FreydCategories-ForCAP

Freyd categories - Formal (co)kernels for additive categories

2019.03.04

4 March 2019

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

Chapter 1

Freyd categories

1.1 Weak kernel

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

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

The triple (K, ι, u) is called a *kernel* of α if the morphisms $u(\tau)$ are uniquely determined up to congruence of morphisms. We denote the object K of such a triple by $\mathrm{KernelObject}(\alpha)$. We say that the morphism $u(\tau)$ is induced by the *universal property of the kernel*.
KernelObject is a functorial operation. This means: for $\mu : A \to A'$, $\nu : B \to B'$, $\alpha : A \to B$, $\alpha' : A' \to B'$ such that $\nu \circ \alpha \sim_{A,B'} \alpha' \circ \mu$, we obtain a morphism $\mathrm{KernelObject}(\alpha) \to \mathrm{KernelObject}(\alpha')$.

1.1.1 WeakKernelObject (for IsCapCategoryMorphism)

▷ `WeakKernelObject(alpha)` (attribute)

Returns: an object

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

1.1.2 WeakKernelEmbedding (for IsCapCategoryMorphism)

▷ `WeakKernelEmbedding(alpha)` (attribute)

Returns: a morphism in $\mathrm{Hom}(\mathrm{KernelObject}(\alpha), A)$

The argument is a morphism $\alpha : A \to B$. The output is the kernel embedding $\iota : \mathrm{KernelObject}(\alpha) \to A$.

1.1.3 WeakKernelEmbeddingWithGivenWeakKernelObject (for IsCapCategoryMorphism, IsCapCategoryObject)

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

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

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

1.1.4 WeakKernelLift (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

Returns: a morphism in $\mathrm{Hom}(T, \mathrm{KernelObject}(\alpha))$

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 $u(\tau) : T \rightarrow \mathrm{KernelObject}(\alpha)$ given by the universal property of the kernel.

1.1.5 WeakKernelLiftWithGivenWeakKernelObject (for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject)

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

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

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 $K = \mathrm{KernelObject}(\alpha)$. The output is the morphism $u(\tau) : T \rightarrow K$ given by the universal property of the 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 operations adds the given function F to the category for the basic operation `KernelObject`. $F : \alpha \mapsto \mathrm{KernelObject}(\alpha)$.

1.1.7 AddWeakKernelEmbedding (for IsCapCategory, IsFunction)

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

Returns: nothing

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

1.1.8 AddWeakKernelEmbeddingWithGivenWeakKernelObject (for IsCapCategory, IsFunction)

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

Returns: nothing

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

1.1.9 AddWeakKernelLift (for IsCapCategory, IsFunction)

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

Returns: nothing

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

1.1.10 AddWeakKernelLiftWithGivenWeakKernelObject (for IsCapCategory, IsFunction)

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

Returns: nothing

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

1.2 Weak cokernel

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

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

The triple (K, ε, u) is called a *cokernel* of α if the morphisms $u(\tau)$ are uniquely determined up to congruence of morphisms. We denote the object K of such a triple by $\mathrm{CokernelObject}(\alpha)$. We say that the morphism $u(\tau)$ is induced by the *universal property of the cokernel*.
CokernelObject is a functorial operation. This means: for $\mu : A \to A'$, $\nu : B \to B'$, $\alpha : A \to B$, $\alpha' : A' \to B'$ such that $\nu \circ \alpha \sim_{A,B'} \alpha' \circ \mu$, we obtain a morphism $\mathrm{CokernelObject}(\alpha) \to \mathrm{CokernelObject}(\alpha')$.

1.2.1 WeakCokernelObject (for IsCapCategoryMorphism)

▷ `WeakCokernelObject(alpha)` (attribute)

Returns: an object

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

1.2.2 WeakCokernelProjection (for IsCapCategoryMorphism)

▷ `WeakCokernelProjection(alpha)` (attribute)

Returns: a morphism in $\mathrm{Hom}(B, \mathrm{CokernelObject}(\alpha))$

The argument is a morphism $\alpha : A \to B$. The output is the cokernel projection $\varepsilon : B \to \mathrm{CokernelObject}(\alpha)$.

1.2.3 WeakCokernelProjectionWithGivenWeakCokernelObject (for IsCapCategoryMorphism, IsCapCategoryObject)

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

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

The arguments are a morphism $\alpha : A \to B$ and an object $K = \mathrm{CokernelObject}(\alpha)$. The output is the cokernel projection $\varepsilon : B \to \mathrm{CokernelObject}(\alpha)$.

