], 1, 2, R ), obj2 );
gap> beta := CategoryOfColumnsMorphism( obj2, HomalgMatrix( [ [ 1, 2 ], [ 3, 4 ] ], 2, 2, R ), obj1 );
gap> comp := PreCompose( alpha, beta );
gap> IsZero( comp );
gap> zero := ZeroMorphism( obj1, obj2 );
gap> IsZero( zero );
gap> ZeroObject( cat );
gap> UniversalMorphismIntoZeroObject( obj2 );
gap> UniversalMorphismFromZeroObject( obj1 );
gap> DirectSum( obj1, obj2 );
gap> DirectSumFunctorial( [ alpha, beta, id ] );
gap> ProjectionInFactorOfDirectSum( [ obj2, obj1, obj2 ], 3 );
gap> UniversalMorphismIntoDirectSum( [ alpha, alpha, alpha ] );
gap> InjectionOfCofactorOfDirectSum( [ obj2, obj2, obj1 ], 2 );
gap> gamma := CategoryOfColumnsMorphism( obj2, HomalgMatrix( [ [ 1, 1 ], [ 1, 1 ] ], 2, 2, R ), obj1 );
gap> IsColiftable( beta, gamma );
false
gap> IsColiftable( gamma, beta );
false
gap> ProjectionInFirstFactorOfWeakBiFiberProduct( gamma, gamma );
gap> ProjectionInFirstFactorOfWeakBiFiberProduct( gamma, ZeroMorphism( Range( gamma ), Range( gamma ) ) );
gap> lift_arg_1 := PreCompose( ProjectionInFirstFactorOfWeakBiFiberProduct( gamma, gamma + gamma ), gamma );
gap> lift_arg_2 := gamma + gamma;
gap> IsLiftable( lift_arg_1, lift_arg_2 );
gap> Lift( lift_arg_1, lift_arg_2 );
gap> pi1 := ProjectionInFirstFactorOfWeakBiFiberProduct( alpha, beta );
gap> pi2 := ProjectionInSecondFactorOfWeakBiFiberProduct( alpha, beta );
gap> IsEqualForMorphisms( PreCompose( pi1, alpha ), PreCompose( pi2, beta ) );

```

```

gap> inj1 := InjectionOfFirstCofactorOfWeakBiPushout( gamma + gamma, gamma );;;
gap> inj2 := InjectionOfSecondCofactorOfWeakBiPushout( gamma + gamma, gamma );;;
gap> IsEqualForMorphisms( PreCompose( gamma + gamma, inj1 ), PreCompose( gamma, inj2 ) );;;
gap> WeakKernelLift( WeakCokernelProjection( gamma ), gamma );;;
gap> pi1 := InjectionOfFirstCofactorOfWeakBiPushout( alpha, alpha );;;
gap> pi2 := InjectionOfSecondCofactorOfWeakBiPushout( alpha, alpha );;;
gap> UniversalMorphismFromWeakBiPushout( alpha, alpha, pi1, pi2 );;;
gap> ## Freyd categories
> freyd := FreydCategory( cat );;;
gap> IsAbelianCategory( freyd );;;
gap> obj_gamma := FreydCategoryObject( gamma );;;
gap> f := FreydCategoryMorphism( obj_gamma, gamma, obj_gamma );;;
gap> witness := MorphismWitness( f );;;
gap> g := FreydCategoryMorphism( obj_gamma, ZeroMorphism( obj2, obj2 ), obj_gamma );;;
gap> IsCongruentForMorphisms( f, g );;;
gap> c := PreCompose( f, f );;;
gap> s := g + g;;
gap> a := CategoryOfColumnsMorphism( obj1, HomalgMatrix( [ [ 2 ] ], 1, 1, R ), obj1 );;;
gap> Z2 := FreydCategoryObject( a );;;
gap> id := IdentityMorphism( Z2 );;;
gap> z := id + id + id;;
gap> d := DirectSumFunctorial( [ z, z, z ] );;;
gap> pr2 := ProjectionInFactorOfDirectSum( [ Z2, Z2, Z2 ], 2 );;;
gap> pr3 := ProjectionInFactorOfDirectSum( [ Z2, Z2, Z2 ], 3 );;;
gap> UniversalMorphismIntoDirectSum( [ pr3, pr2 ] );;;
gap> inj1 := InjectionOfCofactorOfDirectSum( [ Z2, Z2, Z2 ], 1 );;;
gap> inj2 := InjectionOfCofactorOfDirectSum( [ Z2, Z2, Z2 ], 2 );;;
gap> UniversalMorphismFromDirectSum( [ inj2, inj1 ] );;;
gap> ZFree := AsFreydCategoryObject( obj1 );;;
gap> id := IdentityMorphism( ZFree );;;
gap> z := id + id;;
gap> CokernelProjection( z );;;
gap> CokernelColift( z, CokernelProjection( z ) );;;
gap> S := HomalgFieldOfRationalsInSingular() * "x,y,z";;
gap> Cols_S := CategoryOfColumns( S );;;
gap> S3 := CategoryOfColumnsObject( 3, Cols_S );;;
gap> S1 := CategoryOfColumnsObject( 1, Cols_S );;;
gap> mor := CategoryOfColumnsMorphism( S3, HomalgMatrix( "[x,y,z]", 1, 3, S ), S1 );;;
gap> biased_w := CategoryOfColumnsMorphism( S3, HomalgMatrix( "[x,0,0,0,x,0,0,0,x]", 3, 3, S ), S3 );;;
gap> biased_h := CategoryOfColumnsMorphism( S3, HomalgMatrix( "[x*y, x*z, y^2]", 3, 3, S ), S3 );;;
gap> BiasedWeakFiberProduct( biased_h, biased_w );;;
gap> ProjectionOfBiasedWeakFiberProduct( biased_h, biased_w );;;
gap> IsCongruentForMorphisms(
>   PreCompose( UniversalMorphismIntoBiasedWeakFiberProduct( biased_h, biased_w, biased_h ), Pro
>   biased_h
> );
true
gap> IsCongruentForMorphisms(
>   PreCompose( InjectionOfBiasedWeakPushout( biased_h, biased_w ), UniversalMorphismFromBiasedWe
>   biased_h
> );
true

```



```

gap> k := FreydCategoryObject( mor );;
gap> w := EpimorphismFromSomeProjectiveObjectForKernelObject( UniversalMorphismIntoZeroObject( k );;
gap> k := KernelEmbedding( w );;
gap> ColiftAlongEpimorphism( CokernelProjection( k ), CokernelProjection( k ) );;
gap> ## Homomorphism structures
> a := InterpretMorphismAsMorphismFromDistinguishedObjectToHomomorphismStructure( gamma );;
gap> IsCongruentForMorphisms( InterpretMorphismFromDistinguishedObjectToHomomorphismStructureAsMorphism( a );;
gap> a := ZeroObjectFunctorial( cat );;
gap> IsCongruentForMorphisms( InterpretMorphismFromDistinguishedObjectToHomomorphismStructureAsMorphism( a );;
gap> Z4 := FreydCategoryObject( AsCategoryOfColumnsMorphism( HomalgMatrix( "[4]", 1, 1, R ), cat );;
gap> Z3 := FreydCategoryObject( AsCategoryOfColumnsMorphism( HomalgMatrix( "[3]", 1, 1, R ), cat );;
gap> HomomorphismStructureOnObjects( Z4, Z2 );;
gap> HomomorphismStructureOnObjects( Z4, Z4 );;
gap> HomomorphismStructureOnObjects( Z2, Z4 );;
gap> HomomorphismStructureOnObjects( Z3, Z4 );;
gap> HomomorphismStructureOnMorphisms( IdentityMorphism( DirectSum( Z4, Z2, Z3 ) ), -IdentityMorphism( Z4, Z2, Z3 );;
gap> ## Lifts
> S2 := CategoryOfColumnsObject( 2, Cols_S );;
gap> S4 := CategoryOfColumnsObject( 4, Cols_S );;
gap> S1_freyd := AsFreydCategoryObject( S1 );;
gap> S2_freyd := AsFreydCategoryObject( S2 );;
gap> S3_freyd := AsFreydCategoryObject( S3 );;
gap> S4_freyd := AsFreydCategoryObject( S4 );;
gap> lift := FreydCategoryMorphism( S1_freyd, CategoryOfColumnsMorphism( S1, HomalgMatrix( "[x]", 1, 1, R );;
gap> gamma := FreydCategoryMorphism( S1_freyd, CategoryOfColumnsMorphism( S1, HomalgMatrix( "[y]", 1, 1, R );;
gap> alpha := PreCompose( lift, gamma );;
gap> Lift( alpha, gamma );;
gap> Colift( lift, alpha );;
gap> IsCongruentForMorphisms( PreCompose( lift, Colift( lift, alpha ) ), alpha );;
gap> lift := FreydCategoryMorphism( S2_freyd, CategoryOfColumnsMorphism( S2, HomalgMatrix( "[x]", 1, 1, R );;
gap> gamma := FreydCategoryMorphism( S3_freyd, CategoryOfColumnsMorphism( S3, HomalgMatrix( "[x]", 1, 1, R );;
gap> alpha := PreCompose( lift, gamma );;
gap> Lift( alpha, gamma );;
gap> Colift( lift, alpha );;
gap> IsCongruentForMorphisms( PreCompose( lift, Colift( lift, alpha ) ), alpha );;
gap> interpretation := InterpretMorphismAsMorphismFromDistinguishedObjectToHomomorphismStructure( gamma );;
gap> IsCongruentForMorphisms( gamma,
> InterpretMorphismFromDistinguishedObjectToHomomorphismStructureAsMorphism( Source( gamma ), Range( gamma );;
gap> ## Opposite
> HomomorphismStructureOnObjects( Opposite( Z4 ), Opposite( Z2 ) );;
gap> HomomorphismStructureOnObjects( Z2, Z4 );;
gap> interpretation := InterpretMorphismAsMorphismFromDistinguishedObjectToHomomorphismStructure( gamma );;
gap> IsCongruentForMorphisms( Opposite( gamma ),
> InterpretMorphismFromDistinguishedObjectToHomomorphismStructureAsMorphism( Source( Opposite( gamma ), Range( Opposite( gamma );;
true

```

12.10 Cokernel image closure in category of rows

Example

```

gap> R := HomalgFieldOfRationalsInSingular() * "x,y,z";;
gap> RowsR := CategoryOfRows( R );;
gap> m := AsCategoryOfRowsMorphism(

```

```

> HomalgMatrix( "[[x],[y],[z]]", 3, 1, R ), RowsR
> );;
gap> mu := AsCokernelImageClosureMorphism( m );;
gap> C := CokernelObject( mu );;
gap> C2 := AsFinitelyPresentedCokernelImageClosureObject( m );;
gap> IsEqualForObjects( C, C2 );
true
gap> n := AsCategoryOfRowsMorphism(
> HomalgMatrix( "[[x,y],[y^2,z]]", 2, 2, R ), RowsR
> );;
gap> nu := AsCokernelImageClosureMorphism( n );;
gap> nu2 := PreCompose( nu, nu );;
gap> IsWellDefined( nu2 );
true
gap> IsCongruentForMorphisms( nu, nu2 );
false
gap> nu + nu;;
gap> nu2 - nu;;
gap> cat := CapCategory( nu );;
gap> ZeroObject( cat );;
gap> ZeroMorphism( Source( nu ), Source( mu ) );;
gap> UniversalMorphismIntoZeroObject( Source( nu ) );;
gap> UniversalMorphismFromZeroObject( Source( nu ) );;
gap> S := Source( mu );;
gap> DirectSum( [S, S, S] );;
gap> DirectSumFunctorial( [ nu2, nu ] );;
gap> UniversalMorphismIntoDirectSum( [ nu, nu ] );;
gap> UniversalMorphismFromDirectSum( [ nu, nu ] );;
gap> ProjectionInFactorOfDirectSum( [ S, S, S ], 2 );;
gap> InjectionOfCofactorOfDirectSum( [ S, S, S, S ], 4 );;
gap> CokernelColift( nu, CokernelProjection( nu ) );;
gap> IsCongruentForMorphisms( nu, PreCompose( CostrictionToImage( nu ), ImageEmbedding( nu ) ) );;
true
gap> u := UniversalMorphismFromImage( nu, [ nu, IdentityMorphism( Range( nu ) ) ] );;
gap> IsWellDefined( u );
true
gap> IsCongruentForMorphisms( nu, PreCompose( CostrictionToImage( nu ), u ) );
true
gap> IsCongruentForMorphisms( u, ImageEmbedding( nu ) );
true
gap> kernel := KernelObject( mu );;
gap> emb := KernelEmbedding( mu );;
gap> p := PreCompose( EpimorphismFromSomeProjectiveObject( kernel ), KernelEmbedding( mu ) );;
gap> KernelLift( mu, p );;
gap> LiftAlongMonomorphism( emb, p );;
gap> I_to_A := FunctorCokernelImageClosureToFreydCategory( RowsR );;
gap> A_to_I := FunctorFreydCategoryToCokernelImageClosure( RowsR );;
gap> ApplyFunctor( I_to_A, kernel );;
gap> ApplyFunctor( A_to_I, ApplyFunctor( I_to_A, kernel ) );;
gap> nu := NaturalIsomorphismFromIdentityToFinitePresentationOfCokernelImageClosureObject( RowsR
gap> mu := NaturalIsomorphismFromFinitePresentationOfCokernelImageClosureObjectToIdentity( RowsR
gap> IsCongruentForMorphisms(

```

```

> IdentityMorphism( kernel ),
> PreCompose( ApplyNaturalTransformation( nu, kernel ), ApplyNaturalTransformation( mu, kernel ) );
> );
true

```

12.11 Cokernel image closure in category of columns

Example

```

gap> R := HomalgFieldOfRationalsInSingular() * "x,y,z";;
gap> ColsR := CategoryOfColumns( R );;
gap> m := AsCategoryOfColumnsMorphism(
> HomalgMatrix( "[[x],[y],[z]]", 1, 3, R ), ColsR
> );;
gap> mu := AsCokernelImageClosureMorphism( m );;
gap> C := CokernelObject( mu );;
gap> C2 := AsFinitelyPresentedCokernelImageClosureObject( m );;
gap> IsEqualForObjects( C, C2 );
true
gap> n := AsCategoryOfColumnsMorphism(
> HomalgMatrix( "[[x,y],[y^2,z]]", 2, 2, R ), ColsR
> );;
gap> nu := AsCokernelImageClosureMorphism( n );;
gap> nu2 := PreCompose( nu, nu );;
gap> IsWellDefined( nu2 );
true
gap> IsCongruentForMorphisms( nu, nu2 );
false
gap> nu + nu;;
gap> nu2 - nu;;
gap> cat := CapCategory( nu );;
gap> ZeroObject( cat );;
gap> ZeroMorphism( Source( nu ), Source( mu ) );;
gap> UniversalMorphismIntoZeroObject( Source( nu ) );;
gap> UniversalMorphismFromZeroObject( Source( nu ) );;
gap> S := Source( mu );;
gap> DirectSum( [S, S, S] );;
gap> DirectSumFunctorial( [ nu2, nu ] );;
gap> UniversalMorphismIntoDirectSum( [ nu, nu ] );;
gap> UniversalMorphismFromDirectSum( [ nu, nu ] );;
gap> ProjectionInFactorOfDirectSum( [ S, S, S ], 2 );;
gap> InjectionOfCofactorOfDirectSum( [ S, S, S ], 4 );;
gap> CokernelColift( nu, CokernelProjection( nu ) );;
gap> IsCongruentForMorphisms( nu, PreCompose( CostrictionToImage( nu ), ImageEmbedding( nu ) ) );
true
gap> u := UniversalMorphismFromImage( nu, [ nu, IdentityMorphism( Range( nu ) ) ] );;
gap> IsWellDefined( u );
true
gap> IsCongruentForMorphisms( nu, PreCompose( CostrictionToImage( nu ), u ) );
true
gap> IsCongruentForMorphisms( u, ImageEmbedding( nu ) );
true
gap> kernel := KernelObject( mu );;

```

```

gap> emb := KernelEmbedding( mu );;
gap> p := PreCompose( EpimorphismFromSomeProjectiveObject( kernel ), KernelEmbedding( mu ) );;
gap> Kernellift( mu, p );;
gap> LiftAlongMonomorphism( emb, p );;
gap> I_to_A := FunctorCokernelImageClosureToFreydCategory( ColsR );;
gap> A_to_I := FunctorFreydCategoryToCokernelImageClosure( ColsR );;
gap> ApplyFunctor( I_to_A, kernel );;
gap> ApplyFunctor( A_to_I, ApplyFunctor( I_to_A, kernel ) );;
gap> nu := NaturalIsomorphismFromIdentityToFinitePresentationOfCokernelImageClosureObject( ColsR
gap> mu := NaturalIsomorphismFromFinitePresentationOfCokernelImageClosureObjectToIdentity( ColsR
gap> IsCongruentForMorphisms(
>   IdentityMorphism( kernel ),
>   PreCompose( ApplyNaturalTransformation( nu, kernel ), ApplyNaturalTransformation( mu, kerne
> );
true

```

12.12 Grade filtration

The sequence of modules computed via satellites behaves in a way that is not understood in the case when the ring is not Auslander regular.

Example

