

CAP

Categories, Algorithms, Programming

2017.07.25

25 July 2017

Sebastian Gutsche

Sebastian Posur

Øystein Skartsæterhagen

Sebastian Gutsche

Email: gutsche@mathematik.uni-siegen.de

Homepage: <http://www.uni-siegen.de/fb6/rmi/>

Address: Department Mathematik
Universität Siegen
Walter-Flex-Straße 3
57068 Siegen
Germany

Sebastian Posur

Email: sebastian.posur@uni-siegen.de

Homepage: <http://www.uni-siegen.de/fb6/rmi/>

Address: Department Mathematik
Universität Siegen
Walter-Flex-Straße 3
57068 Siegen
Germany

Øystein Skartsæterhagen

Email: oysteini@math.ntnu.no

Homepage: <http://www.math.ntnu.no/~oysteini/>

Address: NTNU
Institutt for matematiske fag
7491 Trondheim
Norway

Contents

1	CAP Categories	5
1.1	Categories	5
1.2	Constructor	6
1.3	Internal Attributes	6
1.4	Logic switcher	7
1.5	Tool functions	7
1.6	Well-Definedness of Cells	8
1.7	Type check	8
2	Category of Categories	9
2.1	The Category Cat	9
2.2	Categories	9
2.3	Constructors	10
2.4	Functors	10
2.5	Natural transformations	12
3	Morphisms	14
3.1	Attributes for the Type of Morphisms	14
3.2	Categorical Properties of Morphisms	14
3.3	Non-Categorical Properties of Morphisms	15
3.4	Equality and Congruence for Morphisms	17
3.5	Basic Operations for Morphisms in Ab-Categories	18
3.6	Subobject and Factorobject Operations	19
3.7	Identity Morphism and Composition of Morphisms	20
3.8	Well-Definedness of Morphisms	21
3.9	Basic Operations for Morphisms in Abelian Categories	22
3.10	Lift/ Colift	22
3.11	Inverses	23
3.12	Tool functions for caches	24
4	Objects	25
4.1	Attributes for the Type of Objects	25
4.2	Equality for Objects	25
4.3	Categorical Properties of Objects	26
4.4	Tool functions for caches	27
4.5	Well-Definedness of Objects	27

5	Category 2-Cells	28
5.1	Attributes for the Type of 2-Cells	28
5.2	Identity 2-Cell and Composition of 2-Cells	28
5.3	Well-Definedness for 2-Cells	30
6	Universal Objects	31
6.1	Kernel	31
6.2	Cokernel	33
6.3	Zero Object	36
6.4	Terminal Object	40
6.5	Initial Object	42
6.6	Direct Sum	43
6.7	Coproduct	50
6.8	Direct Product	54
6.9	Fiber Product	57
6.10	Pushout	62
6.11	Image	68
6.12	Coimage	71
7	Tensor Product and Internal Hom	77
7.1	Monoidal Categories	77
7.2	Braided Monoidal Categories	83
7.3	Symmetric Monoidal Categories	85
7.4	Symmetric Closed Monoidal Categories	85
7.5	Rigid Symmetric Closed Monoidal Categories	97
8	Managing Derived Methods	99
8.1	Info Class	99
8.2	Derivation Objects	99
8.3	Derivation Graphs	101
8.4	Managing Derivations in a Category	104
8.5	Min Heaps for Strings	106
9	Add Functions	108
9.1	Functions Installed by Add	108
9.2	Add Method	109
9.3	InstallAdd Function	110
9.4	Install All Adds	111
10	Technical Details	113
10.1	The Category Cat	113
10.2	Install Functions for IsWellDefined	113
11	Examples and Tests	116
11.1	Spectral Sequences	116
11.2	Monoidal Categories	119
11.3	Generalized Morphisms Category	121
11.4	IsWellDefined	128

11.5 Kernel 128
11.6 FiberProduct 130

Index **132**

Chapter 1

CAP Categories

Categories are the main GAP objects in CAP. They are used to associate GAP objects which represent objects and morphisms with their category. By associating a GAP object to the category, one of two filters belonging to the category (`ObjectFilter/MorphismFilter`) are set to true. Via `Add` methods, functions for specific existential quantifiers can be associated to the category and after that can be applied to GAP objects in the category. A GAP category object also knows which constructions are currently possible in this category.

1.1 Categories

1.1.1 `IsCapCategory` (for `IsObject`)

▷ `IsCapCategory(object)` (filter)

Returns: true or false

The GAP category of CAP categories. Objects of this type handle the CAP category information, the caching, and filters for objects in the CAP category. Please note that the object itself is not related to methods, you only need it as a handler and a presentation of the CAP category.

1.1.2 `IsCapCategoryCell` (for `IsObject`)

▷ `IsCapCategoryCell(object)` (filter)

Returns: true or false

The GAP category of CAP category cells. Every object, morphism, and 2-cell of a CAP category lies in this GAP category.

1.1.3 `IsCapCategoryObject` (for `IsCapCategoryCell`)

▷ `IsCapCategoryObject(object)` (filter)

Returns: true or false

The GAP category of CAP category objects. Every object of a CAP category lies in this GAP category.

1.1.4 IsCapCategoryMorphism (for IsCapCategoryCell)

▷ IsCapCategoryMorphism(*object*) (filter)

Returns: true or false

The GAP category of CAP category morphisms. Every morphism of a CAP category lies in this GAP category.

1.1.5 IsCapCategoryTwoCell (for IsCapCategoryCell)

▷ IsCapCategoryTwoCell(*object*) (filter)

Returns: true or false

The GAP category of CAP category 2-cells. Every 2-cell of a CAP category lies in this GAP category.

1.2 Constructor

1.2.1 CreateCapCategory

▷ CreateCapCategory() (operation)

Returns: a category

Creates a new CAP category from scratch. It gets a generic name.

1.2.2 CreateCapCategory (for IsString)

▷ CreateCapCategory(*s*) (operation)

Returns: a category

The argument is a string *s*. This operation creates a new CAP category from scratch. Its name is set to *s*.

1.3 Internal Attributes

Each category *C* stores various filters. They are used to apply the right functions in the method selection.

1.3.1 CategoryFilter (for IsCapCategory)

▷ CategoryFilter(*C*) (attribute)

Returns: a filter

The argument is a category *C*. The output is a filter in which *C* lies.

1.3.2 CellFilter (for IsCapCategory)

▷ CellFilter(*C*) (attribute)

Returns: a filter

The argument is a category *C*. The output is a filter in which all cells of *C* shall lie.

1.3.3 ObjectFilter (for IsCapCategory)

- ▷ ObjectFilter(C) (attribute)
Returns: a filter
 The argument is a category C . The output is a filter in which all objects of C shall lie.

1.3.4 MorphismFilter (for IsCapCategory)

- ▷ MorphismFilter(C) (attribute)
Returns: a filter
 The argument is a category C . The output is a filter in which all morphisms of C shall lie.

1.3.5 TwoCellFilter (for IsCapCategory)

- ▷ TwoCellFilter(C) (attribute)
Returns: a filter
 The argument is a category C . The output is a filter in which all 2-cells of C shall lie.

1.4 Logic switcher

1.4.1 CapCategorySwitchLogicOn

- ▷ CapCategorySwitchLogicOn(C) (function)
Returns:
 Activates the predicate implication logic for the category C .

1.4.2 CapCategorySwitchLogicOff

- ▷ CapCategorySwitchLogicOff(C) (function)
Returns:
 Deactivates the predicate implication logic for the category C .

1.5 Tool functions

1.5.1 CanCompute (for IsCapCategory, IsString)

- ▷ CanCompute(C, s) (operation)
Returns: true or false
 The argument is a category C and a string s , which should be the name of a primitive operation, e.g., PreCompose. If applying this method is possible in C , the method returns true, false otherwise. If the string is not the name of a primitive operation, an error is raised.

1.5.2 CheckConstructivenessOfCategory (for IsCapCategory, IsString)

- ▷ CheckConstructivenessOfCategory(C, s) (operation)
Returns: a list

The arguments are a category C and a string s . If s is a categorical property (e.g. "IsAbelianCategory"), the output is a list of strings with basic operations which are missing in C to have the categorical property constructively. If s is not a categorical property, an error is raised.

1.6 Well-Definedness of Cells

1.6.1 IsWellDefined (for IsCapCategoryCell)

▷ `IsWellDefined(c)` (property)

Returns: a boolean

The argument is a cell c . The output is `true` if c is well-defined, otherwise the output is `false`.

1.7 Type check

1.7.1 DisableBasicOperationTypeCheck

▷ `DisableBasicOperationTypeCheck(category)` (function)

▷ `EnableBasicOperationTypeCheck(arg)` (function)

Returns:

Most basic operations have a prefunction, which does a (sometimes partial) typecheck at the beginning of the operation. These functions enable or disable this check for a category. (Enabled by default)

Chapter 2

Category of Categories

Categories itself with functors as morphisms form a category. So the data structure of `CapCategory`s is designed to be objects in a category. This category is implemented in `CapCat`. For every category, the corresponding object in `Cat` can be obtained via `AsCatObject`. The implementation of the category of categories offers a data structure for functors. Those are implemented as morphisms in this category, so functors can be handled like morphisms in a category. Also convenience functions to install functors as methods are implemented (in order to avoid `ApplyFunctor`).

2.1 The Category Cat

2.1.1 CapCat

▷ `CapCat` (global variable)

This variable stores the category of categories. Every category object is constructed as an object in this category, so `Cat` is constructed when loading the package.

2.2 Categories

2.2.1 IsCapCategoryAsCatObject (for IsCapCategoryObject)

▷ `IsCapCategoryAsCatObject(object)` (filter)

Returns: true or false

The GAP category of CAP categories seen as object in `Cat`.

2.2.2 IsCapFunctor (for IsCapCategoryMorphism)

▷ `IsCapFunctor(object)` (filter)

Returns: true or false

The GAP category of functors.

2.2.3 IsCapNaturalTransformation (for IsCapCategoryTwoCell)

▷ `IsCapNaturalTransformation(object)` (filter)

Returns: true or false

The GAP category of natural transformations.

2.3 Constructors

2.3.1 AsCatObject (for IsCapCategory)

▷ `AsCatObject(C)` (attribute)

Returns:

Given a CAP category C , this method returns the corresponding object in `Cat`. For technical reasons, the filter `IsCapCategory` must not imply the filter `IsCapCategoryObject`. For example, if `InitialObject` is applied to an object, it returns the initial object of its category. If it is applied to a category, it returns the initial object of the category. If a CAP category would be a category object itself, this would be ambiguous. So categories must be wrapped in a `CatObject` to be an object in `Cat`. This method returns the wrapper object. The category can be reobtained by `AsCapCategory`.

2.3.2 AsCapCategory (for IsCapCategoryAsCatObject)

▷ `AsCapCategory(C)` (attribute)

Returns:

For an object C in `Cat`, this method returns the underlying CAP category. This method is inverse to `AsCatObject`, i.e. `AsCapCategory(AsCatObject(A)) = A`.

2.4 Functors

Functors are morphisms in `Cat`, thus they have source and target which are categories. A multivariate functor can be constructed via a product category as source, a presheaf is constructed via the opposite category as source. Moreover, an object and a morphism function can be added to a functor, to apply it to objects or morphisms in the source category.

2.4.1 CapFunctor (for IsString, IsCapCategory, IsCapCategory)

▷ `CapFunctor(name, A, B)` (operation)

▷ `CapFunctor(name, A, B)` (operation)

▷ `CapFunctor(name, A, B)` (operation)

▷ `CapFunctor(name, A, B)` (operation)

▷ `CapFunctor(name, A, B)` (operation)

▷ `CapFunctor(name, A, B)` (operation)

Returns:

These methods construct a CAP functor, i.e. a morphism in `Cat`. Name should be a unique name for the functor, it is also used when the functor is installed as a method. A and B are source and target. Both can be given as object in `Cat` or as category itself.

2.4.2 AddObjectFunction (for IsCapFunctor, IsFunction)

▷ AddObjectFunction(*functor*, *function*) (operation)

Returns:

This operation adds a function to the functor which can then be applied to objects in the source. The given function *function* has to take one argument which must be an object in the source category and should return a CapCategoryObject. The object is automatically added to the range of the functor when it is applied to the object.

2.4.3 FunctorObjectOperation (for IsCapFunctor)

▷ FunctorObjectOperation(*F*) (attribute)

Returns: a GAP operation

The argument is a functor *F*. The output is the GAP operation realizing the action of *F* on objects.

2.4.4 AddMorphismFunction (for IsCapFunctor, IsFunction)

▷ AddMorphismFunction(*functor*, *function*) (operation)

Returns:

This operation adds a function to the functor which can then be applied to morphisms in the source. The given function *function* has to take three arguments *A*, τ , *B*. When the functor *functor* is applied to the morphism τ , *A* is the result of *functor* applied to the source of τ , *B* is the result of *functor* applied to the range.

2.4.5 FunctorMorphismOperation (for IsCapFunctor)

▷ FunctorMorphismOperation(*F*) (attribute)

Returns: a GAP operation

The argument is a functor *F*. The output is the GAP operation realizing the action of *F* on morphisms.

2.4.6 ApplyFunctor

▷ ApplyFunctor(*func*, *A*) (function)

Returns: IsCapCategoryCell

Applies the functor *func* to the object or morphism *A*.

2.4.7 InstallFunctor (for IsCapFunctor, IsString)

▷ InstallFunctor(*functor*, *method_name*) (operation)

Returns:

TODO

2.4.8 IdentityFunctor (for IsCapCategory)

▷ IdentityFunctor(*category*) (attribute)

Returns: a functor

Returns the identity functor of the category *cat* viewed as an object in the category of categories.

2.4.9 FunctorCanonicalizeZeroObjects (for IsCapCategory)

▷ `FunctorCanonicalizeZeroObjects(category)` (attribute)

Returns: a functor

Returns the endofunctor of the category `cat` with zero which maps each (object isomorphic to the) zero object to `ZeroObject(cat)` and to itself otherwise. This functor is equivalent to the identity functor.

2.4.10 NaturalIsomorphismFromIdentityToCanonicalizeZeroObjects (for IsCap-Category)

▷ `NaturalIsomorphismFromIdentityToCanonicalizeZeroObjects(category)` (attribute)

Returns: a natural transformation

Returns the natural isomorphism from the identity functor to `FunctorCanonicalizeZeroObjects(cat)`.

2.4.11 FunctorCanonicalizeZeroMorphisms (for IsCapCategory)

▷ `FunctorCanonicalizeZeroMorphisms(category)` (attribute)

Returns: a functor

Returns the endofunctor of the category `cat` with zero which maps each object to itself, each morphism ϕ to itself, unless it is congruent to the zero morphism; in this case it is mapped to `ZeroMorphism(Source(ϕ), Range(ϕ))`. This functor is equivalent to the identity functor.

2.4.12 NaturalIsomorphismFromIdentityToCanonicalizeZeroMorphisms (for Is-CapCategory)

▷ `NaturalIsomorphismFromIdentityToCanonicalizeZeroMorphisms(category)` (attribute)

Returns: a natural transformation

Returns the natural isomorphism from the identity functor to `FunctorCanonicalizeZeroMorphisms(cat)`.

2.5 Natural transformations

2.5.1 Name (for IsCapNaturalTransformation)

▷ `Name(arg)` (attribute)

Returns: a string

As every functor, every natural transformation has a name attribute. It has to be a string and will be set by the Constructor.

2.5.2 NaturalTransformation (for IsCapFunctor, IsCapFunctor)

▷ `NaturalTransformation([name,]F, G)` (operation)

Returns: a natural transformation

Constructs a natural transformation between the functors $F: A \rightarrow B$ and $G: A \rightarrow B$. The string `name` is optional, and, if not given, set automatically from the names of the functors

2.5.3 AddNaturalTransformationFunction (for IsCapNaturalTransformation, IsFunction)

▷ AddNaturalTransformationFunction(N , $func$) (operation)

Returns:

Adds the function (or list of functions) $func$ to the natural transformation N . The function or each function in the list should take three arguments. If $N : F \rightarrow G$, the arguments should be $F(A), A, G(A)$. The output should be a morphism, $F(A) \rightarrow G(A)$.

2.5.4 ApplyNaturalTransformation

▷ ApplyNaturalTransformation(N , A) (function)

Returns: a morphism

Given a natural transformation $N : F \rightarrow G$ and an object A , this function should return the morphism $F(A) \rightarrow G(A)$, corresponding to N .

2.5.5 InstallNaturalTransformation (for IsCapNaturalTransformation, IsString)

▷ InstallNaturalTransformation(N , $name$) (operation)

Returns:

Installs the natural transformation N as operation with the name $name$. Argument for this operation is an object, output is a morphism.

2.5.6 HorizontalPreComposeNaturalTransformationWithFunctor (for IsCapNaturalTransformation, IsCapFunctor)

▷ HorizontalPreComposeNaturalTransformationWithFunctor(N , F) (operation)

Returns: a natural transformation

Computes the horizontal composition of the natural transformation N and

2.5.7 HorizontalPreComposeFunctorWithNaturalTransformation (for IsCapFunctor, IsCapNaturalTransformation)

▷ HorizontalPreComposeFunctorWithNaturalTransformation(F , N) (operation)

Returns: a natural transformation

Computes the horizontal composition of the functor F and the natural transformation N .

Chapter 3

Morphisms

Any GAP object satisfying `IsCapCategoryMorphism` can be added to a category and then becomes a morphism in this category. Any morphism can belong to one or no category. After a GAP object is added to the category, it knows which things can be computed in its category and to which category it belongs. It knows categorical properties and attributes, and the functions for existential quantifiers can be applied to the morphism.

3.1 Attributes for the Type of Morphisms

3.1.1 `CapCategory` (for `IsCapCategoryMorphism`)

- ▷ `CapCategory(alpha)` (attribute)
Returns: a category
The argument is a morphism α . The output is the category \mathbf{C} to which α was added.

3.1.2 `Source` (for `IsCapCategoryMorphism`)

- ▷ `Source(alpha)` (attribute)
Returns: an object
The argument is a morphism $\alpha : a \rightarrow b$. The output is its source a .

3.1.3 `Range` (for `IsCapCategoryMorphism`)

- ▷ `Range(alpha)` (attribute)
Returns: an object
The argument is a morphism $\alpha : a \rightarrow b$. The output is its range b .

3.2 Categorical Properties of Morphisms

3.2.1 `AddIsMonomorphism` (for `IsCapCategory`, `IsFunction`)

- ▷ `AddIsMonomorphism(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 `IsMonomorphism`. $F : \alpha \mapsto \text{IsMonomorphism}(\alpha)$.

3.2.2 AddIsEpimorphism (for IsCapCategory, IsFunction)

▷ AddIsEpimorphism(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 IsEpimorphism. $F : \alpha \mapsto \text{IsEpimorphism}(\alpha)$.

3.2.3 AddIsIsomorphism (for IsCapCategory, IsFunction)

▷ AddIsIsomorphism(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 IsIsomorphism. $F : \alpha \mapsto \text{IsIsomorphism}(\alpha)$.

3.2.4 AddIsSplitMonomorphism (for IsCapCategory, IsFunction)

▷ AddIsSplitMonomorphism(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 IsSplitMonomorphism. $F : \alpha \mapsto \text{IsSplitMonomorphism}(\alpha)$.

3.2.5 AddIsSplitEpimorphism (for IsCapCategory, IsFunction)

▷ AddIsSplitEpimorphism(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 IsSplitEpimorphism. $F : \alpha \mapsto \text{IsSplitEpimorphism}(\alpha)$.

3.2.6 AddIsOne (for IsCapCategory, IsFunction)

▷ AddIsOne(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 IsOne. $F : \alpha \mapsto \text{IsOne}(\alpha)$.

3.2.7 AddIsIdempotent (for IsCapCategory, IsFunction)

▷ AddIsIdempotent(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 IsIdempotent. $F : \alpha \mapsto \text{IsIdempotent}(\alpha)$.

3.3 Non-Categorical Properties of Morphisms

Non-categorical properties are not stable under equivalences of categories.

3.3.1 IsIdenticalToIdentityMorphism (for IsCapCategoryMorphism)

▷ `IsIdenticalToIdentityMorphism(α)` (property)

Returns: a boolean

The argument is a morphism $\alpha : a \rightarrow b$. The output is `true` if $\alpha = \text{id}_a$, otherwise the output is `false`.

3.3.2 AddIsIdenticalToIdentityMorphism (for IsCapCategory, IsFunction)

▷ `AddIsIdenticalToIdentityMorphism(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 `IsIdenticalToIdentityMorphism`. $F : \alpha \mapsto \text{IsIdenticalToIdentityMorphism}(\alpha)$.

3.3.3 IsIdenticalToZeroMorphism (for IsCapCategoryMorphism)

▷ `IsIdenticalToZeroMorphism(α)` (property)

Returns: a boolean

The argument is a morphism $\alpha : a \rightarrow b$. The output is `true` if $\alpha = 0$, otherwise the output is `false`.

3.3.4 AddIsIdenticalToZeroMorphism (for IsCapCategory, IsFunction)

▷ `AddIsIdenticalToZeroMorphism(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 `IsIdenticalToZeroMorphism`. $F : \alpha \mapsto \text{IsIdenticalToZeroMorphism}(\alpha)$.

3.3.5 AddIsEndomorphism (for IsCapCategory, IsFunction)

▷ `AddIsEndomorphism(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 `IsEndomorphism`. $F : \alpha \mapsto \text{IsEndomorphism}(\alpha)$.

3.3.6 AddIsAutomorphism (for IsCapCategory, IsFunction)

▷ `AddIsAutomorphism(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 `IsAutomorphism`. $F : \alpha \mapsto \text{IsAutomorphism}(\alpha)$.

3.4 Equality and Congruence for Morphisms

3.4.1 IsCongruentForMorphisms (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ IsCongruentForMorphisms(*alpha*, *beta*) (operation)
Returns: a boolean
 The arguments are two morphisms $\alpha, \beta : a \rightarrow b$. The output is true if $\alpha \sim_{a,b} \beta$, otherwise the output is false.

3.4.2 AddIsCongruentForMorphisms (for IsCapCategory, IsFunction)

▷ AddIsCongruentForMorphisms(*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 IsCongruentForMorphisms. $F : (\alpha, \beta) \mapsto \text{IsCongruentForMorphisms}(\alpha, \beta)$.

3.4.3 IsEqualForMorphisms (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ IsEqualForMorphisms(*alpha*, *beta*) (operation)
Returns: a boolean
 The arguments are two morphisms $\alpha, \beta : a \rightarrow b$. The output is true if $\alpha = \beta$, otherwise the output is false.

3.4.4 AddIsEqualForMorphisms (for IsCapCategory, IsFunction)

▷ AddIsEqualForMorphisms(*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 IsEqualForMorphisms. $F : (\alpha, \beta) \mapsto \text{IsEqualForMorphisms}(\alpha, \beta)$.

3.4.5 IsEqualForMorphismsOnMor (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ IsEqualForMorphismsOnMor(*alpha*, *beta*) (operation)
Returns: a boolean
 The arguments are two morphisms $\alpha : a \rightarrow b, \beta : c \rightarrow d$. The output is true if $\alpha = \beta$, otherwise the output is false.

3.4.6 AddIsEqualForMorphismsOnMor (for IsCapCategory, IsFunction)

▷ AddIsEqualForMorphismsOnMor(*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 IsEqualForMorphismsOnMor. $F : (\alpha, \beta) \mapsto \text{IsEqualForMorphismsOnMor}(\alpha, \beta)$.

3.5 Basic Operations for Morphisms in Ab-Categories

3.5.1 IsZeroForMorphisms (for IsCapCategoryMorphism)

▷ `IsZeroForMorphisms(alpha)` (operation)

Returns: a boolean

The argument is a morphism $\alpha : a \rightarrow b$. The output is true if $\alpha \sim_{a,b} 0$, otherwise the output is false.

3.5.2 AddIsZeroForMorphisms (for IsCapCategory, IsFunction)

▷ `AddIsZeroForMorphisms(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 `IsZeroForMorphisms`. $F : \alpha \mapsto \text{IsZeroForMorphisms}(\alpha)$.

3.5.3 AdditionForMorphisms (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

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

The arguments are two morphisms $\alpha, \beta : a \rightarrow b$. The output is the addition $\alpha + \beta$.

3.5.4 AddAdditionForMorphisms (for IsCapCategory, IsFunction)

▷ `AddAdditionForMorphisms(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 `AdditionForMorphisms`. $F : (\alpha, \beta) \mapsto \alpha + \beta$.

3.5.5 AdditiveInverseForMorphisms (for IsCapCategoryMorphism)

▷ `AdditiveInverseForMorphisms(alpha)` (operation)

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

The argument is a morphism $\alpha : a \rightarrow b$. The output is its additive inverse $-\alpha$.

3.5.6 AddAdditiveInverseForMorphisms (for IsCapCategory, IsFunction)

▷ `AddAdditiveInverseForMorphisms(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 `AdditiveInverseForMorphisms`. $F : \alpha \mapsto -\alpha$.

3.5.7 ZeroMorphism (for IsCapCategoryObject, IsCapCategoryObject)

▷ `ZeroMorphism(a, b)` (operation)

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

The arguments are two objects a and b . The output is the zero morphism $0 : a \rightarrow b$.

3.5.8 AddZeroMorphism (for IsCapCategory, IsFunction)

▷ AddZeroMorphism(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 ZeroMorphism. $F : (a, b) \mapsto (0 : a \rightarrow b)$.

3.6 Subobject and Factorobject Operations

Subobjects of an object c are monomorphisms with range c and a special function for comparison. Similarly, factorobjects of an object c are epimorphisms with source c and a special function for comparison.

3.6.1 IsEqualAsSubobjects (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ IsEqualAsSubobjects($alpha$, $beta$) (operation)

Returns: a boolean

The arguments are two subobjects $\alpha : a \rightarrow c$, $\beta : b \rightarrow c$. The output is true if there exists an isomorphism $\iota : a \rightarrow b$ such that $\beta \circ \iota \sim_{a,c} \alpha$, otherwise the output is false.

3.6.2 AddIsEqualAsSubobjects (for IsCapCategory, IsFunction)

▷ AddIsEqualAsSubobjects(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 IsEqualAsSubobjects. $F : (\alpha, \beta) \mapsto \text{IsEqualAsSubobjects}(\alpha, \beta)$.

3.6.3 IsEqualAsFactorobjects (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ IsEqualAsFactorobjects($alpha$, $beta$) (operation)

Returns: a boolean

The arguments are two factorobjects $\alpha : c \rightarrow a$, $\beta : c \rightarrow b$. The output is true if there exists an isomorphism $\iota : b \rightarrow a$ such that $\iota \circ \beta \sim_{c,a} \alpha$, otherwise the output is false.

3.6.4 AddIsEqualAsFactorobjects (for IsCapCategory, IsFunction)

▷ AddIsEqualAsFactorobjects(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 IsEqualAsFactorobjects. $F : (\alpha, \beta) \mapsto \text{IsEqualAsFactorobjects}(\alpha, \beta)$.

3.6.5 IsDominating (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ IsDominating($alpha$, $beta$) (operation)

Returns: a boolean

In short: Returns true iff α is smaller than β .

Full description: The arguments are two subobjects $\alpha : a \rightarrow c$, $\beta : b \rightarrow c$. The output is true if there exists a morphism $\iota : a \rightarrow b$ such that $\beta \circ \iota \sim_{a,c} \alpha$, otherwise the output is false.

3.6.6 AddIsDominating (for IsCapCategory, IsFunction)

▷ AddIsDominating(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 IsDominating. $F : (\alpha, \beta) \mapsto \text{IsDominating}(\alpha, \beta)$.

3.6.7 IsCodominating (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ IsCodominating($alpha$, $beta$) (operation)

Returns: a boolean

In short: Returns true iff α is smaller than β .

Full description: The arguments are two factorobjects $\alpha : c \rightarrow a$, $\beta : c \rightarrow b$. The output is true if there exists a morphism $\iota : b \rightarrow a$ such that $\iota \circ \beta \sim_{c,a} \alpha$, otherwise the output is false.

3.6.8 AddIsCodominating (for IsCapCategory, IsFunction)

▷ AddIsCodominating(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 IsCodominating. $F : (\alpha, \beta) \mapsto \text{IsCodominating}(\alpha, \beta)$.

3.7 Identity Morphism and Composition of Morphisms

3.7.1 IdentityMorphism (for IsCapCategoryObject)

▷ IdentityMorphism(a) (attribute)

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

The argument is an object a . The output is its identity morphism id_a .

3.7.2 AddIdentityMorphism (for IsCapCategory, IsFunction)

▷ AddIdentityMorphism(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 IdentityMorphism. $F : a \mapsto \text{id}_a$.

3.7.3 PreCompose (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ PreCompose($alpha$, $beta$) (operation)

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

The arguments are two morphisms $\alpha : a \rightarrow b$, $\beta : b \rightarrow c$. The output is the composition $\beta \circ \alpha : a \rightarrow c$.

3.7.4 PreCompose (for IsList)

▷ `PreCompose(L)` (operation)

Returns: a morphism in $\text{Hom}(a_1, a_{n+1})$

This is a convenience method. The argument is a list of morphisms $L = (\alpha_1 : a_1 \rightarrow a_2, \alpha_2 : a_2 \rightarrow a_3, \dots, \alpha_n : a_n \rightarrow a_{n+1})$. The output is the composition $\alpha_n \circ (\alpha_{n-1} \circ (\dots (\alpha_2 \circ \alpha_1)))$.

3.7.5 AddPreCompose (for IsCapCategory, IsFunction)

▷ `AddPreCompose(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 `PreCompose`. $F : (\alpha, \beta) \mapsto \beta \circ \alpha$.

3.7.6 PostCompose (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

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

The arguments are two morphisms $\beta : b \rightarrow c$, $\alpha : a \rightarrow b$. The output is the composition $\beta \circ \alpha : a \rightarrow c$.

3.7.7 PostCompose (for IsList)

▷ `PostCompose(L)` (operation)

Returns: a morphism in $\text{Hom}(a_1, a_{n+1})$

This is a convenience method. The argument is a list of morphisms $L = (\alpha_n : a_n \rightarrow a_{n+1}, \alpha_{n-1} : a_{n-1} \rightarrow a_n, \dots, \alpha_1 : a_1 \rightarrow a_2)$. The output is the composition $((\alpha_n \circ \alpha_{n-1}) \circ \dots \alpha_2) \circ \alpha_1$.

3.7.8 AddPostCompose (for IsCapCategory, IsFunction)

▷ `AddPostCompose(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 `PostCompose`. $F : (\alpha, \beta) \mapsto \alpha \circ \beta$.

3.8 Well-Definedness of Morphisms

3.8.1 IsWellDefinedForMorphisms (for IsCapCategoryMorphism)

▷ `IsWellDefinedForMorphisms(alpha)` (operation)

Returns: a boolean

The argument is a morphism α . The output is true if α is well-defined, otherwise the output is false.

3.8.2 AddIsWellDefinedForMorphisms (for IsCapCategory, IsFunction)

▷ `AddIsWellDefinedForMorphisms(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 `IsWellDefinedForMorphisms`. $F : \alpha \mapsto \text{IsWellDefinedForMorphisms}(\alpha)$.

3.9 Basic Operations for Morphisms in Abelian Categories

3.9.1 LiftAlongMonomorphism (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `LiftAlongMonomorphism(iota, tau)` (operation)

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

The arguments are a monomorphism $\iota : k \hookrightarrow a$ and a morphism $\tau : t \rightarrow a$ such that there is a morphism $u : t \rightarrow k$ with $\iota \circ u \sim_{t, a} \tau$. The output is such a u .

3.9.2 AddLiftAlongMonomorphism (for IsCapCategory, IsFunction)

▷ `AddLiftAlongMonomorphism(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 `LiftAlongMonomorphism`. The function F maps a pair (ι, τ) to a lift u if it exists, and to fail otherwise.

3.9.3 ColiftAlongEpimorphism (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `ColiftAlongEpimorphism(epsilon, tau)` (operation)

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

The arguments are an epimorphism $\varepsilon : a \rightarrow c$ and a morphism $\tau : a \rightarrow t$ such that there is a morphism $u : c \rightarrow t$ with $u \circ \varepsilon \sim_{a, t} \tau$. The output is such a u .

3.9.4 AddColiftAlongEpimorphism (for IsCapCategory, IsFunction)

▷ `AddColiftAlongEpimorphism(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 `ColiftAlongEpimorphism`. The function F maps a pair (ε, τ) to a lift u if it exists, and to fail otherwise.

3.10 Lift/ Colift

- For any pair of morphisms $\alpha : a \rightarrow c, \beta : b \rightarrow c$, we call each morphism $\alpha/\beta : a \rightarrow b$ such that $\beta \circ (\alpha/\beta) \sim_{a, c} \alpha$ a *lift of α along β* .
- For any pair of morphisms $\alpha : a \rightarrow c, \beta : a \rightarrow b$, we call each morphism $\alpha \setminus \beta : c \rightarrow b$ such that $(\alpha \setminus \beta) \circ \alpha \sim_{a, b} \beta$ a *colift of β along α* .

Note that such lifts (or colifts) do not have to be unique. So in general, we do not expect that algorithms computing lifts (or colifts) do this in a functorial way. Thus the operations `Lift` and `Colift` are not regarded as categorical operations, but only as set-theoretic operations.

3.10.1 Lift (for `IsCapCategoryMorphism`, `IsCapCategoryMorphism`)

▷ `Lift(alpha, beta)` (operation)

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

The arguments are two morphisms $\alpha : a \rightarrow c$, $\beta : b \rightarrow c$ such that there is a lift $\alpha/\beta : a \rightarrow b$ of α along β , i.e., a morphism such that $\beta \circ (\alpha/\beta) \sim_{a,c} \alpha$. The output is such a lift or `fail` if it doesn't exist.

3.10.2 AddLift (for `IsCapCategory`, `IsFunction`)

▷ `AddLift(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 `Lift`. The function F maps a pair (α, β) to a lift α/β if it exists, and to `fail` otherwise.

3.10.3 Colift (for `IsCapCategoryMorphism`, `IsCapCategoryMorphism`)

▷ `Colift(alpha, beta)` (operation)

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

The arguments are two morphisms $\alpha : a \rightarrow c$, $\beta : a \rightarrow b$ such that there is a colift $\alpha \setminus \beta : c \rightarrow b$ of β along α , i.e., a morphism such that $(\alpha \setminus \beta) \circ \alpha \sim_{a,b} \beta$. The output is such a colift or `fail` if it doesn't exist.

3.10.4 AddColift (for `IsCapCategory`, `IsFunction`)

▷ `AddColift(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 `Colift`. The function F maps a pair (α, β) to a colift $\alpha \setminus \beta$ if it exists, and to `fail` otherwise.

3.11 Inverses

Let $\alpha : a \rightarrow b$ be a morphism. An inverse of α is a morphism $\alpha^{-1} : b \rightarrow a$ such that $\alpha \circ \alpha^{-1} \sim_{b,b} \text{id}_b$ and $\alpha^{-1} \circ \alpha \sim_{a,a} \text{id}_a$.