1.2.4 WeakCokernelColift (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

Returns: a morphism in $\mathrm{Hom}(\mathrm{CokernelObject}(\alpha), T)$

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

1.2.5 WeakCokernelColiftWithGivenWeakCokernelObject (for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject)

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

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

The arguments are a morphism $\alpha : A \to B$, a test morphism $\tau : B \to T$ satisfying $\tau \circ \alpha \sim_{A,T} 0$, and an object $K = \mathrm{CokernelObject}(\alpha)$. The output is the morphism $u(\tau) : K \to T$ given by the universal property of the 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 operations adds the given function F to the category for the basic operation `CokernelObject`. $F : \alpha \mapsto K$.

1.2.7 AddWeakCokernelProjection (for IsCapCategory, IsFunction)

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

Returns: nothing

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

1.2.8 AddWeakCokernelProjectionWithGivenWeakCokernelObject (for IsCapCategory, IsFunction)

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

Returns: nothing

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

1.2.9 AddWeakCokernelColift (for IsCapCategory, IsFunction)

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

Returns: nothing

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

1.2.10 AddWeakCokernelColiftWithGivenWeakCokernelObject (for IsCapCategory, IsFunction)

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

Returns: nothing

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

1.3 Weak bi-fiber product

1.3.1 AddWeakBiFiberProduct (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `FiberProduct`. $F : ((\beta_i : P_i \rightarrow B)_{i=1\dots n}) \mapsto P$

1.3.2 AddProjectionInFirstFactorOfWeakBiFiberProduct (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `ProjectionInFactorOfFiberProduct`. $F : ((\beta_i : P_i \rightarrow B)_{i=1\dots n}, k) \mapsto \pi_k$

1.3.3 AddProjectionInFirstFactorOfWeakBiFiberProductWithGivenWeakBiFiberProduct (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `ProjectionInFactorOfFiberProductWithGivenFiberProduct`. $F : ((\beta_i : P_i \rightarrow B)_{i=1\dots n}, k, P) \mapsto \pi_k$

1.3.4 AddUniversalMorphismIntoWeakBiFiberProduct (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `UniversalMorphismIntoFiberProduct`. $F : ((\beta_i : P_i \rightarrow B)_{i=1\dots n}, \tau) \mapsto u(\tau)$

1.3.5 AddUniversalMorphismIntoWeakBiFiberProductWithGivenWeakBiFiberProduct (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `UniversalMorphismIntoFiberProductWithGivenFiberProduct`. $F : ((\beta_i : P_i \rightarrow B)_{i=1\dots n}, \tau, P) \mapsto u(\tau)$

1.4 Biased weak fiber product

1.4.1 AddBiasedWeakFiberProduct (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `FiberProduct`. $F : ((\beta_i : P_i \rightarrow B)_{i=1\dots n}) \mapsto P$

1.4.2 AddProjectionOfBiasedWeakFiberProduct (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `ProjectionInFactorOfFiberProduct`. $F : ((\beta_i : P_i \rightarrow B)_{i=1\dots n}, k) \mapsto \pi_k$

1.4.3 AddProjectionOfBiasedWeakFiberProductWithGivenBiasedWeakFiberProduct (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `ProjectionInFactorOfFiberProductWithGivenFiberProduct`. $F : ((\beta_i : P_i \rightarrow B)_{i=1\dots n}, k, P) \mapsto \pi_k$

1.4.4 AddUniversalMorphismIntoBiasedWeakFiberProduct (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `UniversalMorphismIntoFiberProduct`. $F : ((\beta_i : P_i \rightarrow B)_{i=1\dots n}, \tau) \mapsto u(\tau)$

1.4.5 AddUniversalMorphismIntoBiasedWeakFiberProductWithGivenBiasedWeakFiberProduct (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `UniversalMorphismIntoFiberProductWithGivenFiberProduct`. $F : ((\beta_i : P_i \rightarrow B)_{i=1...n}, \tau, P) \mapsto u(\tau)$

1.5 Weak bi-pushout

1.5.1 AddWeakBiPushout (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `Pushout`. $F : ((\beta_i : B \rightarrow I_i)_{i=1...n}) \mapsto I$

1.5.2 AddInjectionOfFirstCofactorOfWeakBiPushout (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `InjectionOfCofactorOfPushout`. $F : ((\beta_i : B \rightarrow I_i)_{i=1...n}, k) \mapsto \iota_k$

1.5.3 AddInjectionOfFirstCofactorOfWeakBiPushoutWithGivenWeakBiPushout (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `InjectionOfCofactorOfPushoutWithGivenPushout`. $F : ((\beta_i : B \rightarrow I_i)_{i=1...n}, k, I) \mapsto \iota_k$

1.5.4 AddUniversalMorphismFromWeakBiPushout (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `UniversalMorphismFromPushout`. $F : ((\beta_i : B \rightarrow I_i)_{i=1...n}, \tau) \mapsto u(\tau)$

1.5.5 AddUniversalMorphismFromWeakBiPushoutWithGivenWeakBiPushout (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `UniversalMorphismFromPushout`. $F : ((\beta_i : B \to I_i)_{i=1\dots n}, \tau, I) \mapsto u(\tau)$