```

gap> R := HomalgFieldOfRationalsInSingular() * "x,y";;
gap> R := R/"x*y"/"x^2";;
gap> RowsR := CategoryOfRows( R );;
gap> Freyd := FreydCategory( RowsR );;
gap> mat := HomalgMatrix( "[x,y]", 1, 2, R );;
gap> M := mat/Freyd;;
gap> mu1 := GradeFiltrationNthMonomorphism( M, 1 );;
gap> IsMonomorphism( mu1 );
true
gap> IsZero( mu1 );
false
gap> IsEpimorphism( mu1 );
false
gap> mu2 := GradeFiltrationNthMonomorphism( M, 2 );;
gap> IsIsomorphism( mu2 );
true
gap> IsZero( mu2 );
false
gap> mu3 := GradeFiltrationNthMonomorphism( M, 3 );;
gap> IsIsomorphism( mu3 );
true
gap> IsZero( mu3 );
false
gap> mu4 := GradeFiltrationNthMonomorphism( M, 4 );;
gap> IsMonomorphism( mu4 );
false
gap> IsEpimorphism( mu4 );
true
gap> IsZero( mu4 );
false

```

Example

```

gap> Qxyz := HomalgFieldOfRationalsInDefaultCAS( ) * "x,y,z";
gap> wmat := HomalgMatrix( "[ \
> x*y, y*z, z, 0, 0, \
> x^3*z,x^2*z^2,0, x*z^2, -z^2, \
> x^4, x^3*z, 0, x^2*z, -x*z, \
> 0, 0, x*y, -y^2, x^2-1,\
> 0, 0, x^2*z, -x*y*z, y*z, \
> 0, 0, x^2*y-x^2,-x*y^2+x*y,y^2-y \
> ]", 6, 5, Qxyz );;
gap> RowsR := CategoryOfRows( Qxyz );;
gap> Freyd := FreydCategory( RowsR );;
gap> Adel := AdelmanCategory( RowsR );;
gap> M := wmat/Freyd;;

```

We compute the grade sequence of functors (it turns out that on the level of functors, we don't get monos)

Example

```

gap> M_tor := M/Adel;;
gap> Mu1 := GradeFiltrationNthNaturalTransformationComponent( M_tor, 1 );;
gap> IsZero( Mu1 );
false
gap> IsMonomorphism( Mu1 );
true
gap> Mu2 := GradeFiltrationNthNaturalTransformationComponent( M_tor, 2 );;
gap> IsZero( Mu2 );
false
gap> IsMonomorphism( Mu2 );
false
gap> Mu3 := GradeFiltrationNthNaturalTransformationComponent( M_tor, 3 );;
gap> IsZero( Mu3 );
false
gap> IsMonomorphism( Mu3 );
false
gap> Mu4 := GradeFiltrationNthNaturalTransformationComponent( M_tor, 4 );;
gap> IsZero( Mu4 );
true

```

We compute the grade sequence of modules (here, we really get monos and thus a filtration)

Example

```

gap> mu1 := GradeFiltrationNthMonomorphism( M, 1 );;
gap> IsZero( mu1 );
false
gap> IsMonomorphism( mu1 );
true
gap> mu2 := GradeFiltrationNthMonomorphism( M, 2 );;
gap> IsZero( mu2 );
false
gap> IsMonomorphism( mu2 );
true
gap> mu3 := GradeFiltrationNthMonomorphism( M, 3 );;
gap> IsZero( mu3 );

```

```

false
gap> IsMonomorphism( mu3 );
true
gap> mu4 := GradeFiltrationNthMonomorphism( M, 4 );;
gap> IsZero( mu4 );
true

```

12.13 Groups as categories

Example

```

gap> G := SymmetricGroup( 3 );;
gap> CG := GroupAsCategory( G );;
#I method installed for IsAutomorphism matches more than one declaration
#I method installed for IsSplitEpimorphism matches more than one declaration
#I method installed for IsSplitMonomorphism matches more than one declaration
gap> u := GroupAsCategoryUniqueObject( CG );;
gap> alpha := GroupAsCategoryMorphism( (1,2,3), CG );;
gap> alpha * Inverse( alpha ) = IdentityMorphism( u );
true
gap> beta := GroupAsCategoryMorphism( (1,2,3,5), CG );;
gap> IsWellDefined( beta );
false
gap> gamma := GroupAsCategoryMorphism( (1,3), CG );;
gap> IsWellDefined( gamma );
true
gap> Lift( alpha, gamma ) * gamma = alpha;
true
gap> alpha * Colift( alpha, gamma ) = gamma;
true
gap> Length( HomomorphismStructureOnObjects( u, u ) ) = Size( G );
true
gap> InterpretMorphismFromDistinguishedObjectToHomomorphismStructureAsMorphism(
>   u,u,
>   PreCompose(
>     InterpretMorphismAsMorphismFromDistinguishedObjectToHomomorphismStructure( alpha ), Hom
>   )
> )
> =
> gamma * alpha * Inverse( gamma );
true
gap> x := (2,3)/CG;;
gap> id := ()/CG;;
gap> IsIdenticalObj( x * x, id );
true

```

12.14 Homomorphisms between f.p. functors based on category of rows

Example

```

gap> R := HomalgFieldOfRationalsInSingular() * "x,y,z";;
gap> Rows_R := CategoryOfRows( R );;
gap> R1 := CategoryOfRowsObject( 1, Rows_R );;

```

```

gap> R3 := CategoryOfRowsObject( 3, Rows_R );;
gap> alpha := CategoryOfRowsMorphism( R3, HomalgMatrix( "[x,y,z]", 3, 1, R ), R1 );;
gap> M := FreydCategoryObject( alpha );;
gap> c0 := CovariantExtAsFreydCategoryObject( M, 0 );;
gap> c1 := CovariantExtAsFreydCategoryObject( M, 1 );;
gap> c2 := CovariantExtAsFreydCategoryObject( M, 2 );;
gap> IsZeroForObjects( HomomorphismStructureOnObjects( c0, c2 ) ); # = Ext^2( M, M )
false

```

12.15 Homomorphisms between f.p. functors based on category of columns

Example

```

gap> R := HomalgFieldOfRationalsInSingular() * "x,y,z";;
gap> Cols_R := CategoryOfColumns( R );;
gap> R1 := CategoryOfColumnsObject( 1, Cols_R );;
gap> R3 := CategoryOfColumnsObject( 3, Cols_R );;
gap> alpha := CategoryOfColumnsMorphism( R3, HomalgMatrix( "[x,y,z]", 1, 3, R ), R1 );;
gap> M := FreydCategoryObject( alpha );;
gap> c0 := CovariantExtAsFreydCategoryObject( M, 0 );;
gap> c1 := CovariantExtAsFreydCategoryObject( M, 1 );;
gap> c2 := CovariantExtAsFreydCategoryObject( M, 2 );;
gap> IsZeroForObjects( HomomorphismStructureOnObjects( c0, c2 ) ); # = Ext^2( M, M )
false

```

12.16 Linear closure of categories

Example

```

gap> G := SymmetricGroup( 3 );;
gap> CG := GroupAsCategory( G );;
#I method installed for IsAutomorphism matches more than one declaration
#I method installed for IsSplitEpimorphism matches more than one declaration
#I method installed for IsSplitMonomorphism matches more than one declaration
gap> compare_func := function( g, h ) return UnderlyingGroupElement( g ) < UnderlyingGroupElement( h );;
gap> ZZ := HomalgRingOfIntegers();;
gap> ZCG := LinearClosure( ZZ, CG, compare_func );;
gap> u := GroupAsCategoryUniqueObject( CG );;
gap> g := GroupAsCategoryMorphism( (1,2,3), CG );;
gap> h := GroupAsCategoryMorphism( (1,2), CG );;
gap> v := LinearClosureObject( ZCG, u );;
gap> elem1 := LinearClosureMorphism( v, [ 1, 2, 3, 4, 5, 6 ], [ g, h, g, h, g, h ], v );;
gap> elem2 := LinearClosureMorphism( v, [ 1, 2, 3, 4, 5, 6 ], [ h, g, h, g, h, g ], v );;
gap> # for i in [ 1 .. 10^6 ] do LinearClosureMorphism( v, [ 1, 2, 3, 4, 5, 6 ], [ g, h, g, h, g, h ], v );;
gap> elem := LinearClosureMorphism( v, [ 0, 0, 0, 0, 0, 0 ], [ g, h, g, h, g, h ], v );;
gap> a := (1,2)/CG/ZCG;;
gap> b := (2,3)/CG/ZCG;;
gap> IsIsomorphism( a + b );
false
gap> Lift( a + b, a ) * a = a + b;
true

```

```
gap> IsLiftable( a + b, -2*a ); ## over Q this is liftable
false
```

12.17 Matrices over $\mathbb{Z}P_K$

Example

```
gap> #Incidence matrix of our proset
> K := [ [1, 1, 1], [0, 1, 1], [0, 1, 1] ];;
gap> #Construction of a tower of categories
> CP_K := ProSetAsCategory( K );;
#I method installed for IsSplitEpimorphism matches more than one declaration
#I method installed for IsSplitMonomorphism matches more than one declaration
gap> ZZ := HomalgRingOfIntegers( );
gap> ZP_K := LinearClosure( ZZ, CP_K, ReturnTrue );
gap> RowsP_K := AdditiveClosure( ZP_K );
gap> a := ProSetAsCategoryObject( 1, CP_K );
gap> b := ProSetAsCategoryObject( 2, CP_K );
gap> c := ProSetAsCategoryObject( 3, CP_K );
gap> #Three random objects in the additive closure
> #Such that there exists morphisms from A->B and B->C:
> rand_coef := List( [ 1 .. 5 ], i -> Random( [ 2 .. 20 ] ) );
gap> A1 := List( [ 1 .. rand_coef[ 1 ] ], i -> a );
gap> A2 := List( [ 1 .. rand_coef[ 2 ] ], i -> b );
gap> A := Concatenation( A1, A2 );
gap> B1 := List( [ 1 .. rand_coef[ 3 ] ], i -> b );
gap> B2 := List( [ 1 .. rand_coef[ 4 ] ], i -> c );
gap> B := Concatenation( B1, B2 );
gap> C := List( [ 1 .. rand_coef[ 5 ] ], i -> c );
gap> #A random lifting problem over ZP_K
> MA_B := List( [ 1 .. rand_coef[ 1 ] + rand_coef[ 2 ] ], i ->
>               List( [ 1 .. rand_coef[ 3 ] + rand_coef[ 4 ] ], j ->
>                   LinearClosureMorphism( LinearClosureObject( A[i], ZP_K ), [Random( [ -20 .. 20 ] )] )
>               );
gap> alpha := MA_B/RowsP_K;;
gap> MB_C := List( [ 1 .. rand_coef[ 3 ] + rand_coef[ 4 ] ], i ->
>               List( [ 1 .. rand_coef[ 5 ] ], j ->
>                   LinearClosureMorphism( LinearClosureObject( B[i], ZP_K ), [Random( [ -20 .. 20 ] )] )
>               );
gap> beta := MB_C/RowsP_K;;
gap> gamma := PreCompose( alpha, beta );
gap> lift := Lift( gamma, beta );
gap> PreCompose(lift, beta) = gamma;
true
```

12.18 Matrices over $\mathbb{Z}G$

Construction of a tower of categories

Example

```
gap> G := SymmetricGroup( 3 );;
gap> CG := GroupAsCategory( G );;
#I method installed for IsAutomorphism matches more than one declaration
#I method installed for IsSplitEpimorphism matches more than one declaration
#I method installed for IsSplitMonomorphism matches more than one declaration
gap> ZZ := HomalgRingOfIntegers( );;
gap> ZCG := LinearClosure( ZZ, CG );;
gap> RowsG := AdditiveClosure( ZCG );;
```

Construction of elements

Example

```
gap> a := (1,2)/CG/ZCG;;
gap> b := (2,3)/CG/ZCG;;
gap> e := ()/CG/ZCG;;
gap> omega := [ [ a - e ], [ b - e ] ]/RowsG;;
gap> u := GroupAsCategoryUniqueObject( CG );;
gap> v := LinearClosureObject( ZCG, u );;
gap> u := AsAdditiveClosureObject( v );;
gap> HomStructure( u, omega );;
```

A random lifting problem over ZG

Example

```
gap> elem := Elements( G );;
gap> elem := List( elem, x -> x/CG/ZCG );;
gap> rand_elem := function() local coeffs; coeffs := List( [ 1 .. 6 ], i -> Random( [ -20 .. 20 ] );;
gap> mat10_11 := List( [ 1 .. 10 ], i ->
>   List( [ 1 .. 11 ], j ->
>     rand_elem()
>   )
> );;
gap> mat11_12 := List( [ 1 .. 11 ], i ->
>   List( [ 1 .. 12 ], j ->
>     rand_elem()
>   )
> );;
gap> alpha := mat10_11/RowsG;;
gap> beta := mat11_12/RowsG;;
gap> gamma := PreCompose( alpha, beta );;
gap> lift := Lift( gamma, beta );;
gap> PreCompose( lift, beta ) = gamma;
true
```

12.19 Prosets

Example

```
gap> K := [ [1, 1, 1], [0, 1, 1], [0, 1, 1] ];;
gap> L := [ [1, 1, 0], [0, 1, 1], [0, 0, 1] ];;
gap> P_K := ProSetAsCategory(K);;
#I method installed for IsSplitEpimorphism matches more than one declaration
#I method installed for IsSplitMonomorphism matches more than one declaration
gap> #ProSetAsCategory(L);
```

```

gap> a := 1/P_K;;
gap> b := ProSetAsCategoryObject(2, P_K);;
gap> c := ProSetAsCategoryObject(3, P_K);;
gap> d := ProSetAsCategoryObject(4, P_K);;
gap> delta := ProSetAsCategoryMorphism(b, a);;
gap> IsWellDefined(a);
true
gap> IsWellDefined(d);
false
gap> IsWellDefined(delta);
false
gap> alpha := ProSetAsCategoryMorphism(a, b);;
gap> beta := ProSetAsCategoryMorphism(b, c);;
gap> gamma := ProSetAsCategoryMorphism(a, c);;
gap> gamma = PreCompose(alpha, beta);
true
gap> id_a := IdentityMorphism(a);;
gap> IsWellDefined(Inverse(alpha));
false
gap> beta*Inverse(beta) = IdentityMorphism(b);
true
gap> alpha = Lift(gamma, beta);
true
gap> fail = Lift(beta, gamma);
true
gap> Colift(alpha, gamma) = beta;
true
gap> alpha = HomStructure(a, b, HomStructure(alpha));
true

```

12.20 Quiver rows basics

Example

```

gap> ## quiver without relations
> QQ := HomalgFieldOfRationals();;
gap> quiver := RightQuiver( "Q(3)[a:1->2,b:1->2,c:2->3]" );;
gap> Av := Vertices( quiver );;
gap> A := PathAlgebra( QQ, quiver );;
gap> a := BasisPaths( CanonicalBasis( A ) );;
gap> a := List( a, p -> PathAsAlgebraElement( A, p ) );;
gap> zA := Zero( A );;
gap> QRowsA := QuiverRows( A );;
gap> mat := [ [ a[1], zA ], [ zA, a[6] ], [ a[1], zA ] ];;
gap> obj1 := QuiverRowsObject( [ [ Av[1], 1 ], [ Av[2], 1 ], [ Av[1], 1 ] ], QRowsA );;
gap> obj2 := QuiverRowsObject( [ [ Av[1], 1 ], [ Av[3], 1 ] ], QRowsA );;
gap> alpha := QuiverRowsMorphism( obj1, mat, obj2 );;
gap> obj3 := QuiverRowsObject( [ [ Av[2], 1 ] ], QRowsA );;
gap> mat := [ [ a[4] ], [ zA ] ];;
gap> beta := QuiverRowsMorphism( obj2, mat, obj3 );;
gap> pre := PreCompose( alpha, beta );;
gap> IsWellDefined( PreCompose( alpha, beta ) );
true

```

```

gap> IsZeroForMorphisms( pre );
false
gap> ze := ZeroMorphism( Source( pre ), Range( pre ) );
gap> IsCongruentForMorphisms( pre + ze, pre );
true
gap> IsCongruentForMorphisms( pre + pre, pre );
false
gap> IsZeroForMorphisms( pre - pre );
true
gap> IsCongruentForMorphisms(
>   PreCompose(
>     UniversalMorphismFromZeroObject( obj1 ),
>     UniversalMorphismIntoZeroObject( obj1 )
>   ),
>   IdentityMorphism( ZeroObject( QRowsA ) )
> );
true
gap> NrSummands( DirectSum( List( [ 1 .. 1000 ], i -> obj1 ) ) ) = 1000 * NrSummands( obj1 );
true
gap> L := [ obj1, obj2, obj3 ];
gap> pi := List( [ 1,2,3 ], i -> ProjectionInFactorOfDirectSum( L, i ) );
gap> iota := List( [ 1,2,3 ], i -> InjectionOfCofactorOfDirectSum( L, i ) );
gap> ForAll( [1,2,3], i ->
>   IsCongruentForMorphisms(
>     PreCompose( iota[i], pi[i] ),
>     IdentityMorphism( L[i] )
>   )
> );
true
gap> IsZeroForMorphisms( PreCompose( iota[2], pi[1] ) );
true
gap> IsCongruentForMorphisms(
>   UniversalMorphismIntoDirectSum( L, pi ),
>   IdentityMorphism( DirectSum( L ) )
> );
true
gap> IsCongruentForMorphisms(
>   UniversalMorphismFromDirectSum( L, iota ),
>   IdentityMorphism( DirectSum( L ) )
> );
true
gap> IsCongruentForMorphisms(
>   InterpretMorphismFromDistinguishedObjectToHomomorphismStructureAsMorphism( obj1, obj2,
>     InterpretMorphismAsMorphismFromDistinguishedObjectToHomomorphismStructure( alpha )
>   ),
>   alpha
> );
true
gap> ## quiver with relations
> quiver := RightQuiver(
>   "Q(8)[a:1->5,b:2->6,c:3->7,d:4->8,e:1->2,f:2->3,g:3->4,h:5->6,i:6->7,j:7->8]"
> );

```