3.11.1 AddInverse (for `IsCapCategory`, `IsFunction`)

▷ `AddInverse(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 `Inverse`. $F : \alpha \mapsto \alpha^{-1}$.

3.12 Tool functions for caches

3.12.1 IsEqualForCacheForMorphisms (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `IsEqualForCacheForMorphisms(phi, psi)` (operation)

Returns: true or false

Compares two objects in the cache

3.12.2 AddIsEqualForCacheForMorphisms (for IsCapCategory, IsFunction)

▷ `AddIsEqualForCacheForMorphisms(c, F)` (operation)

Returns: nothing

By default, CAP uses caches to store the values of Categorical operations. To get a value out of the cache, one needs to compare the input of a basic operation with its previous input. To compare morphisms in the category, `IsEqualForCacheForMorphism` is used. By default this is an alias for `IsEqualForMorphismsOnMor`, where `fail` is substituted by `false`. If you add a function, this function is used instead. A function $F : a, b \mapsto bool$ is expected here. The output has to be true or false. Fail is not allowed in this context.

Chapter 4

Objects

Any GAP object which is `IsCapCategoryObject` can be added to a category and then becomes an object in this category. Any object can belong to one or no category. After a GAP object is added to the category, it knows which things can be computed in its category and to which category it belongs. It knows categorial properties and attributes, and the functions for existential quantifiers can be applied to the object.

4.1 Attributes for the Type of Objects

4.1.1 `CapCategory` (for `IsCapCategoryObject`)

▷ `CapCategory(a)` (attribute)

Returns: a category

The argument is an object a . The output is the category C to which a was added.

4.2 Equality for Objects

4.2.1 `IsEqualForObjects` (for `IsCapCategoryObject`, `IsCapCategoryObject`)

▷ `IsEqualForObjects(a, b)` (operation)

Returns: a boolean

The arguments are two objects a and b . The output is `true` if $a = b$, otherwise the output is `false`.

4.2.2 `AddIsEqualForObjects` (for `IsCapCategory`, `IsFunction`)

▷ `AddIsEqualForObjects(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 `IsEqualForObjects`. $F : (a, b) \mapsto \text{IsEqualForObjects}(a, b)$.

4.3 Categorical Properties of Objects

4.3.1 AddIsProjective (for IsCapCategory, IsFunction)

▷ `AddIsProjective(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 `IsProjective`. $F : a \mapsto \text{IsProjective}(a)$.

4.3.2 AddIsInjective (for IsCapCategory, IsFunction)

▷ `AddIsInjective(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 `IsInjective`. $F : a \mapsto \text{IsInjective}(a)$.

4.3.3 AddIsTerminal (for IsCapCategory, IsFunction)

▷ `AddIsTerminal(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 `IsTerminal`. $F : a \mapsto \text{IsTerminal}(a)$.

4.3.4 AddIsInitial (for IsCapCategory, IsFunction)

▷ `AddIsInitial(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 `IsInitial`. $F : a \mapsto \text{IsInitial}(a)$.

4.3.5 IsZeroForObjects (for IsCapCategoryObject)

▷ `IsZeroForObjects(a)` (operation)
Returns: a boolean

The argument is an object a of a category C . The output is `true` if a is isomorphic to the zero object of C , otherwise the output is `false`.

4.3.6 AddIsZeroForObjects (for IsCapCategory, IsFunction)

▷ `AddIsZeroForObjects(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 `IsZeroForObjects`. $F : a \mapsto \text{IsZeroForObjects}(a)$.

4.4 Tool functions for caches

4.4.1 IsEqualForCacheForObjects (for IsCapCategoryObject, IsCapCategoryObject)

- ▷ `IsEqualForCacheForObjects(phi, psi)` (operation)
Returns: true or false
 Compares two objects in the cache

4.4.2 AddIsEqualForCacheForObjects (for IsCapCategory, IsFunction)

- ▷ `AddIsEqualForCacheForObjects(c, F)` (operation)
Returns: nothing

By default, CAP uses caches to store the values of Categorical operations. To get a value out of the cache, one needs to compare the input of a basic operation with its previous input. To compare objects in the category, `IsEqualForCacheForObject` is used. By default this is an alias for `IsEqualForObjects`, where `fail` is substituted by `false`. If you add a function, this function used instead. A function $F : a, b \mapsto bool$ is expected here. The output has to be true or false. Fail is not allowed in this context.

4.5 Well-Definedness of Objects

4.5.1 IsWellDefinedForObjects (for IsCapCategoryObject)

- ▷ `IsWellDefinedForObjects(a)` (operation)
Returns: a boolean
 The argument is an object a . The output is true if a is well-defined, otherwise the output is false.

4.5.2 AddIsWellDefinedForObjects (for IsCapCategory, IsFunction)

- ▷ `AddIsWellDefinedForObjects(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 `IsWellDefinedForObjects`. $F : a \mapsto IsWellDefinedForObjects(a)$.

Chapter 5

Category 2-Cells

5.1 Attributes for the Type of 2-Cells

5.1.1 Source (for IsCapCategoryTwoCell)

- ▷ `Source(c)` (attribute)
Returns: a morphism
The argument is a 2-cell $c : \alpha \rightarrow \beta$. The output is its source α .

5.1.2 Range (for IsCapCategoryTwoCell)

- ▷ `Range(c)` (attribute)
Returns: a morphism
The argument is a 2-cell $c : \alpha \rightarrow \beta$. The output is its range β .

5.2 Identity 2-Cell and Composition of 2-Cells

5.2.1 IdentityTwoCell (for IsCapCategoryMorphism)

- ▷ `IdentityTwoCell(alpha)` (attribute)
Returns: a 2-cell
The argument is a morphism α . The output is its identity 2-cell $\text{id}_\alpha : \alpha \rightarrow \alpha$.

5.2.2 AddIdentityTwoCell (for IsCapCategory, IsFunction)

- ▷ `AddIdentityTwoCell(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 `IdentityTwoCell`. $F : \alpha \mapsto \text{id}_\alpha$.

5.2.3 HorizontalPreCompose (for IsCapCategoryTwoCell, IsCapCategoryTwoCell)

- ▷ `HorizontalPreCompose(c, d)` (operation)
Returns: a 2-cell
The arguments are two 2-cells $c : \alpha \rightarrow \beta$, $d : \gamma \rightarrow \delta$ between morphisms $\alpha, \beta : a \rightarrow b$ and $\gamma, \delta : b \rightarrow c$. The output is their horizontal composition $d * c : (\gamma \circ \alpha) \rightarrow (\delta \circ \beta)$.

5.2.4 AddHorizontalPreCompose (for IsCapCategory, IsFunction)

▷ AddHorizontalPreCompose(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 HorizontalPreCompose. $F : (c, d) \mapsto d * c$.

5.2.5 HorizontalPostCompose (for IsCapCategoryTwoCell, IsCapCategoryTwoCell)

▷ HorizontalPostCompose(d, c) (operation)

Returns: a 2-cell

The arguments are two 2-cells $d : \gamma \rightarrow \delta, c : \alpha \rightarrow \beta$ between morphisms $\alpha, \beta : a \rightarrow b$ and $\gamma, \delta : b \rightarrow c$. The output is their horizontal composition $d * c : (\gamma \circ \alpha) \rightarrow (\delta \circ \beta)$.

5.2.6 AddHorizontalPostCompose (for IsCapCategory, IsFunction)

▷ AddHorizontalPostCompose(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 HorizontalPostCompose. $F : (d, c) \mapsto d * c$.

5.2.7 VerticalPreCompose (for IsCapCategoryTwoCell, IsCapCategoryTwoCell)

▷ VerticalPreCompose(c, d) (operation)

Returns: a 2-cell

The arguments are two 2-cells $c : \alpha \rightarrow \beta, d : \beta \rightarrow \gamma$ between morphisms $\alpha, \beta, \gamma : a \rightarrow b$. The output is their vertical composition $d \circ c : \alpha \rightarrow \gamma$.

5.2.8 AddVerticalPreCompose (for IsCapCategory, IsFunction)

▷ AddVerticalPreCompose(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 VerticalPreCompose. $F : (c, d) \mapsto d \circ c$.

5.2.9 VerticalPostCompose (for IsCapCategoryTwoCell, IsCapCategoryTwoCell)

▷ VerticalPostCompose(d, c) (operation)

Returns: a 2-cell

The arguments are two 2-cells $d : \beta \rightarrow \gamma, c : \alpha \rightarrow \beta$ between morphisms $\alpha, \beta, \gamma : a \rightarrow b$. The output is their vertical composition $d \circ c : \alpha \rightarrow \gamma$.

5.2.10 AddVerticalPostCompose (for IsCapCategory, IsFunction)

▷ AddVerticalPostCompose(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 VerticalPostCompose. $F : (d, c) \mapsto d \circ c$.

5.3 Well-Definedness for 2-Cells

5.3.1 IsWellDefinedForTwoCells (for IsCapCategoryTwoCell)

▷ `IsWellDefinedForTwoCells(c)` (operation)

Returns: a boolean

The argument is a 2-cell c . The output is true if c is well-defined, otherwise the output is false.

5.3.2 AddIsWellDefinedForTwoCells (for IsCapCategory, IsFunction)

▷ `AddIsWellDefinedForTwoCells(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 `IsWellDefinedForTwoCells`. $F : c \mapsto \text{IsWellDefinedForMorphisms}(c)$.

Chapter 6

Universal Objects

6.1 Kernel

For a given morphism $\alpha : A \rightarrow B$, a 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 *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 $\text{KernelObject}(\alpha)$. We say that the morphism $u(\tau)$ is induced by the *universal property of the kernel*.

KernelObject is a functorial operation. This means: for $\mu : A \rightarrow A'$, $\nu : B \rightarrow B'$, $\alpha : A \rightarrow B$, $\alpha' : A' \rightarrow B'$ such that $\nu \circ \alpha \sim_{A,B'} \alpha' \circ \mu$, we obtain a morphism $\text{KernelObject}(\alpha) \rightarrow \text{KernelObject}(\alpha')$.

6.1.1 KernelObject (for IsCapCategoryMorphism)

▷ `KernelObject(alpha)` (attribute)

Returns: an object

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

6.1.2 KernelEmbedding (for IsCapCategoryMorphism)

▷ `KernelEmbedding(alpha)` (attribute)

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

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

6.1.3 KernelEmbeddingWithGivenKernelObject (for IsCapCategoryMorphism, IsCapCategoryObject)

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

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

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

6.1.4 KernelLift (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

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

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

6.1.5 KernelLiftWithGivenKernelObject (for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject)

▷ `KernelLiftWithGivenKernelObject(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 $K = \text{KernelObject}(\alpha)$. The output is the morphism $u(\tau) : T \rightarrow K$ given by the universal property of the kernel.

6.1.6 AddKernelObject (for IsCapCategory, IsFunction)

▷ `AddKernelObject(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`. $F : \alpha \mapsto \text{KernelObject}(\alpha)$.

6.1.7 AddKernelEmbedding (for IsCapCategory, IsFunction)

▷ `AddKernelEmbedding(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`. $F : \alpha \mapsto \iota$.

6.1.8 AddKernelEmbeddingWithGivenKernelObject (for IsCapCategory, IsFunction)

▷ `AddKernelEmbeddingWithGivenKernelObject(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`. $F : (\alpha, K) \mapsto \iota$.

6.1.9 AddKernelLift (for IsCapCategory, IsFunction)

▷ `AddKernelLift(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`. $F : (\alpha, \tau) \mapsto u(\tau)$.

6.1.10 AddKernelLiftWithGivenKernelObject (for IsCapCategory, IsFunction)

▷ AddKernelLiftWithGivenKernelObject(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. $F : (\alpha, \tau, K) \mapsto u$.

6.1.11 KernelObjectFunctorial (for IsList)

▷ KernelObjectFunctorial(L) (operation)

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

The argument is a list $L = [\alpha : A \rightarrow B, [\mu : A \rightarrow A', \nu : B \rightarrow B'], \alpha' : A' \rightarrow B']$ of morphisms. The output is the morphism $\text{KernelObject}(\alpha) \rightarrow \text{KernelObject}(\alpha')$ given by the functoriality of the kernel.

6.1.12 KernelObjectFunctorial (for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ KernelObjectFunctorial($alpha, mu, alpha_prime$) (operation)

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

The arguments are three morphisms $\alpha : A \rightarrow B, \mu : A \rightarrow A', \alpha' : A' \rightarrow B'$. The output is the morphism $\text{KernelObject}(\alpha) \rightarrow \text{KernelObject}(\alpha')$ given by the functoriality of the kernel.

6.1.13 KernelObjectFunctorialWithGivenKernelObjects (for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject)

▷ KernelObjectFunctorialWithGivenKernelObjects($s, alpha, mu, alpha_prime, r$) (operation)

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

The arguments are an object $s = \text{KernelObject}(\alpha)$, three morphisms $\alpha : A \rightarrow B, \mu : A \rightarrow A', \alpha' : A' \rightarrow B'$, and an object $r = \text{KernelObject}(\alpha')$. The output is the morphism $\text{KernelObject}(\alpha) \rightarrow \text{KernelObject}(\alpha')$ given by the functoriality of the kernel.

6.1.14 AddKernelObjectFunctorialWithGivenKernelObjects (for IsCapCategory, IsFunction)

▷ AddKernelObjectFunctorialWithGivenKernelObjects(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 KernelObjectFunctorialWithGivenKernelObjects. $F : (\text{KernelObject}(\alpha), \alpha, \mu, \alpha', \text{KernelObject}(\alpha')) \mapsto (\text{KernelObject}(\alpha) \rightarrow \text{KernelObject}(\alpha'))$.

6.2 Cokernel

For a given morphism $\alpha : A \rightarrow B$, a 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 *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 $\text{CokernelObject}(\alpha)$. We say that the morphism $u(\tau)$ is induced by the *universal property of the cokernel*.

CokernelObject is a functorial operation. This means: for $\mu : A \rightarrow A'$, $\nu : B \rightarrow B'$, $\alpha : A \rightarrow B$, $\alpha' : A' \rightarrow B'$ such that $\nu \circ \alpha \sim_{A,B'} \alpha' \circ \mu$, we obtain a morphism $\text{CokernelObject}(\alpha) \rightarrow \text{CokernelObject}(\alpha')$.

6.2.1 CokernelObject (for IsCapCategoryMorphism)

▷ $\text{CokernelObject}(\alpha)$ (attribute)

Returns: an object

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

6.2.2 CokernelProjection (for IsCapCategoryMorphism)

▷ $\text{CokernelProjection}(\alpha)$ (attribute)

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

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

6.2.3 CokernelProjectionWithGivenCokernelObject (for IsCapCategoryMorphism, IsCapCategoryObject)

▷ $\text{CokernelProjectionWithGivenCokernelObject}(\alpha, K)$ (operation)

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

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

6.2.4 CokernelColift (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ $\text{CokernelColift}(\alpha, \tau)$ (operation)

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

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 $u(\tau) : \text{CokernelObject}(\alpha) \rightarrow T$ given by the universal property of the cokernel.

6.2.5 CokernelColiftWithGivenCokernelObject (for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject)

▷ $\text{CokernelColiftWithGivenCokernelObject}(\alpha, \tau, K)$ (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 $K = \text{CokernelObject}(\alpha)$. The output is the morphism $u(\tau) : K \rightarrow T$ given by the universal property of the cokernel.

6.2.6 AddCokernelObject (for IsCapCategory, IsFunction)

▷ AddCokernelObject(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. $F : \alpha \mapsto K$.

6.2.7 AddCokernelProjection (for IsCapCategory, IsFunction)

▷ AddCokernelProjection(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. $F : \alpha \mapsto \varepsilon$.

6.2.8 AddCokernelProjectionWithGivenCokernelObject (for IsCapCategory, IsFunction)

▷ AddCokernelProjectionWithGivenCokernelObject(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. $F : (\alpha, K) \mapsto \varepsilon$.

6.2.9 AddCokernelColift (for IsCapCategory, IsFunction)

▷ AddCokernelColift(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. $F : (\alpha, \tau) \mapsto u(\tau)$.

6.2.10 AddCokernelColiftWithGivenCokernelObject (for IsCapCategory, IsFunction)

▷ AddCokernelColiftWithGivenCokernelObject(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. $F : (\alpha, \tau, K) \mapsto u(\tau)$.

6.2.11 CokernelObjectFunctorial (for IsList)

▷ CokernelObjectFunctorial(L) (operation)

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

The argument is a list $L = [\alpha : A \rightarrow B, [\mu : A \rightarrow A', \nu : B \rightarrow B'], \alpha' : A' \rightarrow B']$. The output is the morphism $\text{CokernelObject}(\alpha) \rightarrow \text{CokernelObject}(\alpha')$ given by the functoriality of the cokernel.

6.2.12 CokernelObjectFunctorial (for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ CokernelObjectFunctorial($alpha, nu, alpha_prime$) (operation)

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

The arguments are three morphisms $\alpha : A \rightarrow B, \nu : B \rightarrow B', \alpha' : A' \rightarrow B'$. The output is the morphism $\text{CokernelObject}(\alpha) \rightarrow \text{CokernelObject}(\alpha')$ given by the functorality of the cokernel.

6.2.13 CokernelObjectFunctorialWithGivenCokernelObjects (for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject)

▷ $\text{CokernelObjectFunctorialWithGivenCokernelObjects}(s, \text{alpha}, \text{nu}, \text{alpha_prime}, r)$ (operation)

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

The arguments are an object $s = \text{CokernelObject}(\alpha)$, three morphisms $\alpha : A \rightarrow B, \nu : B \rightarrow B', \alpha' : A' \rightarrow B'$, and an object $r = \text{CokernelObject}(\alpha')$. The output is the morphism $\text{CokernelObject}(\alpha) \rightarrow \text{CokernelObject}(\alpha')$ given by the functorality of the cokernel.

6.2.14 AddCokernelObjectFunctorialWithGivenCokernelObjects (for IsCapCategory, IsFunction)

▷ $\text{AddCokernelObjectFunctorialWithGivenCokernelObjects}(C, F)$ (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 $\text{CokernelObjectFunctorialWithGivenCokernelObjects}$. $F : (\text{CokernelObject}(\alpha), \alpha, \nu, \alpha', \text{CokernelObject}(\alpha')) \mapsto (\text{CokernelObject}(\alpha) \rightarrow \text{CokernelObject}(\alpha'))$.

6.3 Zero Object

A zero object consists of three parts:

- an object Z ,
- a function u_{in} mapping each object A to a morphism $u_{\text{in}}(A) : A \rightarrow Z$,
- a function u_{out} mapping each object A to a morphism $u_{\text{out}}(A) : Z \rightarrow A$.

The triple $(Z, u_{\text{in}}, u_{\text{out}})$ is called a *zero object* if the morphisms $u_{\text{in}}(A), u_{\text{out}}(A)$ are uniquely determined up to congruence of morphisms. We denote the object Z of such a triple by ZeroObject . We say that the morphisms $u_{\text{in}}(A)$ and $u_{\text{out}}(A)$ are induced by the *universal property of the zero object*.

6.3.1 ZeroObject (for IsCapCategory)

▷ $\text{ZeroObject}(C)$ (attribute)

Returns: an object

The argument is a category C . The output is a zero object Z of C .

6.3.2 ZeroObject (for IsCapCategoryCell)

▷ $\text{ZeroObject}(c)$ (attribute)

Returns: an object

This is a convenience method. The argument is a cell c . The output is a zero object Z of the category C for which $c \in C$.

6.3.3 MorphismFromZeroObject (for IsCapCategoryObject)

- ▷ `MorphismFromZeroObject(A)` (attribute)
Returns: a morphism in $\text{Hom}(\text{ZeroObject}, A)$
 This is a convenience method. The argument is an object A . It calls `UniversalMorphismFromZeroObject` on A .

6.3.4 MorphismIntoZeroObject (for IsCapCategoryObject)

- ▷ `MorphismIntoZeroObject(A)` (attribute)
Returns: a morphism in $\text{Hom}(A, \text{ZeroObject})$
 This is a convenience method. The argument is an object A . It calls `UniversalMorphismIntoZeroObject` on A .

6.3.5 UniversalMorphismFromZeroObject (for IsCapCategoryObject)

- ▷ `UniversalMorphismFromZeroObject(A)` (attribute)
Returns: a morphism in $\text{Hom}(\text{ZeroObject}, A)$
 The argument is an object A . The output is the universal morphism $u_{\text{out}} : \text{ZeroObject} \rightarrow A$.

6.3.6 UniversalMorphismFromZeroObjectWithGivenZeroObject (for IsCapCategoryObject, IsCapCategoryObject)

- ▷ `UniversalMorphismFromZeroObjectWithGivenZeroObject(A, Z)` (operation)
Returns: a morphism in $\text{Hom}(Z, A)$
 The arguments are an object A , and a zero object $Z = \text{ZeroObject}$. The output is the universal morphism $u_{\text{out}} : Z \rightarrow A$.

6.3.7 UniversalMorphismIntoZeroObject (for IsCapCategoryObject)

- ▷ `UniversalMorphismIntoZeroObject(A)` (attribute)
Returns: a morphism in $\text{Hom}(A, \text{ZeroObject})$
 The argument is an object A . The output is the universal morphism $u_{\text{in}} : A \rightarrow \text{ZeroObject}$.

6.3.8 UniversalMorphismIntoZeroObjectWithGivenZeroObject (for IsCapCategoryObject, IsCapCategoryObject)

- ▷ `UniversalMorphismIntoZeroObjectWithGivenZeroObject(A, Z)` (operation)
Returns: a morphism in $\text{Hom}(A, Z)$
 The arguments are an object A , and a zero object $Z = \text{ZeroObject}$. The output is the universal morphism $u_{\text{in}} : A \rightarrow Z$.

6.3.9 IsomorphismFromZeroObjectToInitialObject (for IsCapCategory)

- ▷ `IsomorphismFromZeroObjectToInitialObject(C)` (attribute)
Returns: a morphism in $\text{Hom}(\text{ZeroObject}, \text{InitialObject})$
 The argument is a category C . The output is the unique isomorphism $\text{ZeroObject} \rightarrow \text{InitialObject}$.

6.3.10 IsomorphismFromInitialObjectToZeroObject (for IsCapCategory)

▷ `IsomorphismFromInitialObjectToZeroObject(C)` (attribute)

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

The argument is a category C . The output is the unique isomorphism $\text{InitialObject} \rightarrow \text{ZeroObject}$.

6.3.11 IsomorphismFromZeroObjectToTerminalObject (for IsCapCategory)

▷ `IsomorphismFromZeroObjectToTerminalObject(C)` (attribute)

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

The argument is a category C . The output is the unique isomorphism $\text{ZeroObject} \rightarrow \text{TerminalObject}$.

6.3.12 IsomorphismFromTerminalObjectToZeroObject (for IsCapCategory)

▷ `IsomorphismFromTerminalObjectToZeroObject(C)` (attribute)

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

The argument is a category C . The output is the unique isomorphism $\text{TerminalObject} \rightarrow \text{ZeroObject}$.

6.3.13 AddZeroObject (for IsCapCategory, IsFunction)

▷ `AddZeroObject(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 `ZeroObject`. $F : () \mapsto \text{ZeroObject}$.

6.3.14 AddUniversalMorphismIntoZeroObject (for IsCapCategory, IsFunction)

▷ `AddUniversalMorphismIntoZeroObject(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 `UniversalMorphismIntoZeroObject`. $F : A \mapsto u_{\text{in}}(A)$.

6.3.15 AddUniversalMorphismIntoZeroObjectWithGivenZeroObject (for IsCapCategory, IsFunction)

▷ `AddUniversalMorphismIntoZeroObjectWithGivenZeroObject(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 `UniversalMorphismIntoZeroObjectWithGivenZeroObject`. $F : (A, Z) \mapsto u_{\text{in}}(A)$.

6.3.16 AddUniversalMorphismFromZeroObject (for IsCapCategory, IsFunction)

▷ `AddUniversalMorphismFromZeroObject(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 `UniversalMorphismFromZeroObject`. $F : A \mapsto u_{\text{out}}(A)$.

6.3.17 AddUniversalMorphismFromZeroObjectWithGivenZeroObject (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismFromZeroObjectWithGivenZeroObject(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 UniversalMorphismFromZeroObjectWithGivenZeroObject. $F : (A, Z) \mapsto u_{\text{out}}(A)$.

6.3.18 AddIsomorphismFromZeroObjectToInitialObject (for IsCapCategory, IsFunction)

▷ AddIsomorphismFromZeroObjectToInitialObject(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 IsomorphismFromZeroObjectToInitialObject. $F : () \mapsto (\text{ZeroObject} \rightarrow \text{InitialObject})$.

6.3.19 AddIsomorphismFromInitialObjectToZeroObject (for IsCapCategory, IsFunction)

▷ AddIsomorphismFromInitialObjectToZeroObject(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 IsomorphismFromInitialObjectToZeroObject. $F : () \mapsto (\text{InitialObject} \rightarrow \text{ZeroObject})$.

6.3.20 AddIsomorphismFromZeroObjectToTerminalObject (for IsCapCategory, IsFunction)

▷ AddIsomorphismFromZeroObjectToTerminalObject(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 IsomorphismFromZeroObjectToTerminalObject. $F : () \mapsto (\text{ZeroObject} \rightarrow \text{TerminalObject})$.

6.3.21 AddIsomorphismFromTerminalObjectToZeroObject (for IsCapCategory, IsFunction)

▷ AddIsomorphismFromTerminalObjectToZeroObject(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 IsomorphismFromTerminalObjectToZeroObject. $F : () \mapsto (\text{TerminalObject} \rightarrow \text{ZeroObject})$.

6.3.22 ZeroObjectFunctorial (for IsCapCategory)

▷ `ZeroObjectFunctorial(C)` (attribute)

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

The argument is a category C . The output is the unique morphism $\text{ZeroObject} \rightarrow \text{ZeroObject}$.

6.3.23 AddZeroObjectFunctorial (for IsCapCategory, IsFunction)

▷ `AddZeroObjectFunctorial(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 `ZeroObjectFunctorial`. $F : () \mapsto (T \rightarrow T)$.

6.4 Terminal Object

A terminal object consists of two parts:

- an object T ,
- a function u mapping each object A to a morphism $u(A) : A \rightarrow T$.

The pair (T, u) is called a *terminal object* if the morphisms $u(A)$ are uniquely determined up to congruence of morphisms. We denote the object T of such a pair by `TerminalObject`. We say that the morphism $u(A)$ is induced by the *universal property of the terminal object*.

`TerminalObject` is a functorial operation. This just means: There exists a unique morphism $T \rightarrow T$.

6.4.1 TerminalObject (for IsCapCategory)

▷ `TerminalObject(C)` (attribute)

Returns: an object

The argument is a category C . The output is a terminal object T of C .

6.4.2 TerminalObject (for IsCapCategoryCell)

▷ `TerminalObject(c)` (attribute)

Returns: an object

This is a convenience method. The argument is a cell c . The output is a terminal object T of the category C for which $c \in C$.

6.4.3 UniversalMorphismIntoTerminalObject (for IsCapCategoryObject)

▷ `UniversalMorphismIntoTerminalObject(A)` (attribute)

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

The argument is an object A . The output is the universal morphism $u(A) : A \rightarrow \text{TerminalObject}$.

6.4.4 UniversalMorphismIntoTerminalObjectWithGivenTerminalObject (for IsCap-CategoryObject, IsCapCategoryObject)

▷ UniversalMorphismIntoTerminalObjectWithGivenTerminalObject(A, T) (operation)

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

The argument are an object A , and an object $T = \text{TerminalObject}$. The output is the universal morphism $u(A) : A \rightarrow T$.

6.4.5 AddTerminalObject (for IsCapCategory, IsFunction)

▷ AddTerminalObject(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 TerminalObject. $F : () \mapsto T$.

6.4.6 AddUniversalMorphismIntoTerminalObject (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismIntoTerminalObject(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 UniversalMorphismIntoTerminalObject. $F : A \mapsto u(A)$.

6.4.7 AddUniversalMorphismIntoTerminalObjectWithGivenTerminalObject (for Is-CapCategory, IsFunction)

▷ AddUniversalMorphismIntoTerminalObjectWithGivenTerminalObject(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 UniversalMorphismIntoTerminalObjectWithGivenTerminalObject. $F : (A, T) \mapsto u(A)$.

6.4.8 TerminalObjectFunctorial (for IsCapCategory)

▷ TerminalObjectFunctorial(C) (attribute)

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

The argument is a category C . The output is the unique morphism $\text{TerminalObject} \rightarrow \text{TerminalObject}$.

6.4.9 AddTerminalObjectFunctorial (for IsCapCategory, IsFunction)

▷ AddTerminalObjectFunctorial(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 TerminalObjectFunctorial. $F : () \mapsto (T \rightarrow T)$.

6.5 Initial Object

An initial object consists of two parts:

- an object I ,
- a function u mapping each object A to a morphism $u(A) : I \rightarrow A$.

The pair (I, u) is called a *initial object* if the morphisms $u(A)$ are uniquely determined up to congruence of morphisms. We denote the object I of such a triple by `InitialObject`. We say that the morphism $u(A)$ is induced by the *universal property of the initial object*.

`InitialObject` is a functorial operation. This just means: There exists a unique morphisms $I \rightarrow I$.

6.5.1 InitialObject (for IsCapCategory)

▷ `InitialObject(C)` (attribute)

Returns: an object

The argument is a category C . The output is an initial object I of C .

6.5.2 InitialObject (for IsCapCategoryCell)

▷ `InitialObject(c)` (attribute)

Returns: an object

This is a convenience method. The argument is a cell c . The output is an initial object I of the category C for which $c \in C$.

6.5.3 UniversalMorphismFromInitialObject (for IsCapCategoryObject)

▷ `UniversalMorphismFromInitialObject(A)` (attribute)

Returns: a morphism in $\text{Hom}(\text{InitialObject} \rightarrow A)$.

The argument is an object A . The output is the universal morphism $u(A) : \text{InitialObject} \rightarrow A$.

6.5.4 UniversalMorphismFromInitialObjectWithGivenInitialObject (for IsCapCategoryObject, IsCapCategoryObject)

▷ `UniversalMorphismFromInitialObjectWithGivenInitialObject(A, I)` (operation)

Returns: a morphism in $\text{Hom}(\text{InitialObject} \rightarrow A)$.

The arguments are an object A , and an object $I = \text{InitialObject}$. The output is the universal morphism $u(A) : \text{InitialObject} \rightarrow A$.

6.5.5 AddInitialObject (for IsCapCategory, IsFunction)

▷ `AddInitialObject(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 `InitialObject`. $F : () \mapsto I$.

6.5.6 AddUniversalMorphismFromInitialObject (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismFromInitialObject(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 UniversalMorphismFromInitialObject. $F : A \mapsto u(A)$.

6.5.7 AddUniversalMorphismFromInitialObjectWithGivenInitialObject (for IsCap-Category, IsFunction)

▷ AddUniversalMorphismFromInitialObjectWithGivenInitialObject(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 UniversalMorphismFromInitialObjectWithGivenInitialObject. $F : (A, I) \mapsto u(A)$.

6.5.8 InitialObjectFunctorial (for IsCapCategory)

▷ InitialObjectFunctorial(C) (attribute)

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

The argument is a category C . The output is the unique morphism $\text{InitialObject} \rightarrow \text{InitialObject}$.

6.5.9 AddInitialObjectFunctorial (for IsCapCategory, IsFunction)

▷ AddInitialObjectFunctorial(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 InitialObjectFunctorial. $F : () \rightarrow (I \rightarrow I)$.

6.6 Direct Sum

For a given list $D = (S_1, \dots, S_n)$ in an Ab-category, a direct sum consists of five parts:

- an object S ,
- a list of morphisms $\pi = (\pi_i : S \rightarrow S_i)_{i=1\dots n}$,
- a list of morphisms $\iota = (\iota_i : S_i \rightarrow S)_{i=1\dots n}$,
- a dependent function u_{in} mapping every list $\tau = (\tau_i : T \rightarrow S_i)_{i=1\dots n}$ to a morphism $u_{\text{in}}(\tau) : T \rightarrow S$ such that $\pi_i \circ u_{\text{in}}(\tau) \sim_{T, S_i} \tau_i$ for all $i = 1, \dots, n$.
- a dependent function u_{out} mapping every list $\tau = (\tau_i : S_i \rightarrow T)_{i=1\dots n}$ to a morphism $u_{\text{out}}(\tau) : S \rightarrow T$ such that $u_{\text{out}}(\tau) \circ \iota_i \sim_{S_i, T} \tau_i$ for all $i = 1, \dots, n$,

such that

- $\sum_{i=1}^n \iota_i \circ \pi_i = \text{id}_S$,
- $\pi_j \circ \iota_i = \delta_{i,j}$,

where $\delta_{i,j} \in \text{Hom}(S_i, S_j)$ is the identity if $i = j$, and 0 otherwise. The 5-tuple $(S, \pi, \iota, u_{\text{in}}, u_{\text{out}})$ is called a *direct sum* of D . We denote the object S of such a 5-tuple by $\bigoplus_{i=1}^n S_i$. We say that the morphisms $u_{\text{in}}(\tau), u_{\text{out}}(\tau)$ are induced by the *universal property of the direct sum*.

DirectSum is a functorial operation. This means: For $(\mu_i : S_i \rightarrow S'_i)_{i=1 \dots n}$, we obtain a morphism $\bigoplus_{i=1}^n S_i \rightarrow \bigoplus_{i=1}^n S'_i$.

6.6.1 DirectSumOp (for IsList, IsCapCategoryObject)

▷ DirectSumOp(D , *method_selection_object*) (operation)

Returns: an object

The argument is a list of objects $D = (S_1, \dots, S_n)$ and an object for method selection. The output is the direct sum $\bigoplus_{i=1}^n S_i$.

6.6.2 ProjectionInFactorOfDirectSum (for IsList, IsInt)

▷ ProjectionInFactorOfDirectSum(D , k) (operation)

Returns: a morphism in $\text{Hom}(\bigoplus_{i=1}^n S_i, S_k)$

The arguments are a list of objects $D = (S_1, \dots, S_n)$ and an integer k . The output is the k -th projection $\pi_k : \bigoplus_{i=1}^n S_i \rightarrow S_k$.

6.6.3 ProjectionInFactorOfDirectSumOp (for IsList, IsInt, IsCapCategoryObject)

▷ ProjectionInFactorOfDirectSumOp(D , k , *method_selection_object*) (operation)

Returns: a morphism in $\text{Hom}(\bigoplus_{i=1}^n S_i, S_k)$

The arguments are a list of objects $D = (S_1, \dots, S_n)$, an integer k , and an object for method selection. The output is the k -th projection $\pi_k : \bigoplus_{i=1}^n S_i \rightarrow S_k$.

6.6.4 ProjectionInFactorOfDirectSumWithGivenDirectSum (for IsList, IsInt, IsCapCategoryObject)

▷ ProjectionInFactorOfDirectSumWithGivenDirectSum(D , k , S) (operation)

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

The arguments are a list of objects $D = (S_1, \dots, S_n)$, an integer k , and an object $S = \bigoplus_{i=1}^n S_i$. The output is the k -th projection $\pi_k : S \rightarrow S_k$.