1.6 Biased weak pushout

1.6.1 AddBiasedWeakPushout (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `Pushout`. $F : ((\beta_i : B \to I_i)_{i=1\dots n}) \mapsto I$

1.6.2 AddInjectionOfBiasedWeakPushout (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `InjectionOfCofactorOfPushout`. $F : ((\beta_i : B \to I_i)_{i=1\dots n}, k) \mapsto \iota_k$

1.6.3 AddInjectionOfBiasedWeakPushoutWithGivenBiasedWeakPushout (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `InjectionOfCofactorOfPushoutWithGivenPushout`. $F : ((\beta_i : B \to I_i)_{i=1\dots n}, k, I) \mapsto \iota_k$

1.6.4 AddUniversalMorphismFromBiasedWeakPushout (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `UniversalMorphismFromPushout`. $F : ((\beta_i : B \to I_i)_{i=1\dots n}, \tau) \mapsto u(\tau)$

1.6.5 AddUniversalMorphismFromBiasedWeakPushoutWithGivenBiasedWeakPushout (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `UniversalMorphismFromPushout`. $F : ((\beta_i : B \rightarrow I_i)_{i=1\dots n}, \tau, I) \mapsto u(\tau)$

1.7 Abelian constructions

1.7.1 EpimorphismFromSomeProjectiveObjectForKernelObject (for IsCapCategoryMorphism)

▷ `EpimorphismFromSomeProjectiveObjectForKernelObject(A)` (attribute)

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

The argument is an object A. The output is an epimorphism $\pi : P \rightarrow A$ with P a projective object that equals the output of SomeProjectiveObject(A).

1.7.2 EpimorphismFromSomeProjectiveObjectForKernelObjectWithGivenSomeProjectiveObjectFo (for IsCapCategoryMorphism, IsCapCategoryObject)

▷ `EpimorphismFromSomeProjectiveObjectForKernelObjectWithGivenSomeProjectiveObjectForKernelOb`
`P)` (operation)

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

The arguments are an object A and a projective object P that equals the output of SomeProjectiveObject(A). The output is an epimorphism $\pi : P \rightarrow A$.

1.7.3 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 `SomeProjectiveObject`. $F : A \mapsto P$.

1.7.4 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 `EpimorphismFromSomeProjectiveObject`. $F : A \mapsto \pi$.

1.7.5 AddEpimorphismFromSomeProjectiveObjectForKernelObjectWithGivenSomeProjectiveObje (for IsCapCategory, IsFunction)

▷ `AddEpimorphismFromSomeProjectiveObjectForKernelObjectWithGivenSomeProjectiveObjectForKernel`
`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

AddEpimorphismFromSomeProjectiveObjectWithGivenSomeProjectiveObject. $F : (A,P) \mapsto \pi$.

Chapter 2

Additive closure

Chapter 3

Adelman category

Chapter 4

Category of rows

4.1 GAP Categories

4.1.1 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 5

Category of graded rows and category of graded columns

5.1 Constructors

5.1.1 CAPCategoryOfGradedColumns (for IsHomalgGradedRing)

▷ `CAPCategoryOfGradedColumns(R)` (attribute)

Returns: a category

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

5.1.2 CAPCategoryOfGradedRows (for IsHomalgGradedRing)

▷ `CAPCategoryOfGradedRows(R)` (attribute)

Returns: a category

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

5.1.3 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], ...] 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.

5.1.4 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.

5.1.5 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], ...] 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.

5.1.6 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.

5.1.7 GradedRowOrColumnMorphism (for IsGradedRowOrColumn, IsHomalgMatrix, IsGradedRowOrColumn)

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

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

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 $S \to T$ in the category of graded rows and columns over R, whose underlying matrix is given by M.

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

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

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

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

5.2 GAP Categories

5.2.1 IsGradedRowOrColumn (for IsCapCategoryObject)

▷ `IsGradedRowOrColumn(object)` (filter)

Returns: `true` or `false`

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

5.2.2 IsGradedRow (for IsGradedRowOrColumn)

▷ `IsGradedRow(object)` (filter)

Returns: `true` or `false`

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

5.2.3 IsGradedColumn (for IsGradedRowOrColumn)

▷ `IsGradedColumn(object)` (filter)

Returns: `true` or `false`

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

5.2.4 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.

5.2.5 IsGradedRowMorphism (for IsGradedRowOrColumnMorphism)

▷ `IsGradedRowMorphism(object)` (filter)

Returns: `true` or `false`

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

5.2.6 IsGradedColumnMorphism (for IsGradedRowOrColumnMorphism)

▷ `IsGradedColumnMorphism(object)` (filter)

Returns: `true` or `false`

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

5.3 Attributes

5.3.1 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.