```

gap> Bv := Vertices( quiver );;
gap> QQ := HomalgFieldOfRationals();;
gap> kQ := PathAlgebra( QQ, quiver );;
gap> B := QuotientOfPathAlgebra( kQ,
> [
>   kQ.e * kQ.f, kQ.f * kQ.g,
>   kQ.h * kQ.i, kQ.i * kQ.j,
>   kQ.e * kQ.b - kQ.a * kQ.h,
>   kQ.f * kQ.c - kQ.b * kQ.i,
>   kQ.g * kQ.d - kQ.c * kQ.j ]
> );;
gap> b := BasisPaths( CanonicalBasis( B ) );;
gap> QRowsB := QuiverRows( B );;
gap> obj := QuiverRowsObject( [ [ Bv[1], 2 ], [ Bv[1], 4 ], [ Bv[1], 4 ], [ Bv[1], 6 ] ], QRowsB
gap> IsWellDefined( obj );
true
gap> IdentityMorphism( obj );;

```

12.21 Quiver rows over the integers

Well-defined morphisms

Example

```

gap> QQ := HomalgFieldOfRationals();;
gap> snake_quiver := RightQuiver( "Q(4)[a:1->2,b:2->3,c:3->4]" );;
gap> vertices := Vertices( snake_quiver );;
gap> A := PathAlgebra( QQ, snake_quiver );;
gap> A := QuotientOfPathAlgebra( A, [ A.abc ] );;
gap> QRowsA := QuiverRowsDescentToZDefinedByBasisPaths( A );;
gap> v1 := AsQuiverRowsObject( vertices[1], QRowsA );;
gap> v2 := AsQuiverRowsObject( vertices[2], QRowsA );;
gap> mat := [ [ 1/2*A.a ] ];;
gap> x := QuiverRowsMorphism( v1, mat, v2 );;
gap> IsWellDefined( x );
false
gap> mat := [ [ 2*A.a ] ];;
gap> x := QuiverRowsMorphism( v1, mat, v2 );;
gap> IsWellDefined( x );
true

```

Snake lemma over the integers

Example

```

gap> a := AsQuiverRowsMorphism( A.a, QRowsA );;
gap> b := AsQuiverRowsMorphism( A.b, QRowsA );;
gap> c := AsQuiverRowsMorphism( A.c, QRowsA );;
gap> aa := AsAdelmanCategoryMorphism( a );;
gap> bb := AsAdelmanCategoryMorphism( b );;
gap> cc := AsAdelmanCategoryMorphism( c );;
gap> dd := CokernelProjection( aa );;
gap> ee := CokernelColift( aa, PreCompose( bb, cc ) );;
gap> ff := KernelEmbedding( ee );;
gap> gg := KernelEmbedding( cc );;

```

```

gap> hh := KernelLift( cc, PreCompose( aa, bb ) );;
gap> ii := CokernelProjection( hh );;
gap> fff := AsGeneralizedMorphism( ff );;
gap> ddd := AsGeneralizedMorphism( dd );;
gap> bbb := AsGeneralizedMorphism( bb );;
gap> ggg := AsGeneralizedMorphism( gg );;
gap> iii := AsGeneralizedMorphism( ii );;
gap> p := PreCompose( [ fff, PseudoInverse( ddd ), bbb, PseudoInverse( ggg ), iii ] );;
gap> IsHonest( p );
true
gap> jj := KernelObjectFunctorial( bb, dd, ee );;
gap> kk := CokernelObjectFunctorial( hh, gg, bb );;
gap> pp := HonestRepresentative( p );;
gap> comp := PreCompose( jj, pp );;
gap> IsZero( comp );
true
gap> comp := PreCompose( pp, kk );;
gap> IsZero( comp );
true
gap> homology := function( alpha, beta ) return CokernelObject( LiftAlongMonomorphism( KernelEmbedding( alpha ), beta ) );;
gap> IsZero( homology( jj, pp ) );
true
gap> IsZero( homology( pp, kk ) );
true

```

Phenomena over the integers

Example

```

gap> quiver := RightQuiver( "Q(2)[a:1->2]" );;
gap> vertices := Vertices( quiver );;
gap> B := PathAlgebra( QQ, quiver );;
gap> QRowsB := QuiverRows( B );;
gap> QRowsB_overZ := QuiverRowsDescentToZDefinedByBasisPaths( B );;
gap> a := AsQuiverRowsMorphism( B.a, QRowsB );;
gap> a_Z := AsQuiverRowsMorphism( B.a, QRowsB_overZ );;
gap> aa := AsAdelmanCategoryMorphism( a );;
gap> aa_Z := AsAdelmanCategoryMorphism( a_Z );;
gap> bb := aa + aa;;
gap> bb_Z := aa_Z + aa_Z;;
gap> K1 := KernelEmbedding( bb );;
gap> K2 := KernelEmbedding( aa );;
gap> IsEqualAsSubobjects( K1, K2 );
true
gap> K1_Z := KernelEmbedding( bb_Z );;
gap> K2_Z := KernelEmbedding( aa_Z );;
gap> IsEqualAsSubobjects( K1_Z, K2_Z );
false

```

12.22 Category of relations

Example

```

gap> F := HomalgRingOfIntegers( 3 );;
gap> vec := CategoryOfRows( F );;

```

```

gap> rel := RelCategory( vec );;
gap> A := 1/vec/rel;;
gap> id := IdentityMorphism( A );;
gap> IsWellDefined( id );
true
gap> alpha := HomalgMatrix( "[ 1, 2 ]", 2, 1, F )/vec;;
gap> alpha_rel := alpha/rel;;
gap> alpha_rel_inv := rel/alpha;;
gap> beta := PreCompose( alpha_rel_inv, alpha_rel );;
gap> IsCongruentForMorphisms( beta, id );
true
gap> IsEqualForMorphisms( beta, id );
false
gap> R := HomalgFieldOfRationalsInSingular() * "t";;
gap> t := IndeterminatesOfPolynomialRing( R )[1];;
gap> cocycle := function( a, b, c ) local e; e := CostrictionToImage( UniversalMorphismIntoDirectSum( a, b, c ) ); return e; end;;
gap> T := TwistedLinearClosure( R, rel, cocycle );;
gap> gamma := beta/T;;
gap> delta := ZeroMorphism( 1/vec, 1/vec )/rel/T;;
gap> IsZero( 3*gamma - 3*gamma );
true
gap> IsCongruentForMorphisms( delta, gamma );
false
gap> beta := PreCompose( alpha_rel_inv/T, alpha_rel/T );;
gap> IsZero( beta - t * IdentityMorphism( Range( alpha_rel/T ) ) );
true
gap> IsZero( ( gamma * delta ) * gamma - gamma * ( delta * gamma ) );
true

```

12.23 Rings as Ab-categories

Example

```

gap> CR := RingAsCategory( Integers );;
gap> u := RingAsCategoryUniqueObject( CR );;
gap> alpha := 2 / CR;
<2>
gap> IsOne( alpha );
false
gap> IsZero( alpha );
false
gap> alpha * alpha;
<4>
gap> -alpha;
<-2>
gap> IsZero( alpha + AdditiveInverse( alpha ) );
true
gap> beta := RingAsCategoryMorphism( 1/2, CR );;
gap> IsWellDefined( beta );
false
gap> gamma := IdentityMorphism( u );
<1>
gap> IsOne( gamma );

```

```

true
gap> delta := ZeroMorphism( u, u );
<0>
gap> IsZero( delta );
true

```

12.24 Snake lemma first proof

Example

```

gap> DeactivateDefaultCaching();
gap> SwitchGeneralizedMorphismStandard( "cospan" );;
gap> snake_quiver := RightQuiver( "Q(6)[a:1->2,b:2->3,c:1->4,d:2->5,e:3->6,f:4->5,g:5->6]" );;
gap> kQ := PathAlgebra( HomalgFieldOfRationals(), snake_quiver );;
gap> Aoid := Algebroid( kQ, [ kQ.ad - kQ.cf, kQ.dg - kQ.be, kQ.ab, kQ.fg ] );;
gap> m := SetOfGeneratingMorphisms( Aoid );;
gap> a := m[1];;
gap> b := m[2];;
gap> c := m[3];;
gap> d := m[4];;
gap> e := m[5];;
gap> f := m[6];;
gap> g := m[7];;
gap> cat := Aoid;;
gap> CapCategorySwitchLogicOff( cat );;
gap> DisableInputSanityChecks( cat );;
gap> cat := AdditiveClosure( cat );;
gap> DisableInputSanityChecks( cat );;
gap> cat := Opposite( cat );;
gap> DisableInputSanityChecks( cat );;
gap> CapCategorySwitchLogicOff( cat );;
gap> CapCategorySwitchLogicOff( Opposite( cat ) );;
gap> cat := FreydCategory( cat );;
gap> CapCategorySwitchLogicOff( cat );;
gap> cat := Opposite( cat );;
gap> CapCategorySwitchLogicOff( cat );;
gap> af := AsMorphismInFreeAbelianCategory( m[1] );;
gap> bf := AsMorphismInFreeAbelianCategory( m[2] );;
gap> cf := AsMorphismInFreeAbelianCategory( m[3] );;
gap> df := AsMorphismInFreeAbelianCategory( m[4] );;
gap> ef := AsMorphismInFreeAbelianCategory( m[5] );;
gap> ff := AsMorphismInFreeAbelianCategory( m[6] );;
gap> gf := AsMorphismInFreeAbelianCategory( m[7] );;
gap> bn := CokernelProjection( af );;
gap> en := CokernelColift( af, PreCompose( df, gf ) );;
gap> fn := KernelEmbedding( gf );;
gap> cn := KernelLift( gf, PreCompose( af, df ) );;
gap> ke := KernelEmbedding( en );;
gap> co := CokernelProjection( cn );;
gap> gk := AsGeneralizedMorphism( ke );;
gap> gb := AsGeneralizedMorphism( bn );;
gap> gd := AsGeneralizedMorphism( df );;
gap> gf := AsGeneralizedMorphism( fn );;

```

```

gap> gc := AsGeneralizedMorphism( co );;
gap> DirectSumFunctorial( [ af, af ] );;
gap> IsZero( PreCompose( ke, en ) );;
gap> timestart := Runtimes().user_time;;
gap> p := PreCompose( [ gk, PseudoInverse( gb ) ] );;
gap> p2 := PreCompose( p, gd );;
gap> p3:= PreCompose( p2, PseudoInverse( gf ) );;
gap> p4:= PreCompose( p3, gc );;
gap> IsHonest( p );
false
gap> IsHonest( p2 );
false
gap> IsHonest( p3 );
false
gap> IsHonest( p4 );
true
gap> timeend := Runtimes().user_time - timestart;;
gap> h := HonestRepresentative( p4 );;

```

12.25 Snake lemma second proof

Example

```

gap> DeactivateDefaultCaching();
gap> SwitchGeneralizedMorphismStandard( "cospan" );;
gap> snake_quiver := RightQuiver( "Q(6)[a:1->2,b:2->3,c:3->4]" );;
gap> kQ := PathAlgebra( HomalgFieldOfRationals(), snake_quiver );;
gap> Aoid := Algebroid( kQ, [ kQ.abc ] );;
gap> m := SetOfGeneratingMorphisms( Aoid );;
gap> a := m[1];;
gap> b := m[2];;
gap> c := m[3];;
gap> cat := Aoid;;
gap> CapCategorySwitchLogicOff( cat );;
gap> DisableInputSanityChecks( cat );;
gap> cat := AdditiveClosure( cat );;
gap> DisableInputSanityChecks( cat );;
gap> cat := Opposite( cat );;
gap> DisableInputSanityChecks( cat );;
gap> CapCategorySwitchLogicOff( cat );;
gap> CapCategorySwitchLogicOff( Opposite( cat ) );;
gap> cat := FreydCategory( cat );;
gap> CapCategorySwitchLogicOff( cat );;
gap> cat := Opposite( cat );;
gap> CapCategorySwitchLogicOff( cat );;
gap> a := AsMorphismInFreeAbelianCategory( a );;
gap> b := AsMorphismInFreeAbelianCategory( b );;
gap> c := AsMorphismInFreeAbelianCategory( c );;
gap> coker_a := CokernelProjection( a );;
gap> colift := CokernelColift( a, PreCompose( b, c ) );;
gap> ker_c := KernelEmbedding( c );;
gap> lift := KernelLift( c, PreCompose( a, b ) );;
gap> p := PreCompose( [

```



```

> AsGeneralizedMorphism( KernelEmbedding( colift ) ),
> GeneralizedInverse( coker_a ),
> AsGeneralizedMorphism( b ),
> GeneralizedInverse( ker_c ),
> AsGeneralizedMorphism( CokernelProjection( lift ) )
> ] );;
gap> IsHonest( p );
true

```

12.26 Subobject lattice

We compute the number of the generic subobject lattice generated by 2 independent subobjects y, z and one subobject x of y .

Example

```

gap> ReadPackage( "FreydCategoriesForCAP", "examples/SubobjectLatticeFunctions.g" );;
gap> quiver := RightQuiver( "Q(4)[a:1->2,b:2->3,c:1->4]" );;
gap> QQ := HomalgFieldOfRationals();;
gap> B := PathAlgebra( QQ, quiver );;
gap> RowsB := QuiverRowsDescentToZDefinedByBasisPaths( B : overhead := false );;
gap> Adel := AdelmanCategory( RowsB : overhead := false );;
gap> a := B.a/RowsB/Adel;;
gap> b := B.b/RowsB/Adel;;
gap> c := B.c/RowsB/Adel;;
gap> x := KernelEmbedding( a );;
gap> y := KernelEmbedding( PreCompose( a, b ) );;
gap> z := KernelEmbedding( c );;
gap> gens := [ x, y, z ];;
gap> Size( GenerateSubobjects( gens ) );
8

```

12.27 Adelman category theorem

Example

```

gap> quiver := RightQuiver( "Q(9)[a:1->2,b:3->2]" );;
gap> kQ := PathAlgebra( HomalgFieldOfRationals(), quiver );;
gap> Aoid := Algebroid( kQ );;
gap> mm := SetOfGeneratingMorphisms( Aoid );;
gap> CapCategorySwitchLogicOff( Aoid );;
gap> Acat := AdditiveClosure( Aoid );;
gap> a := AsAdditiveClosureMorphism( mm[1] );;
gap> b := AsAdditiveClosureMorphism( mm[2] );;
gap> a := AsAdelmanCategoryMorphism( a );;
gap> b := AsAdelmanCategoryMorphism( b );;
gap> pi1 := ProjectionInFactorOfFiberProduct( [ a, b ], 1 );;
gap> pi2 := ProjectionInFactorOfFiberProduct( [ a, b ], 1 );;
gap> c := CokernelColift( pi1, PreCompose( a, CokernelProjection( b ) ) );;
gap> IsMonomorphism( c );
true

```

Chapter 13

Linear closure of a category

13.1 Functors

13.1.1 `ExtendFunctorToLinearClosureOfSource` (for `IsCapFunctor`, `IsLinearClosure`, `IsFunction`)

▷ `ExtendFunctorToLinearClosureOfSource(F, linear_closure, ring_map)` (operation)

The arguments are a functor $F: C \rightarrow D$, some linear closure $linear_closure$ of C over some commutative ring S and a function $ring_map$; where D is a linear category over some commutative ring R . The $ring_map$ is a function that converts an element s in S to an element in R , such that $ring_map$ defines a ring homomorphism. The output is the linear extension functor of F from $linear_closure$ to D .

13.1.2 `ExtendFunctorToLinearClosureOfSource` (for `IsCapFunctor`, `IsLinearClosure`)

▷ `ExtendFunctorToLinearClosureOfSource(F, linear_closure)` (operation)

The arguments are a functor $F: C \rightarrow D$, some linear closure $linear_closure$ of C over some commutative ring S ; where D is a linear category over S . The output is the linear extension functor of F from $linear_closure$ to D .

Chapter 14

Examples on graded rows and columns

14.1 Freyd category of graded rows

Example

```
gap> Q := HomalgFieldOfRationalsInSingular();
Q
gap> S := GradedRing( Q * "x_1, x_2, x_3, x_4" );
Q[x_1,x_2,x_3,x_4]
(weights: yet unset)
gap> SetWeightsOfIndeterminates( S, [[1,0],[1,0],[0,1],[0,1]] );

gap> cat := CategoryOfGradedRows( S );
Category of graded rows over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])
gap> obj1 := GradedRow( [ [[1,1],1] ], S );
<A graded row of rank 1>
gap> obj2 := GradedRow( [ [[1,1],2] ], S );
<A graded row of rank 2>
gap> gamma := GradedRowOrColumnMorphism( obj2,
> HomalgMatrix( [ [ 1, 1 ], [ 1, 1 ] ], 2, 2, S ), obj2 );
<A morphism in Category of graded rows over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> freyd := FreydCategory( cat );
Category of f.p. graded left modules over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])
gap> IsAbelianCategory( freyd );
true
gap> obj_gamma := FreydCategoryObject( gamma );
<An object in Category of f.p. graded left modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> f := FreydCategoryMorphism( obj_gamma, gamma, obj_gamma );
<A morphism in Category of f.p. graded left modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> witness := MorphismWitness( f );
<A morphism in Category of graded rows over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
```

Example