6.6.5 InjectionOfCofactorOfDirectSum (for IsList, IsInt)

▷ InjectionOfCofactorOfDirectSum(D , k) (operation)

Returns: a morphism in $\text{Hom}(S_k, \bigoplus_{i=1}^n S_i)$

The arguments are a list of objects $D = (S_1, \dots, S_n)$ and an integer k . The output is the k -th injection $\iota_k : S_k \rightarrow \bigoplus_{i=1}^n S_i$.

6.6.6 InjectionOfCofactorOfDirectSumOp (for IsList, IsInt, IsCapCategoryObject)

▷ InjectionOfCofactorOfDirectSumOp(D , k , *method_selection_object*) (operation)

Returns: a morphism in $\text{Hom}(S_k, \bigoplus_{i=1}^n S_i)$

The arguments are a list of objects $D = (S_1, \dots, S_n)$, an integer k , and an object for method selection. The output is the k -th injection $\iota_k : S_k \rightarrow \bigoplus_{i=1}^n S_i$.

6.6.7 InjectionOfCofactorOfDirectSumWithGivenDirectSum (for IsList, IsInt, IsCapCategoryObject)

▷ `InjectionOfCofactorOfDirectSumWithGivenDirectSum(D, k, S)` (operation)

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

The arguments are a list of objects $D = (S_1, \dots, S_n)$, an integer k , and an object $S = \bigoplus_{i=1}^n S_i$. The output is the k -th injection $\iota_k : S_k \rightarrow S$.

6.6.8 UniversalMorphismIntoDirectSum

▷ `UniversalMorphismIntoDirectSum(arg)` (function)

Returns: a morphism in $\text{Hom}(T, \bigoplus_{i=1}^n S_i)$

This is a convenience method. There are three different ways to use this method:

- The arguments are a list of objects $D = (S_1, \dots, S_n)$ and a list of morphisms $\tau = (\tau_i : T \rightarrow S_i)_{i=1..n}$.
- The argument is a list of morphisms $\tau = (\tau_i : T \rightarrow S_i)_{i=1..n}$.
- The arguments are morphisms $\tau_1 : T \rightarrow S_1, \dots, \tau_n : T \rightarrow S_n$.

The output is the morphism $u_{\text{in}}(\tau) : T \rightarrow \bigoplus_{i=1}^n S_i$ given by the universal property of the direct sum.

6.6.9 UniversalMorphismIntoDirectSumOp (for IsList, IsList, IsCapCategoryObject)

▷ `UniversalMorphismIntoDirectSumOp(D, tau, method_selection_object)` (operation)

Returns: a morphism in $\text{Hom}(T, \bigoplus_{i=1}^n S_i)$

The arguments are a list of objects $D = (S_1, \dots, S_n)$, a list of morphisms $\tau = (\tau_i : T \rightarrow S_i)_{i=1..n}$, and an object for method selection. The output is the morphism $u_{\text{in}}(\tau) : T \rightarrow \bigoplus_{i=1}^n S_i$ given by the universal property of the direct sum.

6.6.10 UniversalMorphismIntoDirectSumWithGivenDirectSum (for IsList, IsList, IsCapCategoryObject)

▷ `UniversalMorphismIntoDirectSumWithGivenDirectSum(D, tau, S)` (operation)

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

The arguments are a list of objects $D = (S_1, \dots, S_n)$, a list of morphisms $\tau = (\tau_i : T \rightarrow S_i)_{i=1..n}$, and an object $S = \bigoplus_{i=1}^n S_i$. The output is the morphism $u_{\text{in}}(\tau) : T \rightarrow S$ given by the universal property of the direct sum.

6.6.11 UniversalMorphismFromDirectSum

▷ `UniversalMorphismFromDirectSum(arg)` (function)

Returns: a morphism in $\text{Hom}(\bigoplus_{i=1}^n S_i, T)$

This is a convenience method. There are three different ways to use this method:

- The arguments are a list of objects $D = (S_1, \dots, S_n)$ and a list of morphisms $\tau = (\tau_i : S_i \rightarrow T)_{i=1..n}$.
- The argument is a list of morphisms $\tau = (\tau_i : S_i \rightarrow T)_{i=1..n}$.

- The arguments are morphisms $S_1 \rightarrow T, \dots, S_n \rightarrow T$.

The output is the morphism $u_{\text{out}}(\tau) : \bigoplus_{i=1}^n S_i \rightarrow T$ given by the universal property of the direct sum.

6.6.12 UniversalMorphismFromDirectSumOp (for IsList, IsList, IsCapCategoryObject)

▷ `UniversalMorphismFromDirectSumOp(D , τ , $\text{method_selection_object}$)` (operation)

Returns: a morphism in $\text{Hom}(\bigoplus_{i=1}^n S_i, T)$

The arguments are a list of objects $D = (S_1, \dots, S_n)$, a list of morphisms $\tau = (\tau_i : S_i \rightarrow T)_{i=1 \dots n}$, and an object for method selection. The output is the morphism $u_{\text{out}}(\tau) : \bigoplus_{i=1}^n S_i \rightarrow T$ given by the universal property of the direct sum.

6.6.13 UniversalMorphismFromDirectSumWithGivenDirectSum (for IsList, IsList, IsCapCategoryObject)

▷ `UniversalMorphismFromDirectSumWithGivenDirectSum(D , τ , S)` (operation)

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

The arguments are a list of objects $D = (S_1, \dots, S_n)$, a list of morphisms $\tau = (\tau_i : S_i \rightarrow T)_{i=1 \dots n}$, and an object $S = \bigoplus_{i=1}^n S_i$. The output is the morphism $u_{\text{out}}(\tau) : S \rightarrow T$ given by the universal property of the direct sum.

6.6.14 IsomorphismFromDirectSumToDirectProduct (for IsList)

▷ `IsomorphismFromDirectSumToDirectProduct(D)` (operation)

Returns: a morphism in $\text{Hom}(\bigoplus_{i=1}^n S_i, \prod_{i=1}^n S_i)$

The argument is a list of objects $D = (S_1, \dots, S_n)$. The output is the canonical isomorphism $\bigoplus_{i=1}^n S_i \rightarrow \prod_{i=1}^n S_i$.

6.6.15 IsomorphismFromDirectSumToDirectProductOp (for IsList, IsCapCategoryObject)

▷ `IsomorphismFromDirectSumToDirectProductOp(D , $\text{method_selection_object}$)` (operation)

Returns: a morphism in $\text{Hom}(\bigoplus_{i=1}^n S_i, \prod_{i=1}^n S_i)$

The arguments are a list of objects $D = (S_1, \dots, S_n)$ and an object for method selection. The output is the canonical isomorphism $\bigoplus_{i=1}^n S_i \rightarrow \prod_{i=1}^n S_i$.

6.6.16 IsomorphismFromDirectProductToDirectSum (for IsList)

▷ `IsomorphismFromDirectProductToDirectSum(D)` (operation)

Returns: a morphism in $\text{Hom}(\prod_{i=1}^n S_i, \bigoplus_{i=1}^n S_i)$

The argument is a list of objects $D = (S_1, \dots, S_n)$. The output is the canonical isomorphism $\prod_{i=1}^n S_i \rightarrow \bigoplus_{i=1}^n S_i$.

6.6.17 IsomorphismFromDirectProductToDirectSumOp (for IsList, IsCapCategoryObject)

▷ `IsomorphismFromDirectProductToDirectSumOp(D , $\text{method_selection_object}$)` (operation)

Returns: a morphism in $\text{Hom}(\prod_{i=1}^n S_i, \bigoplus_{i=1}^n S_i)$

The argument is a list of objects $D = (S_1, \dots, S_n)$ and an object for method selection. The output is the canonical isomorphism $\prod_{i=1}^n S_i \rightarrow \bigoplus_{i=1}^n S_i$.

6.6.18 IsomorphismFromDirectSumToCoproduct (for IsList)

▷ `IsomorphismFromDirectSumToCoproduct(D)` (operation)

Returns: a morphism in $\text{Hom}(\bigoplus_{i=1}^n S_i, \bigsqcup_{i=1}^n S_i)$

The argument is a list of objects $D = (S_1, \dots, S_n)$. The output is the canonical isomorphism $\bigoplus_{i=1}^n S_i \rightarrow \bigsqcup_{i=1}^n S_i$.

6.6.19 IsomorphismFromDirectSumToCoproductOp (for IsList, IsCapCategoryObject)

▷ `IsomorphismFromDirectSumToCoproductOp(D, method_selection_object)` (operation)

Returns: a morphism in $\text{Hom}(\bigoplus_{i=1}^n S_i, \bigsqcup_{i=1}^n S_i)$

The argument is a list of objects $D = (S_1, \dots, S_n)$ and an object for method selection. The output is the canonical isomorphism $\bigoplus_{i=1}^n S_i \rightarrow \bigsqcup_{i=1}^n S_i$.

6.6.20 IsomorphismFromCoproductToDirectSum (for IsList)

▷ `IsomorphismFromCoproductToDirectSum(D)` (operation)

Returns: a morphism in $\text{Hom}(\bigsqcup_{i=1}^n S_i, \bigoplus_{i=1}^n S_i)$

The argument is a list of objects $D = (S_1, \dots, S_n)$. The output is the canonical isomorphism $\bigsqcup_{i=1}^n S_i \rightarrow \bigoplus_{i=1}^n S_i$.

6.6.21 IsomorphismFromCoproductToDirectSumOp (for IsList, IsCapCategoryObject)

▷ `IsomorphismFromCoproductToDirectSumOp(D, method_selection_object)` (operation)

Returns: a morphism in $\text{Hom}(\bigsqcup_{i=1}^n S_i, \bigoplus_{i=1}^n S_i)$

The argument is a list of objects $D = (S_1, \dots, S_n)$ and an object for method selection. The output is the canonical isomorphism $\bigsqcup_{i=1}^n S_i \rightarrow \bigoplus_{i=1}^n S_i$.

6.6.22 MorphismBetweenDirectSums (for IsList)

▷ `MorphismBetweenDirectSums(M)` (operation)

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

Returns: a morphism in $\text{Hom}(\bigoplus_{i=1}^m A_i, \bigoplus_{j=1}^n B_j)$

The argument $M = ((\phi_{i,j} : A_i \rightarrow B_j)_{j=1 \dots n})_{i=1 \dots m}$ is a list of lists of morphisms. The output is the morphism $\bigoplus_{i=1}^m A_i \rightarrow \bigoplus_{j=1}^n B_j$ defined by the matrix M . The extra arguments $S = \bigoplus_{i=1}^m A_i$ and $T = \bigoplus_{j=1}^n B_j$ are source and target of the output, respectively. They must be provided in case M is an empty list or a list of empty lists.

6.6.23 MorphismBetweenDirectSumsOp (for IsList, IsInt, IsInt, IsCapCategoryMorphism)

▷ `MorphismBetweenDirectSumsOp(M, m, n, method_selection_morphism)` (operation)

Returns: a morphism in $\text{Hom}(\bigoplus_{i=1}^m A_i, \bigoplus_{j=1}^n B_j)$

The arguments are a list $M = (\phi_{1,1}, \phi_{1,2}, \dots, \phi_{1,n}, \phi_{2,1}, \dots, \phi_{m,n})$ of morphisms $\phi_{i,j} : A_i \rightarrow B_j$, an integer m , an integer n , and a method selection morphism. The output is the morphism $\bigoplus_{i=1}^m A_i \rightarrow \bigoplus_{j=1}^n B_j$ defined by the list M regarded as a matrix of dimension $m \times n$.

6.6.24 AddProjectionInFactorOfDirectSum (for IsCapCategory, IsFunction)

▷ AddProjectionInFactorOfDirectSum(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 ProjectionInFactorOfDirectSum. $F : (D, k) \mapsto \pi_k$.

6.6.25 AddProjectionInFactorOfDirectSumWithGivenDirectSum (for IsCapCategory, IsFunction)

▷ AddProjectionInFactorOfDirectSumWithGivenDirectSum(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 ProjectionInFactorOfDirectSumWithGivenDirectSum. $F : (D, k, S) \mapsto \pi_k$.

6.6.26 AddInjectionOfCofactorOfDirectSum (for IsCapCategory, IsFunction)

▷ AddInjectionOfCofactorOfDirectSum(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 InjectionOfCofactorOfDirectSum. $F : (D, k) \mapsto \iota_k$.

6.6.27 AddInjectionOfCofactorOfDirectSumWithGivenDirectSum (for IsCapCategory, IsFunction)

▷ AddInjectionOfCofactorOfDirectSumWithGivenDirectSum(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 InjectionOfCofactorOfDirectSumWithGivenDirectSum. $F : (D, k, S) \mapsto \iota_k$.

6.6.28 AddUniversalMorphismIntoDirectSum (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismIntoDirectSum(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 UniversalMorphismIntoDirectSum. $F : (D, \tau) \mapsto u_{\text{in}}(\tau)$.

6.6.29 AddUniversalMorphismIntoDirectSumWithGivenDirectSum (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismIntoDirectSumWithGivenDirectSum(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 `UniversalMorphismIntoDirectSumWithGivenDirectSum`. $F : (D, \tau, S) \mapsto u_{\text{in}}(\tau)$.

6.6.30 AddUniversalMorphismFromDirectSum (for IsCapCategory, IsFunction)

▷ `AddUniversalMorphismFromDirectSum(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 `UniversalMorphismFromDirectSum`. $F : (D, \tau) \mapsto u_{\text{out}}(\tau)$.

6.6.31 AddUniversalMorphismFromDirectSumWithGivenDirectSum (for IsCapCategory, IsFunction)

▷ `AddUniversalMorphismFromDirectSumWithGivenDirectSum(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 `UniversalMorphismFromDirectSumWithGivenDirectSum`. $F : (D, \tau, S) \mapsto u_{\text{out}}(\tau)$.

6.6.32 AddIsomorphismFromDirectSumToDirectProduct (for IsCapCategory, IsFunction)

▷ `AddIsomorphismFromDirectSumToDirectProduct(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 `IsomorphismFromDirectSumToDirectProduct`. $F : D \mapsto (\bigoplus_{i=1}^n S_i \rightarrow \prod_{i=1}^n S_i)$.

6.6.33 AddIsomorphismFromDirectProductToDirectSum (for IsCapCategory, IsFunction)

▷ `AddIsomorphismFromDirectProductToDirectSum(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 `IsomorphismFromDirectProductToDirectSum`. $F : D \mapsto (\prod_{i=1}^n S_i \rightarrow \bigoplus_{i=1}^n S_i)$.

6.6.34 AddIsomorphismFromDirectSumToCoproduct (for IsCapCategory, IsFunction)

▷ `AddIsomorphismFromDirectSumToCoproduct(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 `IsomorphismFromDirectSumToCoproduct`. $F : D \mapsto (\bigoplus_{i=1}^n S_i \rightarrow \bigsqcup_{i=1}^n S_i)$.

6.6.35 AddIsomorphismFromCoproductToDirectSum (for IsCapCategory, IsFunction)

▷ AddIsomorphismFromCoproductToDirectSum(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 IsomorphismFromCoproductToDirectSum. $F : D \mapsto (\bigsqcup_{i=1}^n S_i \rightarrow \bigoplus_{i=1}^n S_i)$.

6.6.36 AddDirectSum (for IsCapCategory, IsFunction)

▷ AddDirectSum(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 DirectSum. $F : D \mapsto \bigoplus_{i=1}^n S_i$.

6.6.37 DirectSumFunctorial (for IsList)

▷ DirectSumFunctorial(L) (operation)

Returns: a morphism in $\text{Hom}(\bigoplus_{i=1}^n S_i, \bigoplus_{i=1}^n S'_i)$

The argument is a list of morphisms $L = (\mu_1 : S_1 \rightarrow S'_1, \dots, \mu_n : S_n \rightarrow S'_n)$. The output is a morphism $\bigoplus_{i=1}^n S_i \rightarrow \bigoplus_{i=1}^n S'_i$ given by the functoriality of the direct sum.

6.6.38 DirectSumFunctorialWithGivenDirectSums (for IsCapCategoryObject, IsList, IsCapCategoryObject)

▷ DirectSumFunctorialWithGivenDirectSums(d_1, L, d_2) (operation)

Returns: a morphism in $\text{Hom}(d_1, d_2)$

The arguments are an object $d_1 = \bigoplus_{i=1}^n S_i$, a list of morphisms $L = (\mu_1 : S_1 \rightarrow S'_1, \dots, \mu_n : S_n \rightarrow S'_n)$, and an object $d_2 = \bigoplus_{i=1}^n S'_i$. The output is a morphism $d_1 \rightarrow d_2$ given by the functoriality of the direct sum.

6.6.39 AddDirectSumFunctorialWithGivenDirectSums (for IsCapCategory, IsFunction)

▷ AddDirectSumFunctorialWithGivenDirectSums(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 DirectSumFunctorialWithGivenDirectSums. $F : (\bigoplus_{i=1}^n S_i, (\mu_1, \dots, \mu_n), \bigoplus_{i=1}^n S'_i) \mapsto (\bigoplus_{i=1}^n S_i \rightarrow \bigoplus_{i=1}^n S'_i)$.

6.7 Coproduct

For a given list of objects $D = (I_1, \dots, I_n)$, a coproduct of D consists of three parts:

- an object I ,
- a list of morphisms $\iota = (\iota_i : I_i \rightarrow I)_{i=1 \dots n}$

- a dependent function u mapping each list of morphisms $\tau = (\tau_i : I_i \rightarrow T)$ to a morphism $u(\tau) : I \rightarrow T$ such that $u(\tau) \circ \iota_i \sim_{I,T} \tau_i$ for all $i = 1, \dots, n$.

The triple (I, ι, u) is called a *coproduct* of D if the morphisms $u(\tau)$ are uniquely determined up to congruence of morphisms. We denote the object I of such a triple by $\bigsqcup_{i=1}^n I_i$. We say that the morphism $u(\tau)$ is induced by the *universal property of the coproduct*.

Coproduct is a functorial operation. This means: For $(\mu_i : I_i \rightarrow I'_i)_{i=1\dots n}$, we obtain a morphism $\bigsqcup_{i=1}^n I_i \rightarrow \bigsqcup_{i=1}^n I'_i$.

6.7.1 Coproduct (for IsList)

▷ Coproduct(D) (attribute)

Returns: an object

The argument is a list of objects $D = (I_1, \dots, I_n)$. The output is the coproduct $\bigsqcup_{i=1}^n I_i$.

6.7.2 Coproduct (for IsCapCategoryObject, IsCapCategoryObject)

▷ Coproduct($I1, I2$) (operation)

Returns: an object

This is a convenience method. The arguments are two objects I_1, I_2 . The output is the coproduct $I_1 \sqcup I_2$.

6.7.3 Coproduct (for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject)

▷ Coproduct($I1, I2$) (operation)

Returns: an object

This is a convenience method. The arguments are three objects I_1, I_2, I_3 . The output is the coproduct $I_1 \sqcup I_2 \sqcup I_3$.

6.7.4 CoproductOp (for IsList, IsCapCategoryObject)

▷ CoproductOp($D, method_selection_object$) (operation)

Returns: an object

The arguments are a list of objects $D = (I_1, \dots, I_n)$ and a method selection object. The output is the coproduct $\bigsqcup_{i=1}^n I_i$.

6.7.5 InjectionOfCofactorOfCoproduct (for IsList, IsInt)

▷ InjectionOfCofactorOfCoproduct(D, k) (operation)

Returns: a morphism in $\text{Hom}(I_k, \bigsqcup_{i=1}^n I_i)$

The arguments are a list of objects $D = (I_1, \dots, I_n)$ and an integer k . The output is the k -th injection $\iota_k : I_k \rightarrow \bigsqcup_{i=1}^n I_i$.

6.7.6 InjectionOfCofactorOfCoproductOp (for IsList, IsInt, IsCapCategoryObject)

▷ InjectionOfCofactorOfCoproductOp($D, k, method_selection_object$) (operation)

Returns: a morphism in $\text{Hom}(I_k, \bigsqcup_{i=1}^n I_i)$

The arguments are a list of objects $D = (I_1, \dots, I_n)$, an integer k , and a method selection object. The output is the k -th injection $\iota_k : I_k \rightarrow \bigsqcup_{i=1}^n I_i$.

6.7.7 InjectionOfCofactorOfCoproductWithGivenCoproduct (for IsList, IsInt, IsCapCategoryObject)

▷ `InjectionOfCofactorOfCoproductWithGivenCoproduct(D, k, I)` (operation)

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

The arguments are a list of objects $D = (I_1, \dots, I_n)$, an integer k , and an object $I = \bigsqcup_{i=1}^n I_i$. The output is the k -th injection $\iota_k : I_k \rightarrow I$.

6.7.8 UniversalMorphismFromCoproduct

▷ `UniversalMorphismFromCoproduct(arg)` (function)

Returns: a morphism in $\text{Hom}(\bigsqcup_{i=1}^n I_i, T)$

This is a convenience method. There are three different ways to use this method.

- The arguments are a list of objects $D = (I_1, \dots, I_n)$, a list of morphisms $\tau = (\tau_i : I_i \rightarrow T)$.
- The argument is a list of morphisms $\tau = (\tau_i : I_i \rightarrow T)$.
- The arguments are morphisms $\tau_1 : I_1 \rightarrow T, \dots, \tau_n : I_n \rightarrow T$

The output is the morphism $u(\tau) : \bigsqcup_{i=1}^n I_i \rightarrow T$ given by the universal property of the coproduct.

6.7.9 UniversalMorphismFromCoproductOp (for IsList, IsList, IsCapCategoryObject)

▷ `UniversalMorphismFromCoproductOp(D, tau, method_selection_object)` (operation)

Returns: a morphism in $\text{Hom}(\bigsqcup_{i=1}^n I_i, T)$

The arguments are a list of objects $D = (I_1, \dots, I_n)$, a list of morphisms $\tau = (\tau_i : I_i \rightarrow T)$, and a method selection object. The output is the morphism $u(\tau) : \bigsqcup_{i=1}^n I_i \rightarrow T$ given by the universal property of the coproduct.

6.7.10 UniversalMorphismFromCoproductWithGivenCoproduct (for IsList, IsList, IsCapCategoryObject)

▷ `UniversalMorphismFromCoproductWithGivenCoproduct(D, tau, I)` (operation)

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

The arguments are a list of objects $D = (I_1, \dots, I_n)$, a list of morphisms $\tau = (\tau_i : I_i \rightarrow T)$, and an object $I = \bigsqcup_{i=1}^n I_i$. The output is the morphism $u(\tau) : I \rightarrow T$ given by the universal property of the coproduct.

6.7.11 AddCoproduct (for IsCapCategory, IsFunction)

▷ `AddCoproduct(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 Coproduct. $F : ((I_1, \dots, I_n)) \mapsto I$.

6.7.12 AddInjectionOfCofactorOfCoproduct (for IsCapCategory, IsFunction)

▷ AddInjectionOfCofactorOfCoproduct(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 InjectionOfCofactorOfCoproduct. $F : ((I_1, \dots, I_n), i) \mapsto i_i$.

6.7.13 AddInjectionOfCofactorOfCoproductWithGivenCoproduct (for IsCapCategory, IsFunction)

▷ AddInjectionOfCofactorOfCoproductWithGivenCoproduct(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 InjectionOfCofactorOfCoproductWithGivenCoproduct. $F : ((I_1, \dots, I_n), i, I) \mapsto i_i$.

6.7.14 AddUniversalMorphismFromCoproduct (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismFromCoproduct(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 UniversalMorphismFromCoproduct. $F : ((I_1, \dots, I_n), \tau) \mapsto u(\tau)$.

6.7.15 AddUniversalMorphismFromCoproductWithGivenCoproduct (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismFromCoproductWithGivenCoproduct(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 UniversalMorphismFromCoproductWithGivenCoproduct. $F : ((I_1, \dots, I_n), \tau, I) \mapsto u(\tau)$.

6.7.16 CoproductFunctorial (for IsList)

▷ CoproductFunctorial(L) (operation)

Returns: a morphism in $\text{Hom}(\bigsqcup_{i=1}^n I_i, \bigsqcup_{i=1}^n I'_i)$

The argument is a list $L = (\mu_1 : I_1 \rightarrow I'_1, \dots, \mu_n : I_n \rightarrow I'_n)$. The output is a morphism $\bigsqcup_{i=1}^n I_i \rightarrow \bigsqcup_{i=1}^n I'_i$ given by the functoriality of the coproduct.

6.7.17 CoproductFunctorialWithGivenCoproducts (for IsCapCategoryObject, IsList, IsCapCategoryObject)

▷ CoproductFunctorialWithGivenCoproducts(s, L, r) (operation)

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

The arguments are an object $s = \bigsqcup_{i=1}^n I_i$, a list $L = (\mu_1 : I_1 \rightarrow I'_1, \dots, \mu_n : I_n \rightarrow I'_n)$, and an object $r = \bigsqcup_{i=1}^n I'_i$. The output is a morphism $\bigsqcup_{i=1}^n I_i \rightarrow \bigsqcup_{i=1}^n I'_i$ given by the functoriality of the coproduct.

6.7.18 AddCoproductFunctorialWithGivenCoproducts (for IsCapCategory, IsFunction)

▷ AddCoproductFunctorialWithGivenCoproducts(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 CoproductFunctorialWithGivenCoproducts. $F : (\bigsqcup_{i=1}^n I_i, (\mu_1, \dots, \mu_n), \bigsqcup_{i=1}^n I'_i) \rightarrow (\bigsqcup_{i=1}^n I_i \rightarrow \bigsqcup_{i=1}^n I'_i)$.

6.8 Direct Product

For a given list of objects $D = (P_1, \dots, P_n)$, a direct product of D consists of three parts:

- an object P ,
- a list of morphisms $\pi = (\pi_i : P \rightarrow P_i)_{i=1 \dots n}$
- a dependent function u mapping each list of morphisms $\tau = (\tau_i : T \rightarrow P_i)_{i=1, \dots, n}$ to a morphism $u(\tau) : T \rightarrow P$ such that $\pi_i \circ u(\tau) \sim_{T, P_i} \tau_i$ for all $i = 1, \dots, n$.

The triple (P, π, u) is called a *direct product* of D if the morphisms $u(\tau)$ are uniquely determined up to congruence of morphisms. We denote the object P of such a triple by $\prod_{i=1}^n P_i$. We say that the morphism $u(\tau)$ is induced by the *universal property of the direct product*.

DirectProduct is a functorial operation. This means: For $(\mu_i : P_i \rightarrow P'_i)_{i=1 \dots n}$, we obtain a morphism $\prod_{i=1}^n P_i \rightarrow \prod_{i=1}^n P'_i$.

6.8.1 DirectProductOp (for IsList, IsCapCategoryObject)

▷ DirectProductOp(D) (operation)

Returns: an object

The arguments are a list of objects $D = (P_1, \dots, P_n)$ and an object for method selection. The output is the direct product $\prod_{i=1}^n P_i$.

6.8.2 ProjectionInFactorOfDirectProduct (for IsList, IsInt)

▷ ProjectionInFactorOfDirectProduct(D, k) (operation)

Returns: a morphism in $\text{Hom}(\prod_{i=1}^n P_i, P_k)$

The arguments are a list of objects $D = (P_1, \dots, P_n)$ and an integer k . The output is the k -th projection $\pi_k : \prod_{i=1}^n P_i \rightarrow P_k$.

6.8.3 ProjectionInFactorOfDirectProductOp (for IsList, IsInt, IsCapCategoryObject)

▷ ProjectionInFactorOfDirectProductOp($D, k, \text{method_selection_object}$) (operation)

Returns: a morphism in $\text{Hom}(\prod_{i=1}^n P_i, P_k)$

The arguments are a list of objects $D = (P_1, \dots, P_n)$, an integer k , and an object for method selection. The output is the k -th projection $\pi_k : \prod_{i=1}^n P_i \rightarrow P_k$.

6.8.4 ProjectionInFactorOfDirectProductWithGivenDirectProduct (for IsList, IsInt, IsCapCategoryObject)

▷ `ProjectionInFactorOfDirectProductWithGivenDirectProduct(D, k, P)` (operation)

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

The arguments are a list of objects $D = (P_1, \dots, P_n)$, an integer k , and an object $P = \prod_{i=1}^n P_i$. The output is the k -th projection $\pi_k : P \rightarrow P_k$.

6.8.5 UniversalMorphismIntoDirectProduct

▷ `UniversalMorphismIntoDirectProduct(arg)` (function)

Returns: a morphism in $\text{Hom}(T, \prod_{i=1}^n P_i)$

This is a convenience method. There are three different ways to use this method.

- The arguments are a list of objects $D = (P_1, \dots, P_n)$ and a list of morphisms $\tau = (\tau_i : T \rightarrow P_i)_{i=1, \dots, n}$.
- The argument is a list of morphisms $\tau = (\tau_i : T \rightarrow P_i)_{i=1, \dots, n}$.
- The arguments are morphisms $\tau_1 : T \rightarrow P_1, \dots, \tau_n : T \rightarrow P_n$.

The output is the morphism $u(\tau) : T \rightarrow \prod_{i=1}^n P_i$ given by the universal property of the direct product.

6.8.6 UniversalMorphismIntoDirectProductOp (for IsList, IsList, IsCapCategoryObject)

▷ `UniversalMorphismIntoDirectProductOp(D, tau, method_selection_object)` (operation)

Returns: a morphism in $\text{Hom}(T, \prod_{i=1}^n P_i)$

The arguments are a list of objects $D = (P_1, \dots, P_n)$, a list of morphisms $\tau = (\tau_i : T \rightarrow P_i)_{i=1, \dots, n}$, and an object for method selection. The output is the morphism $u(\tau) : T \rightarrow \prod_{i=1}^n P_i$ given by the universal property of the direct product.

6.8.7 UniversalMorphismIntoDirectProductWithGivenDirectProduct (for IsList, IsList, IsCapCategoryObject)

▷ `UniversalMorphismIntoDirectProductWithGivenDirectProduct(D, tau, P)` (operation)

Returns: a morphism in $\text{Hom}(T, \prod_{i=1}^n P_i)$

The arguments are a list of objects $D = (P_1, \dots, P_n)$, a list of morphisms $\tau = (\tau_i : T \rightarrow P_i)_{i=1, \dots, n}$, and an object $P = \prod_{i=1}^n P_i$. The output is the morphism $u(\tau) : T \rightarrow \prod_{i=1}^n P_i$ given by the universal property of the direct product.

6.8.8 AddDirectProduct (for IsCapCategory, IsFunction)

▷ `AddDirectProduct(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 `DirectProduct`. $F : ((P_1, \dots, P_n)) \mapsto P$

6.8.9 AddProjectionInFactorOfDirectProduct (for IsCapCategory, IsFunction)

▷ AddProjectionInFactorOfDirectProduct(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 ProjectionInFactorOfDirectProduct. $F : ((P_1, \dots, P_n), k) \mapsto \pi_k$

6.8.10 AddProjectionInFactorOfDirectProductWithGivenDirectProduct (for IsCapCategory, IsFunction)

▷ AddProjectionInFactorOfDirectProductWithGivenDirectProduct(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 ProjectionInFactorOfDirectProductWithGivenDirectProduct. $F : ((P_1, \dots, P_n), k, P) \mapsto \pi_k$

6.8.11 AddUniversalMorphismIntoDirectProduct (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismIntoDirectProduct(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 UniversalMorphismIntoDirectProduct. $F : ((P_1, \dots, P_n), \tau) \mapsto u(\tau)$

6.8.12 AddUniversalMorphismIntoDirectProductWithGivenDirectProduct (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismIntoDirectProductWithGivenDirectProduct(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 UniversalMorphismIntoDirectProductWithGivenDirectProduct. $F : ((P_1, \dots, P_n), \tau, P) \mapsto u(\tau)$

6.8.13 DirectProductFunctorial (for IsList)

▷ DirectProductFunctorial(L) (operation)

Returns: a morphism in $\text{Hom}(\prod_{i=1}^n P_i, \prod_{i=1}^n P'_i)$

The argument is a list of morphisms $L = (\mu_i : P_i \rightarrow P'_i)_{i=1 \dots n}$. The output is a morphism $\prod_{i=1}^n P_i \rightarrow \prod_{i=1}^n P'_i$ given by the functoriality of the direct product.

6.8.14 DirectProductFunctorialWithGivenDirectProducts (for IsCapCategoryObject, IsList, IsCapCategoryObject)

▷ DirectProductFunctorialWithGivenDirectProducts(s, L, r) (operation)

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

The arguments are an object $s = \prod_{i=1}^n P_i$, a list of morphisms $L = (\mu_i : P_i \rightarrow P'_i)_{i=1\dots n}$, and an object $r = \prod_{i=1}^n P'_i$. The output is a morphism $\prod_{i=1}^n P_i \rightarrow \prod_{i=1}^n P'_i$ given by the functoriality of the direct product.

6.8.15 AddDirectProductFunctorialWithGivenDirectProducts (for IsCapCategory, IsFunction)

▷ `AddDirectProductFunctorialWithGivenDirectProducts(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 `DirectProductFunctorialWithGivenDirectProducts`.
 $F : (\prod_{i=1}^n P_i, (\mu_i : P_i \rightarrow P'_i)_{i=1\dots n}, \prod_{i=1}^n P'_i) \mapsto (\prod_{i=1}^n P_i \rightarrow \prod_{i=1}^n P'_i)$

6.9 Fiber Product

For a given list of morphisms $D = (\beta_i : P_i \rightarrow B)_{i=1\dots n}$, a fiber product of D consists of three parts:

- an object P ,
- a list of morphisms $\pi = (\pi_i : P \rightarrow P_i)_{i=1\dots n}$ such that $\beta_i \circ \pi_i \sim_{P,B} \beta_j \circ \pi_j$ for all pairs i, j ,
- a dependent function u mapping each list of morphisms $\tau = (\tau_i : T \rightarrow P_i)$ such that $\beta_i \circ \tau_i \sim_{T,B} \beta_j \circ \tau_j$ for all pairs i, j to a morphism $u(\tau) : T \rightarrow P$ such that $\pi_i \circ u(\tau) \sim_{T,P_i} \tau_i$ for all $i = 1, \dots, n$.

The triple (P, π, u) is called a *fiber product* of D if the morphisms $u(\tau)$ are uniquely determined up to congruence of morphisms. We denote the object P of such a triple by `FiberProduct(D)`. We say that the morphism $u(\tau)$ is induced by the *universal property of the fiber product*.

`FiberProduct` is a functorial operation. This means: For a second diagram $D' = (\beta'_i : P'_i \rightarrow B')_{i=1\dots n}$ and a natural morphism between pullback diagrams (i.e., a collection of morphisms $(\mu_i : P_i \rightarrow P'_i)_{i=1\dots n}$ and $\beta : B \rightarrow B'$ such that $\beta'_i \circ \mu_i \sim_{P_i,B'} \beta \circ \beta_i$ for $i = 1, \dots, n$) we obtain a morphism `FiberProduct(D) → FiberProduct(D')`.