5.3.2 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 redcued 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]].

5.3.3 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.

5.3.4 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.

5.3.5 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.

5.4 Printing all information about a morphism

5.4.1 FullInformation (for IsGradedRowOrColumnMorphism)

▷ `FullInformation(m)` (operation)

Returns: detailed information about the morphism

The argument is a morphism m in the category of graded rows or columns. For such a morphism, this methods will print its source, range and mapping matrix.

5.5 Tools to simplify code

5.5.1 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.

5.5.2 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.

5.5.3 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.

5.5.4 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.

5.5.5 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 multipicity 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 6

Cokernel image closure

Chapter 7

Freyd category

Chapter 8

Examples and Tests

8.1 Adelman category basics

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), UniversalMorphis
>                            );
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( Cok
true
gap> k := KernelEmbedding( c );;
gap> IsZeroForMorphisms( PreCompose( k, c ) );
true
gap> IsCongruentForMorphisms( KernelLift( m, KernelEmbedding( m ) ), IdentityMorphism( KernelObje
true
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
```

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 := InterpretMorphismAsMorphismFromDinstinguishedObjectToHomomorphismStructure( a );;
gap> int := InterpretMorphismFromDinstinguishedObjectToHomomorphismStructureAsMorphism( A, B, mor
gap> IsCongruentForMorphisms( int, a );
true
```

8.2 Adelman snake lemma

Example

```
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> 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
```

8.3 Basics

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( gam
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 := 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> 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 := InterpretMorphismAsMorphismFromDinstinguishedObjectToHomomorphismStructure( gamma );;
gap> IsCongruentForMorphisms( InterpretMorphismFromDinstinguishedObjectToHomomorphismStructureAsM
gap> a := ZeroObjectFunctorial( cat );;
gap> IsCongruentForMorphisms( InterpretMorphismFromDinstinguishedObjectToHomomorphismStructureAsM
gap> Z4 := FreydCategoryObject( AsCategoryOfRowsMorphism( HomalgMatrix( "[4]", 1, 1, R ) ) );;
gap> Z3 := FreydCategoryObject( AsCategoryOfRowsMorphism( HomalgMatrix( "[3]", 1, 1, R ) ) );;
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> Lift( alpha, gamma );;
gap> Colift( lift, alpha );;
gap> IsCongruentForMorphisms( PreCompose( lift, Colift( lift, alpha ) ), alpha );;
gap> lift := FreydCategoryMorphism( S2_freyd, CategoryOfRowsMorphism( S2, HomalgMatrix( "[x,y,z,x
gap> gamma := FreydCategoryMorphism( S3_freyd, CategoryOfRowsMorphism( S3, HomalgMatrix( "[x,y,z,
gap> alpha := PreCompose( lift, gamma );;
gap> Lift( alpha, gamma );;
gap> Colift( lift, alpha );;
gap> IsCongruentForMorphisms( PreCompose( lift, Colift( lift, alpha ) ), alpha );;
gap> interpretation := InterpretMorphismAsMorphismFromDinstinguishedObjectToHomomorphismStructure
gap> IsCongruentForMorphisms( gamma,
> InterpretMorphismFromDinstinguishedObjectToHomomorphismStructureAsMorphism( Source( gamma ), Ra
gap> ## Opposite
> HomomorphismStructureOnObjects( Opposite( Z4 ), Opposite( Z2 ) );;
gap> HomomorphismStructureOnObjects( Z2, Z4 );;
gap> interpretation := InterpretMorphismAsMorphismFromDinstinguishedObjectToHomomorphismStructure
gap> IsCongruentForMorphisms( Opposite( gamma ),
```

```
> InterpretMorphismFromDinstinguishedObjectToHomomorphismStructureAsMorphism( Source( Opposite( g
true
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> SetIsAbCategory( Aoid, true );;
gap> s := SetOfObjects( Aoid );;
gap> m := SetOfGeneratingMorphisms( Aoid );;
gap> interpretation := InterpretMorphismAsMorphismFromDinstinguishedObjectToHomomorphismStructure
gap> InterpretMorphismFromDinstinguishedObjectToHomomorphismStructureAsMorphism( Source( 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 := InterpretMorphismAsMorphismFromDinstinguishedObjectToHomomorphismStructure
gap> IsCongruentForMorphisms(
>   InterpretMorphismFromDinstinguishedObjectToHomomorphismStructureAsMorphism( Source( mor ), Ra
>   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
```

8.4 Cokernel image closure

Example