```
gap> Display( witness );
A morphism in Category of graded rows over Q[x_1,x_2,x_3,x_4]
```

```
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])
```

Source:

A graded row over $Q[x_1, x_2, x_3, x_4]$ (with weights
[[1, 0], [1, 0], [0, 1], [0, 1]]) of rank 2 and degrees:
[[(1, 1), 2]]

Matrix:

2,0,
2,0
(over a graded ring)

Range:

A graded row over $Q[x_1, x_2, x_3, x_4]$ (with weights
[[1, 0], [1, 0], [0, 1], [0, 1]]) of rank 2 and degrees:
[[(1, 1), 2]]

Example

```
gap> g := FreydCategoryMorphism( obj_gamma,
>                               ZeroMorphism( obj2, obj2 ), obj_gamma );
<A morphism in Category of f.p. graded left modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsCongruentForMorphisms( f, g );
true
gap> c := PreCompose( f, f );
<A morphism in Category of f.p. graded left modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
```

Example

```
gap> Display( c );
A morphism in Category of f.p. graded left modules over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ] )

-----

Source:
A morphism in Category of graded rows over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ] )

Source:
A graded row over Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ] )
of rank 2 and degrees:
[ [ ( 1, 1 ), 2 ] ]

Matrix:
1,1,
1,1
(over a graded ring)

Range:
A graded row over Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ] )
of rank 2 and degrees:
[ [ ( 1, 1 ), 2 ] ]
```

```

-----

Morphism datum:
A morphism in Category of graded rows over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])

Source:
A graded row over Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])
of rank 2 and degrees:
[ [ ( 1, 1 ), 2 ] ]

Matrix:
2,2,
2,2
(over a graded ring)

Range:
A graded row over Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])
of rank 2 and degrees:
[ [ ( 1, 1 ), 2 ] ]

-----

Range:
A morphism in Category of graded row over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])

Source:
A graded row over Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])
of rank 2 and degrees:
[ [ ( 1, 1 ), 2 ] ]

Matrix:
1,1,
1,1
(over a graded ring)

Range:
A graded row over Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])
of rank 2 and degrees:
[ [ ( 1, 1 ), 2 ] ]

-----

```

Example

```

gap> s := g + g;
<A morphism in Category of f.p. graded left modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> a := GradedRowOrColumnMorphism( obj1,
>                                     HomalgMatrix( [ [ 2 ] ], 1, 1, S ), obj1 );
<A morphism in Category of graded rows over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>

```

Example

```
gap> Display( a );
A morphism in Category of graded rows over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])

Source:
A graded row over Q[x_1,x_2,x_3,x_4] (with weights
[ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ]) of rank 1 and degrees:
[ [ ( 1, 1 ), 1 ] ]

Matrix:
2
(over a graded ring)

Range:
A graded row over Q[x_1,x_2,x_3,x_4] (with weights
[ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ]) of rank 1 and degrees:
[ [ ( 1, 1 ), 1 ] ]
```

Example

```
gap> Z2 := FreydCategoryObject( a );
<An object in Category of f.p. graded left modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
```

Example

```
gap> Display( Z2 );
An object in Freyd( Category of graded rows over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ] ) )

Relation morphism:
A morphism in Category of graded rows over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])

Source:
A graded row over Q[x_1,x_2,x_3,x_4] (with weights
[ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ]) of rank 1 and degrees:
[ [ ( 1, 1 ), 1 ] ]

Matrix:
2
(over a graded ring)

Range:
A graded row over Q[x_1,x_2,x_3,x_4] (with weights
[ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ]) of rank 1 and degrees:
[ [ ( 1, 1 ), 1 ] ]
```

Example

```
gap> id := IdentityMorphism( Z2 );
<An identity morphism in Category of f.p. graded left modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> z := id + id + id;
<A morphism in Category of f.p. graded left modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
```

```

gap> d := DirectSumFunctorial( [ z, z, z ] );
<A morphism in Category of f.p. graded left modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> pr2 := ProjectionInFactorOfDirectSum( [ Z2, Z2, Z2 ], 2 );
<A morphism in Category of f.p. graded left modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> pr3 := ProjectionInFactorOfDirectSum( [ Z2, Z2, Z2 ], 3 );
<A morphism in Category of f.p. graded left modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> uni := UniversalMorphismIntoDirectSum( [ pr3, pr2 ] );
<A morphism in Category of f.p. graded left modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> inj1 := InjectionOfCofactorOfDirectSum( [ Z2, Z2, Z2 ], 1 );
<A morphism in Category of f.p. graded left modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> inj2 := InjectionOfCofactorOfDirectSum( [ Z2, Z2, Z2 ], 2 );
<A morphism in Category of f.p. graded left modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> uni2 := UniversalMorphismFromDirectSum( [ inj2, inj1 ] );
<A morphism in Category of f.p. graded left modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> ZFree := AsFreydCategoryObject( obj1 );
<A projective object in Category of f.p. graded left modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>

```

Example

```

gap> Display( ZFree );
A projective object in Freyd( Category of graded rows over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ] ) )

Relation morphism:
A morphism in Category of graded rows over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ] )

Source:
A graded row over Q[x_1,x_2,x_3,x_4] (with weights
[ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ] ) of rank 0 and degrees:
[ ]

Matrix:
(an empty 0 x 1 matrix)

Range:
A graded row over Q[x_1,x_2,x_3,x_4] (with weights
[ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ] ) of rank 1 and degrees:
[ [ ( 1, 1 ), 1 ] ]

```

Example

```

gap> id := IdentityMorphism( ZFree );
<An identity morphism in Category of f.p. graded left modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> z := id + id;
<A morphism in Category of f.p. graded left modules over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>

```

```

gap> coker_proj := CokernelProjection( z );
<An epimorphism in Category of f.p. graded left modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> cokernel_colift := CokernelColift( z, CokernelProjection( z ) );
<A morphism in Category of f.p. graded left modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> a := ZFree;
<A projective object in Category of f.p. graded left modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> b := obj_gamma;
<An object in Category of f.p. graded left modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> c := TensorProductOnObjects( ZFree, obj_gamma );
<An object in Category of f.p. graded left modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> KaxbKxc := TensorProductOnObjects( TensorProductOnObjects( a, b ), c );
<An object in Category of f.p. graded left modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsEqualForObjects( KaxbKxc, ZeroObject( freyd ) );
false
gap> tensor_product_morphism := TensorProductOnMorphisms( cokernel_colift, coker_proj );
<A morphism in Category of f.p. graded left modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsEpimorphism( tensor_product_morphism );
true
gap> IsEqualForObjects( Source( tensor_product_morphism ), Range( tensor_product_morphism ) );
false
gap> unit := TensorUnit( freyd );
<An object in Category of f.p. graded left modules over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsEqualForObjects( TensorProductOnObjects( a, unit ), a );
true
gap> axKbxcK := TensorProductOnObjects( a, TensorProductOnObjects( b, c ) );
<An object in Category of f.p. graded left modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> ass_left_to_right := AssociatorLeftToRightWithGivenTensorProducts( KaxbKxc, a, b, c, axKbxcK );
<A morphism in Category of f.p. graded left modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsIsomorphism( ass_left_to_right );
true
gap> ass_right_to_left := AssociatorLeftToRightWithGivenTensorProducts( axKbxcK, a, b, c, KaxbKxc );
<A morphism in Category of f.p. graded left modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsMonomorphism( ass_right_to_left );
true
gap> IsEpimorphism( ass_right_to_left );
true
gap> LeftUnitor( a );
<A morphism in Category of f.p. graded left modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> LeftUnitorInverse( axKbxcK );
<A morphism in Category of f.p. graded left modules over

```



```

Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> RightUnitor( b );
<A morphism in Category of f.p. graded left modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> RightUnitorInverse( TensorProductOnObjects( axKbxcK, axKbxcK ) );
<A morphism in Category of f.p. graded left modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> Braiding( axKbxcK, KaxbKxc );
<A morphism in Category of f.p. graded left modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> braiding := Braiding( a, b );
<A morphism in Category of f.p. graded left modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( braiding );
true
gap> hom := InternalHomOnObjects( axKbxcK, axKbxcK );
<An object in Category of f.p. graded left modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsZero( hom );
false
gap> free_mod1 := AsFreydCategoryObject( GradedRow( [ [[0,0],1] ], S ) );
<A projective object in Category of f.p. graded left modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> free_mod2 := AsFreydCategoryObject( GradedRow( [ [[1,1],1] ], S ) );
<A projective object in Category of f.p. graded left modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> hom2 := InternalHomOnObjects( free_mod1, free_mod2 );
<An object in Category of f.p. graded left modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsZero( hom2 );
false
gap> IsZero( Source( RelationMorphism( hom2 ) ) );
true
gap> Rank( Range( RelationMorphism( hom2 ) ) );
1
gap> hom3 := InternalHomOnObjects( free_mod2, free_mod1 );
<An object in Category of f.p. graded left modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsZero( hom3 );
false
gap> InternalHomOnMorphisms( ass_left_to_right, ass_right_to_left );
<A morphism in Category of f.p. graded left modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> eval := EvaluationMorphism( a, b );
<A morphism in Category of f.p. graded left modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsEpimorphism( eval );
true
gap> IsMonomorphism( eval );
true
gap> coeval := CoevaluationMorphism( a, b );
<A morphism in Category of f.p. graded left modules over

```

```

Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsEpimorphism( coeval );
true
gap> IsMonomorphism( coeval );
true

```

14.2 Freyd category of graded columns

Example

```

gap> Q := HomalgFieldOfRationalsInSingular();
Q
gap> S := GradedRing( Q * "x_1, x_2, x_3, x_4" );
Q[x_1,x_2,x_3,x_4]
(weights: yet unset)
gap> SetWeightsOfIndeterminates( S, [[1,0],[1,0],[0,1],[0,1]] );

gap> cat := CategoryOfGradedColumns( S );
Category of graded columns over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])
gap> obj1 := GradedColumn( [ [[1,1],1] ], S );
<A graded column of rank 1>
gap> obj2 := GradedColumn( [ [[1,1],2] ], S );
<A graded column of rank 2>
gap> gamma := GradedRowOrColumnMorphism( obj2,
> HomalgMatrix( [ [ 1, 1 ], [ 1, 1 ] ], 2, 2, S ), obj2 );
<A morphism in Category of graded columns over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> freyd := FreydCategory( cat );
Category of f.p. graded right modules over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])
gap> IsAbelianCategory( freyd );
true
gap> obj_gamma := FreydCategoryObject( gamma );
<An object in Category of f.p. graded right modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> f := FreydCategoryMorphism( obj_gamma, gamma, obj_gamma );
<A morphism in Category of f.p. graded right modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> witness := MorphismWitness( f );
<A morphism in Category of graded columns over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>

```

Example

```

gap> Display( witness );
A morphism in Category of graded columns over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])

Source:
A graded column over Q[x_1,x_2,x_3,x_4] (with weights
[ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ]) of rank 2 and degrees:
[ [ ( 1, 1 ), 2 ] ]

Matrix:

```

```
2,2,
0,0
(over a graded ring)
```

Range:

A graded column over $Q[x_1, x_2, x_3, x_4]$ (with weights
 $\begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$) of rank 2 and degrees:
 $\begin{bmatrix} (1, 1) & 2 \end{bmatrix}$

Example

```
gap> g := FreydCategoryMorphism( obj_gamma,
>                               ZeroMorphism( obj2, obj2 ), obj_gamma );
<A morphism in Category of f.p. graded right modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsCongruentForMorphisms( f, g );
true
gap> c := PreCompose( f, f );
<A morphism in Category of f.p. graded right modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
```

Example

```
gap> Display( c );
A morphism in Category of f.p. graded right modules over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ] )

-----

Source:
A morphism in Category of graded columns over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ] )

Source:
A graded column over Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ] )
of rank 2 and degrees:
[ [ ( 1, 1 ), 2 ] ]

Matrix:
1,1,
1,1
(over a graded ring)

Range:
A graded column over Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ] )
of rank 2 and degrees:
[ [ ( 1, 1 ), 2 ] ]

-----

Morphism datum:
A morphism in Category of graded columns over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ] )

Source:
A graded column over Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ] )
```

```
of rank 2 and degrees:
[ [ ( 1, 1 ), 2 ] ]
```

```
Matrix:
2,2,
2,2
(over a graded ring)
```

```
Range:
A graded column over Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])
of rank 2 and degrees:
[ [ ( 1, 1 ), 2 ] ]
```

```
-----
```

```
Range:
A morphism in Category of graded columns over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])
```

```
Source:
A graded column over Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])
of rank 2 and degrees:
[ [ ( 1, 1 ), 2 ] ]
```

```
Matrix:
1,1,
1,1
(over a graded ring)
```

```
Range:
A graded column over Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])
of rank 2 and degrees:
[ [ ( 1, 1 ), 2 ] ]
```

```
-----
```

Example

```
gap> s := g + g;
<A morphism in Category of f.p. graded right modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> a := GradedRowOrColumnMorphism( obj1,
>                                     HomalgMatrix( [ [ 2 ] ], 1, 1, S ), obj1 );
<A morphism in Category of graded columns over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
```

Example

```
gap> Display( a );
A morphism in Category of graded columns over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])
```

Source:

```
A graded column over Q[x_1,x_2,x_3,x_4] (with weights
[ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ]) of rank 1 and degrees:
[ [ ( 1, 1 ), 1 ] ]
```

Matrix:

2

(over a graded ring)

Range:

A graded column over $Q[x_1, x_2, x_3, x_4]$

(with weights $\begin{bmatrix} 1, 0 \\ 1, 0 \\ 0, 1 \\ 0, 1 \end{bmatrix}$) of rank 1 and degrees:
 $\begin{bmatrix} (1, 1), 1 \end{bmatrix}$

Example

```
gap> Z2 := FreydCategoryObject( a );
<An object in Category of f.p. graded right modules over
Q[x_1,x_2,x_3,x_4] (with weights  $\begin{bmatrix} 1, 0 \\ 1, 0 \\ 0, 1 \\ 0, 1 \end{bmatrix}$ )>
```

Example

```
gap> Display( Z2 );
An object in Freyd( Category of graded columns over
Q[x_1,x_2,x_3,x_4] (with weights  $\begin{bmatrix} 1, 0 \\ 1, 0 \\ 0, 1 \\ 0, 1 \end{bmatrix}$ ) )
```

Relation morphism:

A morphism in Category of graded columns over $Q[x_1, x_2, x_3, x_4]$
(with weights $\begin{bmatrix} 1, 0 \\ 1, 0 \\ 0, 1 \\ 0, 1 \end{bmatrix}$)

Source:

A graded column over $Q[x_1, x_2, x_3, x_4]$ (with weights
 $\begin{bmatrix} 1, 0 \\ 1, 0 \\ 0, 1 \\ 0, 1 \end{bmatrix}$) of rank 1 and degrees:
 $\begin{bmatrix} (1, 1), 1 \end{bmatrix}$

Matrix:

2

(over a graded ring)

Range:

A graded column over $Q[x_1, x_2, x_3, x_4]$ (with weights
 $\begin{bmatrix} 1, 0 \\ 1, 0 \\ 0, 1 \\ 0, 1 \end{bmatrix}$) of rank 1 and degrees:
 $\begin{bmatrix} (1, 1), 1 \end{bmatrix}$

Example

```
gap> id := IdentityMorphism( Z2 );
<An identity morphism in Category of f.p. graded right modules over
Q[x_1,x_2,x_3,x_4] (with weights  $\begin{bmatrix} 1, 0 \\ 1, 0 \\ 0, 1 \\ 0, 1 \end{bmatrix}$ )>
gap> z := id + id + id;
<A morphism in Category of f.p. graded right modules over
Q[x_1,x_2,x_3,x_4] (with weights  $\begin{bmatrix} 1, 0 \\ 1, 0 \\ 0, 1 \\ 0, 1 \end{bmatrix}$ )>
gap> d := DirectSumFunctorial( [ z, z, z ] );
<A morphism in Category of f.p. graded right modules over
Q[x_1,x_2,x_3,x_4] (with weights  $\begin{bmatrix} 1, 0 \\ 1, 0 \\ 0, 1 \\ 0, 1 \end{bmatrix}$ )>
gap> pr2 := ProjectionInFactorOfDirectSum( [ Z2, Z2, Z2 ], 2 );
<A morphism in Category of f.p. graded right modules over
Q[x_1,x_2,x_3,x_4] (with weights  $\begin{bmatrix} 1, 0 \\ 1, 0 \\ 0, 1 \\ 0, 1 \end{bmatrix}$ )>
gap> pr3 := ProjectionInFactorOfDirectSum( [ Z2, Z2, Z2 ], 3 );
<A morphism in Category of f.p. graded right modules over
Q[x_1,x_2,x_3,x_4] (with weights  $\begin{bmatrix} 1, 0 \\ 1, 0 \\ 0, 1 \\ 0, 1 \end{bmatrix}$ )>
```