6.9.1 IsomorphismFromFiberProductToKernelOfDiagonalDifference (for IsList)

▷ `IsomorphismFromFiberProductToKernelOfDiagonalDifference(D)` (operation)

Returns: a morphism in $\text{Hom}(\text{FiberProduct}(D), \Delta)$

The argument is a list of morphisms $D = (\beta_i : P_i \rightarrow B)_{i=1\dots n}$. The output is a morphism `FiberProduct(D) → Δ`, where Δ denotes the kernel object equalizing the morphisms β_i .

6.9.2 IsomorphismFromFiberProductToKernelOfDiagonalDifferenceOp (for IsList, IsCapCategoryMorphism)

▷ `IsomorphismFromFiberProductToKernelOfDiagonalDifferenceOp(D, method_selection_morphism)` (operation)

Returns: a morphism in $\text{Hom}(\text{FiberProduct}(D), \Delta)$

The arguments are a list of morphisms $D = (\beta_i : P_i \rightarrow B)_{i=1\dots n}$ and a morphism for method selection. The output is a morphism `FiberProduct(D) → Δ`, where Δ denotes the kernel object equalizing the morphisms β_i .

6.9.3 AddIsomorphismFromFiberProductToKernelOfDiagonalDifference (for IsCap-Category, IsFunction)

▷ AddIsomorphismFromFiberProductToKernelOfDiagonalDifference(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 IsomorphismFromFiberProductToKernelOfDiagonalDifference. $F : ((\beta_i : P_i \rightarrow B)_{i=1\dots n}) \mapsto \text{FiberProduct}(D) \rightarrow \Delta$

6.9.4 IsomorphismFromKernelOfDiagonalDifferenceToFiberProduct (for IsList)

▷ IsomorphismFromKernelOfDiagonalDifferenceToFiberProduct(D) (operation)

Returns: a morphism in $\text{Hom}(\Delta, \text{FiberProduct}(D))$

The argument is a list of morphisms $D = (\beta_i : P_i \rightarrow B)_{i=1\dots n}$. The output is a morphism $\Delta \rightarrow \text{FiberProduct}(D)$, where Δ denotes the kernel object equalizing the morphisms β_i .

6.9.5 IsomorphismFromKernelOfDiagonalDifferenceToFiberProductOp (for IsList, IsCapCategoryMorphism)

▷ IsomorphismFromKernelOfDiagonalDifferenceToFiberProductOp(D) (operation)

Returns: a morphism in $\text{Hom}(\Delta, \text{FiberProduct}(D))$

The argument is a list of morphisms $D = (\beta_i : P_i \rightarrow B)_{i=1\dots n}$ and a morphism for method selection. The output is a morphism $\Delta \rightarrow \text{FiberProduct}(D)$, where Δ denotes the kernel object equalizing the morphisms β_i .

6.9.6 AddIsomorphismFromKernelOfDiagonalDifferenceToFiberProduct (for IsCap-Category, IsFunction)

▷ AddIsomorphismFromKernelOfDiagonalDifferenceToFiberProduct(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 IsomorphismFromKernelOfDiagonalDifferenceToFiberProduct. $F : ((\beta_i : P_i \rightarrow B)_{i=1\dots n}) \mapsto \Delta \rightarrow \text{FiberProduct}(D)$

6.9.7 DirectSumDiagonalDifference (for IsList)

▷ DirectSumDiagonalDifference(D) (operation)

Returns: a morphism in $\text{Hom}(\bigoplus_{i=1}^n P_i, B)$

The argument is a list of morphisms $D = (\beta_i : P_i \rightarrow B)_{i=1\dots n}$. The output is a morphism $\bigoplus_{i=1}^n P_i \rightarrow B$ such that its kernel equalizes the β_i .

6.9.8 DirectSumDiagonalDifferenceOp (for IsList, IsCapCategoryMorphism)

▷ DirectSumDiagonalDifferenceOp($D, \text{method_selection_morphism}$) (operation)

Returns: a morphism in $\text{Hom}(\bigoplus_{i=1}^n P_i, B)$

The argument is a list of morphisms $D = (\beta_i : P_i \rightarrow B)_{i=1\dots n}$ and a morphism for method selection. The output is a morphism $\bigoplus_{i=1}^n P_i \rightarrow B$ such that its kernel equalizes the β_i .

6.9.9 AddDirectSumDiagonalDifference (for IsCapCategory, IsFunction)

▷ `AddDirectSumDiagonalDifference(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 `DirectSumDiagonalDifference`. $F : (D) \mapsto \text{DirectSumDiagonalDifference}(D)$

6.9.10 FiberProductEmbeddingInDirectSum (for IsList)

▷ `FiberProductEmbeddingInDirectSum(D)` (operation)

Returns: a morphism in $\text{Hom}(\text{FiberProduct}(D), \bigoplus_{i=1}^n P_i)$

The argument is a list of morphisms $D = (\beta_i : P_i \rightarrow B)_{i=1\dots n}$. The output is the natural embedding $\text{FiberProduct}(D) \rightarrow \bigoplus_{i=1}^n P_i$.

6.9.11 FiberProductEmbeddingInDirectSumOp (for IsList, IsCapCategoryMorphism)

▷ `FiberProductEmbeddingInDirectSumOp(D, method_selection_morphism)` (operation)

Returns: a morphism in $\text{Hom}(\text{FiberProduct}(D), \bigoplus_{i=1}^n P_i)$

The argument is a list of morphisms $D = (\beta_i : P_i \rightarrow B)_{i=1\dots n}$ and a morphism for method selection. The output is the natural embedding $\text{FiberProduct}(D) \rightarrow \bigoplus_{i=1}^n P_i$.

6.9.12 AddFiberProductEmbeddingInDirectSum (for IsCapCategory, IsFunction)

▷ `AddFiberProductEmbeddingInDirectSum(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 `FiberProductEmbeddingInDirectSum`. $F : ((\beta_i : P_i \rightarrow B)_{i=1\dots n}) \mapsto \text{FiberProduct}(D) \rightarrow \bigoplus_{i=1}^n P_i$

6.9.13 FiberProduct

▷ `FiberProduct(arg)` (function)

Returns: an object

This is a convenience method. There are two different ways to use this method:

- The argument is a list of morphisms $D = (\beta_i : P_i \rightarrow B)_{i=1\dots n}$.
- The arguments are morphisms $\beta_1 : P_1 \rightarrow B, \dots, \beta_n : P_n \rightarrow B$.

The output is the fiber product $\text{FiberProduct}(D)$.

6.9.14 FiberProductOp (for IsList, IsCapCategoryMorphism)

▷ `FiberProductOp(D, method_selection_morphism)` (operation)

Returns: an object

The arguments are a list of morphisms $D = (\beta_i : P_i \rightarrow B)_{i=1\dots n}$ and a morphism for method selection. The output is the fiber product $\text{FiberProduct}(D)$.

6.9.15 ProjectionInFactorOfFiberProduct (for IsList, IsInt)

▷ `ProjectionInFactorOfFiberProduct(D, k)` (operation)

Returns: a morphism in $\text{Hom}(\text{FiberProduct}(D), P_k)$

The arguments are a list of morphisms $D = (\beta_i : P_i \rightarrow B)_{i=1\dots n}$ and an integer k . The output is the k -th projection $\pi_k : \text{FiberProduct}(D) \rightarrow P_k$.

6.9.16 ProjectionInFactorOfFiberProductOp (for IsList, IsInt, IsCapCategoryMorphism)

▷ `ProjectionInFactorOfFiberProductOp(D, k, method_selection_morphism)` (operation)

Returns: a morphism in $\text{Hom}(\text{FiberProduct}(D), P_k)$

The arguments are a list of morphisms $D = (\beta_i : P_i \rightarrow B)_{i=1\dots n}$, an integer k , and a morphism for method selection. The output is the k -th projection $\pi_k : \text{FiberProduct}(D) \rightarrow P_k$.

6.9.17 ProjectionInFactorOfFiberProductWithGivenFiberProduct (for IsList, IsInt, IsCapCategoryObject)

▷ `ProjectionInFactorOfFiberProductWithGivenFiberProduct(D, k, P)` (operation)

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

The arguments are a list of morphisms $D = (\beta_i : P_i \rightarrow B)_{i=1\dots n}$, an integer k , and an object $P = \text{FiberProduct}(D)$. The output is the k -th projection $\pi_k : P \rightarrow P_k$.

6.9.18 UniversalMorphismIntoFiberProduct

▷ `UniversalMorphismIntoFiberProduct(arg)` (function)

Returns:

This is a convenience method. There are three different ways to use this method:

- The arguments are a list of morphisms $D = (\beta_i : P_i \rightarrow B)_{i=1\dots n}$ and a list of morphisms $\tau = (\tau_i : T \rightarrow P_i)$ such that $\beta_i \circ \tau_i \sim_{T,B} \beta_j \circ \tau_j$ for all pairs i, j . The output is the morphism $u(\tau) : T \rightarrow \text{FiberProduct}(D)$ given by the universal property of the fiber product.
- The arguments are a list of morphisms $D = (\beta_i : P_i \rightarrow B)_{i=1\dots n}$ and morphisms $\tau_1 : T \rightarrow P_1, \dots, \tau_n : T \rightarrow P_n$ such that $\beta_i \circ \tau_i \sim_{T,B} \beta_j \circ \tau_j$ for all pairs i, j . The output is the morphism $u(\tau) : T \rightarrow \text{FiberProduct}(D)$ given by the universal property of the fiber product.

6.9.19 UniversalMorphismIntoFiberProductOp (for IsList, IsList, IsCapCategoryMorphism)

▷ `UniversalMorphismIntoFiberProductOp(D, tau, method_selection_morphism)` (operation)

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

The arguments are a list of morphisms $D = (\beta_i : P_i \rightarrow B)_{i=1\dots n}$, a list of morphisms $\tau = (\tau_i : T \rightarrow P_i)$ such that $\beta_i \circ \tau_i \sim_{T,B} \beta_j \circ \tau_j$ for all pairs i, j , and a morphism for method selection. The output is the morphism $u(\tau) : T \rightarrow \text{FiberProduct}(D)$ given by the universal property of the fiber product.

6.9.20 UniversalMorphismIntoFiberProductWithGivenFiberProduct (for IsList, IsList, IsCapCategoryObject)

▷ `UniversalMorphismIntoFiberProductWithGivenFiberProduct(D, tau, P)` (operation)

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

The arguments are a list of morphisms $D = (\beta_i : P_i \rightarrow B)_{i=1\dots n}$, a list of morphisms $\tau = (\tau_i : T \rightarrow P_i)$ such that $\beta_i \circ \tau_i \sim_{T,B} \beta_j \circ \tau_j$ for all pairs i, j , and an object $P = \text{FiberProduct}(D)$. The output is the morphism $u(\tau) : T \rightarrow P$ given by the universal property of the fiber product.

6.9.21 AddFiberProduct (for IsCapCategory, IsFunction)

▷ `AddFiberProduct(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`. $F : ((\beta_i : P_i \rightarrow B)_{i=1\dots n}) \mapsto P$

6.9.22 AddProjectionInFactorOfFiberProduct (for IsCapCategory, IsFunction)

▷ `AddProjectionInFactorOfFiberProduct(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`. $F : ((\beta_i : P_i \rightarrow B)_{i=1\dots n}, k) \mapsto \pi_k$

6.9.23 AddProjectionInFactorOfFiberProductWithGivenFiberProduct (for IsCapCategory, IsFunction)

▷ `AddProjectionInFactorOfFiberProductWithGivenFiberProduct(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`. $F : ((\beta_i : P_i \rightarrow B)_{i=1\dots n}, k, P) \mapsto \pi_k$

6.9.24 AddUniversalMorphismIntoFiberProduct (for IsCapCategory, IsFunction)

▷ `AddUniversalMorphismIntoFiberProduct(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`. $F : ((\beta_i : P_i \rightarrow B)_{i=1\dots n}, \tau) \mapsto u(\tau)$

6.9.25 AddUniversalMorphismIntoFiberProductWithGivenFiberProduct (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismIntoFiberProductWithGivenFiberProduct(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 UniversalMorphismIntoFiberProductWithGivenFiberProduct. $F : ((\beta_i : P_i \rightarrow B)_{i=1\dots n}, \tau, P) \mapsto u(\tau)$

6.9.26 FiberProductFunctorial (for IsList)

▷ FiberProductFunctorial(L) (operation)

Returns: a morphism in $\text{Hom}(\text{FiberProduct}((\beta_i)_{i=1\dots n}), \text{FiberProduct}((\beta'_i)_{i=1\dots n}))$

The argument is a list of triples of morphisms $L = ((\beta_i : P_i \rightarrow B, \mu_i : P_i \rightarrow P'_i, \beta'_i : P'_i \rightarrow B')_{i=1\dots n})$ such that there exists a morphism $\beta : B \rightarrow B'$ such that $\beta'_i \circ \mu_i \sim_{P_i, B'} \beta \circ \beta_i$ for $i = 1, \dots, n$. The output is the morphism $\text{FiberProduct}((\beta_i)_{i=1\dots n}) \rightarrow \text{FiberProduct}((\beta'_i)_{i=1\dots n})$ given by the functoriality of the fiber product.

6.9.27 FiberProductFunctorialWithGivenFiberProducts (for IsCapCategoryObject, IsList, IsCapCategoryObject)

▷ FiberProductFunctorialWithGivenFiberProducts(s, L, r) (operation)

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

The arguments are an object $s = \text{FiberProduct}((\beta_i)_{i=1\dots n})$, a list of triples of morphisms $L = ((\beta_i : P_i \rightarrow B, \mu_i : P_i \rightarrow P'_i, \beta'_i : P'_i \rightarrow B')_{i=1\dots n})$ such that there exists a morphism $\beta : B \rightarrow B'$ such that $\beta'_i \circ \mu_i \sim_{P_i, B'} \beta \circ \beta_i$ for $i = 1, \dots, n$, and an object $r = \text{FiberProduct}((\beta'_i)_{i=1\dots n})$. The output is the morphism $s \rightarrow r$ given by the functoriality of the fiber product.

6.9.28 AddFiberProductFunctorialWithGivenFiberProducts (for IsCapCategory, IsFunction)

▷ AddFiberProductFunctorialWithGivenFiberProducts(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 FiberProductFunctorialWithGivenFiberProducts. $F : (\text{FiberProduct}((\beta_i)_{i=1\dots n}), (\beta_i : P_i \rightarrow B, \mu_i : P_i \rightarrow P'_i, \beta'_i : P'_i \rightarrow B')_{i=1\dots n}, \text{FiberProduct}((\beta'_i)_{i=1\dots n})) \mapsto (\text{FiberProduct}((\beta_i)_{i=1\dots n}) \rightarrow \text{FiberProduct}((\beta'_i)_{i=1\dots n}))$

6.10 Pushout

For a given list of morphisms $D = (\beta_i : B \rightarrow I_i)_{i=1\dots n}$, a pushout of D consists of three parts:

- an object I ,
- a list of morphisms $\iota = (\iota_i : I_i \rightarrow I)_{i=1\dots n}$ such that $\iota_i \circ \beta_i \sim_{B, I} \iota_j \circ \beta_j$ for all pairs i, j ,
- a dependent function u mapping each list of morphisms $\tau = (\tau_i : I_i \rightarrow T)_{i=1\dots n}$ such that $\tau_i \circ \beta_i \sim_{B, T} \tau_j \circ \beta_j$ to a morphism $u(\tau) : I \rightarrow T$ such that $u(\tau) \circ \iota_i \sim_{I, T} \tau_i$ for all $i = 1, \dots, n$.

The triple (I, ι, u) is called a *pushout* of D if the morphisms $u(\tau)$ are uniquely determined up to congruence of morphisms. We denote the object I of such a triple by $\text{Pushout}(D)$. We say that the morphism $u(\tau)$ is induced by the *universal property of the pushout*.

Pushout is a functorial operation. This means: For a second diagram $D' = (\beta'_i : B' \rightarrow I'_i)_{i=1\dots n}$ and a natural morphism between pushout diagrams (i.e., a collection of morphisms $(\mu_i : I_i \rightarrow I'_i)_{i=1\dots n}$ and $\beta : B \rightarrow B'$ such that $\beta'_i \circ \beta \sim_{B, I'_i} \mu_i \circ \beta_i$ for $i = 1, \dots, n$) we obtain a morphism $\text{Pushout}(D) \rightarrow \text{Pushout}(D')$.

6.10.1 IsomorphismFromPushoutToCokernelOfDiagonalDifference (for IsList)

▷ `IsomorphismFromPushoutToCokernelOfDiagonalDifference(D)` (operation)

Returns: a morphism in $\text{Hom}(\text{Pushout}(D), \Delta)$

The argument is a list of morphisms $D = (\beta_i : B \rightarrow I_i)_{i=1\dots n}$. The output is a morphism $\text{Pushout}(D) \rightarrow \Delta$, where Δ denotes the cokernel object coequalizing the morphisms β_i .

6.10.2 IsomorphismFromPushoutToCokernelOfDiagonalDifferenceOp (for IsList, IsCapCategoryMorphism)

▷ `IsomorphismFromPushoutToCokernelOfDiagonalDifferenceOp(D, method_selection_morphism)` (operation)

Returns: a morphism in $\text{Hom}(\text{Pushout}(D), \Delta)$

The argument is a list of morphisms $D = (\beta_i : B \rightarrow I_i)_{i=1\dots n}$ and a morphism for method selection. The output is a morphism $\text{Pushout}(D) \rightarrow \Delta$, where Δ denotes the cokernel object coequalizing the morphisms β_i .

6.10.3 AddIsomorphismFromPushoutToCokernelOfDiagonalDifference (for IsCapCategory, IsFunction)

▷ `AddIsomorphismFromPushoutToCokernelOfDiagonalDifference(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 `IsomorphismFromPushoutToCokernelOfDiagonalDifference`. $F : ((\beta_i : B \rightarrow I_i)_{i=1\dots n}) \mapsto (\text{Pushout}(D) \rightarrow \Delta)$

6.10.4 IsomorphismFromCokernelOfDiagonalDifferenceToPushout (for IsList)

▷ `IsomorphismFromCokernelOfDiagonalDifferenceToPushout(D)` (operation)

Returns: a morphism in $\text{Hom}(\Delta, \text{Pushout}(D))$

The argument is a list of morphisms $D = (\beta_i : B \rightarrow I_i)_{i=1\dots n}$. The output is a morphism $\Delta \rightarrow \text{Pushout}(D)$, where Δ denotes the cokernel object coequalizing the morphisms β_i .

6.10.5 IsomorphismFromCokernelOfDiagonalDifferenceToPushoutOp (for IsList, IsCapCategoryMorphism)

▷ `IsomorphismFromCokernelOfDiagonalDifferenceToPushoutOp(D, method_selection_morphism)` (operation)

Returns: a morphism in $\text{Hom}(\Delta, \text{Pushout}(D))$

The argument is a list of morphisms $D = (\beta_i : B \rightarrow I_i)_{i=1\dots n}$ and a morphism for method selection. The output is a morphism $\Delta \rightarrow \text{Pushout}(D)$, where Δ denotes the cokernel object coequalizing the morphisms β_i .

6.10.6 AddIsomorphismFromCokernelOfDiagonalDifferenceToPushout (for IsCapCategory, IsFunction)

▷ `AddIsomorphismFromCokernelOfDiagonalDifferenceToPushout(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 `IsomorphismFromCokernelOfDiagonalDifferenceToPushout`. $F : ((\beta_i : B \rightarrow I_i)_{i=1\dots n}) \mapsto (\Delta \rightarrow \text{Pushout}(D))$

6.10.7 DirectSumCodiagonalDifference (for IsList)

▷ `DirectSumCodiagonalDifference(D)` (operation)

Returns: a morphism in $\text{Hom}(B, \bigoplus_{i=1}^n I_i)$

The argument is a list of morphisms $D = (\beta_i : B \rightarrow I_i)_{i=1\dots n}$. The output is a morphism $B \rightarrow \bigoplus_{i=1}^n I_i$ such that its cokernel coequalizes the β_i .

6.10.8 DirectSumCodiagonalDifferenceOp (for IsList, IsCapCategoryMorphism)

▷ `DirectSumCodiagonalDifferenceOp(D, method_selection_morphism)` (operation)

Returns: a morphism in $\text{Hom}(B, \bigoplus_{i=1}^n I_i)$

The argument is a list of morphisms $D = (\beta_i : B \rightarrow I_i)_{i=1\dots n}$ and a morphism for method selection. The output is a morphism $B \rightarrow \bigoplus_{i=1}^n I_i$ such that its cokernel coequalizes the β_i .

6.10.9 AddDirectSumCodiagonalDifference (for IsCapCategory, IsFunction)

▷ `AddDirectSumCodiagonalDifference(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 `DirectSumCodiagonalDifference`. $F : (D) \mapsto \text{DirectSumCodiagonalDifference}(D)$

6.10.10 DirectSumProjectionInPushout (for IsList)

▷ `DirectSumProjectionInPushout(D)` (operation)

Returns: a morphism in $\text{Hom}(\bigoplus_{i=1}^n I_i, \text{Pushout}(D))$

The argument is a list of morphisms $D = (\beta_i : B \rightarrow I_i)_{i=1\dots n}$. The output is the natural projection $\bigoplus_{i=1}^n I_i \rightarrow \text{Pushout}(D)$.

6.10.11 DirectSumProjectionInPushoutOp (for IsList, IsCapCategoryMorphism)

▷ `DirectSumProjectionInPushoutOp(D, method_selection_morphism)` (operation)

Returns: a morphism in $\text{Hom}(\bigoplus_{i=1}^n I_i, \text{Pushout}(D))$

The argument is a list of morphisms $D = (\beta_i : B \rightarrow I_i)_{i=1\dots n}$ and a morphism for method selection. The output is the natural projection $\bigoplus_{i=1}^n I_i \rightarrow \text{Pushout}(D)$.

6.10.12 AddDirectSumProjectionInPushout (for IsCapCategory, IsFunction)

▷ AddDirectSumProjectionInPushout(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 DirectSumProjectionInPushout. $F : ((\beta_i : B \rightarrow I_i)_{i=1\dots n}) \mapsto (\bigoplus_{i=1}^n I_i \rightarrow \text{Pushout}(D))$

6.10.13 Pushout (for IsList)

▷ Pushout(D) (operation)

Returns: an object

The argument is a list of morphisms $D = (\beta_i : B \rightarrow I_i)_{i=1\dots n}$. The output is the pushout $\text{Pushout}(D)$.

6.10.14 Pushout (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ Pushout(D) (operation)

Returns: an object

This is a convenience method. The arguments are a morphism α and a morphism β . The output is the pushout $\text{Pushout}(\alpha, \beta)$.

6.10.15 PushoutOp (for IsList, IsCapCategoryMorphism)

▷ PushoutOp(D) (operation)

Returns: an object

The arguments are a list of morphisms $D = (\beta_i : B \rightarrow I_i)_{i=1\dots n}$ and a morphism for method selection. The output is the pushout $\text{Pushout}(D)$.

6.10.16 InjectionOfCofactorOfPushout (for IsList, IsInt)

▷ InjectionOfCofactorOfPushout(D, k) (operation)

Returns: a morphism in $\text{Hom}(I_k, \text{Pushout}(D))$.

The arguments are a list of morphisms $D = (\beta_i : B \rightarrow I_i)_{i=1\dots n}$ and an integer k . The output is the k -th injection $\iota_k : I_k \rightarrow \text{Pushout}(D)$.

6.10.17 InjectionOfCofactorOfPushoutOp (for IsList, IsInt, IsCapCategoryMorphism)

▷ InjectionOfCofactorOfPushoutOp($D, k, \text{method_selection_morphism}$) (operation)

Returns: a morphism in $\text{Hom}(I_k, \text{Pushout}(D))$.

The arguments are a list of morphisms $D = (\beta_i : B \rightarrow I_i)_{i=1\dots n}$, an integer k , and a morphism for method selection. The output is the k -th injection $\iota_k : I_k \rightarrow \text{Pushout}(D)$.

6.10.18 InjectionOfCofactorOfPushoutWithGivenPushout (for IsList, IsInt, IsCapCategoryObject)

▷ InjectionOfCofactorOfPushoutWithGivenPushout(D, k, I) (operation)

Returns: a morphism in $\text{Hom}(I_k, \text{Pushout}(D))$.

The arguments are a list of morphisms $D = (\beta_i : B \rightarrow I_i)_{i=1\dots n}$, an integer k , and an object $I = \text{Pushout}(D)$. The output is the k -th injection $\iota_k : I_k \rightarrow \text{Pushout}(D)$.

6.10.19 UniversalMorphismFromPushout

▷ `UniversalMorphismFromPushout(arg)` (function)

Returns:

This is a convenience method. There are three different ways to use this method:

- The arguments are a list of morphisms $D = (\beta_i : B \rightarrow I_i)_{i=1\dots n}$ and a list of morphisms $\tau = (\tau_i : I_i \rightarrow T)_{i=1\dots n}$ such that $\tau_i \circ \beta_i \sim_{B,T} \tau_j \circ \beta_j$. The output is the morphism $u(\tau) : \text{Pushout}(D) \rightarrow T$ given by the universal property of the pushout.
- The arguments are a list of morphisms $D = (\beta_i : B \rightarrow I_i)_{i=1\dots n}$ and morphisms $\tau_1 : I_1 \rightarrow T, \dots, \tau_n : I_n \rightarrow T$ such that $\tau_i \circ \beta_i \sim_{B,T} \tau_j \circ \beta_j$. The output is the morphism $u(\tau) : \text{Pushout}(D) \rightarrow T$ given by the universal property of the pushout.

6.10.20 UniversalMorphismFromPushoutOp (for IsList, IsList, IsCapCategoryMorphism)

▷ `UniversalMorphismFromPushoutOp(D, tau, method_selection_morphism)` (operation)

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

The arguments are a list of morphisms $D = (\beta_i : B \rightarrow I_i)_{i=1\dots n}$, a list of morphisms $\tau = (\tau_i : I_i \rightarrow T)_{i=1\dots n}$ such that $\tau_i \circ \beta_i \sim_{B,T} \tau_j \circ \beta_j$, and a morphism for method selection. The output is the morphism $u(\tau) : \text{Pushout}(D) \rightarrow T$ given by the universal property of the pushout.

6.10.21 UniversalMorphismFromPushoutWithGivenPushout (for IsList, IsList, IsCapCategoryObject)

▷ `UniversalMorphismFromPushoutWithGivenPushout(D, tau, I)` (operation)

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

The arguments are a list of morphisms $D = (\beta_i : B \rightarrow I_i)_{i=1\dots n}$, a list of morphisms $\tau = (\tau_i : I_i \rightarrow T)_{i=1\dots n}$ such that $\tau_i \circ \beta_i \sim_{B,T} \tau_j \circ \beta_j$, and an object $I = \text{Pushout}(D)$. The output is the morphism $u(\tau) : I \rightarrow T$ given by the universal property of the pushout.

6.10.22 AddPushout (for IsCapCategory, IsFunction)

▷ `AddPushout(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`. $F : ((\beta_i : B \rightarrow I_i)_{i=1\dots n}) \mapsto I$

6.10.23 AddInjectionOfCofactorOfPushout (for IsCapCategory, IsFunction)

▷ `AddInjectionOfCofactorOfPushout(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`. $F : ((\beta_i : B \rightarrow I_i)_{i=1\dots n}, k) \mapsto \iota_k$

6.10.24 AddInjectionOfCofactorOfPushoutWithGivenPushout (for IsCapCategory, IsFunction)

▷ AddInjectionOfCofactorOfPushoutWithGivenPushout(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. $F : ((\beta_i : B \rightarrow I_i)_{i=1\dots n}, k, I) \mapsto \iota_k$

6.10.25 AddUniversalMorphismFromPushout (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismFromPushout(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. $F : ((\beta_i : B \rightarrow I_i)_{i=1\dots n}, \tau) \mapsto u(\tau)$

6.10.26 AddUniversalMorphismFromPushoutWithGivenPushout (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismFromPushoutWithGivenPushout(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. $F : ((\beta_i : B \rightarrow I_i)_{i=1\dots n}, \tau, I) \mapsto u(\tau)$

6.10.27 PushoutFunctorial (for IsList)

▷ PushoutFunctorial(L) (operation)

Returns: a morphism in $\text{Hom}(\text{Pushout}((\beta_i)_{i=1}^n), \text{Pushout}((\beta'_i)_{i=1}^n))$

The argument is a list $L = ((\beta_i : B \rightarrow I_i, \mu_i : I_i \rightarrow I'_i, \beta'_i : B' \rightarrow I'_i)_{i=1\dots n})$ such that there exists a morphism $\beta : B \rightarrow B'$ such that $\beta'_i \circ \beta \sim_{B, I'_i} \mu_i \circ \beta_i$ for $i = 1, \dots, n$. The output is the morphism $\text{Pushout}((\beta_i)_{i=1}^n) \rightarrow \text{Pushout}((\beta'_i)_{i=1}^n)$ given by the functoriality of the pushout.

6.10.28 PushoutFunctorialWithGivenPushouts (for IsCapCategoryObject, IsList, IsCapCategoryObject)

▷ PushoutFunctorialWithGivenPushouts(s, L, r) (operation)

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

The arguments are an object $s = \text{Pushout}((\beta_i)_{i=1}^n)$, a list $L = ((\beta_i : B \rightarrow I_i, \mu_i : I_i \rightarrow I'_i, \beta'_i : B' \rightarrow I'_i)_{i=1\dots n})$ such that there exists a morphism $\beta : B \rightarrow B'$ such that $\beta'_i \circ \beta \sim_{B, I'_i} \mu_i \circ \beta_i$ for $i = 1, \dots, n$, and an object $r = \text{Pushout}((\beta'_i)_{i=1}^n)$. The output is the morphism $s \rightarrow r$ given by the functoriality of the pushout.

6.10.29 AddPushoutFunctorialWithGivenPushouts (for IsCapCategory, IsFunction)

▷ AddPushoutFunctorialWithGivenPushouts(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 `PushoutFunctorial`. $F : (\text{Pushout}((\beta_i)_{i=1}^n), (\beta_i : B \rightarrow I_i, \mu_i : I_i \rightarrow I'_i, \beta'_i : B' \rightarrow I'_i)_{i=1 \dots n}, \text{Pushout}((\beta'_i)_{i=1}^n)) \mapsto (\text{Pushout}((\beta_i)_{i=1}^n) \rightarrow \text{Pushout}((\beta'_i)_{i=1}^n))$

6.11 Image

For a given morphism $\alpha : A \rightarrow B$, an image of α consists of four parts:

- an object I ,
- a morphism $c : A \rightarrow I$,
- a monomorphism $\iota : I \hookrightarrow B$ such that $\iota \circ c \sim_{A,B} \alpha$,
- a dependent function u mapping each pair of morphisms $\tau = (\tau_1 : A \rightarrow T, \tau_2 : T \hookrightarrow B)$ where τ_2 is a monomorphism such that $\tau_2 \circ \tau_1 \sim_{A,B} \alpha$ to a morphism $u(\tau) : I \rightarrow T$ such that $\tau_2 \circ u(\tau) \sim_{I,B} \iota$ and $u(\tau) \circ c \sim_{A,T} \tau_1$.

The 4-tuple (I, c, ι, u) is called an *image* of α if the morphisms $u(\tau)$ are uniquely determined up to congruence of morphisms. We denote the object I of such a 4-tuple by $\text{im}(\alpha)$. We say that the morphism $u(\tau)$ is induced by the *universal property of the image*.

6.11.1 IsomorphismFromImageObjectToKernelOfCokernel (for IsCapCategoryMorphism)

▷ `IsomorphismFromImageObjectToKernelOfCokernel(alpha)` (attribute)
Returns: a morphism in $\text{Hom}(\text{im}(\alpha), \text{KernelObject}(\text{CokernelProjection}(\alpha)))$
 The argument is a morphism α . The output is the canonical morphism $\text{im}(\alpha) \rightarrow \text{KernelObject}(\text{CokernelProjection}(\alpha))$.

6.11.2 AddIsomorphismFromImageObjectToKernelOfCokernel (for IsCapCategory, IsFunction)

▷ `AddIsomorphismFromImageObjectToKernelOfCokernel(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 `IsomorphismFromImageObjectToKernelOfCokernel`. $F : \alpha \mapsto (\text{im}(\alpha) \rightarrow \text{KernelObject}(\text{CokernelProjection}(\alpha)))$

6.11.3 IsomorphismFromKernelOfCokernelToImageObject (for IsCapCategoryMorphism)

▷ `IsomorphismFromKernelOfCokernelToImageObject(alpha)` (attribute)
Returns: a morphism in $\text{Hom}(\text{KernelObject}(\text{CokernelProjection}(\alpha)), \text{im}(\alpha))$
 The argument is a morphism α . The output is the canonical morphism $\text{KernelObject}(\text{CokernelProjection}(\alpha)) \rightarrow \text{im}(\alpha)$.

6.11.4 AddIsomorphismFromKernelOfCokernelToImageObject (for IsCapCategory, IsFunction)

▷ AddIsomorphismFromKernelOfCokernelToImageObject(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 IsomorphismFromKernelOfCokernelToImageObject. $F : \alpha \mapsto (\text{KernelObject}(\text{CokernelProjection}(\alpha)) \rightarrow \text{im}(\alpha))$

6.11.5 ImageObject (for IsCapCategoryMorphism)

▷ ImageObject($alpha$) (attribute)

Returns: an object

The argument is a morphism α . The output is the image $\text{im}(\alpha)$.

6.11.6 ImageEmbedding (for IsCapCategoryMorphism)

▷ ImageEmbedding($alpha$) (attribute)

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

The argument is a morphism $\alpha : A \rightarrow B$. The output is the image embedding $\iota : \text{im}(\alpha) \hookrightarrow B$.

6.11.7 ImageEmbeddingWithGivenImageObject (for IsCapCategoryMorphism, IsCapCategoryObject)

▷ ImageEmbeddingWithGivenImageObject($alpha, I$) (operation)

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

The argument is a morphism $\alpha : A \rightarrow B$ and an object $I = \text{im}(\alpha)$. The output is the image embedding $\iota : I \hookrightarrow B$.

6.11.8 CostrictionToImage (for IsCapCategoryMorphism)

▷ CostrictionToImage($alpha$) (attribute)

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

The argument is a morphism $\alpha : A \rightarrow B$. The output is the costriction to image $c : A \rightarrow \text{im}(\alpha)$.

6.11.9 CostrictionToImageWithGivenImageObject (for IsCapCategoryMorphism, IsCapCategoryObject)

▷ CostrictionToImageWithGivenImageObject($alpha, I$) (operation)

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

The argument is a morphism $\alpha : A \rightarrow B$ and an object $I = \text{im}(\alpha)$. The output is the costriction to image $c : A \rightarrow I$.