```
gap> R := HomalgFieldOfRationalsInSingular() * "x,y,z";;
gap> RowsR := CategoryOfRows( R );;
gap> m := AsCategoryOfRowsMorphism(
>     HomalgMatrix( "[[x],[y],[z]]", 3, 1, R )
> );;
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 )
> );;
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( CoastrictionToImage( nu ), ImageEmbedding( nu ) ) )
true
gap> u := UniversalMorphismFromImage( nu, [ nu, IdentityMorphism( Range( nu ) ) ] );;
gap> IsWellDefined( u );
true
gap> IsCongruentForMorphisms( nu, PreCompose( CoastrictionToImage( 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, kerne
> );
```

```
true
```

8.5 Homomorphisms between f.p. functors

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
```

8.6 Snake lemma first proof

Example

```
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> SetIsAbCategory( Aoid, true );;
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> cat := AdditiveClosure( cat );;
gap> cat := Opposite( 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 );;
```

8.7 Snake lemma second proof

Example

```
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> SetIsAbCategory( Aoid, true );;
gap> m := SetOfGeneratingMorphisms( Aoid );;
gap> a := m[1];;
gap> b := m[2];;
gap> c := m[3];;
gap> cat := Aoid;;
gap> CapCategorySwitchLogicOff( cat );;
gap> cat := AdditiveClosure( cat );;
gap> cat := Opposite( 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
```

8.8 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 9

Examples on graded rows and columns

9.1 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 occurances 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

$$\varphi : A \to B$$

given by a homomgeneous 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) occurances 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
```

9.2 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 the 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> 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)
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 the 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> 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)
```

9.3 The GAP categories

Example

```
gap> categoryL := CapCategory( Q1L );
CAP 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 );
CAP category of graded columns over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])
```

9.4 A few categorical constructions for graded rows

Example

```
gap> ZeroObject( categoryL );
<A graded row of rank 0>
```

```
gap> O1L := GradedRow( [ [[-1,0],2] ], S );
<A graded row of rank 2>
gap> Display( ZeroMorphism( ZeroObject( categoryL ), O1L ) );
A morphism in the category of graded rows over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [0, 1 ] ])
with matrix:
(an empty 0 x 2 matrix)
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>
gap> Display( IdentityMorphism( O2L ) );
A morphism in the category of graded rows over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [0, 1 ] ])
with matrix:
1
(over a graded ring)
gap> IsWellDefined( IdentityMorphism( Q2L ) );
true
gap> directSumL := DirectSum( [ O1L, O2L ] );
<A graded row of rank 3>
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 ] ]
gap> i1L := InjectionOfCofactorOfDirectSum( [ O1L, O2L ], 1 );
<A morphism in the 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
gap> Display( UnderlyingHomalgMatrix( i1L ) );
1,0,0,
0,1,0
(over a graded ring)
gap> i2L := InjectionOfCofactorOfDirectSum( [ O1L, O2L ], 2 );
<A morphism in the 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
gap> Display( UnderlyingHomalgMatrix( i2L ) );
0,0,1
(over a graded ring)
gap> proj1L := ProjectionInFactorOfDirectSum( [ O1L, O2L ], 1 );
<A morphism in the 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
gap> Display( UnderlyingHomalgMatrix( proj1L ) );
1,0,
0,1,
0,0
(over a graded ring)
```

```
gap> proj2L := ProjectionInFactorOfDirectSum( [ O1L, O2L ], 2 );
<A morphism in the 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
gap> Display( UnderlyingHomalgMatrix( proj2L ) );
0,
0,
1
(over a graded ring)
gap> kL := WeakKernelEmbedding( proj1L );
<A morphism in the 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