```

gap> uni := UniversalMorphismIntoDirectSum( [ pr3, pr2 ] );
<A morphism in Category of f.p. graded right modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> inj1 := InjectionOfCofactorOfDirectSum( [ Z2, Z2, Z2 ], 1 );
<A morphism in Category of f.p. graded right modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> inj2 := InjectionOfCofactorOfDirectSum( [ Z2, Z2, Z2 ], 2 );
<A morphism in Category of f.p. graded right modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> uni2 := UniversalMorphismFromDirectSum( [ inj2, inj1 ] );
<A morphism in Category of f.p. graded right modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> ZFree := AsFreydCategoryObject( obj1 );
<A projective object in Category of f.p. graded right modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>

```

Example

```

gap> Display( ZFree );
A projective object in Category of f.p. graded right modules over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ] )

Relation morphism:
A morphism in Category of graded columns over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ] )

Source:
A graded column over Q[x_1,x_2,x_3,x_4] (with weights
[ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ] ) of rank 0 and degrees:
[ ]

Matrix:
(an empty 1 x 0 matrix)

Range:
A graded column over Q[x_1,x_2,x_3,x_4] (with weights
[ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ] ) of rank 1 and degrees:
[ [ ( 1, 1 ), 1 ] ]

```

Example

```

gap> id := IdentityMorphism( ZFree );
<An identity morphism in Category of f.p. graded right modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> z := id + id;
<A morphism in Category of f.p. graded right modules over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> coker_proj := CokernelProjection( z );
<An epimorphism in Category of f.p. graded right modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> cokernel_colift := CokernelColift( z, CokernelProjection( z ) );
<A morphism in Category of f.p. graded right modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> a := ZFree;
<A projective object in Category of f.p. graded right modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>

```

```

gap> b := obj_gamma;
<An object in Category of f.p. graded right modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> c := TensorProductOnObjects( a, b );
<An object in Category of f.p. graded right modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> KaxbKxc := TensorProductOnObjects( TensorProductOnObjects( a, b ), c );
<An object in Category of f.p. graded right modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsEqualForObjects( KaxbKxc, ZeroObject( freyd ) );
false
gap> tensor_product_morphism := TensorProductOnMorphisms( cokernel_colift, coker_proj );
<A morphism in Category of f.p. graded right modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsEpimorphism( tensor_product_morphism );
true
gap> IsEqualForObjects( Source( tensor_product_morphism ), Range( tensor_product_morphism ) );
false
gap> unit := TensorUnit( freyd );
<An object in Category of f.p. graded right modules over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsEqualForObjects( TensorProductOnObjects( a, unit ), a );
true
gap> axKbxcK := TensorProductOnObjects( a, TensorProductOnObjects( b, c ) );
<An object in Category of f.p. graded right modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> ass_left_to_right := AssociatorLeftToRightWithGivenTensorProducts( KaxbKxc, a, b, c, axKbxcK );
<A morphism in Category of f.p. graded right modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsIsomorphism( ass_left_to_right );
true
gap> ass_right_to_left := AssociatorLeftToRightWithGivenTensorProducts( axKbxcK, a, b, c, KaxbKxc );
<A morphism in Category of f.p. graded right modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsMonomorphism( ass_right_to_left );
true
gap> IsEpimorphism( ass_right_to_left );
true
gap> LeftUnitor( a );
<A morphism in Category of f.p. graded right modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> LeftUnitorInverse( axKbxcK );
<A morphism in Category of f.p. graded right modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> RightUnitor( b );
<A morphism in Category of f.p. graded right modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> RightUnitorInverse( TensorProductOnObjects( axKbxcK, axKbxcK ) );
<A morphism in Category of f.p. graded right modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> Braiding( axKbxcK, KaxbKxc );
<A morphism in Category of f.p. graded right modules over

```

```

Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> braiding := Braiding( a, b );
<A morphism in Category of f.p. graded right modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( braiding );
true
gap> hom := InternalHomOnObjects( axKbxcK, axKbxcK );
<An object in Category of f.p. graded right modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsZero( hom );
false
gap> free_mod1 := AsFreydCategoryObject( GradedColumn( [ [0,0],1] ], S );
<A projective object in Category of f.p. graded right modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> free_mod2 := AsFreydCategoryObject( GradedColumn( [ [1,1],1] ], S );
<A projective object in Category of f.p. graded right modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> hom2 := InternalHomOnObjects( free_mod1, free_mod2 );
<An object in Category of f.p. graded right modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsZero( hom2 );
false
gap> IsZero( Source( RelationMorphism( hom2 ) ) );
true
gap> Rank( Range( RelationMorphism( hom2 ) ) );
1
gap> hom3 := InternalHomOnObjects( free_mod2, free_mod1 );
<An object in Category of f.p. graded right modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsZero( hom3 );
false
gap> InternalHomOnMorphisms( ass_left_to_right, ass_right_to_left );
<A morphism in Category of f.p. graded right modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> eval := EvaluationMorphism( a, b );
<A morphism in Category of f.p. graded right modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsEpimorphism( eval );
true
gap> IsMonomorphism( eval );
true
gap> coeval := CoevaluationMorphism( a, b );
<A morphism in Category of f.p. graded right modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsEpimorphism( coeval );
true
gap> IsMonomorphism( coeval );
true

```


14.3 Constructors of objects and reduction of degree lists

Example

```
gap> Q := HomalgFieldOfRationalsInSingular();
Q
gap> S := GradedRing( Q * "x_1, x_2, x_3, x_4" );
Q[x_1,x_2,x_3,x_4]
(weights: yet unset)
gap> SetWeightsOfIndeterminates( S, [[1,0],[1,0],[0,1],[0,1]] );

gap> ObjectL := GradedRow( [ [[1,0],2] ], S );
<A graded row of rank 2>
gap> DegreeList( ObjectL );
[ [ ( 1, 0 ), 2 ] ]
gap> Object2L := GradedRow( [ [[1,0],2],
> [[1,0],3],[[0,1],2],[[1,0],1] ], S );
<A graded row of rank 8>
gap> DegreeList( Object2L );
[ [ ( 1, 0 ), 5 ], [ ( 0, 1 ), 2 ], [ ( 1, 0 ), 1 ] ]
gap> UnzipDegreeList( Object2L );
[ ( 1, 0 ), ( 1, 0 ), ( 1, 0 ), ( 1, 0 ), ( 1, 0 ), ( 0, 1 ), ( 0, 1 ), ( 1, 0 ) ]
gap> ObjectR := GradedColumn( [ [[1,0],2] ], S );
<A graded column of rank 2>
gap> DegreeList( ObjectR );
[ [ ( 1, 0 ), 2 ] ]
gap> Object2R := GradedColumn( [ [[1,0],2],
> [[1,0],3],[[0,1],2],[[1,0],1] ], S );
<A graded column of rank 8>
gap> DegreeList( Object2R );
[ [ ( 1, 0 ), 5 ], [ ( 0, 1 ), 2 ], [ ( 1, 0 ), 1 ] ]
gap> UnzipDegreeList( Object2R );
[ ( 1, 0 ), ( 1, 0 ), ( 1, 0 ), ( 1, 0 ), ( 1, 0 ), ( 0, 1 ), ( 0, 1 ), ( 1, 0 ) ]
gap> S2 := GradedRing( Q * "x" );
gap> SetWeightsOfIndeterminates( S2, [ 1 ] );
gap> IsWellDefined( GradedRow( [ [ [ 1 ], 1 ] ], S2 ) );
true
gap> IsWellDefined( GradedColumn( [ [ [ 1 ], 1 ] ], S2 ) );
true
```

Whenever the object constructor is called, it tries to simplify the given degree list. To this end it checks if subsequent degree group elements match. If so, their multiplicities are added. So, as in the example above we have:

$$[(1,0),2],[(1,0),3],[(0,1),2],[(1,0),1] \mapsto [(1,0),5],[(0,1),2],[(1,0),1]$$

Note that, even though there are two occurrences of $(1,0)$ in the final degree list, we do not simplify further. The reason for this is as follows. Assume that we have a map of graded rows

$$\varphi: A \rightarrow B$$

given by a homogeneous matrix M and that we want to compute the weak kernel embedding of this mapping. To this end we first compute the row syzygies of M . Let us call the corresponding matrix N .

Then we deduce the degree list of the weak kernel object from N and from the graded row A . Once this degree list is known, we would call the object constructor. If this object constructor summarised all (and not only subsequent) occurrences of one degree element in the degree list, then in order to make sure that the weak kernel embedding is a mapping of graded rows, the rows of the matrix N would have to be shuffled. The latter we do not wish to perform.

Note that the 'IsEqualForObjects' methods returns true whenever the degree lists of two graded rows/columns are identical. So in particular it returns false, if the degree lists are mere permutations of one another. Here is an example.

Example

```
gap> Object2LShuffle := GradedRow( [ [[0,1],1],
>      [[1,0],2],[[0,1],1],[[1,0],4] ], S );
<A graded row of rank 8>
gap> IsEqualForObjects( Object2L, Object2LShuffle );
false
gap> Object2RShuffle := GradedColumn( [ [[0,1],1],
>      [[1,0],2],[[0,1],1],[[1,0],4] ], S );
<A graded column of rank 8>
gap> IsEqualForObjects( Object2R, Object2RShuffle );
false
```

14.4 Constructors of morphisms

Example

```
gap> Q1L := GradedRow( [ [[0,0],1] ], S );
<A graded row of rank 1>
gap> IsWellDefined( Q1L );
true
gap> Q2L := GradedRow( [ [[1,0],2] ], S );
<A graded row of rank 2>
gap> m1L := GradedRowOrColumnMorphism(
>      Q1L, HomalgMatrix( ["x_1","x_2"], S ), Q2L );
<A morphism in Category of graded rows over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( m1L );
true
```

Example

```
gap> Display( Source( m1L ) );
A graded row over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ]) of rank 1 and degrees:
[ [ 0, 1 ] ]
gap> Display( Range( m1L ) );
A graded row over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ]) of rank 2 and degrees:
[ [ ( 1, 0 ), 2 ] ]
gap> Display( UnderlyingHomalgMatrix( m1L ) );
x_1,x_2
(over a graded ring)
```

Example

```
gap> Q1R := GradedColumn( [ [[0,0],1] ], S );
<A graded column of rank 1>
```

```

gap> IsWellDefined( Q1R );
true
gap> Q2R := GradedColumn( [ [[1,0],2] ], S );
<A graded column of rank 2>
gap> m1R := GradedRowOrColumnMorphism(
>      Q1R, HomalgMatrix( ["x_1"],["x_2"], S ), Q2R );
<A morphism in Category of graded columns over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( m1R );
true

```

Example

```

gap> Display( Source( m1R ) );
A graded column over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ]) of rank 1 and degrees:
[ [ 0, 1 ] ]
gap> Display( Range( m1R ) );
A graded column over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ]) of rank 2 and degrees:
[ [ ( 1, 0 ), 2 ] ]
gap> Display( UnderlyingHomalgMatrix( m1R ) );
x_1,
x_2
(over a graded ring)

```

14.5 The GAP categories

Example

```

gap> categoryL := CapCategory( Q1L );
Category of graded rows over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])
gap> categoryR := CapCategory( Q1R );
Category of graded columns over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])

```

14.6 A few categorical constructions for graded rows

Example

```

gap> ZeroObject( categoryL );
<A graded row of rank 0>
gap> 01L := GradedRow( [ [-1,0],2] ], S );
<A graded row of rank 2>

```

Example

```

gap> Display( ZeroMorphism( ZeroObject( categoryL ), 01L ) );
A morphism in Category of graded rows over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])

Source:
A graded row over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])
of rank 0 and degrees:

```

[]

Matrix:

(an empty 0 x 2 matrix)

Range:

A graded row over $Q[x_1, x_2, x_3, x_4]$

(with weights [[1, 0], [1, 0], [0, 1], [0, 1]])

of rank 2 and degrees:

[[(-1, 0), 2]]

Example

```
gap> O2L := GradedRow( [ [[0,0],1] ], S );
<A graded row of rank 1>
gap> obj3L := GradedRow( [ [[-1,0],1] ], S );
<A graded row of rank 1>
```

Example

```
gap> Display( IdentityMorphism( O2L ) );
A morphism in Category of graded rows over  $Q[x_1, x_2, x_3, x_4]$ 
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])
```

Source:

A graded row over $Q[x_1, x_2, x_3, x_4]$

(with weights [[1, 0], [1, 0], [0, 1], [0, 1]])

of rank 1 and degrees:

[[0, 1]]

Matrix:

1

(over a graded ring)

Range:

A graded row over $Q[x_1, x_2, x_3, x_4]$

(with weights [[1, 0], [1, 0], [0, 1], [0, 1]])

of rank 1 and degrees:

[[0, 1]]

Example

```
gap> IsWellDefined( IdentityMorphism( Q2L ) );
true
gap> directSumL := DirectSum( [ O1L, O2L ] );
<A graded row of rank 3>
```

Example

```
gap> Display( directSumL );
A graded row over  $Q[x_1, x_2, x_3, x_4]$ 
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ]) of rank 3 and degrees:
[ [ ( -1, 0 ), 2 ], [ 0, 1 ] ]
```

Example

```
gap> i1L := InjectionOfCofactorOfDirectSum( [ O1L, O2L ], 1 );
<A morphism in Category of graded rows over
 $Q[x_1, x_2, x_3, x_4]$  (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( i1L );
true
```

Example

```
gap> Display( UnderlyingHomalgMatrix( i1L ) );
1,0,0,
0,1,0
(over a graded ring)
```

Example

```
gap> i2L := InjectionOfCofactorOfDirectSum( [ 01L, 02L ], 2 );
<A morphism in Category of graded rows over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( i2L );
true
```

Example

```
gap> Display( UnderlyingHomalgMatrix( i2L ) );
0,0,1
(over a graded ring)
```

Example

```
gap> proj1L := ProjectionInFactorOfDirectSum( [ 01L, 02L ], 1 );
<A morphism in Category of graded rows over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( proj1L );
true
```

Example

```
gap> Display( UnderlyingHomalgMatrix( proj1L ) );
1,0,
0,1,
0,0
(over a graded ring)
```

Example

```
gap> proj2L := ProjectionInFactorOfDirectSum( [ 01L, 02L ], 2 );
<A morphism in Category of graded rows over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( proj2L );
true
```

Example

```
gap> Display( UnderlyingHomalgMatrix( proj2L ) );
0,
0,
1
(over a graded ring)
```

Example

```
gap> kL := WeakKernelEmbedding( proj1L );
<A morphism in Category of graded rows over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( kL );
true
```

Example

```
gap> Display( UnderlyingHomalgMatrix( kL ) );
0,0,1
(over a graded ring)
```

Example

```
gap> ckL := WeakCokernelProjection( kL );
<A morphism in Category of graded rows over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( ckL );
true
```

Example

```
gap> Display( UnderlyingHomalgMatrix( ckL ) );
1,0,
0,1,
0,0
(over a graded ring)
```

Example

```
gap> IsMonomorphism( kL );
true
gap> IsEpimorphism( kL );
false
gap> IsMonomorphism( ckL );
false
gap> IsEpimorphism( ckL );
true
gap> m1L := GradedRowOrColumnMorphism( 01L,
> HomalgMatrix( [[ "x_1" ], [ "x_2" ] ], S ), 02L );
<A morphism in Category of graded rows over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( m1L );
true
gap> m2L := IdentityMorphism( 02L );
<A morphism in Category of graded rows over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( m2L );
true
gap> obj1L := GradedRow( [ [0,0],1], [[-1,0],1], S );
<A graded row of rank 2>
gap> m1L := GradedRowOrColumnMorphism( obj1L,
> HomalgMatrix( [[ 1 ], [ "x_2" ] ], S ), 02L );
<A morphism in Category of graded rows over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( m1L );
true
gap> m3L := GradedRowOrColumnMorphism( obj3L,
> HomalgMatrix( [[ "x_1" ] ], S ), 02L );
<A morphism in Category of graded rows over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( m3L );
true
gap> liftL := Lift( m3L, m1L );
<A morphism in Category of graded rows over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( liftL );
true
```

Example

```
gap> Display( UnderlyingHomalgMatrix( liftL ) );
x_1, 0
(over a graded ring)
```

Example

```
gap> O3L := GradedRow( [ [1,0],2 ] , S );
<A graded row of rank 2>
gap> morL := GradedRowOrColumnMorphism(
>      O2L, HomalgMatrix( [ [ "x_1, x_2" ] ], S ), O3L );
<A morphism in Category of graded rows over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( morL );
true
gap> coliftL := Colift( m2L, morL );
<A morphism in Category of graded rows over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( coliftL );
true
```

Example

```
gap> Display( UnderlyingHomalgMatrix( coliftL ) );
x_1,x_2
(over a graded ring)
```

Example

```
gap> fpL := WeakBiFiberProduct( m1L, m2L );
<A graded row of rank 2>
gap> fp_proj1L := ProjectionInFirstFactorOfWeakBiFiberProduct( m1L, m2L );
<A morphism in Category of graded rows over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( fp_proj1L );
true
```

Example

```
gap> Display( UnderlyingHomalgMatrix( fp_proj1L ) );
1,0,
0,1
(over a graded ring)
```

Example

```
gap> fp_proj2L := ProjectionInSecondFactorOfWeakBiFiberProduct( m1L, m2L );
<A morphism in Category of graded rows over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( fp_proj2L );
true
```

Example