6.11.10 UniversalMorphismFromImage (for IsCapCategoryMorphism, IsList)

▷ UniversalMorphismFromImage($alpha, tau$) (operation)

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

The arguments are a morphism $\alpha : A \rightarrow B$ and a pair of morphisms $\tau = (\tau_1 : A \rightarrow T, \tau_2 : T \hookrightarrow B)$ where τ_2 is a monomorphism such that $\tau_2 \circ \tau_1 \sim_{A,B} \alpha$. The output is the morphism $u(\tau) : \text{im}(\alpha) \rightarrow T$ given by the universal property of the image.

6.11.11 UniversalMorphismFromImageWithGivenImageObject (for IsCapCategory-Morphism, IsList, IsCapCategoryObject)

▷ `UniversalMorphismFromImageWithGivenImageObject(alpha, tau, I)` (operation)

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

The arguments are a morphism $\alpha : A \rightarrow B$, a pair of morphisms $\tau = (\tau_1 : A \rightarrow T, \tau_2 : T \hookrightarrow B)$ where τ_2 is a monomorphism such that $\tau_2 \circ \tau_1 \sim_{A,B} \alpha$, and an object $I = \text{im}(\alpha)$. The output is the morphism $u(\tau) : \text{im}(\alpha) \rightarrow T$ given by the universal property of the image.

6.11.12 AddImageObject (for IsCapCategory, IsFunction)

▷ `AddImageObject(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 `ImageObject`. $F : \alpha \mapsto I$.

6.11.13 AddImageEmbedding (for IsCapCategory, IsFunction)

▷ `AddImageEmbedding(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 `ImageEmbedding`. $F : \alpha \mapsto \iota$.

6.11.14 AddImageEmbeddingWithGivenImageObject (for IsCapCategory, IsFunction)

▷ `AddImageEmbeddingWithGivenImageObject(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 `ImageEmbeddingWithGivenImageObject`. $F : (\alpha, I) \mapsto \iota$.

6.11.15 AddCostrictionToImage (for IsCapCategory, IsFunction)

▷ `AddCostrictionToImage(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 `CostrictionToImage`. $F : \alpha \mapsto c$.

6.11.16 AddCostrictionToImageWithGivenImageObject (for IsCapCategory, IsFunction)

▷ `AddCostrictionToImageWithGivenImageObject(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 `CostrictionToImageWithGivenImageObject`. $F : (\alpha, I) \mapsto c$.

6.11.17 AddUniversalMorphismFromImage (for IsCapCategory, IsFunction)

▷ `AddUniversalMorphismFromImage(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 `UniversalMorphismFromImage`. $F : (\alpha, \tau) \mapsto u(\tau)$.

6.11.18 AddUniversalMorphismFromImageWithGivenImageObject (for IsCapCategory, IsFunction)

▷ `AddUniversalMorphismFromImageWithGivenImageObject(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 `UniversalMorphismFromImageWithGivenImageObject`. $F : (\alpha, \tau, I) \mapsto u(\tau)$.

6.12 Coimage

For a given morphism $\alpha : A \rightarrow B$, a coimage of α consists of four parts:

- an object C ,
- an epimorphism $\pi : A \rightarrow C$,
- a morphism $a : C \rightarrow B$ such that $a \circ \pi \sim_{A,B} \alpha$,
- a dependent function u mapping each pair of morphisms $\tau = (\tau_1 : A \rightarrow T, \tau_2 : T \rightarrow B)$ where τ_1 is an epimorphism such that $\tau_2 \circ \tau_1 \sim_{A,B} \alpha$ to a morphism $u(\tau) : T \rightarrow C$ such that $u(\tau) \circ \tau_1 \sim_{A,C} \pi$ and $a \circ u(\tau) \sim_{T,B} \tau_2$.

The 4-tuple (C, π, a, u) is called a *coimage* of α if the morphisms $u(\tau)$ are uniquely determined up to congruence of morphisms. We denote the object C of such a 4-tuple by $\text{coim}(\alpha)$. We say that the morphism $u(\tau)$ is induced by the *universal property of the coimage*.

6.12.1 MorphismFromCoimageToImage (for IsCapCategoryMorphism)

▷ `MorphismFromCoimageToImage(alpha)` (attribute)

Returns: a morphism in $\text{Hom}(\text{coim}(\alpha), \text{im}(\alpha))$

The argument is a morphism $\alpha : A \rightarrow B$. The output is the canonical morphism (in a preabelian category) $\text{coim}(\alpha) \rightarrow \text{im}(\alpha)$.

6.12.2 MorphismFromCoimageToImageWithGivenObjects (for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryObject)

▷ `MorphismFromCoimageToImageWithGivenObjects(alpha)` (operation)

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

The argument is an object $C = \text{coim}(\alpha)$, a morphism $\alpha : A \rightarrow B$, and an object $I = \text{im}(\alpha)$. The output is the canonical morphism (in a preabelian category) $C \rightarrow I$.

6.12.3 AddMorphismFromCoimageToImageWithGivenObjects (for IsCapCategory, IsFunction)

▷ AddMorphismFromCoimageToImageWithGivenObjects(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 MorphismFromCoimageToImageWithGivenObjects. $F : (C, \alpha, I) \mapsto (C \rightarrow I)$.

6.12.4 InverseMorphismFromCoimageToImage (for IsCapCategoryMorphism)

▷ InverseMorphismFromCoimageToImage($alpha$) (attribute)

Returns: a morphism in $\text{Hom}(\text{im}(\alpha), \text{coim}(\alpha))$

The argument is a morphism $\alpha : A \rightarrow B$. The output is the inverse of the canonical morphism (in an abelian category) $\text{im}(\alpha) \rightarrow \text{coim}(\alpha)$.

6.12.5 InverseMorphismFromCoimageToImageWithGivenObjects (for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryObject)

▷ InverseMorphismFromCoimageToImageWithGivenObjects($alpha$) (operation)

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

The argument is an object $C = \text{coim}(\alpha)$, a morphism $\alpha : A \rightarrow B$, and an object $I = \text{im}(\alpha)$. The output is the inverse of the canonical morphism (in an abelian category) $I \rightarrow C$.

6.12.6 AddInverseMorphismFromCoimageToImageWithGivenObjects (for IsCapCategory, IsFunction)

▷ AddInverseMorphismFromCoimageToImageWithGivenObjects(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 MorphismFromCoimageToImageWithGivenObjects. $F : (C, \alpha, I) \mapsto (I \rightarrow C)$.

6.12.7 IsomorphismFromCoimageToCokernelOfKernel (for IsCapCategoryMorphism)

▷ IsomorphismFromCoimageToCokernelOfKernel($alpha$) (attribute)

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

The argument is a morphism $\alpha : A \rightarrow B$. The output is the canonical morphism $\text{coim}(\alpha) \rightarrow \text{CokernelObject}(\text{KernelEmbedding}(\alpha))$.

6.12.8 AddIsomorphismFromCoimageToCokernelOfKernel (for IsCapCategory, Is-Function)

▷ AddIsomorphismFromCoimageToCokernelOfKernel(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 IsomorphismFromCoimageToCokernelOfKernel. $F : \alpha \mapsto (\text{coim}(\alpha) \rightarrow \text{CokernelObject}(\text{KernelEmbedding}(\alpha)))$.

6.12.9 IsomorphismFromCokernelOfKernelToCoimage (for IsCapCategoryMorphism)

▷ IsomorphismFromCokernelOfKernelToCoimage($alpha$) (attribute)

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

The argument is a morphism $\alpha : A \rightarrow B$. The output is the canonical morphism $\text{CokernelObject}(\text{KernelEmbedding}(\alpha)) \rightarrow \text{coim}(\alpha)$.

6.12.10 AddIsomorphismFromCokernelOfKernelToCoimage (for IsCapCategory, Is-Function)

▷ AddIsomorphismFromCokernelOfKernelToCoimage(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 IsomorphismFromCokernelOfKernelToCoimage. $F : \alpha \mapsto (\text{CokernelObject}(\text{KernelEmbedding}(\alpha)) \rightarrow \text{coim}(\alpha))$.

6.12.11 Coimage (for IsCapCategoryMorphism)

▷ Coimage($alpha$) (attribute)

Returns: an object

The argument is a morphism α . The output is the coimage $\text{coim}(\alpha)$.

6.12.12 CoimageProjection (for IsCapCategoryObject)

▷ CoimageProjection(C) (attribute)

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

This is a convenience method. The argument is an object C which was created as a coimage of a morphism $\alpha : A \rightarrow B$. The output is the coimage projection $\pi : A \twoheadrightarrow C$.

6.12.13 CoimageProjection (for IsCapCategoryMorphism)

▷ CoimageProjection($alpha$) (attribute)

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

The argument is a morphism $\alpha : A \rightarrow B$. The output is the coimage projection $\pi : A \twoheadrightarrow \text{coim}(\alpha)$.

6.12.14 CoimageProjectionWithGivenCoimage (for IsCapCategoryMorphism, Is-CapCategoryObject)

▷ `CoimageProjectionWithGivenCoimage(alpha, C)` (operation)

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

The arguments are a morphism $\alpha : A \rightarrow B$ and an object $C = \text{coim}(\alpha)$. The output is the coimage projection $\pi : A \twoheadrightarrow C$.

6.12.15 AstrictionToCoimage (for IsCapCategoryObject)

▷ `AstrictionToCoimage(C)` (attribute)

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

This is a convenience method. The argument is an object C which was created as a coimage of a morphism $\alpha : A \rightarrow B$. The output is the astriction to coimage $a : C \rightarrow B$.

6.12.16 AstrictionToCoimage (for IsCapCategoryMorphism)

▷ `AstrictionToCoimage(alpha)` (attribute)

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

The argument is a morphism $\alpha : A \rightarrow B$. The output is the astriction to coimage $a : \text{coim}(\alpha) \rightarrow B$.

6.12.17 AstrictionToCoimageWithGivenCoimage (for IsCapCategoryMorphism, Is-CapCategoryObject)

▷ `AstrictionToCoimageWithGivenCoimage(alpha, C)` (operation)

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

The argument are a morphism $\alpha : A \rightarrow B$ and an object $C = \text{coim}(\alpha)$. The output is the astriction to coimage $a : C \rightarrow B$.

6.12.18 UniversalMorphismIntoCoimage (for IsCapCategoryMorphism, IsList)

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

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

The arguments are a morphism $\alpha : A \rightarrow B$ and a pair of morphisms $\tau = (\tau_1 : A \twoheadrightarrow T, \tau_2 : T \rightarrow B)$ where τ_1 is an epimorphism such that $\tau_2 \circ \tau_1 \sim_{A,B} \alpha$. The output is the morphism $u(\tau) : T \rightarrow \text{coim}(\alpha)$ given by the universal property of the coimage.

6.12.19 UniversalMorphismIntoCoimageWithGivenCoimage (for IsCapCategory-Morphism, IsList, IsCapCategoryObject)

▷ `UniversalMorphismIntoCoimageWithGivenCoimage(alpha, tau, C)` (operation)

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

The arguments are a morphism $\alpha : A \rightarrow B$, a pair of morphisms $\tau = (\tau_1 : A \twoheadrightarrow T, \tau_2 : T \rightarrow B)$ where τ_1 is an epimorphism such that $\tau_2 \circ \tau_1 \sim_{A,B} \alpha$, and an object $C = \text{coim}(\alpha)$. The output is the morphism $u(\tau) : T \rightarrow C$ given by the universal property of the coimage.

6.12.20 AddCoimage (for IsCapCategory, IsFunction)

▷ AddCoimage(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 Coimage. $F : \alpha \mapsto C$

6.12.21 AddCoimageProjection (for IsCapCategory, IsFunction)

▷ AddCoimageProjection(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 CoimageProjection. $F : \alpha \mapsto \pi$

6.12.22 AddCoimageProjectionWithGivenCoimage (for IsCapCategory, IsFunction)

▷ AddCoimageProjectionWithGivenCoimage(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 CoimageProjectionWithGivenCoimage. $F : (\alpha, C) \mapsto \pi$

6.12.23 AddAstrictionToCoimage (for IsCapCategory, IsFunction)

▷ AddAstrictionToCoimage(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 AstrictionToCoimage. $F : \alpha \mapsto a$

6.12.24 AddAstrictionToCoimageWithGivenCoimage (for IsCapCategory, IsFunction)

▷ AddAstrictionToCoimageWithGivenCoimage(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 AstrictionToCoimageWithGivenCoimage. $F : (\alpha, C) \mapsto a$

6.12.25 AddUniversalMorphismIntoCoimage (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismIntoCoimage(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 UniversalMorphismIntoCoimage. $F : (\alpha, \tau) \mapsto u(\tau)$

6.12.26 AddUniversalMorphismIntoCoimageWithGivenCoimage (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismIntoCoimageWithGivenCoimage(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 `UniversalMorphismIntoCoimageWithGivenCoimage`. $F : (\alpha, \tau, C) \mapsto u(\tau)$

Chapter 7

Tensor Product and Internal Hom

7.1 Monoidal Categories

A 6-tuple $(\mathbf{C}, \otimes, 1, \alpha, \lambda, \rho)$ consisting of

- a category \mathbf{C} ,
- a functor $\otimes : \mathbf{C} \times \mathbf{C} \rightarrow \mathbf{C}$,
- an object $1 \in \mathbf{C}$,
- a natural isomorphism $\alpha_{a,b,c} : a \otimes (b \otimes c) \cong (a \otimes b) \otimes c$,
- a natural isomorphism $\lambda_a : 1 \otimes a \cong a$,
- a natural isomorphism $\rho_a : a \otimes 1 \cong a$,

is called a *monoidal category*, if

- for all objects a, b, c, d , the pentagon identity holds: $(\alpha_{a,b,c} \otimes \text{id}_d) \circ \alpha_{a,b \otimes c, d} \circ (\text{id}_a \otimes \alpha_{b,c,d}) = \alpha_{a \otimes b, c, d} \circ \alpha_{a, b, c \otimes d}$,
- for all objects a, c , the triangle identity holds: $(\rho_a \otimes \text{id}_c) \circ \alpha_{a, 1, c} = \text{id}_a \otimes \lambda_c$.

The corresponding GAP property is given by `IsMonoidalCategory`.

7.1.1 TensorProductOnObjects (for IsCapCategoryObject, IsCapCategoryObject)

▷ `TensorProductOnObjects(a, b)` (operation)

Returns: an object

The arguments are two objects a, b . The output is the tensor product $a \otimes b$.

7.1.2 AddTensorProductOnObjects (for IsCapCategory, IsFunction)

▷ `AddTensorProductOnObjects(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 `TensorProductOnObjects`. $F : (a, b) \mapsto a \otimes b$.

7.1.3 TensorProductOnMorphisms (for IsCapCategoryMorphism, IsCapCategoryMorphism)

- ▷ TensorProductOnMorphisms(*alpha*, *beta*) (operation)
Returns: a morphism in $\text{Hom}(a \otimes b, a' \otimes b')$
 The arguments are two morphisms $\alpha : a \rightarrow a', \beta : b \rightarrow b'$. The output is the tensor product $\alpha \otimes \beta$.

7.1.4 TensorProductOnMorphismsWithGivenTensorProducts (for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject)

- ▷ TensorProductOnMorphismsWithGivenTensorProducts(*s*, *alpha*, *beta*, *r*) (operation)
Returns: a morphism in $\text{Hom}(a \otimes b, a' \otimes b')$
 The arguments are an object $s = a \otimes b$, two morphisms $\alpha : a \rightarrow a', \beta : b \rightarrow b'$, and an object $r = a' \otimes b'$. The output is the tensor product $\alpha \otimes \beta$.

7.1.5 AddTensorProductOnMorphismsWithGivenTensorProducts (for IsCapCategory, IsFunction)

- ▷ AddTensorProductOnMorphismsWithGivenTensorProducts(*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 TensorProductOnMorphismsWithGivenTensorProducts. $F : (a \otimes b, \alpha : a \rightarrow a', \beta : b \rightarrow b', a' \otimes b') \mapsto \alpha \otimes \beta$.

7.1.6 AssociatorRightToLeft (for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject)

- ▷ AssociatorRightToLeft(*a*, *b*, *c*) (operation)
Returns: a morphism in $\text{Hom}(a \otimes (b \otimes c), (a \otimes b) \otimes c)$.
 The arguments are three objects *a, b, c*. The output is the associator $\alpha_{a,(b,c)} : a \otimes (b \otimes c) \rightarrow (a \otimes b) \otimes c$.

7.1.7 AssociatorRightToLeftWithGivenTensorProducts (for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject)

- ▷ AssociatorRightToLeftWithGivenTensorProducts(*s*, *a*, *b*, *c*, *r*) (operation)
Returns: a morphism in $\text{Hom}(a \otimes (b \otimes c), (a \otimes b) \otimes c)$.
 The arguments are an object $s = a \otimes (b \otimes c)$, three objects *a, b, c*, and an object $r = (a \otimes b) \otimes c$. The output is the associator $\alpha_{a,(b,c)} : a \otimes (b \otimes c) \rightarrow (a \otimes b) \otimes c$.

7.1.8 AddAssociatorRightToLeftWithGivenTensorProducts (for IsCapCategory, IsFunction)

- ▷ AddAssociatorRightToLeftWithGivenTensorProducts(*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 `AssociatorRightToLeftWithGivenTensorProducts`. $F : (a \otimes (b \otimes c), a, b, c, (a \otimes b) \otimes c) \mapsto \alpha_{a,(b,c)}$.

7.1.9 `AssociatorLeftToRight` (for `IsCapCategoryObject`, `IsCapCategoryObject`, `IsCapCategoryObject`)

▷ `AssociatorLeftToRight(a, b, c)` (operation)

Returns: a morphism in $\text{Hom}((a \otimes b) \otimes c \rightarrow a \otimes (b \otimes c))$.

The arguments are three objects a, b, c . The output is the associator $\alpha_{(a,b),c} : (a \otimes b) \otimes c \rightarrow a \otimes (b \otimes c)$.

7.1.10 `AssociatorLeftToRightWithGivenTensorProducts` (for `IsCapCategoryObject`, `IsCapCategoryObject`, `IsCapCategoryObject`, `IsCapCategoryObject`, `IsCapCategoryObject`)

▷ `AssociatorLeftToRightWithGivenTensorProducts(s, a, b, c, r)` (operation)

Returns: a morphism in $\text{Hom}((a \otimes b) \otimes c \rightarrow a \otimes (b \otimes c))$.

The arguments are an object $s = (a \otimes b) \otimes c$, three objects a, b, c , and an object $r = a \otimes (b \otimes c)$. The output is the associator $\alpha_{(a,b),c} : (a \otimes b) \otimes c \rightarrow a \otimes (b \otimes c)$.

7.1.11 `AddAssociatorLeftToRightWithGivenTensorProducts` (for `IsCapCategory`, `IsFunction`)

▷ `AddAssociatorLeftToRightWithGivenTensorProducts(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 `AssociatorLeftToRightWithGivenTensorProducts`. $F : ((a \otimes b) \otimes c, a, b, c, a \otimes (b \otimes c)) \mapsto \alpha_{(a,b),c}$.

7.1.12 `TensorUnit` (for `IsCapCategory`)

▷ `TensorUnit(C)` (attribute)

Returns: an object

The argument is a category C . The output is the tensor unit 1 of C .

7.1.13 `AddTensorUnit` (for `IsCapCategory`, `IsFunction`)

▷ `AddTensorUnit(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 `TensorUnit`. $F : () \mapsto 1$.

7.1.14 `LeftUnitor` (for `IsCapCategoryObject`)

▷ `LeftUnitor(a)` (attribute)

Returns: a morphism in $\text{Hom}(1 \otimes a, a)$

The argument is an object a . The output is the left unitor $\lambda_a : 1 \otimes a \rightarrow a$.

7.1.15 LeftUnitorWithGivenTensorProduct (for IsCapCategoryObject, IsCapCategoryObject)

- ▷ LeftUnitorWithGivenTensorProduct(a, s) (operation)
Returns: a morphism in $\text{Hom}(1 \otimes a, a)$
 The arguments are an object a and an object $s = 1 \otimes a$. The output is the left unitor $\lambda_a : 1 \otimes a \rightarrow a$.

7.1.16 AddLeftUnitorWithGivenTensorProduct (for IsCapCategory, IsFunction)

- ▷ AddLeftUnitorWithGivenTensorProduct(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 LeftUnitorWithGivenTensorProduct. $F : (a, 1 \otimes a) \mapsto \lambda_a$.

7.1.17 LeftUnitorInverse (for IsCapCategoryObject)

- ▷ LeftUnitorInverse(a) (attribute)
Returns: a morphism in $\text{Hom}(a, 1 \otimes a)$
 The argument is an object a . The output is the inverse of the left unitor $\lambda_a^{-1} : a \rightarrow 1 \otimes a$.

7.1.18 LeftUnitorInverseWithGivenTensorProduct (for IsCapCategoryObject, IsCapCategoryObject)

- ▷ LeftUnitorInverseWithGivenTensorProduct(a, r) (operation)
Returns: a morphism in $\text{Hom}(a, 1 \otimes a)$
 The argument is an object a and an object $r = 1 \otimes a$. The output is the inverse of the left unitor $\lambda_a^{-1} : a \rightarrow 1 \otimes a$.

7.1.19 AddLeftUnitorInverseWithGivenTensorProduct (for IsCapCategory, IsFunction)

- ▷ AddLeftUnitorInverseWithGivenTensorProduct(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 LeftUnitorInverseWithGivenTensorProduct. $F : (a, 1 \otimes a) \mapsto \lambda_a^{-1}$.

7.1.20 RightUnitor (for IsCapCategoryObject)

- ▷ RightUnitor(a) (attribute)
Returns: a morphism in $\text{Hom}(a \otimes 1, a)$
 The argument is an object a . The output is the right unitor $\rho_a : a \otimes 1 \rightarrow a$.

7.1.21 RightUnitorWithGivenTensorProduct (for IsCapCategoryObject, IsCapCategoryObject)

- ▷ RightUnitorWithGivenTensorProduct(a, s) (operation)
Returns: a morphism in $\text{Hom}(a \otimes 1, a)$
 The arguments are an object a and an object $s = a \otimes 1$. The output is the right unitor $\rho_a : a \otimes 1 \rightarrow a$.

7.1.22 AddRightUnitorWithGivenTensorProduct (for IsCapCategory, IsFunction)

▷ AddRightUnitorWithGivenTensorProduct(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 RightUnitorWithGivenTensorProduct. $F : (a, a \otimes 1) \mapsto \rho_a$.

7.1.23 RightUnitorInverse (for IsCapCategoryObject)

▷ RightUnitorInverse(a) (attribute)

Returns: a morphism in $\text{Hom}(a, a \otimes 1)$

The argument is an object a . The output is the inverse of the right unitor $\rho_a^{-1} : a \rightarrow a \otimes 1$.

7.1.24 RightUnitorInverseWithGivenTensorProduct (for IsCapCategoryObject, IsCapCategoryObject)

▷ RightUnitorInverseWithGivenTensorProduct(a, r) (operation)

Returns: a morphism in $\text{Hom}(a, a \otimes 1)$

The arguments are an object a and an object $r = a \otimes 1$. The output is the inverse of the right unitor $\rho_a^{-1} : a \rightarrow a \otimes 1$.

7.1.25 AddRightUnitorInverseWithGivenTensorProduct (for IsCapCategory, IsFunction)

▷ AddRightUnitorInverseWithGivenTensorProduct(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 RightUnitorInverseWithGivenTensorProduct. $F : (a, a \otimes 1) \mapsto \rho_a^{-1}$.

7.1.26 LeftDistributivityExpanding (for IsCapCategoryObject, IsList)

▷ LeftDistributivityExpanding(a, L) (operation)

Returns: a morphism in $\text{Hom}(a \otimes (b_1 \oplus \dots \oplus b_n), (a \otimes b_1) \oplus \dots \oplus (a \otimes b_n))$

The arguments are an object a and a list of objects $L = (b_1, \dots, b_n)$. The output is the left distributivity morphism $a \otimes (b_1 \oplus \dots \oplus b_n) \rightarrow (a \otimes b_1) \oplus \dots \oplus (a \otimes b_n)$.

7.1.27 LeftDistributivityExpandingWithGivenObjects (for IsCapCategoryObject, IsCapCategoryObject, IsList, IsCapCategoryObject)

▷ LeftDistributivityExpandingWithGivenObjects(s, a, L, r) (operation)

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

The arguments are an object $s = a \otimes (b_1 \oplus \dots \oplus b_n)$, an object a , a list of objects $L = (b_1, \dots, b_n)$, and an object $r = (a \otimes b_1) \oplus \dots \oplus (a \otimes b_n)$. The output is the left distributivity morphism $s \rightarrow r$.

7.1.28 AddLeftDistributivityExpandingWithGivenObjects (for IsCapCategory, IsFunction)

▷ AddLeftDistributivityExpandingWithGivenObjects(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 LeftDistributivityExpandingWithGivenObjects. $F : (a \otimes (b_1 \oplus \dots \oplus b_n), a, L, (a \otimes b_1) \oplus \dots \oplus (a \otimes b_n)) \mapsto \text{LeftDistributivityExpandingWithGivenObjects}(a, L)$.

7.1.29 LeftDistributivityFactoring (for IsCapCategoryObject, IsList)

▷ LeftDistributivityFactoring(a, L) (operation)

Returns: a morphism in $\text{Hom}((a \otimes b_1) \oplus \dots \oplus (a \otimes b_n), a \otimes (b_1 \oplus \dots \oplus b_n))$

The arguments are an object a and a list of objects $L = (b_1, \dots, b_n)$. The output is the left distributivity morphism $(a \otimes b_1) \oplus \dots \oplus (a \otimes b_n) \rightarrow a \otimes (b_1 \oplus \dots \oplus b_n)$.

7.1.30 LeftDistributivityFactoringWithGivenObjects (for IsCapCategoryObject, IsCapCategoryObject, IsList, IsCapCategoryObject)

▷ LeftDistributivityFactoringWithGivenObjects(s, a, L, r) (operation)

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

The arguments are an object $s = (a \otimes b_1) \oplus \dots \oplus (a \otimes b_n)$, an object a , a list of objects $L = (b_1, \dots, b_n)$, and an object $r = a \otimes (b_1 \oplus \dots \oplus b_n)$. The output is the left distributivity morphism $s \rightarrow r$.

7.1.31 AddLeftDistributivityFactoringWithGivenObjects (for IsCapCategory, IsFunction)

▷ AddLeftDistributivityFactoringWithGivenObjects(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 LeftDistributivityFactoringWithGivenObjects. $F : ((a \otimes b_1) \oplus \dots \oplus (a \otimes b_n), a, L, a \otimes (b_1 \oplus \dots \oplus b_n)) \mapsto \text{LeftDistributivityFactoringWithGivenObjects}(a, L)$.

7.1.32 RightDistributivityExpanding (for IsList, IsCapCategoryObject)

▷ RightDistributivityExpanding(L, a) (operation)

Returns: a morphism in $\text{Hom}((b_1 \oplus \dots \oplus b_n) \otimes a, (b_1 \otimes a) \oplus \dots \oplus (b_n \otimes a))$

The arguments are a list of objects $L = (b_1, \dots, b_n)$ and an object a . The output is the right distributivity morphism $(b_1 \oplus \dots \oplus b_n) \otimes a \rightarrow (b_1 \otimes a) \oplus \dots \oplus (b_n \otimes a)$.

7.1.33 RightDistributivityExpandingWithGivenObjects (for IsCapCategoryObject, IsList, IsCapCategoryObject, IsCapCategoryObject)

▷ RightDistributivityExpandingWithGivenObjects(s, L, a, r) (operation)

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

The arguments are an object $s = (b_1 \oplus \dots \oplus b_n) \otimes a$, a list of objects $L = (b_1, \dots, b_n)$, an object a , and an object $r = (b_1 \otimes a) \oplus \dots \oplus (b_n \otimes a)$. The output is the right distributivity morphism $s \rightarrow r$.

7.1.34 AddRightDistributivityExpandingWithGivenObjects (for IsCapCategory, IsFunction)

▷ AddRightDistributivityExpandingWithGivenObjects(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 RightDistributivityExpandingWithGivenObjects. $F : ((b_1 \oplus \dots \oplus b_n) \otimes a, L, a, (b_1 \otimes a) \oplus \dots \oplus (b_n \otimes a)) \mapsto \text{RightDistributivityExpandingWithGivenObjects}(L, a)$.

7.1.35 RightDistributivityFactoring (for IsList, IsCapCategoryObject)

▷ RightDistributivityFactoring(L , a) (operation)

Returns: a morphism in $\text{Hom}((b_1 \otimes a) \oplus \dots \oplus (b_n \otimes a), (b_1 \oplus \dots \oplus b_n) \otimes a)$

The arguments are a list of objects $L = (b_1, \dots, b_n)$ and an object a . The output is the right distributivity morphism $(b_1 \otimes a) \oplus \dots \oplus (b_n \otimes a) \rightarrow (b_1 \oplus \dots \oplus b_n) \otimes a$.

7.1.36 RightDistributivityFactoringWithGivenObjects (for IsCapCategoryObject, IsList, IsCapCategoryObject, IsCapCategoryObject)

▷ RightDistributivityFactoringWithGivenObjects(s , L , a , r) (operation)

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

The arguments are an object $s = (b_1 \otimes a) \oplus \dots \oplus (b_n \otimes a)$, a list of objects $L = (b_1, \dots, b_n)$, an object a , and an object $r = (b_1 \oplus \dots \oplus b_n) \otimes a$. The output is the right distributivity morphism $s \rightarrow r$.

7.1.37 AddRightDistributivityFactoringWithGivenObjects (for IsCapCategory, IsFunction)

▷ AddRightDistributivityFactoringWithGivenObjects(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 RightDistributivityFactoringWithGivenObjects. $F : ((b_1 \otimes a) \oplus \dots \oplus (b_n \otimes a), L, a, (b_1 \oplus \dots \oplus b_n) \otimes a) \mapsto \text{RightDistributivityFactoringWithGivenObjects}(L, a)$.

7.2 Braided Monoidal Categories

A monoidal category \mathbf{C} equipped with a natural isomorphism $B_{a,b} : a \otimes b \cong b \otimes a$ is called a *braided monoidal category* if

- $\lambda_a \circ B_{a,1} = \rho_a$,
- $(B_{c,a} \otimes \text{id}_b) \circ \alpha_{c,a,b} \circ B_{a \otimes b, c} = \alpha_{a,c,b} \circ (\text{id}_a \otimes B_{b,c}) \circ \alpha_{a,b,c}^{-1}$,
- $(\text{id}_b \otimes B_{c,a}) \circ \alpha_{b,c,a}^{-1} \circ B_{a,b \otimes c} = \alpha_{b,a,c}^{-1} \circ (B_{a,b} \otimes \text{id}_c) \circ \alpha_{a,b,c}$.

The corresponding GAP property is given by IsBraidedMonoidalCategory.

7.2.1 Braiding (for IsCapCategoryObject, IsCapCategoryObject)

▷ Braiding(a, b) (operation)

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

The arguments are two objects a, b . The output is the braiding $B_{a,b} : a \otimes b \rightarrow b \otimes a$.

7.2.2 BraidingWithGivenTensorProducts (for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject)

▷ BraidingWithGivenTensorProducts(s, a, b, r) (operation)

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

The arguments are an object $s = a \otimes b$, two objects a, b , and an object $r = b \otimes a$. The output is the braiding $B_{a,b} : a \otimes b \rightarrow b \otimes a$.

7.2.3 AddBraidingWithGivenTensorProducts (for IsCapCategory, IsFunction)

▷ AddBraidingWithGivenTensorProducts(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 BraidingWithGivenTensorProducts. $F : (a \otimes b, a, b, b \otimes a) \rightarrow B_{a,b}$.

7.2.4 BraidingInverse (for IsCapCategoryObject, IsCapCategoryObject)

▷ BraidingInverse(a, b) (operation)

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

The arguments are two objects a, b . The output is the inverse of the braiding $B_{a,b}^{-1} : b \otimes a \rightarrow a \otimes b$.

7.2.5 BraidingInverseWithGivenTensorProducts (for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject)

▷ BraidingInverseWithGivenTensorProducts(s, a, b, r) (operation)

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

The arguments are an object $s = b \otimes a$, two objects a, b , and an object $r = a \otimes b$. The output is the braiding $B_{a,b}^{-1} : b \otimes a \rightarrow a \otimes b$.

7.2.6 AddBraidingInverseWithGivenTensorProducts (for IsCapCategory, IsFunction)

▷ AddBraidingInverseWithGivenTensorProducts(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 BraidingInverseWithGivenTensorProducts. $F : (b \otimes a, a, b, a \otimes b) \rightarrow B_{a,b}^{-1}$.

7.3 Symmetric Monoidal Categories

A braided monoidal category \mathbf{C} is called *symmetric monoidal category* if $B_{a,b}^{-1} = B_{b,a}$. The corresponding GAP property is given by `IsSymmetricMonoidalCategory`.

7.4 Symmetric Closed Monoidal Categories

A symmetric monoidal category \mathbf{C} which has for each functor $- \otimes b : \mathbf{C} \rightarrow \mathbf{C}$ a right adjoint (denoted by $\underline{\text{Hom}}(b, -)$) is called a *symmetric closed monoidal category*. The corresponding GAP property is given by `IsSymmetricClosedMonoidalCategory`.

7.4.1 InternalHomOnObjects (for IsCapCategoryObject, IsCapCategoryObject)

▷ `InternalHomOnObjects(a, b)` (operation)

Returns: an object

The arguments are two objects a, b . The output is the internal hom object $\underline{\text{Hom}}(a, b)$.

7.4.2 AddInternalHomOnObjects (for IsCapCategory, IsFunction)

▷ `AddInternalHomOnObjects(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 `InternalHomOnObjects`. $F : (a, b) \mapsto \underline{\text{Hom}}(a, b)$.

7.4.3 InternalHomOnMorphisms (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

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

The arguments are two morphisms $\alpha : a \rightarrow a', \beta : b \rightarrow b'$. The output is the internal hom morphism $\underline{\text{Hom}}(\alpha, \beta) : \underline{\text{Hom}}(a', b) \rightarrow \underline{\text{Hom}}(a, b')$.

7.4.4 InternalHomOnMorphismsWithGivenInternalHoms (for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject)

▷ `InternalHomOnMorphismsWithGivenInternalHoms(s, alpha, beta, r)` (operation)

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

The arguments are an object $s = \underline{\text{Hom}}(a', b)$, two morphisms $\alpha : a \rightarrow a', \beta : b \rightarrow b'$, and an object $r = \underline{\text{Hom}}(a, b')$. The output is the internal hom morphism $\underline{\text{Hom}}(\alpha, \beta) : \underline{\text{Hom}}(a', b) \rightarrow \underline{\text{Hom}}(a, b')$.

7.4.5 AddInternalHomOnMorphismsWithGivenInternalHoms (for IsCapCategory, IsFunction)

▷ `AddInternalHomOnMorphismsWithGivenInternalHoms(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 `InternalHomOnMorphismsWithGivenInternalHoms`. $F : (\underline{\text{Hom}}(a', b), \alpha : a \rightarrow a', \beta : b \rightarrow b', \underline{\text{Hom}}(a, b')) \mapsto \underline{\text{Hom}}(\alpha, \beta)$.