gap> Display( UnderlyingHomalgMatrix( kL ) );
0,0,1
(over a graded ring)
gap> ckL := WeakCokernelProjection( kL );
<A morphism in the 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
gap> Display( UnderlyingHomalgMatrix( ckL ) );
1,0,
0,1,
0,0
(over a graded ring)
gap> IsMonomorphism( kL );
true
gap> IsEpimorphism( kL );
false
gap> IsMonomorphism( ckL );
false
gap> IsEpimorphism( ckL );
true
gap> m1L := GradedRowOrColumnMorphism( O1L,
>       HomalgMatrix( [[ "x_1" ], [ "x_2" ]], S ), O2L );
<A morphism in the 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( O2L );
<A morphism in the 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 ), O2L );
<A morphism in the 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 ), O2L );
<A morphism in the 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 the 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
gap> Display( UnderlyingHomalgMatrix( liftL ) );
x_1, 0
(over a graded ring)
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 the 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 the 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
gap> Display( UnderlyingHomalgMatrix( coliftL ) );
x_1,x_2
(over a graded ring)
gap> fpL := WeakBiFiberProduct( m1L, m2L );
<A graded row of rank 2>
gap> fp_proj1L := ProjectionInFirstFactorOfWeakBiFiberProduct( m1L, m2L );
<A morphism in the 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
gap> Display( UnderlyingHomalgMatrix( fp_proj1L ) );
1,0,
0,1
(over a graded ring)
gap> fp_proj2L := ProjectionInSecondFactorOfWeakBiFiberProduct( m1L, m2L );
<A morphism in the 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
gap> Display( UnderlyingHomalgMatrix( fp_proj2L ) );
1,
x_2
```

```
(over a graded ring)
gap> BiasedWeakFiberProduct( m1L, m2L );
<A graded row of rank 2>
gap> pbwfprow := ProjectionOfBiasedWeakFiberProduct( m1L, m2L );
<A morphism in the 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
gap> Display( pbwfprow );
A morphism in the category of graded rows over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])
with matrix:
1,0,
0,1
(over a graded ring)
gap> poL := WeakBiPushout( morL, m2L );
<A graded row of rank 2>
gap> inj1L := InjectionOfFirstCofactorOfWeakBiPushout( morL, m2L );
<A morphism in the 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
gap> Display( UnderlyingHomalgMatrix( inj1L ) );
1,0,
0,1
(over a graded ring)
gap> inj2L := InjectionOfSecondCofactorOfWeakBiPushout( morL, m2L );
<A morphism in the 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
gap> Display( UnderlyingHomalgMatrix( inj2L ) );
x_1,x_2
(over a graded ring)
gap> injectionL := InjectionOfBiasedWeakPushout( morL, m2L );
<A morphism in the 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
gap> Display( injectionL );
A morphism in the category of graded rows over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])
with matrix:
1,0,
0,1
(over a graded ring)
gap> tensorProductL := TensorProductOnObjects( O1L, O2L );
<A graded row of rank 2>
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 ] ]
```

```
gap> tensorProductMorphismL := TensorProductOnMorphisms( m2L, morL );
<A morphism in the 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
gap> Display( tensorProductMorphismL );
A morphism in the category of graded rows over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [0, 1 ] ])
with matrix:
x_1,x_2
(over a graded ring)
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 ] ]
gap> IsWellDefined( DualOnMorphisms( m1L ) );
true
gap> Display( DualOnMorphisms( m1L ) );
A morphism in the category of graded rows over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [0, 1 ] ])
with matrix:
1,x_2
(over a graded ring)
gap> IsWellDefined( EvaluationForDualWithGivenTensorProduct( TensorProductOnObjects(
> DualOnObjects( ObjectL ), ObjectL ), ObjectL, TensorUnit( categoryL ) ) );
true
gap> Display( EvaluationForDualWithGivenTensorProduct( TensorProductOnObjects(
> DualOnObjects( ObjectL ), ObjectL ), ObjectL, TensorUnit( categoryL ) ) );
A morphism in the category of graded rows over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [0, 1 ] ])
with matrix:
1,
0,
0,
1
(over a graded ring)
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 ] ]
```