```
gap> Display( UnderlyingHomalgMatrix( fp_proj2L ) );
1,
x_2
(over a graded ring)
```

Example

```
gap> BiasedWeakFiberProduct( m1L, m2L );
<A graded row of rank 2>
```

```
gap> pbwfprow := ProjectionOfBiasedWeakFiberProduct( m1L, m2L );
<A morphism in Category of graded rows over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( pbwfprow );
true
```

Example

```
gap> Display( pbwfprow );
A morphism in Category of graded rows over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])

Source:
A graded row over Q[x_1,x_2,x_3,x_4] (with weights
[ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])
of rank 2 and degrees:
[ [ 0, 1 ], [ ( -1, 0 ), 1 ] ]

Matrix:
1,0,
0,1
(over a graded ring)

Range:
A graded row over Q[x_1,x_2,x_3,x_4] (with weights
[ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])
of rank 2 and degrees:
[ [ 0, 1 ], [ ( -1, 0 ), 1 ] ]
```

Example

```
gap> poL := WeakBiPushout( morL, m2L );
<A graded row of rank 2>
gap> inj1L := InjectionOfFirstCofactorOfWeakBiPushout( morL, m2L );
<A morphism in Category of graded rows over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( inj1L );
true
```

Example

```
gap> Display( UnderlyingHomalgMatrix( inj1L ) );
1,0,
0,1
(over a graded ring)
```

Example

```
gap> inj2L := InjectionOfSecondCofactorOfWeakBiPushout( morL, m2L );
<A morphism in Category of graded rows over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( inj2L );
true
```

Example

```
gap> Display( UnderlyingHomalgMatrix( inj2L ) );
x_1,x_2
(over a graded ring)
```


Example

```
gap> injectionL := InjectionOfBiasedWeakPushout( morL, m2L );
<A morphism in Category of graded rows over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( injectionL );
true
```

Example

```
gap> Display( injectionL );
A morphism in Category of graded rows over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])

Source:
A graded row over Q[x_1,x_2,x_3,x_4] (with weights
[ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ]) of rank 2 and degrees:
[ [ ( 1, 0 ), 2 ] ]

Matrix:
1,0,
0,1
(over a graded ring)

Range:
A graded row over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])
of rank 2 and degrees:
[ [ ( 1, 0 ), 2 ] ]
```

Example

```
gap> tensorProductL := TensorProductOnObjects( O1L, O2L );
<A graded row of rank 2>
```

Example

```
gap> Display( tensorProductL );
A graded row over Q[x_1,x_2,x_3,x_4] (with weights
[ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ]) of rank 2 and degrees:
[ [ ( -1, 0 ), 2 ] ]
```

Example

```
gap> tensorProductMorphismL := TensorProductOnMorphisms( m2L, morL );
<A morphism in Category of graded rows over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( tensorProductMorphismL );
true
```

Example

```
gap> Display( tensorProductMorphismL );
A morphism in Category of graded rows over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])

Source:
A graded row over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])
of rank 1 and degrees:
[ [ 0, 1 ] ]
```

```

Matrix:
x_1,x_2
(over a graded ring)

Range:
A graded row over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])
of rank 2 and degrees:
[ [ ( 1, 0 ), 2 ] ]
gap> Display( DualOnObjects( TensorProductOnObjects( ObjectL, Object2L ) ) );
A graded row over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ]) of rank 16 and degrees:
[ [ ( -2, 0 ), 5 ], [ ( -1, -1 ), 2 ], [ ( -2, 0 ), 6 ], [ ( -1, -1 ), 2 ],
[ ( -2, 0 ), 1 ] ]

```

Example

```

gap> IsWellDefined( DualOnMorphisms( m1L ) );
true

```

Example

```

gap> Display( DualOnMorphisms( m1L ) );
A morphism in Category of graded rows over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])

Source:
A graded row over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])
of rank 1 and degrees:
[ [ 0, 1 ] ]

Matrix:
1,x_2
(over a graded ring)

Range:
A graded row over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])
of rank 2 and degrees:
[ [ 0, 1 ], [ ( 1, 0 ), 1 ] ]

```

Example

```

gap> IsWellDefined( EvaluationForDualWithGivenTensorProduct( TensorProductOnObjects(
> DualOnObjects( ObjectL ), ObjectL ), ObjectL, TensorUnit( categoryL ) ) );
true

```

Example

```

gap> Display( EvaluationForDualWithGivenTensorProduct( TensorProductOnObjects(
> DualOnObjects( ObjectL ), ObjectL ), ObjectL, TensorUnit( categoryL ) ) );
A morphism in Category of graded rows over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])

Source:
A graded row over Q[x_1,x_2,x_3,x_4] (with weights
[ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ]) of rank 4 and degrees:

```

```

[ [ 0, 4 ] ]

Matrix:
1,
0,
0,
1
(over a graded ring)

Range:
A graded row over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])
of rank 1 and degrees:
[ [ 0, 1 ] ]
gap> Display( InternalHomOnObjects( ObjectL, ObjectL ) );
A graded row over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])
of rank 4 and degrees:
[ [ 0, 4 ] ]

```

14.7 A few categorical constructions for graded columns

Example

```

gap> ZeroObject( categoryR );
<A graded column of rank 0>
gap> 01R := GradedColumn( [ [-1,0],2 ] , S );
<A graded column of rank 2>

```

Example

```

gap> Display( ZeroMorphism( ZeroObject( categoryR ), 01R ) );
A morphism in Category of graded columns over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])

Source:
A graded column over Q[x_1,x_2,x_3,x_4] (with weights
[ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])
of rank 0 and degrees:
[ ]

Matrix:
(an empty 2 x 0 matrix)

Range:
A graded column over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])
of rank 2 and degrees:
[ [ ( -1, 0 ), 2 ] ]

```

Example

```

gap> 02R := GradedColumn( [ [0,0],1 ] , S );
<A graded column of rank 1>
gap> obj3R := GradedColumn( [ [-1,0],1 ] , S );
<A graded column of rank 1>

```

Example

```
gap> Display( IdentityMorphism( O2R ) );
A morphism in Category of graded columns over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])
```

Source:

A graded column over Q[x_1,x_2,x_3,x_4]
(with weights [[1, 0], [1, 0], [0, 1], [0, 1]])
of rank 1 and degrees:
[[0, 1]]

Matrix:

1
(over a graded ring)

Range:

A graded column over Q[x_1,x_2,x_3,x_4]
(with weights [[1, 0], [1, 0], [0, 1], [0, 1]])
of rank 1 and degrees:
[[0, 1]]

Example

```
gap> IsWellDefined( IdentityMorphism( Q2R ) );
true
gap> directSumR := DirectSum( [ O1R, O2R ] );
<A graded column of rank 3>
```

Example

```
gap> Display( directSumR );
A graded column over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ]) of rank 3 and degrees:
[ [ ( -1, 0 ), 2 ], [ 0, 1 ] ]
```

Example

```
gap> i1R := InjectionOfCofactorOfDirectSum( [ O1R, O2R ], 1 );
<A morphism in Category of graded columns over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( i1R );
true
```

Example

```
gap> Display( UnderlyingHomalgMatrix( i1R ) );
1,0,
0,1,
0,0
(over a graded ring)
```

Example

```
gap> i2R := InjectionOfCofactorOfDirectSum( [ O1R, O2R ], 2 );
<A morphism in Category of graded columns over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( i2R );
true
```

Example

```
gap> Display( UnderlyingHomalgMatrix( i2R ) );
0,
```

```
0,
1
(over a graded ring)
```

Example

```
gap> proj1R := ProjectionInFactorOfDirectSum( [ 01R, 02R ], 1 );
<A morphism in Category of graded columns over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( proj1R );
true
```

Example

```
gap> Display( UnderlyingHomalgMatrix( proj1R ) );
1,0,0,
0,1,0
(over a graded ring)
```

Example

```
gap> proj2R := ProjectionInFactorOfDirectSum( [ 01R, 02R ], 2 );
<A morphism in Category of graded columns over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( proj2R );
true
```

Example

```
gap> Display( UnderlyingHomalgMatrix( proj2R ) );
0,0,1
(over a graded ring)
```

Example

```
gap> kR := WeakKernelEmbedding( proj1R );
<A morphism in Category of graded columns over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( kR );
true
```

Example

```
gap> Display( UnderlyingHomalgMatrix( kR ) );
0,
0,
1
(over a graded ring)
```

Example

```
gap> ckR := WeakCokernelProjection( kR );
<A morphism in Category of graded columns over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( ckR );
true
```

Example

```
gap> Display( UnderlyingHomalgMatrix( ckR ) );
1,0,0,
0,1,0
(over a graded ring)
```

Example

```

gap> IsMonomorphism( kR );
true
gap> IsEpimorphism( kR );
false
gap> IsMonomorphism( ckR );
false
gap> IsEpimorphism( ckR );
true
gap> m1R := GradedRowOrColumnMorphism( 01R,
>      HomalgMatrix( [[ "x_1", "x_2" ]], S ), 02R );
<A morphism in Category of graded columns over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( m1R );
true
gap> m2R := IdentityMorphism( 02R );
<A morphism in Category of graded columns over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( m2R );
true
gap> obj1R := GradedColumn( [ [[0,0],1], [[-1,0],1] ], S );
<A graded column of rank 2>
gap> m1R := GradedRowOrColumnMorphism( obj1R,
>      HomalgMatrix( [ [ 1, "x_2" ] ], S ), 02R );
<A morphism in Category of graded columns over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( m1R );
true
gap> m3R := GradedRowOrColumnMorphism( obj3R,
>      HomalgMatrix( [[ "x_1" ]], S ), 02R );
<A morphism in Category of graded columns over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( m3R );
true
gap> liftR := Lift( m3R, m1R );
<A morphism in Category of graded columns over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( liftR );
true

```

Example

```

gap> Display( UnderlyingHomalgMatrix( liftR ) );
x_1,
0
(over a graded ring)

```

Example

```

gap> 03R := GradedColumn( [ [[1,0],2] ], S );
<A graded column of rank 2>
gap> morR := GradedRowOrColumnMorphism(
>      02R, HomalgMatrix( [[ "x_1" ], [ "x_2" ] ], S ), 03R );
<A morphism in Category of graded columns over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( morR );

```

```

true
gap> coliftR := Colift( m2R, morR );
<A morphism in Category of graded columns over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( coliftR );
true

```

— Example —

```

gap> Display( UnderlyingHomalgMatrix( coliftR ) );
x_1,
x_2
(over a graded ring)

```

— Example —

```

gap> fpR := WeakBiFiberProduct( m1R, m2R );
<A graded column of rank 2>
gap> fp_proj1R := ProjectionInFirstFactorOfWeakBiFiberProduct( m1R, m2R );
<A morphism in Category of graded columns over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( fp_proj1R );
true

```

— Example —

```

gap> Display( UnderlyingHomalgMatrix( fp_proj1R ) );
1,0,
0,1
(over a graded ring)

```

— Example —

```

gap> fp_proj2R := ProjectionInSecondFactorOfWeakBiFiberProduct( m1R, m2R );
<A morphism in Category of graded columns over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( fp_proj2R );
true

```

— Example —

```

gap> Display( UnderlyingHomalgMatrix( fp_proj2R ) );
1, x_2
(over a graded ring)

```

— Example —

```

gap> BiasedWeakFiberProduct( m1R, m2R );
<A graded column of rank 2>
gap> pbwfpcol := ProjectionOfBiasedWeakFiberProduct( m1R, m2R );
<A morphism in Category of graded columns over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( pbwfpcol );
true

```

— Example —

```

gap> Display( pbwfpcol );
A morphism in Category of graded columns over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])

Source:
A graded column over Q[x_1,x_2,x_3,x_4]

```

```
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])
of rank 2 and degrees:
[ [ 0, 1 ], [ ( -1, 0 ), 1 ] ]
```

```
Matrix:
1,0,
0,1
(over a graded ring)
```

```
Range:
A graded column over  $\mathbb{Q}[x_1, x_2, x_3, x_4]$ 
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])
of rank 2 and degrees:
[ [ 0, 1 ], [ ( -1, 0 ), 1 ] ]
```

Example

```
gap> poR := WeakBiPushout( morR, m2R );
<A graded column of rank 2>
gap> inj1R := InjectionOfFirstCofactorOfWeakBiPushout( morR, m2R );
<A morphism in Category of graded columns over
 $\mathbb{Q}[x_1, x_2, x_3, x_4]$  (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( inj1R );
true
```

Example

```
gap> Display( UnderlyingHomalgMatrix( inj1R ) );
1,0,
0,1
(over a graded ring)
```

Example

```
gap> inj2R := InjectionOfSecondCofactorOfWeakBiPushout( morR, m2R );
<A morphism in Category of graded columns over
 $\mathbb{Q}[x_1, x_2, x_3, x_4]$  (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( inj2R );
true
```

Example

```
gap> Display( UnderlyingHomalgMatrix( inj2R ) );
x_1,
x_2
(over a graded ring)
```

Example

```
gap> injectionR := InjectionOfBiasedWeakPushout( morR, m2R );
<A morphism in Category of graded columns over
 $\mathbb{Q}[x_1, x_2, x_3, x_4]$  (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( injectionR );
true
```

Example

```
gap> Display( injectionR );
A morphism in Category of graded columns over  $\mathbb{Q}[x_1, x_2, x_3, x_4]$ 
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])

Source:
```



```
A graded column over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])
of rank 2 and degrees:
[ [ ( 1, 0 ), 2 ] ]
```

```
Matrix:
1,0,
0,1
(over a graded ring)
```

```
Range:
A graded column over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])
of rank 2 and degrees:
[ [ ( 1, 0 ), 2 ] ]
```

Example

```
gap> tensorProductR := TensorProductOnObjects( O1R, O2R );
<A graded column of rank 2>
```

Example

```
gap> Display( tensorProductR );
A graded column over Q[x_1,x_2,x_3,x_4] (with weights
[ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ]) of rank 2 and degrees:
[ [ ( -1, 0 ), 2 ] ]
```

Example

```
gap> tensorProductMorphismR := TensorProductOnMorphisms( m2R, morR );
<A morphism in Category of graded columns over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( tensorProductMorphismR );
true
```

Example

```
gap> Display( tensorProductMorphismR );
A morphism in Category of graded columns over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])

Source:
A graded column over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])
of rank 1 and degrees:
[ [ 0, 1 ] ]

Matrix:
x_1,
x_2
(over a graded ring)

Range:
A graded column over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])
of rank 2 and degrees:
[ [ ( 1, 0 ), 2 ] ]
gap> Display( DualOnObjects( TensorProductOnObjects( ObjectR, Object2R ) ) );
```

```
A graded column over  $\mathbb{Q}[x_1, x_2, x_3, x_4]$ 
(with weights  $\begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$ ) of rank 16 and degrees:
 $\begin{bmatrix} (-2, 0), 5 \\ (-1, -1), 2 \\ (-2, 0), 6 \\ (-1, -1), 2 \\ (-2, 0), 1 \end{bmatrix}$ 
```

Example

```
gap> IsWellDefined( DualOnMorphisms( m1R ) );
true
```

Example

```
gap> Display( DualOnMorphisms( m1R ) );
A morphism in Category of graded columns over  $\mathbb{Q}[x_1, x_2, x_3, x_4]$ 
(with weights  $\begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$ )

Source:
A graded column over  $\mathbb{Q}[x_1, x_2, x_3, x_4]$ 
(with weights  $\begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$ )
of rank 1 and degrees:
 $\begin{bmatrix} 0, 1 \end{bmatrix}$ 

Matrix:
1,
x_2
(over a graded ring)

Range:
A graded column over  $\mathbb{Q}[x_1, x_2, x_3, x_4]$ 
(with weights  $\begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$ )
of rank 2 and degrees:
 $\begin{bmatrix} 0, 1 \\ (1, 0), 1 \end{bmatrix}$ 
```

Example

```
gap> IsWellDefined( EvaluationForDualWithGivenTensorProduct( TensorProductOnObjects(
> DualOnObjects( ObjectR ), ObjectR ), ObjectR, TensorUnit( categoryR ) ) );
true
```

Example

```
gap> Display( EvaluationForDualWithGivenTensorProduct( TensorProductOnObjects(
> DualOnObjects( ObjectR ), ObjectR ), ObjectR, TensorUnit( categoryR ) ) );
A morphism in Category of graded columns over  $\mathbb{Q}[x_1, x_2, x_3, x_4]$ 
(with weights  $\begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$ )

Source:
A graded column over  $\mathbb{Q}[x_1, x_2, x_3, x_4]$ 
(with weights  $\begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$ )
of rank 4 and degrees:
 $\begin{bmatrix} 0, 4 \end{bmatrix}$ 

Matrix:
1,0,0,1
(over a graded ring)

Range:
A graded column over  $\mathbb{Q}[x_1, x_2, x_3, x_4]$ 
(with weights  $\begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$ )
```

```

of rank 1 and degrees:
[ [ 0, 1 ] ]
gap> Display( InternalHomOnObjects( ObjectR, ObjectR ) );
A graded column over  $\mathbb{Q}[x_1, x_2, x_3, x_4]$ 
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])
of rank 4 and degrees:
[ [ 0, 4 ] ]

```

14.8 Additional examples on monoidal structure for graded rows

Example