7.4.6 EvaluationMorphism (for IsCapCategoryObject, IsCapCategoryObject)

▷ `EvaluationMorphism(a, b)` (operation)

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

The arguments are two objects a, b . The output is the evaluation morphism $\text{ev}_{a,b} : \underline{\text{Hom}}(a, b) \otimes a \rightarrow b$, i.e., the counit of the tensor hom adjunction.

7.4.7 EvaluationMorphismWithGivenSource (for IsCapCategoryObject, IsCapCategoryObject)

▷ `EvaluationMorphismWithGivenSource(a, b, s)` (operation)

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

The arguments are two objects a, b and an object $s = \underline{\text{Hom}}(a, b) \otimes a$. The output is the evaluation morphism $\text{ev}_{a,b} : \underline{\text{Hom}}(a, b) \otimes a \rightarrow b$, i.e., the counit of the tensor hom adjunction.

7.4.8 AddEvaluationMorphismWithGivenSource (for IsCapCategory, IsFunction)

▷ `AddEvaluationMorphismWithGivenSource(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 `EvaluationMorphismWithGivenSource`. $F : (a, b, \underline{\text{Hom}}(a, b) \otimes a) \mapsto \text{ev}_{a,b}$.

7.4.9 CoevaluationMorphism (for IsCapCategoryObject, IsCapCategoryObject)

▷ `CoevaluationMorphism(a, b)` (operation)

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

The arguments are two objects a, b . The output is the coevaluation morphism $\text{coev}_{a,b} : a \rightarrow \underline{\text{Hom}}(b, a \otimes b)$, i.e., the unit of the tensor hom adjunction.

7.4.10 CoevaluationMorphismWithGivenRange (for IsCapCategoryObject, IsCapCategoryObject)

▷ `CoevaluationMorphismWithGivenRange(a, b, r)` (operation)

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

The arguments are two objects a, b and an object $r = \underline{\text{Hom}}(b, a \otimes b)$. The output is the coevaluation morphism $\text{coev}_{a,b} : a \rightarrow \underline{\text{Hom}}(b, a \otimes b)$, i.e., the unit of the tensor hom adjunction.

7.4.11 AddCoevaluationMorphismWithGivenRange (for IsCapCategory, IsFunction)

▷ `AddCoevaluationMorphismWithGivenRange(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 `CoevaluationMorphismWithGivenRange`. $F : (a, b, \underline{\text{Hom}}(b, a \otimes b)) \mapsto \text{coev}_{a,b}$.

7.4.12 TensorProductToInternalHomAdjunctionMap (for IsCapCategoryObject, Is-CapCategoryObject, IsCapCategoryMorphism)

▷ `TensorProductToInternalHomAdjunctionMap(a, b, f)` (operation)

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

The arguments are objects a, b and a morphism $f : a \otimes b \rightarrow c$. The output is a morphism $g : a \rightarrow \underline{\text{Hom}}(b, c)$ corresponding to f under the tensor hom adjunction.

7.4.13 AddTensorProductToInternalHomAdjunctionMap (for IsCapCategory, Is-Function)

▷ `AddTensorProductToInternalHomAdjunctionMap(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 `TensorProductToInternalHomAdjunctionMap`. $F : (a, b, f : a \otimes b \rightarrow c) \mapsto (g : a \rightarrow \underline{\text{Hom}}(b, c))$.

7.4.14 InternalHomToTensorProductAdjunctionMap (for IsCapCategoryObject, Is-CapCategoryObject, IsCapCategoryMorphism)

▷ `InternalHomToTensorProductAdjunctionMap(b, c, g)` (operation)

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

The arguments are objects b, c and a morphism $g : a \rightarrow \underline{\text{Hom}}(b, c)$. The output is a morphism $f : a \otimes b \rightarrow c$ corresponding to g under the tensor hom adjunction.

7.4.15 AddInternalHomToTensorProductAdjunctionMap (for IsCapCategory, Is-Function)

▷ `AddInternalHomToTensorProductAdjunctionMap(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 `InternalHomToTensorProductAdjunctionMap`. $F : (b, c, g : a \rightarrow \underline{\text{Hom}}(b, c)) \mapsto (g : a \otimes b \rightarrow c)$.

7.4.16 MonoidalPreComposeMorphism (for IsCapCategoryObject, IsCapCategory-Object, IsCapCategoryObject)

▷ `MonoidalPreComposeMorphism(a, b, c)` (operation)

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

The arguments are three objects a, b, c . The output is the precomposition morphism `MonoidalPreComposeMorphismWithGivenObjects` $_{a,b,c} : \underline{\text{Hom}}(a, b) \otimes \underline{\text{Hom}}(b, c) \rightarrow \underline{\text{Hom}}(a, c)$.

7.4.17 MonoidalPreComposeMorphismWithGivenObjects (for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject)

▷ MonoidalPreComposeMorphismWithGivenObjects(s, a, b, c, r) (operation)

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

The arguments are an object $s = \underline{\text{Hom}}(a, b) \otimes \underline{\text{Hom}}(b, c)$, three objects a, b, c , and an object $r = \underline{\text{Hom}}(a, c)$. The output is the precomposition morphism $\text{MonoidalPreComposeMorphismWithGivenObjects}_{a,b,c} : \underline{\text{Hom}}(a, b) \otimes \underline{\text{Hom}}(b, c) \rightarrow \underline{\text{Hom}}(a, c)$.

7.4.18 AddMonoidalPreComposeMorphismWithGivenObjects (for IsCapCategory, IsFunction)

▷ AddMonoidalPreComposeMorphismWithGivenObjects(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 $\text{MonoidalPreComposeMorphismWithGivenObjects}$. $F : (\underline{\text{Hom}}(a, b) \otimes \underline{\text{Hom}}(b, c), a, b, c, \underline{\text{Hom}}(a, c)) \mapsto \text{MonoidalPreComposeMorphismWithGivenObjects}_{a,b,c}$.

7.4.19 MonoidalPostComposeMorphism (for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject)

▷ MonoidalPostComposeMorphism(a, b, c) (operation)

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

The arguments are three objects a, b, c . The output is the postcomposition morphism $\text{MonoidalPostComposeMorphismWithGivenObjects}_{a,b,c} : \underline{\text{Hom}}(b, c) \otimes \underline{\text{Hom}}(a, b) \rightarrow \underline{\text{Hom}}(a, c)$.

7.4.20 MonoidalPostComposeMorphismWithGivenObjects (for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject)

▷ MonoidalPostComposeMorphismWithGivenObjects(s, a, b, c, r) (operation)

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

The arguments are an object $s = \underline{\text{Hom}}(b, c) \otimes \underline{\text{Hom}}(a, b)$, three objects a, b, c , and an object $r = \underline{\text{Hom}}(a, c)$. The output is the postcomposition morphism $\text{MonoidalPostComposeMorphismWithGivenObjects}_{a,b,c} : \underline{\text{Hom}}(b, c) \otimes \underline{\text{Hom}}(a, b) \rightarrow \underline{\text{Hom}}(a, c)$.

7.4.21 AddMonoidalPostComposeMorphismWithGivenObjects (for IsCapCategory, IsFunction)

▷ AddMonoidalPostComposeMorphismWithGivenObjects(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 $\text{MonoidalPostComposeMorphismWithGivenObjects}$. $F : (\underline{\text{Hom}}(b, c) \otimes \underline{\text{Hom}}(a, b), a, b, c, \underline{\text{Hom}}(a, c)) \mapsto \text{MonoidalPostComposeMorphismWithGivenObjects}_{a,b,c}$.

7.4.22 DualOnObjects (for IsCapCategoryObject)

- ▷ `DualOnObjects(a)` (attribute)
Returns: an object
 The argument is an object a . The output is its dual object a^\vee .

7.4.23 AddDualOnObjects (for IsCapCategory, IsFunction)

- ▷ `AddDualOnObjects(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 `DualOnObjects`. $F : a \mapsto a^\vee$.

7.4.24 DualOnMorphisms (for IsCapCategoryMorphism)

- ▷ `DualOnMorphisms(alpha)` (attribute)
Returns: a morphism in $\text{Hom}(b^\vee, a^\vee)$.
 The argument is a morphism $\alpha : a \rightarrow b$. The output is its dual morphism $\alpha^\vee : b^\vee \rightarrow a^\vee$.

7.4.25 DualOnMorphismsWithGivenDuals (for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryObject)

- ▷ `DualOnMorphismsWithGivenDuals(s, alpha, r)` (operation)
Returns: a morphism in $\text{Hom}(b^\vee, a^\vee)$.
 The argument is an object $s = b^\vee$, a morphism $\alpha : a \rightarrow b$, and an object $r = a^\vee$. The output is the dual morphism $\alpha^\vee : b^\vee \rightarrow a^\vee$.

7.4.26 AddDualOnMorphismsWithGivenDuals (for IsCapCategory, IsFunction)

- ▷ `AddDualOnMorphismsWithGivenDuals(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 `DualOnMorphismsWithGivenDuals`. $F : (b^\vee, \alpha, a^\vee) \mapsto \alpha^\vee$.

7.4.27 EvaluationForDual (for IsCapCategoryObject)

- ▷ `EvaluationForDual(a)` (attribute)
Returns: a morphism in $\text{Hom}(a^\vee \otimes a, 1)$.
 The argument is an object a . The output is the evaluation morphism $\text{ev}_a : a^\vee \otimes a \rightarrow 1$.

7.4.28 EvaluationForDualWithGivenTensorProduct (for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject)

- ▷ `EvaluationForDualWithGivenTensorProduct(s, a, r)` (operation)
Returns: a morphism in $\text{Hom}(a^\vee \otimes a, 1)$.
 The arguments are an object $s = a^\vee \otimes a$, an object a , and an object $r = 1$. The output is the evaluation morphism $\text{ev}_a : a^\vee \otimes a \rightarrow 1$.

7.4.29 AddEvaluationForDualWithGivenTensorProduct (for IsCapCategory, IsFunction)

▷ AddEvaluationForDualWithGivenTensorProduct(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 EvaluationForDualWithGivenTensorProduct. $F : (a^\vee \otimes a, a, 1) \mapsto \text{ev}_a$.

7.4.30 CoevaluationForDual (for IsCapCategoryObject)

▷ CoevaluationForDual(a) (attribute)

Returns: a morphism in $\text{Hom}(1, a \otimes a^\vee)$.

The argument is an object a . The output is the coevaluation morphism $\text{coev}_a : 1 \rightarrow a \otimes a^\vee$.

7.4.31 CoevaluationForDualWithGivenTensorProduct (for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject)

▷ CoevaluationForDualWithGivenTensorProduct(s, a, r) (operation)

Returns: a morphism in $\text{Hom}(1, a \otimes a^\vee)$.

The arguments are an object $s = 1$, an object a , and an object $r = a \otimes a^\vee$. The output is the coevaluation morphism $\text{coev}_a : 1 \rightarrow a \otimes a^\vee$.

7.4.32 AddCoevaluationForDualWithGivenTensorProduct (for IsCapCategory, IsFunction)

▷ AddCoevaluationForDualWithGivenTensorProduct(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 CoevaluationForDualWithGivenTensorProduct. $F : (1, a, a \otimes a^\vee) \mapsto \text{coev}_a$.

7.4.33 MorphismToBidual (for IsCapCategoryObject)

▷ MorphismToBidual(a) (attribute)

Returns: a morphism in $\text{Hom}(a, (a^\vee)^\vee)$.

The argument is an object a . The output is the morphism to the bidual $a \rightarrow (a^\vee)^\vee$.

7.4.34 MorphismToBidualWithGivenBidual (for IsCapCategoryObject, IsCapCategoryObject)

▷ MorphismToBidualWithGivenBidual(a, r) (operation)

Returns: a morphism in $\text{Hom}(a, (a^\vee)^\vee)$.

The arguments are an object a , and an object $r = (a^\vee)^\vee$. The output is the morphism to the bidual $a \rightarrow (a^\vee)^\vee$.

7.4.35 AddMorphismToBidualWithGivenBidual (for IsCapCategory, IsFunction)

▷ AddMorphismToBidualWithGivenBidual(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 MorphismToBidualWithGivenBidual. $F : (a, (a^\vee)^\vee) \mapsto (a \rightarrow (a^\vee)^\vee)$.

7.4.36 TensorProductInternalHomCompatibilityMorphism (for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject)

▷ TensorProductInternalHomCompatibilityMorphism(a, a', b, b') (operation)

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

The arguments are four objects a, a', b, b' . The output is the natural morphism $\text{TensorProductInternalHomCompatibilityMorphismWithGivenObjects}_{a, a', b, b'} : \underline{\text{Hom}}(a, a') \otimes \underline{\text{Hom}}(b, b') \rightarrow \underline{\text{Hom}}(a \otimes b, a' \otimes b')$.

7.4.37 TensorProductInternalHomCompatibilityMorphismWithGivenObjects (for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsList)

▷ TensorProductInternalHomCompatibilityMorphismWithGivenObjects(a, a', b, b', L) (operation)

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

The arguments are four objects a, a', b, b' , and a list $L = [\underline{\text{Hom}}(a, a') \otimes \underline{\text{Hom}}(b, b'), \underline{\text{Hom}}(a \otimes b, a' \otimes b')]$. The output is the natural morphism $\text{TensorProductInternalHomCompatibilityMorphismWithGivenObjects}_{a, a', b, b'} : \underline{\text{Hom}}(a, a') \otimes \underline{\text{Hom}}(b, b') \rightarrow \underline{\text{Hom}}(a \otimes b, a' \otimes b')$.

7.4.38 AddTensorProductInternalHomCompatibilityMorphismWithGivenObjects (for IsCapCategory, IsFunction)

▷ AddTensorProductInternalHomCompatibilityMorphismWithGivenObjects(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 $\text{TensorProductInternalHomCompatibilityMorphismWithGivenObjects}$. $F : (a, a', b, b', [\underline{\text{Hom}}(a, a') \otimes \underline{\text{Hom}}(b, b'), \underline{\text{Hom}}(a \otimes b, a' \otimes b')]) \mapsto \text{TensorProductInternalHomCompatibilityMorphismWithGivenObjects}_{a, a', b, b'}$.

7.4.39 TensorProductDualityCompatibilityMorphism (for IsCapCategoryObject, IsCapCategoryObject)

▷ TensorProductDualityCompatibilityMorphism(a, b) (operation)

Returns: a morphism in $\text{Hom}(a^\vee \otimes b^\vee, (a \otimes b)^\vee)$.

The arguments are two objects a, b . The output is the natural morphism $\text{TensorProductDualityCompatibilityMorphismWithGivenObjects} : a^\vee \otimes b^\vee \rightarrow (a \otimes b)^\vee$.

7.4.40 TensorProductDualityCompatibilityMorphismWithGivenObjects (for IsCap-CategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject)

▷ TensorProductDualityCompatibilityMorphismWithGivenObjects(s, a, b, r) (operation)
Returns: a morphism in $\text{Hom}(a^\vee \otimes b^\vee, (a \otimes b)^\vee)$.

The arguments are an object $s = a^\vee \otimes b^\vee$, two objects a, b , and an object $r = (a \otimes b)^\vee$. The output is the natural morphism $\text{TensorProductDualityCompatibilityMorphismWithGivenObjects}_{a,b} : a^\vee \otimes b^\vee \rightarrow (a \otimes b)^\vee$.

7.4.41 AddTensorProductDualityCompatibilityMorphismWithGivenObjects (for Is-CapCategory, IsFunction)

▷ AddTensorProductDualityCompatibilityMorphismWithGivenObjects(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 $\text{TensorProductDualityCompatibilityMorphismWithGivenObjects}$. $F : (a^\vee \otimes b^\vee, a, b, (a \otimes b)^\vee) \mapsto \text{TensorProductDualityCompatibilityMorphismWithGivenObjects}_{a,b}$.

7.4.42 MorphismFromTensorProductToInternalHom (for IsCapCategoryObject, Is-CapCategoryObject)

▷ MorphismFromTensorProductToInternalHom(a, b) (operation)
Returns: a morphism in $\text{Hom}(a^\vee \otimes b, \underline{\text{Hom}}(a, b))$.

The arguments are two objects a, b . The output is the natural morphism $\text{MorphismFromTensorProductToInternalHomWithGivenObjects}_{a,b} : a^\vee \otimes b \rightarrow \underline{\text{Hom}}(a, b)$.

7.4.43 MorphismFromTensorProductToInternalHomWithGivenObjects (for IsCap-CategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject)

▷ MorphismFromTensorProductToInternalHomWithGivenObjects(s, a, b, r) (operation)
Returns: a morphism in $\text{Hom}(a^\vee \otimes b, \underline{\text{Hom}}(a, b))$.

The arguments are an object $s = a^\vee \otimes b$, two objects a, b , and an object $r = \underline{\text{Hom}}(a, b)$. The output is the natural morphism $\text{MorphismFromTensorProductToInternalHomWithGivenObjects}_{a,b} : a^\vee \otimes b \rightarrow \underline{\text{Hom}}(a, b)$.

7.4.44 AddMorphismFromTensorProductToInternalHomWithGivenObjects (for Is-CapCategory, IsFunction)

▷ AddMorphismFromTensorProductToInternalHomWithGivenObjects(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 $\text{MorphismFromTensorProductToInternalHomWithGivenObjects}$. $F : (a^\vee \otimes b, a, b, \underline{\text{Hom}}(a, b)) \mapsto \text{MorphismFromTensorProductToInternalHomWithGivenObjects}_{a,b}$.

7.4.45 IsomorphismFromTensorProductToInternalHom (for IsCapCategoryObject, IsCapCategoryObject)

▷ IsomorphismFromTensorProductToInternalHom(a, b) (operation)

Returns: a morphism in $\text{Hom}(a^\vee \otimes b, \underline{\text{Hom}}(a, b))$.

The arguments are two objects a, b . The output is the natural morphism $\text{IsomorphismFromTensorProductToInternalHom}_{a,b} : a^\vee \otimes b \rightarrow \underline{\text{Hom}}(a, b)$.

7.4.46 AddIsomorphismFromTensorProductToInternalHom (for IsCapCategory, IsFunction)

▷ AddIsomorphismFromTensorProductToInternalHom(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 $\text{IsomorphismFromTensorProductToInternalHom}$. $F : (a, b) \mapsto \text{IsomorphismFromTensorProductToInternalHom}_{a,b}$.

7.4.47 MorphismFromInternalHomToTensorProduct (for IsCapCategoryObject, IsCapCategoryObject)

▷ MorphismFromInternalHomToTensorProduct(a, b) (operation)

Returns: a morphism in $\text{Hom}(\underline{\text{Hom}}(a, b), a^\vee \otimes b)$.

The arguments are two objects a, b . The output is the inverse of $\text{MorphismFromTensorProductToInternalHomWithGivenObjects}$, namely $\text{MorphismFromInternalHomToTensorProductWithGivenObjects}_{a,b} : \underline{\text{Hom}}(a, b) \rightarrow a^\vee \otimes b$.

7.4.48 MorphismFromInternalHomToTensorProductWithGivenObjects (for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject)

▷ MorphismFromInternalHomToTensorProductWithGivenObjects(s, a, b, r) (operation)

Returns: a morphism in $\text{Hom}(\underline{\text{Hom}}(a, b), a^\vee \otimes b)$.

The arguments are an object $s = \underline{\text{Hom}}(a, b)$, two objects a, b , and an object $r = a^\vee \otimes b$. The output is the inverse of $\text{MorphismFromTensorProductToInternalHomWithGivenObjects}$, namely $\text{MorphismFromInternalHomToTensorProductWithGivenObjects}_{a,b} : \underline{\text{Hom}}(a, b) \rightarrow a^\vee \otimes b$.

7.4.49 AddMorphismFromInternalHomToTensorProductWithGivenObjects (for IsCapCategory, IsFunction)

▷ AddMorphismFromInternalHomToTensorProductWithGivenObjects(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 $\text{MorphismFromInternalHomToTensorProductWithGivenObjects}$. $F : (\underline{\text{Hom}}(a, b), a, b, a^\vee \otimes b) \mapsto \text{MorphismFromInternalHomToTensorProductWithGivenObjects}_{a,b}$.

7.4.50 IsomorphismFromInternalHomToTensorProduct (for IsCapCategoryObject, IsCapCategoryObject)

▷ IsomorphismFromInternalHomToTensorProduct(a, b) (operation)

Returns: a morphism in $\text{Hom}(\underline{\text{Hom}}(a, b), a^\vee \otimes b)$.

The arguments are two objects a, b . The output is the inverse of IsomorphismFromTensorProductToInternalHom, namely IsomorphismFromInternalHomToTensorProduct $_{a,b} : \underline{\text{Hom}}(a, b) \rightarrow a^\vee \otimes b$.

7.4.51 AddIsomorphismFromInternalHomToTensorProduct (for IsCapCategory, IsFunction)

▷ AddIsomorphismFromInternalHomToTensorProduct(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 IsomorphismFromInternalHomToTensorProduct. $F : (a, b) \mapsto \text{IsomorphismFromInternalHomToTensorProduct}_{a,b}$.

7.4.52 TraceMap (for IsCapCategoryMorphism)

▷ TraceMap(α) (attribute)

Returns: a morphism in $\text{Hom}(1, 1)$.

The argument is a morphism α . The output is the trace morphism $\text{trace}_\alpha : 1 \rightarrow 1$.

7.4.53 AddTraceMap (for IsCapCategory, IsFunction)

▷ AddTraceMap(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 TraceMap. $F : \alpha \mapsto \text{trace}_\alpha$

7.4.54 RankMorphism (for IsCapCategoryObject)

▷ RankMorphism(a) (attribute)

Returns: a morphism in $\text{Hom}(1, 1)$.

The argument is an object a . The output is the rank morphism $\text{rank}_a : 1 \rightarrow 1$.

7.4.55 AddRankMorphism (for IsCapCategory, IsFunction)

▷ AddRankMorphism(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 RankMorphism. $F : a \mapsto \text{rank}_a$

7.4.56 IsomorphismFromDualToInternalHom (for IsCapCategoryObject)

▷ IsomorphismFromDualToInternalHom(a) (attribute)

Returns: a morphism in $\text{Hom}(a^\vee, \text{Hom}(a, 1))$.

The argument is an object a . The output is the isomorphism $\text{IsomorphismFromDualToInternalHom}_a : a^\vee \rightarrow \text{Hom}(a, 1)$.

7.4.57 AddIsomorphismFromDualToInternalHom (for IsCapCategory, IsFunction)

▷ $\text{AddIsomorphismFromDualToInternalHom}(C, F)$ (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 $\text{IsomorphismFromDualToInternalHom}$. $F : a \mapsto \text{IsomorphismFromDualToInternalHom}_a$

7.4.58 IsomorphismFromInternalHomToDual (for IsCapCategoryObject)

▷ $\text{IsomorphismFromInternalHomToDual}(a)$ (attribute)

Returns: a morphism in $\text{Hom}(\text{Hom}(a, 1), a^\vee)$.

The argument is an object a . The output is the isomorphism $\text{IsomorphismFromInternalHomToDual}_a : \text{Hom}(a, 1) \rightarrow a^\vee$.

7.4.59 AddIsomorphismFromInternalHomToDual (for IsCapCategory, IsFunction)

▷ $\text{AddIsomorphismFromInternalHomToDual}(C, F)$ (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 $\text{IsomorphismFromInternalHomToDual}$. $F : a \mapsto \text{IsomorphismFromInternalHomToDual}_a$

7.4.60 UniversalPropertyOfDual (for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryMorphism)

▷ $\text{UniversalPropertyOfDual}(t, a, \alpha)$ (operation)

Returns: a morphism in $\text{Hom}(t, a^\vee)$.

The arguments are two objects t, a , and a morphism $\alpha : t \otimes a \rightarrow 1$. The output is the morphism $t \rightarrow a^\vee$ given by the universal property of a^\vee .

7.4.61 AddUniversalPropertyOfDual (for IsCapCategory, IsFunction)

▷ $\text{AddUniversalPropertyOfDual}(C, F)$ (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 $\text{UniversalPropertyOfDual}$. $F : (t, a, \alpha : t \otimes a \rightarrow 1) \mapsto (t \rightarrow a^\vee)$.

7.4.62 LambdaIntroduction (for IsCapCategoryMorphism)

▷ $\text{LambdaIntroduction}(\alpha)$ (attribute)

Returns: a morphism in $\text{Hom}(1, \underline{\text{Hom}}(a, b))$.

The argument is a morphism $\alpha : a \rightarrow b$. The output is the corresponding morphism $1 \rightarrow \underline{\text{Hom}}(a, b)$ under the tensor hom adjunction.

7.4.63 AddLambdaIntroduction (for IsCapCategory, IsFunction)

▷ AddLambdaIntroduction(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 LambdaIntroduction. $F : (\alpha : a \rightarrow b) \mapsto (1 \rightarrow \underline{\text{Hom}}(a, b))$.

7.4.64 LambdaElimination (for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryMorphism)

▷ LambdaElimination(a, b, α) (operation)

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

The arguments are two objects a, b , and a morphism $\alpha : 1 \rightarrow \underline{\text{Hom}}(a, b)$. The output is a morphism $a \rightarrow b$ corresponding to α under the tensor hom adjunction.

7.4.65 AddLambdaElimination (for IsCapCategory, IsFunction)

▷ AddLambdaElimination(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 LambdaElimination. $F : (a, b, \alpha : 1 \rightarrow \underline{\text{Hom}}(a, b)) \mapsto (a \rightarrow b)$.

7.4.66 IsomorphismFromObjectToInternalHom (for IsCapCategoryObject)

▷ IsomorphismFromObjectToInternalHom(a) (attribute)

Returns: a morphism in $\text{Hom}(a, \underline{\text{Hom}}(1, a))$.

The argument is an object a . The output is the natural isomorphism $a \rightarrow \underline{\text{Hom}}(1, a)$.

7.4.67 IsomorphismFromObjectToInternalHomWithGivenInternalHom (for IsCapCategoryObject, IsCapCategoryObject)

▷ IsomorphismFromObjectToInternalHomWithGivenInternalHom(a, r) (operation)

Returns: a morphism in $\text{Hom}(a, \underline{\text{Hom}}(1, a))$.

The argument is an object a , and an object $r = \underline{\text{Hom}}(1, a)$. The output is the natural isomorphism $a \rightarrow \underline{\text{Hom}}(1, a)$.

7.4.68 AddIsomorphismFromObjectToInternalHomWithGivenInternalHom (for IsCapCategory, IsFunction)

▷ AddIsomorphismFromObjectToInternalHomWithGivenInternalHom(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 IsomorphismFromObjectToInternalHomWithGivenInternalHom. $F : (a, \underline{\text{Hom}}(1, a)) \mapsto (a \rightarrow \underline{\text{Hom}}(1, a))$.

7.4.69 IsomorphismFromInternalHomToObject (for IsCapCategoryObject)

- ▷ `IsomorphismFromInternalHomToObject(a)` (attribute)
Returns: a morphism in $\text{Hom}(\underline{\text{Hom}}(1, a), a)$.
 The argument is an object a . The output is the natural isomorphism $\underline{\text{Hom}}(1, a) \rightarrow a$.

7.4.70 IsomorphismFromInternalHomToObjectWithGivenInternalHom (for IsCapCategoryObject, IsCapCategoryObject)

- ▷ `IsomorphismFromInternalHomToObjectWithGivenInternalHom(a, s)` (operation)
Returns: a morphism in $\text{Hom}(\underline{\text{Hom}}(1, a), a)$.
 The argument is an object a , and an object $s = \underline{\text{Hom}}(1, a)$. The output is the natural isomorphism $\underline{\text{Hom}}(1, a) \rightarrow a$.

7.4.71 AddIsomorphismFromInternalHomToObjectWithGivenInternalHom (for IsCapCategory, IsFunction)

- ▷ `AddIsomorphismFromInternalHomToObjectWithGivenInternalHom(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 `IsomorphismFromInternalHomToObjectWithGivenInternalHom`. $F : (a, \underline{\text{Hom}}(1, a)) \mapsto (\underline{\text{Hom}}(1, a) \rightarrow a)$.

7.5 Rigid Symmetric Closed Monoidal Categories

A symmetric closed monoidal category \mathbf{C} satisfying

- the natural morphism $\underline{\text{Hom}}(a_1, b_1) \otimes \underline{\text{Hom}}(a_2, b_2) \rightarrow \underline{\text{Hom}}(a_1 \otimes a_2, b_1 \otimes b_2)$ is an isomorphism,
- the natural morphism $a \rightarrow \underline{\text{Hom}}(\underline{\text{Hom}}(a, 1), 1)$ is an isomorphism

is called a *rigid symmetric closed monoidal category*.

7.5.1 TensorProductInternalHomCompatibilityMorphismInverse (for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject)

- ▷ `TensorProductInternalHomCompatibilityMorphismInverse(a, a', b, b')` (operation)
Returns: a morphism in $\text{Hom}(\underline{\text{Hom}}(a \otimes b, a' \otimes b'), \underline{\text{Hom}}(a, a') \otimes \underline{\text{Hom}}(b, b'))$.
 The arguments are four objects a, a', b, b' . The output is the natural morphism `TensorProductInternalHomCompatibilityMorphismInverseWithGivenObjects` $_{a, a', b, b'} : \underline{\text{Hom}}(a \otimes b, a' \otimes b') \rightarrow \underline{\text{Hom}}(a, a') \otimes \underline{\text{Hom}}(b, b')$.

7.5.2 TensorProductInternalHomCompatibilityMorphismInverseWithGivenObjects (for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsList)

- ▷ `TensorProductInternalHomCompatibilityMorphismInverseWithGivenObjects(a, a', b, b', L)` (operation)
Returns: a morphism in $\text{Hom}(\underline{\text{Hom}}(a \otimes b, a' \otimes b'), \underline{\text{Hom}}(a, a') \otimes \underline{\text{Hom}}(b, b'))$.

The arguments are four objects a, a', b, b' , and a list $L = [\underline{\text{Hom}}(a, a') \otimes \underline{\text{Hom}}(b, b'), \underline{\text{Hom}}(a \otimes b, a' \otimes b')]$. The output is the natural morphism $\text{TensorProductInternalHomCompatibilityMorphismInverseWithGivenObjects}_{a, a', b, b'} : \underline{\text{Hom}}(a \otimes b, a' \otimes b') \rightarrow \underline{\text{Hom}}(a, a') \otimes \underline{\text{Hom}}(b, b')$.

7.5.3 AddTensorProductInternalHomCompatibilityMorphismInverseWithGivenObjects (for IsCapCategory, IsFunction)

▷ $\text{AddTensorProductInternalHomCompatibilityMorphismInverseWithGivenObjects}(C, F)$ (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 $\text{TensorProductInternalHomCompatibilityMorphismInverseWithGivenObjects}$.

$F : (a, a', b, b', [\underline{\text{Hom}}(a, a') \otimes \underline{\text{Hom}}(b, b'), \underline{\text{Hom}}(a \otimes b, a' \otimes b')]) \mapsto \text{TensorProductInternalHomCompatibilityMorphismInverseWithGivenObjects}_{a, a', b, b'}$.

7.5.4 MorphismFromBidual (for IsCapCategoryObject)

▷ $\text{MorphismFromBidual}(a)$ (attribute)

Returns: a morphism in $\text{Hom}((a^\vee)^\vee, a)$.

The argument is an object a . The output is the inverse of the morphism to the bidual $(a^\vee)^\vee \rightarrow a$.

7.5.5 MorphismFromBidualWithGivenBidual (for IsCapCategoryObject, IsCapCategoryObject)

▷ $\text{MorphismFromBidualWithGivenBidual}(a, s)$ (operation)

Returns: a morphism in $\text{Hom}((a^\vee)^\vee, a)$.

The argument is an object a , and an object $s = (a^\vee)^\vee$. The output is the inverse of the morphism to the bidual $(a^\vee)^\vee \rightarrow a$.

7.5.6 AddMorphismFromBidualWithGivenBidual (for IsCapCategory, IsFunction)

▷ $\text{AddMorphismFromBidualWithGivenBidual}(C, F)$ (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 $\text{MorphismFromBidualWithGivenBidual}$. $F : (a, (a^\vee)^\vee) \mapsto ((a^\vee)^\vee \rightarrow a)$.

Chapter 8

Managing Derived Methods

8.1 Info Class

8.1.1 DerivationInfo

▷ `DerivationInfo` (info class)

Info class for derivations.

8.1.2 ActivateDerivationInfo

▷ `ActivateDerivationInfo(arg)` (function)

Returns:

8.1.3 DeactivateDerivationInfo

▷ `DeactivateDerivationInfo(arg)` (function)

Returns:

8.2 Derivation Objects

8.2.1 IsDerivedMethod (for IsObject)

▷ `IsDerivedMethod(arg)` (filter)

Returns: `true` or `false`

A derivation object describes a derived method. It contains information about which operation the derived method implements, and which other operations it relies on.

8.2.2 MakeDerivation (for IsString, IsFunction, IsDenseList, IsPosInt, IsDenseList, IsFunction)

▷ `MakeDerivation(name, target_op, used_ops_with_multiples, weight, implementations_with_extra_filters, category_filter)` (operation)

Returns:

Creates a new derivation object. The argument `name` is an arbitrary name used to identify this derivation, and is useful only for debugging purposes. The argument `target_op` is the operation

which the derived method implements. The argument *used_ops_with_multiples* contains each operation used by the derived method, together with a positive integer specifying how many times that operation is used. This is given as a list of lists, where each sublist has as first entry an operation and as second entry an integer. The argument *weight* is an additional number to add when calculating the resulting weight of the target operation using this derivation. Unless there is any particular reason to regard the derivation as exceedingly expensive, this number should be 1. The argument *implementations_with_extra_filters* contains one or more functions with the actual implementation of the derived method, together with lists of extra argument filters for each function. The argument is a list with entries of the form `[fun, filters]`, where `fun` is a function and `filters` is a (not necessarily dense) list of argument filters. If only one function is given, then `filters` should be the empty list; in this case the argument's value would be `[[fun,[]]]`, where `fun` is the function. The argument *category_filter* is a filter describing which categories the derivation is valid for. If it is valid for all categories, then this argument should have the value `IsCapCategory`.

8.2.3 DerivationName (for IsDerivedMethod)

▷ `DerivationName(d)` (attribute)

Returns:

The name of the derivation. This is a name identifying this particular derivation, and normally not the same as the name of the operation implemented by the derivation.

8.2.4 DerivationWeight (for IsDerivedMethod)

▷ `DerivationWeight(d)` (attribute)

Returns:

Extra weight for the derivation.

8.2.5 DerivationFunctionsWithExtraFilters (for IsDerivedMethod)

▷ `DerivationFunctionsWithExtraFilters(d)` (attribute)

Returns:

The implementation(s) of the derivation, together with lists of extra filters for each implementation.

8.2.6 CategoryFilter (for IsDerivedMethod)

▷ `CategoryFilter(d)` (attribute)

Returns:

Filter describing which categories the derivation is valid for.