9.5 A few categorical constructions for graded columns

Example

```
gap> ZeroObject( categoryR );
<A graded column of rank 0>
gap> O1R := GradedColumn( [ [[-1,0],2] ], S );
<A graded column of rank 2>
gap> Display( ZeroMorphism( ZeroObject( categoryR ), O1R ) );
A morphism in the category of graded columns over
```

```
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])
with matrix:
(an empty 2 x 0 matrix)
gap> O2R := GradedColumn( [ [[0,0],1] ], S );
<A graded column of rank 1>
gap> obj3R := GradedColumn( [ [[-1,0],1] ], S );
<A graded column of rank 1>
gap> Display( IdentityMorphism( O2R ) );
A morphism in the category of graded columns over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])
with matrix:
1
(over a graded ring)
gap> IsWellDefined( IdentityMorphism( Q2R ) );
true
gap> directSumR := DirectSum( [ O1R, O2R ] );
<A graded column of rank 3>
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 ] ]
gap> i1R := InjectionOfCofactorOfDirectSum( [ O1R, O2R ], 1 );
<A morphism in the 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
gap> Display( UnderlyingHomalgMatrix( i1R ) );
1,0,
0,1,
0,0
(over a graded ring)
gap> i2R := InjectionOfCofactorOfDirectSum( [ O1R, O2R ], 2 );
<A morphism in the 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
gap> Display( UnderlyingHomalgMatrix( i2R ) );
0,
0,
1
(over a graded ring)
gap> proj1R := ProjectionInFactorOfDirectSum( [ O1R, O2R ], 1 );
<A morphism in the 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
gap> Display( UnderlyingHomalgMatrix( proj1R ) );
1,0,0,
0,1,0
(over a graded ring)
gap> proj2R := ProjectionInFactorOfDirectSum( [ O1R, O2R ], 2 );
<A morphism in the 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
gap> Display( UnderlyingHomalgMatrix( proj2R ) );
0,0,1
(over a graded ring)
gap> kR := WeakKernelEmbedding( proj1R );
<A morphism in the 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
gap> Display( UnderlyingHomalgMatrix( kR ) );
0,
0,
1
(over a graded ring)
gap> ckR := WeakCokernelProjection( kR );
<A morphism in the 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
gap> Display( UnderlyingHomalgMatrix( ckR ) );
1,0,0,
0,1,0
(over a graded ring)
gap> IsMonomorphism( kR );
true
gap> IsEpimorphism( kR );
false
gap> IsMonomorphism( ckR );
false
gap> IsEpimorphism( ckR );
true
gap> m1R := GradedRowOrColumnMorphism( O1R,
>       HomalgMatrix( [[ "x_1", "x_2" ]], S ), O2R );
<A morphism in the 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( O2R );
<A morphism in the 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 ), O2R );
<A morphism in the 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 ), O2R );
<A morphism in the 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 the 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
gap> Display( UnderlyingHomalgMatrix( liftR ) );
x_1,
0
(over a graded ring)
gap> O3R := GradedColumn( [ [[1,0],2] ], S );
<A graded column of rank 2>
gap> morR := GradedRowOrColumnMorphism(
>        O2R, HomalgMatrix( [[ "x_1" ], [ "x_2" ]], S ), O3R );
<A morphism in the 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 the 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
gap> Display( UnderlyingHomalgMatrix( coliftR ) );
x_1,
x_2
(over a graded ring)
gap> fpR := WeakBiFiberProduct( m1R, m2R );
<A graded column of rank 2>
gap> fp_proj1R := ProjectionInFirstFactorOfWeakBiFiberProduct( m1R, m2R );
<A morphism in the 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
gap> Display( UnderlyingHomalgMatrix( fp_proj1R ) );
1,0,
0,1
(over a graded ring)
gap> fp_proj2R := ProjectionInSecondFactorOfWeakBiFiberProduct( m1R, m2R );
<A morphism in the 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
gap> Display( UnderlyingHomalgMatrix( fp_proj2R ) );
1, x_2
(over a graded ring)
gap> BiasedWeakFiberProduct( m1R, m2R );
```

```
<A graded column of rank 2>
gap> pbwfpcol := ProjectionOfBiasedWeakFiberProduct( m1R, m2R );
<A morphism in the 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
gap> Display( pbwfpcol );
A morphism in the category of graded columns over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])
with matrix:
1,0,
0,1
(over a graded ring)
gap> poR := WeakBiPushout( morR, m2R );
<A graded column of rank 2>
gap> inj1R := InjectionOfFirstCofactorOfWeakBiPushout( morR, m2R );
<A morphism in the 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( inj1R );
true
gap> Display( UnderlyingHomalgMatrix( inj1R ) );
1,0,
0,1
(over a graded ring)
gap> inj2R := InjectionOfSecondCofactorOfWeakBiPushout( morR, m2R );
<A morphism in the 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( inj2R );
true
gap> Display( UnderlyingHomalgMatrix( inj2R ) );
x_1,
x_2
(over a graded ring)
gap> injectionR := InjectionOfBiasedWeakPushout( morR, m2R );
<A morphism in the 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( injectionR );
true
gap> Display( injectionR );
A morphism in the category of graded columns over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])
with matrix:
1,0,
0,1
(over a graded ring)
gap> tensorProductR := TensorProductOnObjects( O1R, O2R );
<A graded column of rank 2>
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 ] ]
gap> tensorProductMorphismR := TensorProductOnMorphisms( m2R, morR );
```

```
<A morphism in the 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
gap> Display( tensorProductMorphismR );
A morphism in the category of graded columns over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [0, 1 ] ])
with matrix:
x_1,
x_2
(over a graded ring)
gap> Display( DualOnObjects( TensorProductOnObjects( ObjectR, Object2R ) ) );
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 16 and degrees:
[ [ ( -2, 0 ), 5 ], [ ( -1, -1 ), 2 ], [ ( -2, 0 ), 6 ], [ ( -1, -1 ), 2 ],
[ ( -2, 0 ), 1 ] ]
gap> IsWellDefined( DualOnMorphisms( m1R ) );
true
gap> Display( DualOnMorphisms( m1R ) );
A morphism in the category of graded columns over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])
with matrix:
1,
x_2
(over a graded ring)
gap> IsWellDefined( EvaluationForDualWithGivenTensorProduct( TensorProductOnObjects(
> DualOnObjects( ObjectR ), ObjectR ), ObjectR, TensorUnit( categoryR ) ) );
true
gap> Display( EvaluationForDualWithGivenTensorProduct( TensorProductOnObjects(
> DualOnObjects( ObjectR ), ObjectR ), ObjectR, TensorUnit( categoryR ) ) );
A morphism in the category of graded columns over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [0, 1 ] ])
with matrix:
1,0,0,1
(over a graded ring)
gap> Display( InternalHomOnObjects( ObjectR, ObjectR ) );
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 4 and degrees:
[ [ 0, 4 ] ]
```