```

gap> aR := GradedRow( [ [ [1,0], 1 ] ], S );
<A graded row of rank 1>
gap> bR := ZeroObject( aR );
<A graded row of rank 0>
gap> coevR := CoevaluationForDual( bR );
<A morphism in Category of graded rows over  $\mathbb{Q}[x_1, x_2, x_3, x_4]$ 
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( coevR );
true
gap> evalR := EvaluationForDual( bR );
<A morphism in Category of graded rows over  $\mathbb{Q}[x_1, x_2, x_3, x_4]$ 
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( evalR );
true
gap> cR := GradedRow( [ [ [2,0], 1 ] ], S );
<A graded row of rank 1>
gap> aR_o_bR := TensorProductOnObjects( aR, bR );
<A graded row of rank 0>
gap> phiR := ZeroMorphism( aR_o_bR, cR );
<A morphism in Category of graded rows over  $\mathbb{Q}[x_1, x_2, x_3, x_4]$ 
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( phiR );
true
gap> tens_mor := TensorProductToInternalHomAdjunctionMap(aR,bR,phiR);
<A morphism in Category of graded rows over  $\mathbb{Q}[x_1, x_2, x_3, x_4]$ 
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( tens_mor );
true

```

14.9 Additional examples on monoidal structure for graded columns

Example

```

gap> aC := GradedColumn( [ [ [1,0], 1 ] ], S );
<A graded column of rank 1>
gap> bC := ZeroObject( aC );
<A graded column of rank 0>
gap> coevC := CoevaluationForDual( bC );
<A morphism in Category of graded columns over  $\mathbb{Q}[x_1, x_2, x_3, x_4]$ 
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( coevC );

```

```

true
gap> evalC := EvaluationForDual( bC );
<A morphism in Category of graded columns over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( evalC );
true
gap> cC := GradedColumn( [ [ [2,0], 1 ] ], S );
<A graded column of rank 1>
gap> aC_o_bC := TensorProductOnObjects( aC, bC );
<A graded column of rank 0>
gap> phiC := ZeroMorphism( aC_o_bC, cC );
<A morphism in Category of graded columns over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( phiC );
true
gap> tens_mor := TensorProductToInternalHomAdjunctionMap(aC,bC,phiC);
<A morphism in Category of graded columns over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( tens_mor );
true

```

14.10 Examples to test Tools methods in graded rows/cols

Example

```

gap> S := GradedRing( Q * "x,y" );
Q[x,y]
(weights: yet unset)
gap> SetWeightsOfIndeterminates( S, [ 1, 1 ] );
gap> mat_1 := HomalgMatrix( "[ x, 0, 0, y ]", 2, 2, S );
<A 2 x 2 matrix over a graded ring>
gap> mat_2 := HomalgMatrix( "[ x, 0, 0, 0 ]", 2, 2, S );
<A 2 x 2 matrix over a graded ring>
gap> a := GradedRow( [ [ [ 1 ], 1 ], [ [ 2 ], 1 ] ], S );
<A graded row of rank 2>
gap> b := GradedColumn( [ [ [ 1 ], 1 ], [ [ 2 ], 1 ] ], S );
<A graded column of rank 2>
gap> map := DeduceMapFromMatrixAndRangeForGradedRows( mat_1, a );
<A morphism in Category of graded rows over Q[x,y] (with weights [ 1, 1 ])>
gap> some_map := DeduceSomeMapFromMatrixAndRangeForGradedRows( mat_1, a );
<A morphism in Category of graded rows over Q[x,y] (with weights [ 1, 1 ])>
gap> IsEqualForMorphisms( map, some_map );
true
gap> map := DeduceMapFromMatrixAndSourceForGradedRows( mat_1, a );
<A morphism in Category of graded rows over Q[x,y] (with weights [ 1, 1 ])>
gap> some_map := DeduceSomeMapFromMatrixAndSourceForGradedRows( mat_1, a );
<A morphism in Category of graded rows over Q[x,y] (with weights [ 1, 1 ])>
gap> IsEqualForMorphisms( map, some_map );
true
gap> some_map := DeduceSomeMapFromMatrixAndRangeForGradedRows( mat_2, a );
<A morphism in Category of graded rows over Q[x,y] (with weights [ 1, 1 ])>
gap> IsWellDefined( some_map );
true

```

```

gap> some_map := DeduceSomeMapFromMatrixAndSourceForGradedRows( mat_2, a );
<A morphism in Category of graded rows over Q[x,y] (with weights [ 1, 1 ])>
gap> IsWellDefined( some_map );
true
gap> map := DeduceMapFromMatrixAndRangeForGradedCols( mat_1, b );
<A morphism in Category of graded columns over Q[x,y] (with weights [ 1, 1 ])>
gap> some_map := DeduceSomeMapFromMatrixAndRangeForGradedCols( mat_1, b );
<A morphism in Category of graded columns over Q[x,y] (with weights [ 1, 1 ])>
gap> IsEqualForMorphisms( map, some_map );
true
gap> map := DeduceMapFromMatrixAndSourceForGradedCols( mat_1, b );
<A morphism in Category of graded columns over Q[x,y] (with weights [ 1, 1 ])>
gap> some_map := DeduceSomeMapFromMatrixAndSourceForGradedCols( mat_1, b );
<A morphism in Category of graded columns over Q[x,y] (with weights [ 1, 1 ])>
gap> IsEqualForMorphisms( map, some_map );
true
gap> some_map := DeduceSomeMapFromMatrixAndRangeForGradedCols( mat_2, b );
<A morphism in Category of graded columns over Q[x,y] (with weights [ 1, 1 ])>
gap> IsWellDefined( some_map );
true
gap> some_map := DeduceSomeMapFromMatrixAndSourceForGradedCols( mat_2, b );
<A morphism in Category of graded columns over Q[x,y] (with weights [ 1, 1 ])>
gap> IsWellDefined( some_map );
true

```

Chapter 15

Category of rows and columns over a field

15.1 Abelian operations for rows

Category of rows over a field

Example

```
gap> Q := HomalgFieldOfRationals();
gap> RowsQ := CategoryOfRows( Q );
gap> a := 3/RowsQ;;
gap> b := 4/RowsQ;;
gap> homalg_matrix := HomalgMatrix( [ [ 1, 0, 0, 0 ],
>                                     [ 0, 1, 0, -1 ],
>                                     [ -1, 0, 2, 1 ] ], 3, 4, Q );
gap> alpha := homalg_matrix/RowsQ;;
gap> homalg_matrix := HomalgMatrix( [ [ 1, 1, 0, 0 ],
>                                     [ 0, 1, 0, -1 ],
>                                     [ -1, 0, 2, 1 ] ], 3, 4, Q );
gap> beta := homalg_matrix/RowsQ;;
gap> IsWellDefined( CokernelObject( alpha ) );
true
gap> c := CokernelProjection( alpha );
gap> gamma := UniversalMorphismIntoDirectSum( [ c, c ] );
gap> colift := CokernelColift( alpha, gamma );
gap> IsEqualForMorphisms( PreCompose( c, colift ), gamma );
true
gap> FiberProduct( alpha, beta );
gap> F := FiberProduct( alpha, beta );
gap> IsWellDefined( F );
true
gap> IsWellDefined( ProjectionInFactorOfFiberProduct( [ alpha, beta ], 1 ) );
true
gap> IsWellDefined( Pushout( alpha, beta ) );
true
gap> i1 := InjectionOfCofactorOfPushout( [ alpha, beta ], 1 );
gap> i2 := InjectionOfCofactorOfPushout( [ alpha, beta ], 2 );
gap> u := UniversalMorphismFromDirectSum( [ b, b ], [ i1, i2 ] );
gap> KernelObjectFunctorial( u, IdentityMorphism( Source( u ) ), u ) = IdentityMorphism( 3/RowsQ );
true
gap> IsZero( CokernelObjectFunctorial( u, IdentityMorphism( Range( u ) ), u ) );
true
```

```

gap> DirectProductFunctorial( [ u, u ] ) = DirectSumFunctorial( [ u, u ] );
true
gap> CoproductFunctorial( [ u, u ] ) = DirectSumFunctorial( [ u, u ] );
true
gap> IsOne( FiberProductFunctorial( [ u, u ], [ IdentityMorphism( Source( u ) ), IdentityMorphism( Range( u ) ) ] ), IdentityMorphism( Source( u ) ) );
true
gap> IsOne( PushoutFunctorial( [ u, u ], [ IdentityMorphism( Range( u ) ), IdentityMorphism( Source( u ) ) ] ), IdentityMorphism( Range( u ) ) );
true
gap> IsCongruentForMorphisms( (1/2) * alpha, alpha * (1/2) );
true
gap> RankOfObject( HomomorphismStructureOnObjects( a, b ) ) = RankOfObject( a ) * RankOfObject( b );
true
gap> IsCongruentForMorphisms(
>   PreCompose( [ u, DualOnMorphisms( i1 ), DualOnMorphisms( alpha ) ] ),
>   InterpretMorphismFromDistinguishedObjectToHomomorphismStructureAsMorphism( Source( u ), Source( b ) ),
>   PreCompose(
>     InterpretMorphismAsMorphismFromDistinguishedObjectToHomomorphismStructure( DualOnMorphisms( i1 ), DualOnMorphisms( alpha ) ),
>     HomomorphismStructureOnMorphisms( u, DualOnMorphisms( alpha ) )
>   )
> );
true

```

Category of columns over a field

Example

```

gap> Q := HomalgFieldOfRationals();
gap> ColsQ := CategoryOfColumns( Q );
gap> a := 3/ColsQ;
gap> b := 4/ColsQ;
gap> homalg_matrix := HomalgMatrix( [ [ 1, 0, 0, 0 ],
>                                     [ 0, 1, 0, -1 ],
>                                     [ -1, 0, 2, 1 ] ], 3, 4, Q );
gap> homalg_matrix := TransposedMatrix( homalg_matrix );
gap> alpha := homalg_matrix/ColsQ;
gap> homalg_matrix := HomalgMatrix( [ [ 1, 1, 0, 0 ],
>                                     [ 0, 1, 0, -1 ],
>                                     [ -1, 0, 2, 1 ] ], 3, 4, Q );
gap> homalg_matrix := TransposedMatrix( homalg_matrix );
gap> beta := homalg_matrix/ColsQ;
gap> IsWellDefined( CokernelObject( alpha ) );
true
gap> c := CokernelProjection( alpha );
gap> gamma := UniversalMorphismIntoDirectSum( [ c, c ] );
gap> colift := CokernelColift( alpha, gamma );
gap> IsEqualForMorphisms( PreCompose( c, colift ), gamma );
true
gap> FiberProduct( alpha, beta );
gap> F := FiberProduct( alpha, beta );
gap> IsWellDefined( F );
true
gap> IsWellDefined( ProjectionInFactorOfFiberProduct( [ alpha, beta ], 1 ) );
true
gap> IsWellDefined( Pushout( alpha, beta ) );

```

```

true
gap> i1 := InjectionOfCofactorOfPushout( [ alpha, beta ], 1 );;
gap> i2 := InjectionOfCofactorOfPushout( [ alpha, beta ], 2 );;
gap> u := UniversalMorphismFromDirectSum( [ b, b ], [ i1, i2 ] );;
gap> KernelObjectFunctorial( u, IdentityMorphism( Source( u ) ), u ) = IdentityMorphism( 3/ColsQ
true
gap> IsZero( CokernelObjectFunctorial( u, IdentityMorphism( Range( u ) ), u ) );
true
gap> DirectProductFunctorial( [ u, u ] ) = DirectSumFunctorial( [ u, u ] );
true
gap> CoproductFunctorial( [ u, u ] ) = DirectSumFunctorial( [ u, u ] );
true
gap> IsOne( FiberProductFunctorial( [ u, u ], [ IdentityMorphism( Source( u ) ), IdentityMorphism
true
gap> IsOne( PushoutFunctorial( [ u, u ], [ IdentityMorphism( Range( u ) ), IdentityMorphism( Rang
true
gap> IsCongruentForMorphisms( (1/2) * alpha, alpha * (1/2) );
true
gap> RankOfObject( HomomorphismStructureOnObjects( a, b ) ) = RankOfObject( a ) * RankOfObject( b
true
gap> IsCongruentForMorphisms(
>   PreCompose( [ u, DualOnMorphisms( i1 ), DualOnMorphisms( alpha ) ] ),
>   InterpretMorphismFromDistinguishedObjectToHomomorphismStructureAsMorphism( Source( u ), Sou
>   PreCompose(
>     InterpretMorphismAsMorphismFromDistinguishedObjectToHomomorphismStructure( DualOnM
>     HomomorphismStructureOnMorphisms( u, DualOnMorphisms( alpha ) )
>   )
> )
> );
true

```


Chapter 16

Example on tensor products in Freyd categories

16.1 Tensor products for categories of rows

Example

```
gap> R := HomalgFieldOfRationalsInSingular() * "a,b,c,d,e,f,g,h,i,j";;
gap> C := CategoryOfRows( R );;
gap> T := TensorUnit( C );;
gap> IsWellDefined( T );
true
```

We test the naturality of the braiding.

Example

```
gap> R2 := DirectSum( T, T );;
gap> R3 := DirectSum( T, R2 );;
gap> R4 := DirectSum( R2, R2 );;
gap> alpha := CategoryOfRowsMorphism( T, HomalgMatrix( "[ a, b, c, d ]", 1, 4, R ), R4 );;
gap> beta := CategoryOfRowsMorphism( R2, HomalgMatrix( "[ e, f, g, h, i, j ]", 2, 3, R ), R3 );;
gap> IsCongruentForMorphisms(
>   PreCompose( Braiding( T, R2 ), TensorProductOnMorphisms( beta, alpha ) ),
>   PreCompose( TensorProductOnMorphisms( alpha, beta ), Braiding( R4, R3 ) )
> );
true
```

We compute the torsion part of a f.p. module with the help of the induced tensor structure on the Freyd category.

Example

```
gap> M := FreydCategoryObject( alpha );;
gap> mu := MorphismToBidual( M );;
gap> co := CoactionToImage( mu );;
gap> IsIsomorphism( co );
true
```

16.2 Tensor products for categories of columns

Example

```
gap> R := HomalgFieldOfRationalsInSingular() * "a,b,c,d,e,f,g,h,i,j";;
gap> C := CategoryOfColumns( R );;
gap> T := TensorUnit( C );;
gap> IsWellDefined( T );
true
```

We test the naturality of the braiding.

Example

```
gap> R2 := DirectSum( T, T );;
gap> R3 := DirectSum( T, R2 );;
gap> R4 := DirectSum( R2, R2 );;
gap> alpha := CategoryOfColumnsMorphism( T, HomalgMatrix( "[ a, b, c, d ]", 4, 1, R ), R4 );;
gap> beta := CategoryOfColumnsMorphism( R2, HomalgMatrix( "[ e, f, g, h, i, j ]", 3, 2, R ), R3 );;
gap> IsCongruentForMorphisms(
>   PreCompose( Braiding( T, R2 ), TensorProductOnMorphisms( beta, alpha ) ),
>   PreCompose( TensorProductOnMorphisms( alpha, beta ), Braiding( R4, R3 ) )
> );
true
```

We compute the torsion part of a f.p. module with the help of the induced tensor structure on the Freyd category.

Example