8.2.7 IsApplicableToCategory (for IsDerivedMethod, IsCapCategory)

▷ `IsApplicableToCategory(d, C)` (operation)

Returns: true if the category *C* is known to satisfy the category filter of the derivation *d*.

Checks if the derivation is known to be valid for a given category.

8.2.8 TargetOperation (for IsDerivedMethod)

- ▷ TargetOperation(*d*) (attribute)
Returns: The name (as a string) of the operation implemented by the derivation *d*

8.2.9 UsedOperations (for IsDerivedMethod)

- ▷ UsedOperations(*d*) (attribute)
Returns: The names (as strings) of the operations used by the derivation *d*

8.2.10 UsedOperationMultiples (for IsDerivedMethod)

- ▷ UsedOperationMultiples(*d*) (attribute)
Returns: Multiplicities of each operation used by the derivation *d*, in order corresponding to the operation names returned by UsedOperations(*d*).

8.2.11 UsedOperationsWithMultiples (for IsDerivedMethod)

- ▷ UsedOperationsWithMultiples(*d*) (attribute)
Returns: The names of the operations used by the derivation *d*, together with their multiplicities. The result is a list consisting of lists of the form [*op_name*, *mult*], where *op_name* is a string and *mult* a positive integer.

8.2.12 InstallDerivationForCategory (for IsDerivedMethod, IsPosInt, IsCapCategory)

- ▷ InstallDerivationForCategory(*d*, *weight*, *C*) (operation)
Returns:
 Install the derived method *d* for the category *C*. The integer *weight* is the computed weight of the operation implemented by this derivation.

8.2.13 DerivationResultWeight (for IsDerivedMethod, IsDenseList)

- ▷ DerivationResultWeight(*d*, *op_weights*) (operation)
Returns:
 Computes the resulting weight of the target operation of this derivation given a list of weights for the operations it uses. The argument *op_weights* should be a list of integers specifying weights for the operations given by UsedOperations(*d*), in the same order.

8.3 Derivation Graphs

8.3.1 IsDerivedMethodGraph (for IsObject)

- ▷ IsDerivedMethodGraph(*arg*) (filter)
Returns: true or false
 A derivation graph consists of a set of operations and a set of derivations specifying how some operations can be implemented in terms of other operations.

8.3.2 MakeDerivationGraph (for IsDenseList)

▷ MakeDerivationGraph(*operations*) (operation)

Returns:

Make a derivation graph containing the given set of operations and no derivations. The argument *operations* should be a list of strings, the names of the operations. The set of operations is fixed once the graph is created. Derivations can be added to the graph by calling AddDerivation.

8.3.3 AddOperationsToDerivationGraph (for IsDerivedMethodGraph, IsDenseList)

▷ AddOperationsToDerivationGraph(*graph*, *operations*) (operation)

Returns:

Adds a list of operation names *operations* to a given derivation graph *graph*. This is used in extensions of CAP which want to have their own primitive operations, but do not want to pollute the CAP kernel any more. Please use it with caution. If a weight list/category was created before it will not be aware of the operations.

8.3.4 AddDerivation (for IsDerivedMethodGraph, IsDerivedMethod)

▷ AddDerivation(*G*, *d*) (operation)

Returns:

Add a derivation to a derivation graph.

8.3.5 AddDerivation (for IsDerivedMethodGraph, IsFunction, IsDenseList, IsObject)

▷ AddDerivation(*arg1*, *arg2*, *arg3*, *arg4*) (operation)

Returns:

8.3.6 AddDerivation (for IsDerivedMethodGraph, IsFunction, IsDenseList)

▷ AddDerivation(*arg1*, *arg2*, *arg3*) (operation)

Returns:

8.3.7 AddDerivation (for IsDerivedMethodGraph, IsFunction, IsFunction)

▷ AddDerivation(*arg1*, *arg2*, *arg3*) (operation)

Returns:

8.3.8 AddDerivationPair (for IsDerivedMethodGraph, IsFunction, IsFunction, IsDenseList, IsDenseList, IsDenseList)

▷ AddDerivationPair(*arg1*, *arg2*, *arg3*, *arg4*, *arg5*, *arg6*) (operation)

Returns:

8.3.9 AddDerivationPair (for IsDerivedMethodGraph, IsFunction, IsFunction, IsDenseList, IsDenseList)

▷ AddDerivationPair(*arg1*, *arg2*, *arg3*, *arg4*, *arg5*) (operation)

Returns:

8.3.10 AddDerivationPair (for IsDerivedMethodGraph, IsFunction, IsFunction, IsDenseList, IsFunction, IsFunction)

▷ AddDerivationPair(*arg1*, *arg2*, *arg3*, *arg4*, *arg5*, *arg6*) (operation)

Returns:

8.3.11 AddDerivationPair (for IsDerivedMethodGraph, IsFunction, IsFunction, IsFunction, IsFunction)

▷ AddDerivationPair(*arg1*, *arg2*, *arg3*, *arg4*, *arg5*) (operation)

Returns:

8.3.12 AddDerivationToCAP

▷ AddDerivationToCAP(*arg*) (function)

Returns:

8.3.13 AddDerivationPairToCAP

▷ AddDerivationPairToCAP(*arg*) (function)

Returns:

8.3.14 AddWithGivenDerivationPairToCAP

▷ AddWithGivenDerivationPairToCAP(*arg*) (function)

Returns:

8.3.15 Operations (for IsDerivedMethodGraph)

▷ Operations(*G*) (attribute)

Returns:

Gives the operations in the graph *G*, as a list of strings.

8.3.16 DerivationsUsingOperation (for IsDerivedMethodGraph, IsString)

▷ DerivationsUsingOperation(*G*, *op_name*) (operation)

Returns:

Finds all the derivations in the graph *G* that use the operation named *op_name*, and returns them as a list.

8.3.17 DerivationsOfOperation (for IsDerivedMethodGraph, IsString)

▷ DerivationsOfOperation(*G*, *op_name*) (operation)

Returns:

Finds all the derivations in the graph *G* targeting the operation named *op_name* (that is, the derivations that provide implementations of this operation), and returns them as a list.

8.4 Managing Derivations in a Category

8.4.1 IsOperationWeightList (for IsObject)

▷ `IsOperationWeightList(arg)` (filter)
Returns: true or false

An operation weight list manages the use of derivations in a single category C . For every operation, it keeps a weight value which indicates how costly it is to perform that operation in the category C . Whenever a new operation is implemented in C , the operation weight list should be notified about this and given a weight to assign to this operation. It will then automatically install all possible derived methods for C in such a way that every operation has the smallest possible weight (the weight of a derived method is computed by using the weights of the operations it uses; see `DerivationResultWeight`).

8.4.2 MakeOperationWeightList (for IsCapCategory, IsDerivedMethodGraph)

▷ `MakeOperationWeightList(C, G)` (operation)
Returns:

Create the operation weight list for a category. This should only be done once for every category, and the category should afterwards remember the returned object. The argument C is the CAP category this operation weight list is associated to, and the argument G is a derivation graph containing operation names and derivations.

8.4.3 DerivationGraph (for IsOperationWeightList)

▷ `DerivationGraph(owl)` (attribute)
Returns:

Returns the derivation graph used by the operation weight list `owl`.

8.4.4 CategoryOfOperationWeightList (for IsOperationWeightList)

▷ `CategoryOfOperationWeightList(owl)` (attribute)
Returns:

Returns the CAP category associated to the operation weight list `owl`.

8.4.5 CurrentOperationWeight (for IsOperationWeightList, IsString)

▷ `CurrentOperationWeight(owl, op_name)` (operation)
Returns:

Returns the current weight of the operation named `op_name`.

8.4.6 OperationWeightUsingDerivation (for IsOperationWeightList, IsDerivedMethod)

▷ `OperationWeightUsingDerivation(owl, d)` (operation)
Returns:

Finds out what the weight of the operation implemented by the derivation d would be if we had used that derivation.

8.4.7 DerivationOfOperation (for IsOperationWeightList, IsString)

▷ `DerivationOfOperation(owl, op_name)` (operation)

Returns:

Returns the derivation which is currently used to implement the operation named `op_name`. If the operation is not implemented by a derivation (that is, either implemented directly or not implemented at all), then `fail` is returned.

8.4.8 InstallDerivationsUsingOperation (for IsOperationWeightList, IsString)

▷ `InstallDerivationsUsingOperation(owl, op_name)` (operation)

Returns:

Performs a search from the operation `op_name`, and installs all derivations that give improvements over the current state. This is used internally by `AddPrimitiveOperation` and `Reevaluate`. It should normally not be necessary to call this function directly.

8.4.9 Reevaluate (for IsOperationWeightList)

▷ `Reevaluate(owl)` (operation)

Returns:

Reevaluate the installed derivations, installing better derivations if possible. This should be called if new derivations become available for the category, either because the category has acquired more knowledge about itself (e.g. it is told that it is abelian) or because new derivations have been added to the graph.

8.4.10 AddPrimitiveOperation (for IsOperationWeightList, IsString, IsInt)

▷ `AddPrimitiveOperation(owl, op_name, weight)` (operation)

Returns:

Add the operation named `op_name` to the operation weight list `owl` with weight `weight`. This causes all operations that can be derived, directly or indirectly, from the newly added operation to be installed as well (unless they are already installed with the same or lower weight).

8.4.11 PrintDerivationTree (for IsOperationWeightList, IsString)

▷ `PrintDerivationTree(owl, op_name)` (operation)

Returns:

Print a tree representation of the way the operation named `op_name` is implemented in the category of the operation weight list `owl`.

8.4.12 PrintTree (for IsObject, IsFunction, IsFunction)

▷ `PrintTree(arg1, arg2, arg3)` (operation)

Returns:

Prints a tree structure.

8.4.13 PrintTreeRec (for IsObject, IsFunction, IsFunction, IsInt)

▷ PrintTreeRec(*arg1*, *arg2*, *arg3*, *arg4*) (operation)
Returns:

8.5 Min Heaps for Strings

This section describes an implementation of min heaps for storing strings with associated integer keys, used internally by operation weight lists.

8.5.1 IsStringMinHeap (for IsObject)

▷ IsStringMinHeap(*arg*) (filter)
Returns: true or false
 A string min heap is a min heap where every node contains a string label and an integer key.

8.5.2 StringMinHeap

▷ StringMinHeap(*arg*) (function)
Returns:
 Create an empty string min heap.

8.5.3 Add (for IsStringMinHeap, IsString, IsInt)

▷ Add(*H*, *string*, *key*) (operation)
Returns:
 Add a new node containing the label *string* and the key *key* to the heap *H*.

8.5.4 ExtractMin (for IsStringMinHeap)

▷ ExtractMin(*H*) (operation)
Returns:
 Remove a node with minimal key value from the heap *H*, and return it. The return value is a list [*label*, *key*], where *label* is the extracted node's label (a string) and *key* is the node's key (an integer).

8.5.5 DecreaseKey (for IsStringMinHeap, IsString, IsInt)

▷ DecreaseKey(*H*, *string*, *key*) (operation)
Returns:
 Decrease the key value for the node with label *string* in the heap *H*. The new key value is given by *key* and must be smaller than the node's current value.

8.5.6 IsEmptyHeap (for IsStringMinHeap)

▷ IsEmptyHeap(*H*) (operation)
Returns:
 Returns true if the heap *H* is empty, false otherwise.

8.5.7 HeapSize (for IsStringMinHeap)

▷ HeapSize(H) (operation)

Returns:

Returns the number of nodes in the heap H .

8.5.8 Contains (for IsStringMinHeap, IsString)

▷ Contains(H , $string$) (operation)

Returns:

Returns true if the heap H contains a node with label $string$, and false otherwise.

8.5.9 Swap (for IsStringMinHeap, IsPosInt, IsPosInt)

▷ Swap(H , i , j) (operation)

Returns:

Swaps two elements in the list used to implement the heap, and updates the heap's internal mapping of labels to list indices. This is an internal function which should only be called from the functions that implement the heap functionality.

8.5.10 Heapify (for IsStringMinHeap, IsPosInt)

▷ Heapify(H , i) (operation)

Returns:

Heapify the heap H , starting from index i . This is an internal function.

Chapter 9

Add Functions

This section describes the overall structure of Add-functions and the functions installed by them.

9.1 Functions Installed by Add

Add functions (up to some exceptions) have the following syntax `DeclareOperation("AddSomeFunc", [IsCapCategory, IsList, IsInt])`; The first argument is the category to which some function (e.g. `KernelObject`) is added, the second is a list containing pairs of functions and additional filters for the arguments, (e.g. if one argument is a morphism, an additional filter could be `IsMorphism`). The third is a weight which will then be the weight for `SomeFunc`. This is described later. If only one function is to be installed, the list can be replaced by the function. Via `InstallMethod`, CAP installs the given function(s) as methods for the install name of `SomeFunc`, as listed in the `MethodRecord`. If no install name is given, the name `SomeFunc` is used. All installed methods follow the following steps, described below:

- Redirect function
- Prefunction
- Function
- Logic
- Postfunction
- Addfunction

Every other part, except from function, does only depend on the name `SomeFunc`. We now explain the steps in detail.

- Redirect function: The redirect is used to redirect the computation from the given functions to some other symbol. If there is for example a `with` given method for some universal property, and the universal object is already computed, the redirect function might detect such a thing, calls the `with` given operation with the universal object as additional argument and then returns the value. In general, the redirect can be an arbitrary function. It is called with the same arguments as the operation `SomeFunc` itself and can return an array containing `[true, something]`, which will cause the installed method to simply return the object `something`, or `[false]`. If the output is

false, the computation will continue with the step Prefunction. Additionally, for every category and every name like SomeFunc, there is a boolean, stored in the category's redirects component under the name of SomeFunc, which, when it is false, will prevent the redirect function from being executed.

- Prefunction: The prefunction should be used for error handling and soft checks of the sanity of the input to SomeFunc (e.g. for KernelLift it should check whether range and source of the morphisms coincide). Generally, the prefunction is defined in the method record and only depends on the name SomeFunc. It is called with the same input as the function itself, and should return either [true], which continues the computation, or [false, "message"], which will cause an error with message "message" and some additional information.
- Function: This will launch the function(s) given as arguments. The result should be as specified in the type of SomeFunc. The resulting object is now named the result.
- Logic: For every function, some logical todos can be implemented in a logic texfile for the category. If there is some logic written down in a file belonging to the category, or belonging to some type of category. Please see the description of logic for more details. If there is some logic and some predicate relations for the function SomeFunc, it is installed in this step for the result.
- Postfunction: The postfunction called with the arguments of the function and the result. It can be an arbitrary function doing some cosmetics. If for example SomeFunc is KernelEmbedding, it will set the KernelObject of the input morphism to result. The postfunction is also taken from the method record and does only depend on the name SomeFunc.
- Addfunction: If the result is a category cell, it is added to the category for which the function was installed.

9.2 Add Method

Except from installing a new method for the name SomeFunc, an Add method does slightly more. Every Add method has the same structure. The steps in the Add method are as follows:

- Weight check: If the current weight of the operation is lower than the given weight of the new functions, then the add function returns and installs nothing.
- Option check: There are two possible options for every add method: SetPrimitive and IsDerivation.
 - SetPrimitive should be a boolean, the default is true. If SetPrimitive is false, then the current call of this add will not set the installed function to be primitive. This is used for derivations.
 - IsDerivation should be a boolean, default is false. If it is true, the add method assumes that the given function is a derivation and does not try to install a corresponding pair (See below).
- Standard weight: If the weight parameter is -1, the Standard weight is assumed, which is 100.

- **Checking for pairs:** If the function is not a with given operation, has a corresponding with given or is a with given, and is newly installed, i.e. the current installation weight which is given to the add function is less than the current weight, the add method is going to install a corresponding pair function, i.e. a function for the corresponding with or without given method, which redirects to the currently installed functions. It also deactivates the redirect for this function. Note that the pair install is only done for primitive functions, and if the current weight is higher than the given weight.
- **Can compute:** Set the corresponding can compute of the category to true
- **Install methods:** Decide on the methods used to install the function. Check whether `InstallMethodWithCache`, `InstallMethodWithToDoForIsWellDefined`, both, or simply `InstallMethod` is used. This is decided by the `ToDo` and the caching flags.
- **Installation:** Next, the method to install the functions is created. It creates the correct filter list, by merging the standard filters for the operation with the particular filters for the given functions, then installs the method as described above.
- **SetPrimitive:** If the set primitive flag is true, it is set as primitive in the weight list of the category.
- **Pair install:** If there is a function pair, as described above, it is installed.

After calling an add method, the corresponding Operation is available in the category. Also, some derivations, which are triggered by the setting of the primitive value, might be available.

9.3 InstallAdd Function

Almost all Add methods in the CAP kernel are installed by the `CapInternalInstallAdd` operation. The definition of this function is as follows: `DeclareOperation("CapInternalInstallAdd", [IsRecord]);` The record can have the following components, used as described:

- **function_name:** The name of the function. This does not have to coincide with the installation name. It is used for the derivation weight.
- **installation_name (optional):** A string which is the name of the operation for which the functions given to the Add method are installed as methods.
- **pre_function (optional):** A function which is used as the prefunction of the installed methods, as described above.
- **redirect_function (optional):** A function which is used as the redirect function of the installed methods, as described above.
- **post_function (optional):** A function which is used as the postfunction of the installed methods, as described above.
- **filter_list:** A list containing the basic filters for the methods installed by the add methods. Possible are filters, or the strings category, object, morphism, or twocell, which will then be replaced at the time the add method is called with the corresponding filters of the category.

- `well_defined_todo` (optional): A boolean, default value is true, which states whether there should be to do list entries which propagate well definedness from the input of the installed methods to their output. Please note that true only makes sense if at least one argument and the output of the installed method is a cell.
- `cache_name` (optional): The name of the cache which is used for the installed methods. If no cache name is given, the caching for the operation is deactivated completely.
- `argument_list` (optional): A list containing integers, which defines which arguments should be used for the additional functions, (e.g. `redirect`, `pre`, ...). This is important for the `Op` method constructions. If no argument list is given, all arguments are used. Please note that if you have a method selection argument for your function, you need to give the `argument_list` to explicitly state which argument is the method selection argument.
- `return_type` (optional): The return type can be `object`, `morphism`, or `twocell`. If it is one of those, the correct add function (see above) is used for the result of the computation. Otherwise, no add function is used after all.
- `is_with_given`: Boolean, marks whether the function which is to be installed is a with given function or not.
- `with_given_without_given_name_pair` (optional): If the currently installed operation has a corresponding with given operation or is the with given of another operation, the names of both should be in this list.
- `functorial` (optional): If an object has a corresponding functorial function, e.g., `KernelObject` and `KernelObjectFunctorial`, the name of the functorial is stored as a string.

Using all those entries, the operation `CapInternalInstallAdd` installs add methods as described above. It first provides a sanity check for all the entries described, then installs the `Add` method in 4 ways, with list or functions as second argument, and with an optional third parameter for the weight.

9.4 Install All Adds

The function `CAP_INTERNAL_INSTALL_ALL_ADDS` does not take any arguments, it is an auxiliary function which iterates over the `CAP_INTERNAL_METHOD_NAME_RECORD` and calls, after some cosmetics, the `CapInternalInstallAdd` with the corresponding method record entry. The steps below are performed for every entry of the method record:

- **No install check:** If the `no_install` component in the record is set to true, the loop continues with the next entry, since this flag indicates that there should be no add function for this operation.
- **Cache check:** If there is no `cache_name`, set it to the name of the method record entry.
- **Function name:** Set the component `function_name` to the entry name.
- **Redirect:** Since the `redirect` function needs the category to work correctly, the given redirects in the method records are packed up to discard the last argument, which is the category.
- **arg_list:** Next, an argument list for `redirect` and `post` function is created, by looking at the filter list in the record. If the first one is a list, the first and the last (method selection argument) is used, otherwise only the first.

- **WithGiven special case:** If the current entry belongs to a `WithGiven` operation, the `with_given_without_given_name_pair` is set, the `with given` flag is set to true, and the `CapInternalInstallAdd` is called with the record. The loop then continues.
- **Non universal object special case:** If the `Operation` does not have a universal type, i.e. does not belong to a universal construction, `CapInternalInstallAdd` is called with the record. The loop then continues.

Please note that we are now in the case where the operation belongs to a universal construction, (e.g. `KernelLift`) and is not a `WithGiven` type of operation.

- `argument_list`: If the method is an `Op` construction, i.e. has a method selection object, the argument list is set to all but the last object and then used as above. Otherwise, the `argument_list` is set to all arguments.
- If the `Operation` constructs a universal object, the postfunction is created and then `CapInternalInstallAdd` is called.
- If the `Operation` constructs a universal morphism, the redirect is created and stored in the record. Also, the postfunction is created. Then `CapInternalInstallAdd` is called.

After one call of this function, all add methods are installed correctly. A second call should not do anything.

Chapter 10

Technical Details

10.1 The Category Cat

10.1.1 ObjectCache (for IsCapFunctor)

- ▷ `ObjectCache(functor)` (attribute)
Returns: `IsCachingObject`
Returns the caching object which stores the results of the functor *functor* applied to objects.

10.1.2 MorphismCache (for IsCapFunctor)

- ▷ `MorphismCache(functor)` (attribute)
Returns: `IsCachingObject`
Returns the caching object which stores the results of the functor *functor* applied to morphisms.

10.2 Install Functions for IsWellDefined

10.2.1 InstallMethodWithToDoForIsWellDefined

- ▷ `InstallMethodWithToDoForIsWellDefined(arg)` (function)
Returns:

The `IsWellDefined` filter is a basic function of CAP. For every categorial construction the outcome is well defined if and only if every input object or morphism of the construction is well defined. So for every implementation of a categorial construction a `ToDoListEntry` needs to be defined which propagates well definedness from the input cells to the output. For not writing this construction in every method, this function can be used to install a method which then installs the correct `ToDoListEntries` for the output. The input syntax works exactly like `InstallMethod`, with one extension: The method creates an auxilliary function which computes the output from the function given to `InstallMethodWithToDoForIsWellDefined`, then installs the `ToDoListEntries`, and then installs the auxilliary function instead of the original one. This is normally done with `InstallMethod`. However, one can change this via the option `InstallMethod`, which can be set to any other function which is then used instead of `InstallMethod`. This is used for the caching functions.

10.2.2 InstallSetWithToDoForIsWellDefined (for IsObject, IsString, IsList)

▷ `InstallSetWithToDoForIsWellDefined(arg1, arg2, arg3)` (operation)

Returns:

For the caching one needs the possibility to install setters for functions with multiple arguments. This is a setter function which automatically adds `ToDoListEntries` for `IsWellDefined` like described above for the manually setted result of a method.

10.2.3 DeclareAttributeWithToDoForIsWellDefined

▷ `DeclareAttributeWithToDoForIsWellDefined(arg)` (function)

Returns:

Since attributes install their setters themselves, one needs to declare attributes in another way to ensure `ToDoListEntries` for `IsWellDefined` in the setter of an attribute. This function works like `DeclareAttribute`, but installs `ToDoListEntries` for the setter of the attribute. Please note that implementations still need to be done with `InstallMethodWithToDoForIsWellDefined`.

10.2.4 DeclareFamilyProperty

▷ `DeclareFamilyProperty(arg)` (function)

Returns:

10.2.5 CAP_INTERNAL_REPLACE_STRINGS_WITH_FILTERS

▷ `CAP_INTERNAL_REPLACE_STRINGS_WITH_FILTERS(list, category)` (function)

Returns: Replaced list

The function takes a list (of lists) of filters or strings, where the strings can be category, cell, object, morphism, or twocell. The strings are then recursively replaced by the corresponding filters of the category. The replaced list is returned.

10.2.6 CAP_INTERNAL_MERGE_FILTER_LISTS

▷ `CAP_INTERNAL_MERGE_FILTER_LISTS(list, additional, list)` (function)

Returns: merged lists

The first argument should be a dense list with filters, the second a sparse list containing filters not longer then the first one. The filters of the second list are then appended (via `and`) to the filters in the first list at the corresponding position, and the resulting list is returned.

10.2.7 CAP_INTERNAL_RETURN_OPTION_OR_DEFAULT

▷ `CAP_INTERNAL_RETURN_OPTION_OR_DEFAULT(string, value)` (function)

Returns: option value

Returns the value of the option with name `string`, or, if this value is fail, the object value.

10.2.8 CAP_INTERNAL_FIND_APPEARANCE_OF_SYMBOL_IN_FUNCTION

▷ `CAP_INTERNAL_FIND_APPEARANCE_OF_SYMBOL_IN_FUNCTION(function, symbol_list, loop_multiple)` (function)

Returns: a list of symbols with multiples

The function searches for the appearance of the strings in symbol list on the function function and returns a list of pairs, containing the name of the symbol and the number of appearance. If the symbol appears in a loop, the number of appearance is counted times the loop multiple.

10.2.9 CAP_INTERNAL_MERGE_PRECONDITIONS_LIST

▷ CAP_INTERNAL_MERGE_PRECONDITIONS_LIST(*list1*, *list2*) (function)

Returns: merge list

The function takes two lists containing pairs of symbols (strings) and multiples. The lists are merged that pairs where the string only appears in one list is then added to the return list, if a pair with a string appears in both lists, the resulting lists only contains this pair once, with the higher multiple from both lists.

Chapter 11

Examples and Tests

11.1 Spectral Sequences

Example

```
gap> ZZ := HomalgRingOfIntegersInSingular( );
Z
gap> C1 := FreeLeftPresentation( 1, ZZ );
<An object in Category of left presentations of Z>
gap> C2 := FreeLeftPresentation( 2, ZZ );
<An object in Category of left presentations of Z>
gap> h1 := PresentationMorphism( C2, HomalgMatrix( [ [ 0 ], [ 4 ] ], ZZ ), C1 );
<A morphism in Category of left presentations of Z>
gap> h2 := PresentationMorphism( C2, HomalgMatrix( [ [ 0 ], [ 2 ] ], ZZ ), C1 );
<A morphism in Category of left presentations of Z>
gap> v1 := PresentationMorphism( C2, HomalgMatrix( [ [ 2, 0 ], [ 1, 2 ] ], ZZ ), C2 );
<A morphism in Category of left presentations of Z>
gap> v2 := PresentationMorphism( C1, HomalgMatrix( [ [ 4 ] ], ZZ ), C1 );
<A morphism in Category of left presentations of Z>
gap> cocomplex_h1 := CocomplexFromMorphismList( [ h1 ] );
<An object in Cocomplex category of Category of left presentations of Z>
gap> cocomplex_h2 := CocomplexFromMorphismList( [ h2 ] );
<An object in Cocomplex category of Category of left presentations of Z>
gap> cocomplex_mor := CochainMap( cocomplex_h2, [ v1, v2 ], cocomplex_h1 );
<A morphism in Cocomplex category of Category of left presentations of Z>
gap> Zmod := CapCategory( C1 );
Category of left presentations of Z
gap> CH0 := CohomologyFunctor( Zmod, 0 );
0-th cohomology functor of Category of left presentations of Z
gap> cmor0 := ApplyFunctor( CH0, cocomplex_mor );
<A morphism in Category of left presentations of Z>
gap> Display( UnderlyingMatrix( cmor0 ) );
2
gap> CH1 := CohomologyFunctor( Zmod, 1 );
1-th cohomology functor of Category of left presentations of Z
gap> cmor1 := ApplyFunctor( CH1, cocomplex_mor );
<A morphism in Category of left presentations of Z>
gap> Display( UnderlyingMatrix( cmor1 ) );
4
gap> ToComplex := CocomplexToComplexFunctor( Zmod );
```

```

Cocomplex to complex functor of Category of left presentations of Z
gap> complex_mor := ApplyFunctor( ToComplex, cocomplex_mor );
<A morphism in Complex category of Category of left presentations of Z>
gap> H0 := HomologyFunctor( Zmod, 0 );
0-th homology functor of Category of left presentations of Z
gap> mor0 := ApplyFunctor( H0, complex_mor );
<A morphism in Category of left presentations of Z>
gap> Display( UnderlyingMatrix( mor0 ) );
2
gap> Hm1 := HomologyFunctor( Zmod, -1 );
-1-th homology functor of Category of left presentations of Z
gap> mor1 := ApplyFunctor( Hm1, complex_mor );
<A morphism in Category of left presentations of Z>
gap> Display( UnderlyingMatrix( mor1 ) );
4

```

————— Example —————

```

gap> QQ := HomalgFieldOfRationalsInSingular( );
gap> R := QQ * "x,y";
Q[x,y]
gap> SetRecursionTrapInterval( 10000 );
gap> category := LeftPresentations( R );
Category of left presentations of Q[x,y]
gap> S := FreeLeftPresentation( 1, R );
<An object in Category of left presentations of Q[x,y]>
gap> object_func := function( i ) return S; end;
function( i ) ... end
gap> morphism_func := function( i ) return IdentityMorphism( S ); end;
function( i ) ... end
gap> C0 := ZFunctorObjectExtendedByInitialAndIdentity( object_func, morphism_func, category, 0,
<An object in Functors from integers into Category of left presentations of Q[x,y]>
gap> S2 := FreeLeftPresentation( 2, R );
<An object in Category of left presentations of Q[x,y]>
gap> C1 := ZFunctorObjectFromMorphismList( [ InjectionOfCofactorOfDirectSum( [ S2, S ], 1 ) ], 2
<An object in Functors from integers into Category of left presentations of Q[x,y]>
gap> C1 := ZFunctorObjectExtendedByInitialAndIdentity( C1, 2, 3 );
<An object in Functors from integers into Category of left presentations of Q[x,y]>
gap> C2 := ZFunctorObjectFromMorphismList( [ InjectionOfCofactorOfDirectSum( [ S, S ], 1 ) ], 3
<An object in Functors from integers into Category of left presentations of Q[x,y]>
gap> C2 := ZFunctorObjectExtendedByInitialAndIdentity( C2, 3, 4 );
<An object in Functors from integers into Category of left presentations of Q[x,y]>
gap> delta_1_3 := PresentationMorphism( C1[3], HomalgMatrix( [ [ "x^2" ], [ "xy" ], [ "y^3" ] ],
<A morphism in Category of left presentations of Q[x,y]>
gap> delta_1_2 := PresentationMorphism( C1[2], HomalgMatrix( [ [ "x^2" ], [ "xy" ] ], 2, 1, R ),
<A morphism in Category of left presentations of Q[x,y]>
gap> delta1 := ZFunctorMorphism( C1, [ UniversalMorphismFromInitialObject( C0[1] ), UniversalMor
<A morphism in Functors from integers into Category of left presentations of Q[x,y]>
gap> delta1 := ZFunctorMorphismExtendedByInitialAndIdentity( delta1, 0, 3 );
<A morphism in Functors from integers into Category of left presentations of Q[x,y]>
gap> delta1 := AsAscendingFilteredMorphism( delta1 );
<A morphism in Ascending filtered object category of Category of left presentations of Q[x,y]>
gap> delta_2_3 := PresentationMorphism( C2[3], HomalgMatrix( [ [ "y", "-x", "0" ] ], 1, 3, R ),
<A morphism in Category of left presentations of Q[x,y]>
gap> delta_2_4 := PresentationMorphism( C2[4], HomalgMatrix( [ [ "y", "-x", "0" ], [ "0", "y^2",

```

```

<A morphism in Category of left presentations of Q[x,y]>
gap> delta2 := ZFunctorMorphism( C2, [ UniversalMorphismFromInitialObject( C1[2] ), delta_2_3,
<A morphism in Functors from integers into Category of left presentations of Q[x,y]>
gap> delta2 := ZFunctorMorphismExtendedByInitialAndIdentity( delta2, 2, 4 );
<A morphism in Functors from integers into Category of left presentations of Q[x,y]>
gap> delta2 := AsAscendingFilteredMorphism( delta2 );
<A morphism in Ascending filtered object category of Category of left presentations of Q[x,y]>
gap> SetIsAdditiveCategory( CategoryOfAscendingFilteredObjects( category ), true );
gap> complex := ZFunctorObjectFromMorphismList( [ delta2, delta1 ], -2 );
<An object in Functors from integers into Ascending filtered object category of Category of left
gap> complex := AsComplex( complex );
<An object in Complex category of Ascending filtered object category of Category of left present
gap> LessGenFunctor := FunctorLessGeneratorsLeft( R );
Less generators for Category of left presentations of Q[x,y]
gap> s := SpectralSequenceEntryOfAscendingFilteredComplex( complex, 0, 0, 0 );
<A morphism in Generalized morphism category of Category of left presentations of Q[x,y]>
gap> Display( UnderlyingMatrix( ApplyFunctor( LessGenFunctor, UnderlyingHonestObject( Source( s
(an empty 0 x 1 matrix)
gap> s := SpectralSequenceEntryOfAscendingFilteredComplex( complex, 1, 0, 0 );
<A morphism in Generalized morphism category of Category of left presentations of Q[x,y]>
gap> Display( UnderlyingMatrix( ApplyFunctor( LessGenFunctor, UnderlyingHonestObject( Source( s
(an empty 0 x 1 matrix)
gap> s := SpectralSequenceEntryOfAscendingFilteredComplex( complex, 2, 0, 0 );
<A morphism in Generalized morphism category of Category of left presentations of Q[x,y]>
gap> Display( UnderlyingMatrix( ApplyFunctor( LessGenFunctor, UnderlyingHonestObject( Source( s
(an empty 0 x 1 matrix)
gap> s := SpectralSequenceEntryOfAscendingFilteredComplex( complex, 3, 0, 0 );
<A morphism in Generalized morphism category of Category of left presentations of Q[x,y]>
gap> Display( UnderlyingMatrix( ApplyFunctor( LessGenFunctor, UnderlyingHonestObject( Source( s
x*y,
x^2
gap> s := SpectralSequenceEntryOfAscendingFilteredComplex( complex, 4, 0, 0 );
<A morphism in Generalized morphism category of Category of left presentations of Q[x,y]>
gap> Display( UnderlyingMatrix( ApplyFunctor( LessGenFunctor, UnderlyingHonestObject( Source( s
x*y,
x^2,
y^3
gap> s := SpectralSequenceEntryOfAscendingFilteredComplex( complex, 5, 0, 0 );
<A morphism in Generalized morphism category of Category of left presentations of Q[x,y]>
gap> Display( UnderlyingMatrix( ApplyFunctor( LessGenFunctor, UnderlyingHonestObject( Source( s
x*y,
x^2,
y^3
gap> s := SpectralSequenceDifferentialOfAscendingFilteredComplex( complex, 3, 3, -2 );
<A morphism in Category of left presentations of Q[x,y]>
gap> Display( UnderlyingMatrix( ApplyFunctor( LessGenFunctor, s ) ) );
y^3
gap> AscToDescFunctor := AscendingToDescendingFilteredObjectFunctor( category );
Ascending to descending filtered object functor of Category of left presentations of Q[x,y]
gap> cocomplex := ZFunctorObjectFromMorphismList( [ ApplyFunctor( AscToDescFunctor, delta2 ), Ap
<An object in Functors from integers into Descending filtered object category of Category of left
gap> SetIsAdditiveCategory( CategoryOfDescendingFilteredObjects( category ), true );
gap> cocomplex := AsCocomplex( cocomplex );