9.6 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 the 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( coevR );
```

```
true
gap> evalR := EvaluationForDual( bR );
<A morphism in the 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( 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 the 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( phiR );
true
gap> tens_mor := TensorProductToInternalHomAdjunctionMap(aR,bR,phiR);
<A morphism in the 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( tens_mor );
true
```

9.7 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 the 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( coevC );
true
gap> evalC := EvaluationForDual( bC );
<A morphism in the 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 the 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 the 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
```

9.8 FreydCategory for graded rows

Example

```
gap> cat := CAPCategoryOfGradedRows( S );
CAP 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 the 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 );
Freyd( CAP 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> IsAbelianCategory( freyd );
true
gap> obj_gamma := FreydCategoryObject( gamma );
<An object in Freyd( CAP 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> f := FreydCategoryMorphism( obj_gamma, gamma, obj_gamma );
<A morphism in Freyd( CAP 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> witness := MorphismWitness( f );
<A morphism in the 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> Display( witness );
A morphism in the category of graded rows over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ]) with matrix:
2,0,
2,0
(over a graded ring)
gap> g := FreydCategoryMorphism( obj_gamma,
>                     ZeroMorphism( obj2, obj2 ), obj_gamma );
<A morphism in Freyd( CAP 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> IsCongruentForMorphisms( f, g );
true
gap> c := PreCompose( f, f );
<A morphism in Freyd( CAP 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> Display( c );
Morphism datum:
A morphism in the category of graded rows over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ]) with matrix:
2,2,
2,2
(over a graded ring)
```

```
gap> s := g + g;
<A morphism in Freyd( CAP 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> Display( s );
Morphism datum:
A morphism in the category of graded rows over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ]) with matrix:
0,0,
0,0
(over a graded ring)
gap> a := GradedRowOrColumnMorphism( obj1,
>                                HomalgMatrix( [ [ 2 ] ], 1, 1, S ), obj1 );
<A morphism in the 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> Display( a );
A morphism in the category of graded rows over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ]) with matrix:
2
(over a graded ring)
gap> Z2 := FreydCategoryObject( a );
<An object in Freyd( CAP 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> Display( Z2 );
Relation morphism:
A morphism in the category of graded rows over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ]) with matrix:
2
(over a graded ring)
gap> id := IdentityMorphism( Z2 );
<An identity morphism in Freyd( CAP 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> z := id + id + id;
<A morphism in Freyd( CAP 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> Display( z );
Morphism datum:
A morphism in the category of graded rows over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ]) with matrix:
3
(over a graded ring)
gap> d := DirectSumFunctorial( [ z, z, z ] );
<A morphism in Freyd( CAP 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> Display( d );
Morphism datum:
A morphism in the category of graded rows over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ]) with matrix:
3,0,0,
0,3,0,
0,0,3
(over a graded ring)
gap> pr2 := ProjectionInFactorOfDirectSum( [ Z2, Z2, Z2 ], 2 );
```

```
<A morphism in Freyd( CAP 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> pr3 := ProjectionInFactorOfDirectSum( [ Z2, Z2, Z2 ], 3 );
<A morphism in Freyd( CAP 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> Display( UniversalMorphismIntoDirectSum( [ pr3, pr2 ] ) );
Morphism datum:
A morphism in the category of graded rows over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ]) with matrix:
0,0,
0,1,
1,0
(over a graded ring)
gap> inj1 := InjectionOfCofactorOfDirectSum( [ Z2, Z2, Z2 ], 1 );
<A morphism in Freyd( CAP 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> inj2 := InjectionOfCofactorOfDirectSum( [ Z2, Z2, Z2 ], 2 );
<A morphism in Freyd( CAP 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> Display( UniversalMorphismFromDirectSum( [ inj2, inj1 ] ) );
Morphism datum:
A morphism in the category of graded rows over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ]) with matrix:
0,1,0,
1,0,0
(over a graded ring)
gap> ZFree := AsFreydCategoryObject( obj1 );
<A projective object in Freyd( CAP 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> Display( ZFree );
Relation morphism:
A morphism in the category of graded rows over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ]) with matrix:
(an empty 0 x 1 matrix)
gap> id := IdentityMorphism( ZFree );
<An identity morphism in Freyd( CAP 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> z := id + id;
<A morphism in Freyd( CAP 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> Display( z );
Morphism datum:
A morphism in the category of graded rows over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ]) with matrix:
2
(over a graded ring)
gap> Display( CokernelProjection( z ) );
Morphism datum:
A morphism in the category of graded rows over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ]) with matrix:
1
(over a graded ring)
```

```
gap> Display( CokernelColift( z, CokernelProjection( z ) ) );
Morphism datum:
A morphism in the category of graded rows over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ]) with matrix:
1
(over a graded ring)
```

Index