```
gap> M := FreydCategoryObject( alpha );;
gap> mu := MorphismToBidual( M );;
gap> co := CoastrictionToImage( mu );;
gap> IsIsomorphism( co );
true
```

Chapter 17

The CAP category of graded module presentations for CAP by use of Freyd categories

17.1 CAP categories

17.1.1 FpGradedLeftModules (for IsHomalgGradedRing)

▷ `FpGradedLeftModules(S)` (attribute)

Returns: a `CapCategory`

Given a graded ring S , one can consider the category of f.p. graded left S -modules, which is captured by this attribute.

17.1.2 FpGradedRightModules (for IsHomalgGradedRing)

▷ `FpGradedRightModules(S)` (attribute)

Returns: a `CapCategory`

Given a graded ring S , one can consider the category of f.p. graded right S -modules, which is captured by this attribute.

17.2 The GAP categories for graded module presentations for CAP

17.2.1 IsFpGradedLeftOrRightModulesObject (for IsFreydCategoryObject)

▷ `IsFpGradedLeftOrRightModulesObject($object$)` (filter)

Returns: true or false

The GAP category of graded left and right module presentations.

17.2.2 IsFpGradedLeftModulesObject (for IsFpGradedLeftOrRightModulesObject)

▷ `IsFpGradedLeftModulesObject($object$)` (filter)

Returns: true or false

The GAP category of objects in the presentation category over the category of projective graded left modules.

17.2.3 IsFpGradedRightModulesObject (for IsFpGradedLeftOrRightModulesObject)

▷ IsFpGradedRightModulesObject(*object*) (filter)

Returns: true or false

The GAP category of objects in the presentation category over the category of projective graded right modules.

17.3 The GAP categories for graded module presentation morphisms for CAP

17.3.1 IsFpGradedLeftOrRightModulesMorphism (for IsFreydCategoryMorphism)

▷ IsFpGradedLeftOrRightModulesMorphism(*object*) (filter)

Returns: true or false

The GAP category of left or right module presentation morphisms

17.3.2 IsFpGradedLeftModulesMorphism (for IsFpGradedLeftOrRightModulesMorphism)

▷ IsFpGradedLeftModulesMorphism(*object*) (filter)

Returns: true or false

The GAP category of morphisms in the presentation category over the category of projective graded left modules.

17.3.3 IsFpGradedRightModulesMorphism (for IsFpGradedLeftOrRightModulesMorphism)

▷ IsFpGradedRightModulesMorphism(*object*) (filter)

Returns: true or false

The GAP category of morphisms in the presentation category over the category of projective graded right modules.

Index

- [
 - for IsAdditiveClosureMorphism, IsInt, IsInt, [27](#)
- AddBiasedWeakFiberProduct
 - for IsCapCategory, IsFunction, [15](#)
- AddBiasedWeakPushout
 - for IsCapCategory, IsFunction, [20](#)
- AddDirectSumMorphismToWeakBiPushout
 - for IsCapCategory, IsFunction, [19](#)
- AddEpimorphismFromSomeProjectiveObjectForKernelObject
 - for IsCapCategory, IsFunction, [23](#)
- AddEpimorphismFromSomeProjectiveObjectForKernelObjectWithGivenSomeProjectiveObjectForKernelObject
 - for IsCapCategory, IsFunction, [23](#)
- AddInjectionOfBiasedWeakPushout
 - for IsCapCategory, IsFunction, [21](#)
- AddInjectionOfBiasedWeakPushoutWithGivenBiasedWeakPushout
 - for IsCapCategory, IsFunction, [21](#)
- AddInjectionOfFirstCofactorOfWeakBiPushout
 - for IsCapCategory, IsFunction, [18](#)
- AddInjectionOfFirstCofactorOfWeakBiPushoutWithGivenWeakBiPushout
 - for IsCapCategory, IsFunction, [18](#)
- AddInjectionOfSecondCofactorOfWeakBiPushout
 - for IsCapCategory, IsFunction, [18](#)
- AddInjectionOfSecondCofactorOfWeakBiPushoutWithGivenWeakBiPushout
 - for IsCapCategory, IsFunction, [18](#)
- AdditiveClosure
 - for IsCapCategory, [24](#)
- AdditiveClosureMorphism
 - for IsAdditiveClosureObject, IsObject, IsAdditiveClosureObject, [25](#)
- AdditiveClosureMorphismListList
 - for IsAdditiveClosureObject, IsList, IsAdditiveClosureObject, [25](#)
- AdditiveClosureObject
 - for IsList, IsAdditiveClosureCategory, [24](#)
- AddMonomorphismToSomeInjectiveObjectForCokernelObject
 - for IsCapCategory, IsFunction, [23](#)
- AddMonomorphismToSomeInjectiveObjectForCokernelObjectWithGivenSomeInjectiveObjectForCokernelObject
 - for IsCapCategory, IsFunction, [23](#)
- AddProjectionInFirstFactorOfWeakBiFiberProduct
 - for IsCapCategory, IsFunction, [12](#)
- AddProjectionInFirstFactorOfWeakBiFiberProductWithGivenWeakBiFiberProduct
 - for IsCapCategory, IsFunction, [12](#)
- AddProjectionInSecondFactorOfWeakBiFiberProduct
 - for IsCapCategory, IsFunction, [12](#)
- AddProjectionInSecondFactorOfWeakBiFiberProductWithGivenWeakBiFiberProduct
 - for IsCapCategory, IsFunction, [12](#)
- AddProjectionOfBiasedWeakFiberProduct
 - for IsCapCategory, IsFunction, [15](#)
- AddProjectionOfBiasedWeakFiberProductWithGivenBiasedWeakFiberProduct
 - for IsCapCategory, IsFunction, [15](#)
- AddSomeInjectiveObjectForCokernelObject
 - for IsCapCategory, IsFunction, [23](#)
- AddSomeProjectiveObjectForKernelObject

- for IsCapCategory, IsFunction, [22](#)
- AddUniversalMorphismFromBiasedWeakPushout
 - for IsCapCategory, IsFunction, [21](#)
- AddUniversalMorphismFromBiasedWeakPushoutWithGivenBiasedWeakPushout
 - for IsCapCategory, IsFunction, [21](#)
- AddUniversalMorphismFromWeakBiPushout
 - for IsCapCategory, IsFunction, [18](#)
- AddUniversalMorphismFromWeakBiPushoutWithGivenWeakBiPushout
 - for IsCapCategory, IsFunction, [19](#)
- AddUniversalMorphismIntoBiasedWeakFiberProduct
 - for IsCapCategory, IsFunction, [15](#)
- AddUniversalMorphismIntoBiasedWeakFiberProductWithGivenBiasedWeakFiberProduct
 - for IsCapCategory, IsFunction, [15](#)
- AddUniversalMorphismIntoWeakBiFiberProduct
 - for IsCapCategory, IsFunction, [12](#)
- AddUniversalMorphismIntoWeakBiFiberProductWithGivenWeakBiFiberProduct
 - for IsCapCategory, IsFunction, [13](#)
- AddWeakBiFiberProduct
 - for IsCapCategory, IsFunction, [11](#)
- AddWeakBiFiberProductMorphismToDirectSum
 - for IsCapCategory, IsFunction, [13](#)
- AddWeakBiPushout
 - for IsCapCategory, IsFunction, [18](#)
- AddWeakCokernelColift
 - for IsCapCategory, IsFunction, [9](#)
- AddWeakCokernelColiftWithGivenWeakCokernelObject
 - for IsCapCategory, IsFunction, [9](#)
- AddWeakCokernelObject
 - for IsCapCategory, IsFunction, [8](#)
- AddWeakCokernelProjection
 - for IsCapCategory, IsFunction, [8](#)
- AddWeakCokernelProjectionWithGivenWeakCokernelObject
 - for IsCapCategory, IsFunction, [9](#)
- AddWeakKernelEmbedding
 - for IsCapCategory, IsFunction, [6](#)
- AddWeakKernelEmbeddingWithGivenWeakKernelObject
 - for IsCapCategory, IsFunction, [6](#)
- AddWeakKernelLift
 - for IsCapCategory, IsFunction, [7](#)
- AddWeakKernelLiftWithGivenWeakKernelObject
 - for IsCapCategory, IsFunction, [7](#)
- AddWeakKernelObject
 - for IsCapCategory, IsFunction, [6](#)
- AdelmanCategory
 - for IsCapCategory, [32](#)
- AdelmanCategoryMorphism
 - for IsAdelmanCategoryObject, IsCapCategoryMorphism, IsAdelmanCategoryObject, [32](#)
- AdelmanCategoryObject
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, [32](#)
- AsAdditiveClosureMorphism
 - for IsCapCategoryMorphism, [25](#)
- AsAdditiveClosureObject
 - for IsCapCategoryObject, [25](#)
- AsAdelmanCategoryMorphism
 - for IsCapCategoryMorphism, [32](#)
- AsAdelmanCategoryObject
 - for IsCapCategoryObject, [32](#)
- BiasedWeakFiberProduct
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, [14](#)
- BiasedWeakPushout
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, [20](#)
- *
 - for IsFreydCategoryMorphism, IsFreydCategoryMorphism, [59](#)
 - for IsFreydCategoryObject, IsFreydCategoryObject, [59](#)
- \/*
 - for IsCapCategoryCell, IsAdditiveClosureCategory, [27](#)
 - for IsCapCategoryMorphism, IsAdelmanCategory, [33](#)

- for IsCapCategoryObject, IsAdelmanCategory, 33
- for IsList, IsAdditiveClosureCategory, 27
- $\backslash[\backslash]$
 - for IsAdditiveClosureObject, IsInt, 27
- $\backslash\sim$
 - for IsFreydCategoryMorphism, IsInt, 59
 - for IsFreydCategoryObject, IsInt, 59
- CategoryOfGradedColumns
 - for IsHomalgGradedRing, 52
- CategoryOfGradedRows
 - for IsHomalgGradedRing, 52
- CorelationMorphism
 - for IsAdelmanCategoryObject, 33
- CorelationWitness
 - for IsAdelmanCategoryMorphism, 34
- DeduceMapFromMatrixAndRangeForGradedCols
 - for IsHomalgMatrix, IsGradedColumn, 56
- DeduceMapFromMatrixAndRangeForGradedRows
 - for IsHomalgMatrix, IsGradedRow, 55
- DeduceMapFromMatrixAndSourceForGradedCols
 - for IsHomalgMatrix, IsGradedColumn, 56
- DeduceMapFromMatrixAndSourceForGradedRows
 - for IsHomalgMatrix, IsGradedRow, 55
- DeduceSomeMapFromMatrixAndRangeForGradedCols
 - for IsHomalgMatrix, IsGradedColumn, 56
- DeduceSomeMapFromMatrixAndRangeForGradedRows
 - for IsHomalgMatrix, IsGradedRow, 55
- DeduceSomeMapFromMatrixAndSourceForGradedCols
 - for IsHomalgMatrix, IsGradedColumn, 56
- DeduceSomeMapFromMatrixAndSourceForGradedRows
 - for IsHomalgMatrix, IsGradedRow, 56
- DegreeList
 - for IsGradedRowOrColumn, 54
- DirectSumMorphismToWeakBiPushout
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, 17
- EpimorphismFromSomeProjectiveObjectForKernelObject
 - for IsCapCategoryMorphism, 22
- EpimorphismFromSomeProjectiveObjectForKernelObjectWithGivenSomeProjectiveObjectForKernelObject
 - for IsCapCategoryMorphism, IsCapCategoryObject, 22
- ExtendFunctorToAdditiveClosureOfSource
 - for IsCapFunctor, 26
- ExtendFunctorToAdditiveClosures
 - for IsCapFunctor, 26
- ExtendFunctorToLinearClosureOfSource
 - for IsCapFunctor, IsLinearClosure, 89
 - for IsCapFunctor, IsLinearClosure, IsFunction, 89
- ExtendFunctorWithAdditiveRangeToFunctorFromAdditiveClosureOfSource
 - for IsCapFunctor, 26
- ExtendNaturalTransformationToAdditiveClosureOfSource
 - for IsCapNaturalTransformation, 26
- FpGradedLeftModules
 - for IsHomalgGradedRing, 130
- FpGradedRightModules
 - for IsHomalgGradedRing, 130
- GradedColumn
 - for IsList, IsHomalgGradedRing, 53
 - for IsList, IsHomalgGradedRing, IsBool, 53
- GradedRow
 - for IsList, IsHomalgGradedRing, 52
 - for IsList, IsHomalgGradedRing, IsBool, 52
- GradedRowOrColumnMorphism
 - for IsGradedRowOrColumn, IsHomalgMatrix, IsGradedRowOrColumn, 53
 - for IsGradedRowOrColumn, IsHomalgMatrix, IsGradedRowOrColumn, IsBool, 53
- InclusionFunctorInAdditiveClosure
 - for IsCapCategory, 25
- InjectionOfBiasedWeakPushout
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, 20

- InjectionOfBiasedWeakPushoutWithGiven-BiasedWeakPushout
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject, [20](#)
- InjectionOfFirstCofactorOfWeakBi-Pushout
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, [16](#)
- InjectionOfFirstCofactorOfWeakBi-PushoutWithGivenWeakBiPushout
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject, [17](#)
- InjectionOfSecondCofactorOfWeakBi-Pushout
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, [16](#)
- InjectionOfSecondCofactorOfWeakBi-PushoutWithGivenWeakBiPushout
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject, [17](#)
- INTERNAL_HOM_EMBEDDING
 - for IsFreydCategoryObject, IsFreydCategoryObject, [59](#)
- IsAdditiveClosureCategory
 - for IsCapCategory, [24](#)
- IsAdditiveClosureMorphism
 - for IsCapCategoryMorphism, [24](#)
- IsAdditiveClosureObject
 - for IsCapCategoryObject, [24](#)
- IsAdelmanCategory
 - for IsCapCategory, [32](#)
- IsAdelmanCategoryMorphism
 - for IsCapCategoryMorphism, [31](#)
- IsAdelmanCategoryObject
 - for IsCapCategoryObject, [31](#)
- IsCategoryOfColumnsObject
 - for IsCapCategoryObject, [45](#)
- IsCategoryOfRowsObject
 - for IsCapCategoryObject, [35](#)
- IsFpGradedLeftModulesMorphism
 - for IsFpGradedLeftOrRightModulesMorphism, [131](#)
- IsFpGradedLeftModulesObject
 - for IsFpGradedLeftOrRightModulesObject, [130](#)
- IsFpGradedLeftOrRightModulesMorphism
 - for IsFreydCategoryMorphism, [131](#)
- IsFpGradedLeftOrRightModulesObject
 - for IsFreydCategoryObject, [130](#)
- IsFpGradedRightModulesMorphism
 - for IsFpGradedLeftOrRightModulesMorphism, [131](#)
- IsFpGradedRightModulesObject
 - for IsFpGradedLeftOrRightModulesObject, [131](#)
- IsGradedColumn
 - for IsGradedRowOrColumn, [54](#)
- IsGradedColumnMorphism
 - for IsGradedRowOrColumnMorphism, [55](#)
- IsGradedRow
 - for IsGradedRowOrColumn, [54](#)
- IsGradedRowMorphism
 - for IsGradedRowOrColumnMorphism, [55](#)
- IsGradedRowOrColumn
 - for IsCapCategoryObject, [54](#)
- IsGradedRowOrColumnMorphism
 - for IsCapCategoryMorphism, [55](#)
- IsSequenceAsAdelmanCategoryObject
 - for IsAdelmanCategoryObject, [34](#)
- MonomorphismToSomeInjectiveObjectFor-CokernelObject
 - for IsCapCategoryMorphism, [22](#)
- MonomorphismToSomeInjectiveObjectFor-CokernelObjectWithGivenSome-InjectiveObjectForCokernel-Object
 - for IsCapCategoryMorphism, IsCapCategoryObject, [22](#)
- MorphismDatum
 - for IsAdelmanCategoryMorphism, [33](#)
- MorphismMatrix
 - for IsAdditiveClosureMorphism, [27](#)
- NrCols
 - for IsAdditiveClosureMorphism, [27](#)
- NrRows
 - for IsAdditiveClosureMorphism, [27](#)
- ObjectList
 - for IsAdditiveClosureObject, [26](#)
- ProjectionInFirstFactorOfWeakBiFiber-Product

- for IsCapCategoryMorphism, IsCapCategoryMorphism, [10](#)
- ProjectionInFirstFactorOfWeakBiFiberProductWithGivenWeakBiFiberProduct
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject, [10](#)
- ProjectionInSecondFactorOfWeakBiFiberProduct
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, [10](#)
- ProjectionInSecondFactorOfWeakBiFiberProductWithGivenWeakBiFiberProduct
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject, [11](#)
- ProjectionOfBiasedWeakFiberProduct
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, [14](#)
- ProjectionOfBiasedWeakFiberProductWithGivenBiasedWeakFiberProduct
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject, [14](#)
- RankOfObject
 - for IsGradedRowOrColumn, [54](#)
- RelationMorphism
 - for IsAdelmanCategoryObject, [33](#)
- RelationWitness
 - for IsAdelmanCategoryMorphism, [34](#)
- SomeInjectiveObjectForCokernelObject
 - for IsCapCategoryMorphism, [22](#)
- SomeProjectiveObjectForKernelObject
 - for IsCapCategoryMorphism, [21](#)
- UnderlyingCategory
 - for IsAdditiveClosureCategory, [26](#)
 - for IsAdelmanCategory, [33](#)
- UnderlyingHomalgGradedRing
 - for IsGradedRowOrColumn, [53](#)
 - for IsGradedRowOrColumnMorphism, [54](#)
- UnderlyingHomalgMatrix
 - for IsGradedRowOrColumnMorphism, [54](#)
- UniversalMorphismFromBiasedWeakPushout
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, [20](#)
- UniversalMorphismFromBiasedWeakPushoutWithGivenBiasedWeakPushout
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject, [20](#)
- UniversalMorphismFromWeakBiPushout
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, [17](#)
- UniversalMorphismFromWeakBiPushoutWithGivenWeakBiPushout
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject, [17](#)
- UniversalMorphismIntoBiasedWeakFiberProduct
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, [14](#)
- UniversalMorphismIntoBiasedWeakFiberProductWithGivenBiasedWeakFiberProduct
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject, [14](#)
- UniversalMorphismIntoWeakBiFiberProduct
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, [11](#)
- UniversalMorphismIntoWeakBiFiberProductWithGivenWeakBiFiberProduct
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject, [11](#)
- UnzipDegreeList
 - for IsGradedRowOrColumn, [56](#)
- WeakBiFiberProduct
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, [10](#)
- WeakBiFiberProductMorphismToDirectSum

- for IsCapCategoryMorphism, IsCapCategoryMorphism, [11](#)
- WeakBiPushout
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, [16](#)
- WeakCokernelColift
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, [8](#)
- WeakCokernelColiftWithGivenWeakCokernelObject
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject, [8](#)
- WeakCokernelObject
 - for IsCapCategoryMorphism, [7](#)
- WeakCokernelProjection
 - for IsCapCategoryMorphism, [8](#)
- WeakCokernelProjectionWithGivenWeakCokernelObject
 - for IsCapCategoryMorphism, IsCapCategoryObject, [8](#)
- WeakKernelEmbedding
 - for IsCapCategoryMorphism, [5](#)
- WeakKernelEmbeddingWithGivenWeakKernelObject
 - for IsCapCategoryMorphism, IsCapCategoryObject, [6](#)
- WeakKernelLift
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, [6](#)
- WeakKernelLiftWithGivenWeakKernelObject
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject, [6](#)
- WeakKernelObject
 - for IsCapCategoryMorphism, [5](#)
- WitnessPairForBeingCongruentToZero
 - for IsAdelmanCategoryMorphism, [34](#)