```

```

<An object in Cocomplex category of Descending filtered object category of Category of left pres
gap> s := SpectralSequenceEntryOfDescendingFilteredCocomplex( cocomplex, 0, -2, 1 );
<A morphism in Generalized morphism category of Category of left presentations of Q[x,y]>
gap> Display( UnderlyingMatrix( ApplyFunctor( LessGenFunctor, UnderlyingHonestObject( Source( s
(an empty 0 x 2 matrix)
gap> s := SpectralSequenceEntryOfDescendingFilteredCocomplex( cocomplex, 1, -2, 1 );
<A morphism in Generalized morphism category of Category of left presentations of Q[x,y]>
gap> Display( UnderlyingMatrix( ApplyFunctor( LessGenFunctor, UnderlyingHonestObject( Source( s
(an empty 0 x 2 matrix)
gap> s := SpectralSequenceEntryOfDescendingFilteredCocomplex( cocomplex, 2, -2, 1 );
<A morphism in Generalized morphism category of Category of left presentations of Q[x,y]>
gap> Display( UnderlyingMatrix( ApplyFunctor( LessGenFunctor, UnderlyingHonestObject( Source( s
-y,x
gap> s := SpectralSequenceEntryOfDescendingFilteredCocomplex( cocomplex, 3, -2, 1 );
<A morphism in Generalized morphism category of Category of left presentations of Q[x,y]>
gap> Display( UnderlyingMatrix( ApplyFunctor( LessGenFunctor, UnderlyingHonestObject( Source( s
(an empty 0 x 0 matrix)
gap> s := SpectralSequenceDifferentialOfDescendingFilteredCocomplex( cocomplex, 2, -2, 1 );
<A morphism in Category of left presentations of Q[x,y]>
gap> Display( UnderlyingMatrix( ApplyFunctor( LessGenFunctor, s ) ) );
x^2,
x*y

```

11.2 Monoidal Categories

Example

```

gap> ZZ := HomalgRingOfIntegers();
gap> M1 := AsLeftPresentation( HomalgMatrix( [ [ 2 ] ], 1, 1, ZZ ) );
<An object in Category of left presentations of Z>
gap> N1 := AsLeftPresentation( HomalgMatrix( [ [ 3 ] ], 1, 1, ZZ ) );
<An object in Category of left presentations of Z>
gap> T1 := TensorProductOnObjects( M1, N1 );
<An object in Category of left presentations of Z>
gap> Display( UnderlyingMatrix( T1 ) );
[ [ 3 ],
  [ 2 ] ]
gap> IsZeroForObjects( T1 );
true
gap> B1 := Braiding( DirectSum( M1, N1 ), DirectSum( M1, M1 ) );
<A morphism in Category of left presentations of Z>
gap> Display( UnderlyingMatrix( B1 ) );
[ [ 1, 0, 0, 0 ],
  [ 0, 0, 1, 0 ],
  [ 0, 1, 0, 0 ],
  [ 0, 0, 0, 1 ] ]
gap> IsWellDefined( B1 );
true
gap> U1 := TensorUnit( CapCategory( M1 ) );
<An object in Category of left presentations of Z>
gap> IntHom1 := InternalHomOnObjects( DirectSum( M1, U1 ), N1 );
<An object in Category of left presentations of Z>
gap> Display( UnderlyingMatrix( IntHom1 ) );

```



```

[ [ -2, -1 ],
  [  1, -1 ] ]
gap> generator_l1 := StandardGeneratorMorphism( IntHomL, 1 );
<A morphism in Category of left presentations of Z>
gap> morphism_l1 := LambdaElimination( DirectSum( M1, U1 ), N1, generator_l1 );
<A morphism in Category of left presentations of Z>
gap> Display( UnderlyingMatrix( morphism_l1 ) );
[ [ 0 ],
  [ 2 ] ]
gap> generator_l2 := StandardGeneratorMorphism( IntHomL, 2 );
<A morphism in Category of left presentations of Z>
gap> morphism_l2 := LambdaElimination( DirectSum( M1, U1 ), N1, generator_l2 );
<A morphism in Category of left presentations of Z>
gap> Display( UnderlyingMatrix( morphism_l2 ) );
[ [ 0 ],
  [ 2 ] ]
gap> IsEqualForMorphisms( LambdaIntroduction( morphism_l1 ), generator_l1 );
false
gap> IsCongruentForMorphisms( LambdaIntroduction( morphism_l1 ), generator_l1 );
true
gap> IsEqualForMorphisms( LambdaIntroduction( morphism_l2 ), generator_l2 );
false
gap> IsCongruentForMorphisms( LambdaIntroduction( morphism_l2 ), generator_l2 );
true
gap> Mr := AsRightPresentation( HomalgMatrix( [ [ 2 ] ], 1, 1, ZZ ) );
<An object in Category of right presentations of Z>
gap> Nr := AsRightPresentation( HomalgMatrix( [ [ 3 ] ], 1, 1, ZZ ) );
<An object in Category of right presentations of Z>
gap> Tr := TensorProductOnObjects( Mr, Nr );
<An object in Category of right presentations of Z>
gap> Display( UnderlyingMatrix( Tr ) );
[ [ 3, 2 ] ]
gap> IsZeroForObjects( Tr );
true
gap> Br := Braiding( DirectSum( Mr, Nr ), DirectSum( Mr, Mr ) );
<A morphism in Category of right presentations of Z>
gap> Display( UnderlyingMatrix( Br ) );
[ [ 1, 0, 0, 0 ],
  [ 0, 0, 1, 0 ],
  [ 0, 1, 0, 0 ],
  [ 0, 0, 0, 1 ] ]
gap> IsWellDefined( Br );
true
gap> Ur := TensorUnit( CapCategory( Mr ) );
<An object in Category of right presentations of Z>
gap> IntHomr := InternalHomOnObjects( DirectSum( Mr, Ur ), Nr );
<An object in Category of right presentations of Z>
gap> Display( UnderlyingMatrix( IntHomr ) );
[ [ -2,  1 ],
  [ -1, -1 ] ]
gap> generator_r1 := StandardGeneratorMorphism( IntHomr, 1 );
<A morphism in Category of right presentations of Z>
gap> morphism_r1 := LambdaElimination( DirectSum( Mr, Ur ), Nr, generator_r1 );

```

```

<A morphism in Category of right presentations of Z>
gap> Display( UnderlyingMatrix( morphism_r1 ) );
[[ 0, 2 ]]
gap> generator_r2 := StandardGeneratorMorphism( IntHom1, 2 );
<A morphism in Category of left presentations of Z>
gap> morphism_r2 := LambdaElimination( DirectSum( M1, U1 ), N1, generator_r2 );
<A morphism in Category of left presentations of Z>
gap> Display( UnderlyingMatrix( morphism_r2 ) );
[[ 0 ],
 [ 2 ]]
gap> IsEqualForMorphisms( LambdaIntroduction( morphism_r1 ), generator_r1 );
false
gap> IsCongruentForMorphisms( LambdaIntroduction( morphism_r1 ), generator_r1 );
true
gap> IsEqualForMorphisms( LambdaIntroduction( morphism_r2 ), generator_r2 );
false
gap> IsCongruentForMorphisms( LambdaIntroduction( morphism_r2 ), generator_r2 );
true

```

11.3 Generalized Morphisms Category

Example

```

gap> vecspaces := CreateCapCategory( "VectorSpacesForGeneralizedMorphismsTest" );
VectorSpacesForGeneralizedMorphismsTest
gap> ReadPackage( "CAP", "examples/testfiles/VectorSpacesAllMethods.gi" );
true
gap> LoadPackage( "GeneralizedMorphismsForCAP" );
true
gap> B := QVectorSpace( 2 );
<A rational vector space of dimension 2>
gap> C := QVectorSpace( 3 );
<A rational vector space of dimension 3>
gap> B_1 := QVectorSpace( 1 );
<A rational vector space of dimension 1>
gap> C_1 := QVectorSpace( 2 );
<A rational vector space of dimension 2>
gap> c1_source_aid := VectorSpaceMorphism( B_1, [[ 1, 0 ]], B );
A rational vector space homomorphism with matrix:
[[ 1, 0 ]]

gap> SetIsSubobject( c1_source_aid, true );
gap> c1_range_aid := VectorSpaceMorphism( C, [[ 1, 0 ], [ 0, 1 ], [ 0, 0 ]], C_1 );
A rational vector space homomorphism with matrix:
[[ 1, 0 ],
 [ 0, 1 ],
 [ 0, 0 ]]

gap> SetIsFactorobject( c1_range_aid, true );
gap> c1_associated := VectorSpaceMorphism( B_1, [[ 1, 1 ]], C_1 );
A rational vector space homomorphism with matrix:
[[ 1, 1 ]]

```

```

gap> c1 := GeneralizedMorphism( c1_source_aid, c1_associated, c1_range_aid );
<A morphism in Generalized morphism category of VectorSpacesForGeneralizedMorphismsTest>
gap> B_2 := QVectorSpace( 1 );
<A rational vector space of dimension 1>
gap> C_2 := QVectorSpace( 2 );
<A rational vector space of dimension 2>
gap> c2_source_aid := VectorSpaceMorphism( B_2, [ [ 2, 0 ] ], B );
A rational vector space homomorphism with matrix:
[ [ 2, 0 ] ]

gap> SetIsSubobject( c2_source_aid, true );
gap> c2_range_aid := VectorSpaceMorphism( C, [ [ 3, 0 ], [ 0, 3 ], [ 0, 0 ] ], C_2 );
A rational vector space homomorphism with matrix:
[ [ 3, 0 ],
  [ 0, 3 ],
  [ 0, 0 ] ]

gap> SetIsFactorobject( c2_range_aid, true );
gap> c2_associated := VectorSpaceMorphism( B_2, [ [ 6, 6 ] ], C_2 );
A rational vector space homomorphism with matrix:
[ [ 6, 6 ] ]

gap> c2 := GeneralizedMorphism( c2_source_aid, c2_associated, c2_range_aid );
<A morphism in Generalized morphism category of VectorSpacesForGeneralizedMorphismsTest>
gap> IsCongruentForMorphisms( c1, c2 );
true
gap> IsCongruentForMorphisms( c1, c1 );
true
gap> c3_associated := VectorSpaceMorphism( B_1, [ [ 2, 2 ] ], C_1 );
A rational vector space homomorphism with matrix:
[ [ 2, 2 ] ]

gap> c3 := GeneralizedMorphism( c1_source_aid, c3_associated, c1_range_aid );
<A morphism in Generalized morphism category of VectorSpacesForGeneralizedMorphismsTest>
gap> IsCongruentForMorphisms( c1, c3 );
false
gap> IsCongruentForMorphisms( c2, c3 );
false
gap> c1 + c2;
<A morphism in Generalized morphism category of VectorSpacesForGeneralizedMorphismsTest>
gap> Arrow( c1 + c2 );
A rational vector space homomorphism with matrix:
[ [ 12, 12 ] ]

```

First composition test:

Example

```

gap> vecspaces := CreateCapCategory( "VectorSpacesForGeneralizedMorphismsTest" );
VectorSpacesForGeneralizedMorphismsTest
gap> ReadPackage( "CAP", "examples/testfiles/VectorSpacesAllMethods.gi" );
true
gap> A := QVectorSpace( 1 );
<A rational vector space of dimension 1>

```

```

gap> B := QVectorSpace( 2 );
<A rational vector space of dimension 2>
gap> C := QVectorSpace( 3 );
<A rational vector space of dimension 3>
gap> phi_tilde_associated := VectorSpaceMorphism( A, [ [ 1, 2, 0 ] ], C );
A rational vector space homomorphism with matrix:
[ [ 1, 2, 0 ] ]

gap> phi_tilde_source_aid := VectorSpaceMorphism( A, [ [ 1, 2 ] ], B );
A rational vector space homomorphism with matrix:
[ [ 1, 2 ] ]

gap> phi_tilde := GeneralizedMorphismWithSourceAid( phi_tilde_source_aid, phi_tilde_associated )
<A morphism in Generalized morphism category of VectorSpacesForGeneralizedMorphismsTest>
gap> psi_tilde_associated := IdentityMorphism( B );
A rational vector space homomorphism with matrix:
[ [ 1, 0 ],
  [ 0, 1 ] ]

gap> psi_tilde_source_aid := VectorSpaceMorphism( B, [ [ 1, 0, 0 ], [ 0, 1, 0 ] ], C );
A rational vector space homomorphism with matrix:
[ [ 1, 0, 0 ],
  [ 0, 1, 0 ] ]

gap> psi_tilde := GeneralizedMorphismWithSourceAid( psi_tilde_source_aid, psi_tilde_associated )
<A morphism in Generalized morphism category of VectorSpacesForGeneralizedMorphismsTest>
gap> composition := PreCompose( phi_tilde, psi_tilde );
<A morphism in Generalized morphism category of VectorSpacesForGeneralizedMorphismsTest>
gap> Arrow( composition );
A rational vector space homomorphism with matrix:
[ [ 1/2, 1 ] ]

gap> SourceAid( composition );
A rational vector space homomorphism with matrix:
[ [ 1/2, 1 ] ]

gap> RangeAid( composition );
A rational vector space homomorphism with matrix:
[ [ 1, 0 ],
  [ 0, 1 ] ]

```

Second composition test

Example

```

gap> vecspaces := CreateCapCategory( "VectorSpacesForGeneralizedMorphismsTest" );
VectorSpacesForGeneralizedMorphismsTest
gap> ReadPackage( "CAP", "examples/testfiles/VectorSpacesAllMethods.gi" );
true
gap> A := QVectorSpace( 1 );
<A rational vector space of dimension 1>
gap> B := QVectorSpace( 2 );
<A rational vector space of dimension 2>
gap> C := QVectorSpace( 3 );
<A rational vector space of dimension 3>

```

```

gap> phi2_tilde_associated := VectorSpaceMorphism( A, [ [ 1, 5 ] ], B );
A rational vector space homomorphism with matrix:
[ [ 1, 5 ] ]

gap> phi2_tilde_range_aid := VectorSpaceMorphism( C, [ [ 1, 0 ], [ 0, 1 ], [ 1, 1 ] ], B );
A rational vector space homomorphism with matrix:
[ [ 1, 0 ],
  [ 0, 1 ],
  [ 1, 1 ] ]

gap> phi2_tilde := GeneralizedMorphismWithRangeAid( phi2_tilde_associated, phi2_tilde_range_aid
<A morphism in Generalized morphism category of VectorSpacesForGeneralizedMorphismsTest>
gap> psi2_tilde_associated := VectorSpaceMorphism( C, [ [ 1 ], [ 3 ], [ 4 ] ], A );
A rational vector space homomorphism with matrix:
[ [ 1 ],
  [ 3 ],
  [ 4 ] ]

gap> psi2_tilde_range_aid := VectorSpaceMorphism( B, [ [ 1 ], [ 1 ] ], A );
A rational vector space homomorphism with matrix:
[ [ 1 ],
  [ 1 ] ]

gap> psi2_tilde := GeneralizedMorphismWithRangeAid( psi2_tilde_associated, psi2_tilde_range_aid
<A morphism in Generalized morphism category of VectorSpacesForGeneralizedMorphismsTest>
gap> composition2 := PreCompose( phi2_tilde, psi2_tilde );
<A morphism in Generalized morphism category of VectorSpacesForGeneralizedMorphismsTest>
gap> Arrow( composition2 );
A rational vector space homomorphism with matrix:
[ [ 16 ] ]

gap> RangeAid( composition2 );
A rational vector space homomorphism with matrix:
[ [ 1 ],
  [ 1 ] ]

gap> SourceAid( composition2 );
A rational vector space homomorphism with matrix:
[ [ 1 ] ]

```

Third composition test

Example

```

gap> vecspaces := CreateCapCategory( "VectorSpacesForGeneralizedMorphismsTest" );
VectorSpacesForGeneralizedMorphismsTest
gap> ReadPackage( "CAP", "examples/testfiles/VectorSpacesAllMethods.gi" );
true
gap> A := QVectorSpace( 3 );
<A rational vector space of dimension 3>
gap> Asub := QVectorSpace( 2 );
<A rational vector space of dimension 2>
gap> B := QVectorSpace( 3 );
<A rational vector space of dimension 3>
gap> Bfac := QVectorSpace( 1 );

```

```

<A rational vector space of dimension 1>
gap> Bsub := QVectorSpace( 2 );
<A rational vector space of dimension 2>
gap> C := QVectorSpace( 3 );
<A rational vector space of dimension 3>
gap> Cfac := QVectorSpace( 1 );
<A rational vector space of dimension 1>
gap> Asub_into_A := VectorSpaceMorphism( Asub, [ [ 1, 0, 0 ], [ 0, 1, 0 ] ], A );
A rational vector space homomorphism with matrix:
[ [ 1, 0, 0 ],
  [ 0, 1, 0 ] ]

gap> Asub_to_Bfac := VectorSpaceMorphism( Asub, [ [ 1 ] ], [ 1 ] ], Bfac );
A rational vector space homomorphism with matrix:
[ [ 1 ],
  [ 1 ] ]

gap> B_onto_Bfac := VectorSpaceMorphism( B, [ [ 1 ], [ 1 ], [ 1 ] ], Bfac );
A rational vector space homomorphism with matrix:
[ [ 1 ],
  [ 1 ],
  [ 1 ] ]

gap> Bsub_into_B := VectorSpaceMorphism( Bsub, [ [ 2, 2, 0 ], [ 0, 2, 2 ] ], B );
A rational vector space homomorphism with matrix:
[ [ 2, 2, 0 ],
  [ 0, 2, 2 ] ]

gap> Bsub_to_Cfac := VectorSpaceMorphism( Bsub, [ [ 3 ] ], [ 0 ] ], Cfac );
A rational vector space homomorphism with matrix:
[ [ 3 ],
  [ 0 ] ]

gap> C_onto_Cfac := VectorSpaceMorphism( C, [ [ 1 ], [ 2 ], [ 3 ] ], Cfac );
A rational vector space homomorphism with matrix:
[ [ 1 ],
  [ 2 ],
  [ 3 ] ]

gap> generalized_morphism1 := GeneralizedMorphism( Asub_into_A, Asub_to_Bfac, B_onto_Bfac );
<A morphism in Generalized morphism category of VectorSpacesForGeneralizedMorphismsTest>
gap> generalized_morphism2 := GeneralizedMorphism( Bsub_into_B, Bsub_to_Cfac, C_onto_Cfac );
<A morphism in Generalized morphism category of VectorSpacesForGeneralizedMorphismsTest>
gap> IsWellDefined( generalized_morphism1 );
true
gap> IsWellDefined( generalized_morphism2 );
true
gap> p := PreCompose( generalized_morphism1, generalized_morphism2 );
<A morphism in Generalized morphism category of VectorSpacesForGeneralizedMorphismsTest>
gap> SourceAid( p );
A rational vector space homomorphism with matrix:
[ [ -1, 1, 0 ],
  [ 1, 0, 0 ] ]

```

```

gap> Arrow( p );
A rational vector space homomorphism with matrix:
(an empty 2 x 0 matrix)

gap> RangeAid( p );
A rational vector space homomorphism with matrix:
(an empty 3 x 0 matrix)
gap> A := QVectorSpace( 3 );
<A rational vector space of dimension 3>
gap> Asub := QVectorSpace( 2 );
<A rational vector space of dimension 2>
gap> B := QVectorSpace( 3 );
<A rational vector space of dimension 3>
gap> Bfac := QVectorSpace( 1 );
<A rational vector space of dimension 1>
gap> Bsub := QVectorSpace( 2 );
<A rational vector space of dimension 2>
gap> C := QVectorSpace( 3 );
<A rational vector space of dimension 3>
gap> Cfac := QVectorSpace( 2 );
<A rational vector space of dimension 2>
gap> Asub_into_A := VectorSpaceMorphism( Asub, [ [ 1, 0, 0 ], [ 0, 1, 0 ] ], A );
A rational vector space homomorphism with matrix:
[ [ 1, 0, 0 ],
  [ 0, 1, 0 ] ]

gap> Asub_to_Bfac := VectorSpaceMorphism( Asub, [ [ 1 ] ], [ 1 ] ], Bfac );
A rational vector space homomorphism with matrix:
[ [ 1 ],
  [ 1 ] ]

gap> B_onto_Bfac := VectorSpaceMorphism( B, [ [ 1 ], [ 1 ], [ 1 ] ], Bfac );
A rational vector space homomorphism with matrix:
[ [ 1 ],
  [ 1 ],
  [ 1 ] ]

gap> Bsub_into_B := VectorSpaceMorphism( Bsub, [ [ 2, 2, 0 ], [ 0, 2, 2 ] ], B );
A rational vector space homomorphism with matrix:
[ [ 2, 2, 0 ],
  [ 0, 2, 2 ] ]

gap> Bsub_to_Cfac := VectorSpaceMorphism( Bsub, [ [ 3, 3 ], [ 0, 0 ] ], Cfac );
A rational vector space homomorphism with matrix:
[ [ 3, 3 ],
  [ 0, 0 ] ]

gap> C_onto_Cfac := VectorSpaceMorphism( C, [ [ 1, 0 ], [ 0, 2 ], [ 3, 3 ] ], Cfac );
A rational vector space homomorphism with matrix:
[ [ 1, 0 ],
  [ 0, 2 ],
  [ 3, 3 ] ]

```

```

gap> generalized_morphism1 := GeneralizedMorphism( Asub_into_A, Asub_to_Bfac, B_onto_Bfac );
<A morphism in Generalized morphism category of VectorSpacesForGeneralizedMorphismsTest>
gap> generalized_morphism2 := GeneralizedMorphism( Bsub_into_B, Bsub_to_Cfac, C_onto_Cfac );
<A morphism in Generalized morphism category of VectorSpacesForGeneralizedMorphismsTest>
gap> IsWellDefined( generalized_morphism1 );
true
gap> IsWellDefined( generalized_morphism2 );
true
gap> p := PreCompose( generalized_morphism1, generalized_morphism2 );
<A morphism in Generalized morphism category of VectorSpacesForGeneralizedMorphismsTest>
gap> SourceAid( p );
A rational vector space homomorphism with matrix:
[ [ -1,  1,  0 ],
  [  1,  0,  0 ] ]

gap> Arrow( p );
A rational vector space homomorphism with matrix:
[ [ 0 ],
  [ 0 ] ]

gap> RangeAid( p );
A rational vector space homomorphism with matrix:
[ [ -1 ],
  [  2 ],
  [  0 ] ]

```

Honest representative test

Example

```

gap> vecspaces := CreateCapCategory( "VectorSpacesForGeneralizedMorphismsTest" );
VectorSpacesForGeneralizedMorphismsTest
gap> ReadPackage( "CAP", "examples/testfiles/VectorSpacesAllMethods.gi" );
true
gap> A := QVectorSpace( 1 );
<A rational vector space of dimension 1>
gap> B := QVectorSpace( 2 );
<A rational vector space of dimension 2>
gap> phi_tilde_source_aid := VectorSpaceMorphism( A, [ [ 2 ] ], A );
A rational vector space homomorphism with matrix:
[ [ 2 ] ]

gap> phi_tilde_associated := VectorSpaceMorphism( A, [ [ 1, 1 ] ], B );
A rational vector space homomorphism with matrix:
[ [ 1, 1 ] ]

gap> phi_tilde_range_aid := VectorSpaceMorphism( B, [ [ 1, 2 ], [ 3, 4 ] ], B );
A rational vector space homomorphism with matrix:
[ [ 1, 2 ],
  [ 3, 4 ] ]

gap> phi_tilde := GeneralizedMorphism( phi_tilde_source_aid, phi_tilde_associated, phi_tilde_range_aid );
<A morphism in Generalized morphism category of VectorSpacesForGeneralizedMorphismsTest>
gap> HonestRepresentative( phi_tilde );

```



```
A rational vector space homomorphism with matrix:
[ [ -1/4,  1/4 ] ]
```

```
gap> IsWellDefined( phi_tilde );
true
gap> IsWellDefined( psi_tilde );
true
```

11.4 IsWellDefined

Example

```
gap> vecspaces := CreateCapCategory( "VectorSpacesForIsWellDefinedTest" );
VectorSpacesForIsWellDefinedTest
gap> ReadPackage( "CAP", "examples/testfiles/VectorSpacesAllMethods.gi" );
true
gap> LoadPackage( "GeneralizedMorphismsForCAP" );
true
gap> A := QVectorSpace( 1 );
<A rational vector space of dimension 1>
gap> B := QVectorSpace( 2 );
<A rational vector space of dimension 2>
gap> alpha := VectorSpaceMorphism( A, [ [ 1, 2 ] ], B );
A rational vector space homomorphism with matrix:
[ [ 1, 2 ] ]

gap> g := GeneralizedMorphism( alpha, alpha, alpha );
<A morphism in Generalized morphism category of VectorSpacesForIsWellDefinedTest>
gap> IsWellDefined( alpha );
true
gap> IsWellDefined( g );
true
```

11.5 Kernel

Example

```
gap> vecspaces := CreateCapCategory( "VectorSpaces01" );
VectorSpaces01
gap> ReadPackage( "CAP", "examples/testfiles/VectorSpacesAddKernel01.gi" );
true
gap> V := QVectorSpace( 2 );
<A rational vector space of dimension 2>
gap> W := QVectorSpace( 3 );
<A rational vector space of dimension 3>
gap> alpha := VectorSpaceMorphism( V, [ [ 1, 1, 1 ], [ -1, -1, -1 ] ], W );
A rational vector space homomorphism with matrix:
[ [ 1, 1, 1 ],
  [ -1, -1, -1 ] ]

gap> k := KernelObject( alpha );
<A rational vector space of dimension 1>
gap> T := QVectorSpace( 2 );
<A rational vector space of dimension 2>
```

```

gap> tau := VectorSpaceMorphism( T, [ [ 2, 2 ], [ 2, 2 ] ], V );
A rational vector space homomorphism with matrix:
[ [ 2, 2 ],
  [ 2, 2 ] ]

gap> k_lift := KernelLift( alpha, tau );
A rational vector space homomorphism with matrix:
[ [ 2 ],
  [ 2 ] ]

gap> HasKernelEmbedding( alpha );
false
gap> KernelEmbedding( alpha );
A rational vector space homomorphism with matrix:
[ [ 1, 1 ] ]

```

Example

```

gap> vecspaces := CreateCapCategory( "VectorSpaces02" );
VectorSpaces02
gap> ReadPackage( "CAP", "examples/testfiles/VectorSpacesAddKernel02.gi" );
true
gap> V := QVectorSpace( 2 );
<A rational vector space of dimension 2>
gap> W := QVectorSpace( 3 );
<A rational vector space of dimension 3>
gap> alpha := VectorSpaceMorphism( V, [ [ 1, 1, 1 ], [ -1, -1, -1 ] ], W );
A rational vector space homomorphism with matrix:
[ [ 1, 1, 1 ],
  [ -1, -1, -1 ] ]

gap> k := KernelObject( alpha );
<A rational vector space of dimension 1>
gap> T := QVectorSpace( 2 );
<A rational vector space of dimension 2>
gap> tau := VectorSpaceMorphism( T, [ [ 2, 2 ], [ 2, 2 ] ], V );
A rational vector space homomorphism with matrix:
[ [ 2, 2 ],
  [ 2, 2 ] ]

gap> k_lift := KernelLift( alpha, tau );
A rational vector space homomorphism with matrix:
[ [ 2 ],
  [ 2 ] ]

gap> HasKernelEmbedding( alpha );
false

```

Example

```

gap> vecspaces := CreateCapCategory( "VectorSpaces03" );
VectorSpaces03
gap> ReadPackage( "CAP", "examples/testfiles/VectorSpacesAddKernel03.gi" );
true
gap> V := QVectorSpace( 2 );

```

```

<A rational vector space of dimension 2>
gap> W := QVectorSpace( 3 );
<A rational vector space of dimension 3>
gap> alpha := VectorSpaceMorphism( V, [ [ 1, 1, 1 ], [ -1, -1, -1 ] ], W );
A rational vector space homomorphism with matrix:
[ [ 1, 1, 1 ],
  [ -1, -1, -1 ] ]

gap> k := KernelObject( alpha );
<A rational vector space of dimension 1>
gap> k_emb := KernelEmbedding( alpha );
A rational vector space homomorphism with matrix:
[ [ 1, 1 ] ]

gap> IsIdenticalObj( Source( k_emb ), k );
true
gap> V := QVectorSpace( 2 );
<A rational vector space of dimension 2>
gap> W := QVectorSpace( 3 );
<A rational vector space of dimension 3>
gap> beta := VectorSpaceMorphism( V, [ [ 1, 1, 1 ], [ -1, -1, -1 ] ], W );
A rational vector space homomorphism with matrix:
[ [ 1, 1, 1 ],
  [ -1, -1, -1 ] ]

gap> k_emb := KernelEmbedding( beta );
A rational vector space homomorphism with matrix:
[ [ 1, 1 ] ]

gap> IsIdenticalObj( Source( k_emb ), KernelObject( beta ) );
true

```

11.6 FiberProduct

Example

```

gap> vecspaces := CreateCapCategory( "VectorSpacesForFiberProductTest" );
VectorSpacesForFiberProductTest
gap> ReadPackage( "CAP", "examples/testfiles/VectorSpacesAllMethods.gi" );
true
gap> A := QVectorSpace( 1 );
<A rational vector space of dimension 1>
gap> B := QVectorSpace( 2 );
<A rational vector space of dimension 2>
gap> C := QVectorSpace( 3 );
<A rational vector space of dimension 3>
gap> AtoC := VectorSpaceMorphism( A, [ [ 1, 2, 0 ] ], C );
A rational vector space homomorphism with matrix:
[ [ 1, 2, 0 ] ]

gap> BtoC := VectorSpaceMorphism( B, [ [ 1, 0, 0 ], [ 0, 1, 0 ] ], C );
A rational vector space homomorphism with matrix:
[ [ 1, 0, 0 ],
  [ 0, 1, 0 ] ]

```

```
[ 0, 1, 0 ] ]

gap> P := FiberProduct( AtoC, BtoC );
<A rational vector space of dimension 1>
gap> p1 := ProjectionInFactorOfFiberProduct( [ AtoC, BtoC ], 1 );
A rational vector space homomorphism with matrix:
[ [ 1/2 ] ]

gap> p2 := ProjectionInFactorOfFiberProduct( [ AtoC, BtoC ], 2 );
A rational vector space homomorphism with matrix:
[ [ 1/2, 1 ] ]
```

Index

- ActivateDerivationInfo, [99](#)
- Add
 - for IsStringMinHeap, IsString, IsInt, [106](#)
- AddAdditionForMorphisms
 - for IsCapCategory, IsFunction, [18](#)
- AddAdditiveInverseForMorphisms
 - for IsCapCategory, IsFunction, [18](#)
- AddAssociatorLeftToRightWithGivenTensorProducts
 - for IsCapCategory, IsFunction, [79](#)
- AddAssociatorRightToLeftWithGivenTensorProducts
 - for IsCapCategory, IsFunction, [78](#)
- AddAstrictionToCoimage
 - for IsCapCategory, IsFunction, [75](#)
- AddAstrictionToCoimageWithGivenCoimage
 - for IsCapCategory, IsFunction, [75](#)
- AddBraidingInverseWithGivenTensorProducts
 - for IsCapCategory, IsFunction, [84](#)
- AddBraidingWithGivenTensorProducts
 - for IsCapCategory, IsFunction, [84](#)
- AddCoastriictionToImage
 - for IsCapCategory, IsFunction, [70](#)
- AddCoastriictionToImageWithGivenImageObject
 - for IsCapCategory, IsFunction, [70](#)
- AddCoevaluationForDualWithGivenTensorProduct
 - for IsCapCategory, IsFunction, [90](#)
- AddCoevaluationMorphismWithGivenRange
 - for IsCapCategory, IsFunction, [86](#)
- AddCoimage
 - for IsCapCategory, IsFunction, [75](#)
- AddCoimageProjection
 - for IsCapCategory, IsFunction, [75](#)
- AddCoimageProjectionWithGivenCoimage
 - for IsCapCategory, IsFunction, [75](#)
- AddCokernelColift
 - for IsCapCategory, IsFunction, [35](#)
- AddCokernelColiftWithGivenCokernelObject
 - for IsCapCategory, IsFunction, [35](#)
- AddCokernelObject
 - for IsCapCategory, IsFunction, [35](#)
- AddCokernelObjectFunctorialWithGivenCokernelObjects
 - for IsCapCategory, IsFunction, [36](#)
- AddCokernelProjection
 - for IsCapCategory, IsFunction, [35](#)
- AddCokernelProjectionWithGivenCokernelObject
 - for IsCapCategory, IsFunction, [35](#)
- AddColift
 - for IsCapCategory, IsFunction, [23](#)
- AddColiftAlongEpimorphism
 - for IsCapCategory, IsFunction, [22](#)
- AddCoproduct
 - for IsCapCategory, IsFunction, [52](#)
- AddCoproductFunctorialWithGivenCoproducts
 - for IsCapCategory, IsFunction, [54](#)
- AddDerivation
 - for IsDerivedMethodGraph, IsDerivedMethod, [102](#)
 - for IsDerivedMethodGraph, IsFunction, IsDenseList, [102](#)
 - for IsDerivedMethodGraph, IsFunction, IsDenseList, IsObject, [102](#)
 - for IsDerivedMethodGraph, IsFunction, IsFunction, [102](#)
- AddDerivationPair
 - for IsDerivedMethodGraph, IsFunction, IsFunction, IsDenseList, IsDenseList, [102](#)
 - for IsDerivedMethodGraph, IsFunction, IsFunction, IsDenseList, IsDenseList, IsDenseList, [102](#)
 - for IsDerivedMethodGraph, IsFunction, Is-

- Function, IsDenseList, IsFunction, IsFunction, 103
- for IsDerivedMethodGraph, IsFunction, IsFunction, IsFunction, IsFunction, 103
- AddDerivationPairToCAP, 103
- AddDerivationToCAP, 103
- AddDirectProduct
 - for IsCapCategory, IsFunction, 55
- AddDirectProductFunctorialWithGivenDirectProducts
 - for IsCapCategory, IsFunction, 57
- AddDirectSum
 - for IsCapCategory, IsFunction, 50
- AddDirectSumCodiagonalDifference
 - for IsCapCategory, IsFunction, 64
- AddDirectSumDiagonalDifference
 - for IsCapCategory, IsFunction, 59
- AddDirectSumFunctorialWithGivenDirectSums
 - for IsCapCategory, IsFunction, 50
- AddDirectSumProjectionInPushout
 - for IsCapCategory, IsFunction, 65
- AddDualOnMorphismsWithGivenDuals
 - for IsCapCategory, IsFunction, 89
- AddDualOnObjects
 - for IsCapCategory, IsFunction, 89
- AddEvaluationForDualWithGivenTensorProduct
 - for IsCapCategory, IsFunction, 90
- AddEvaluationMorphismWithGivenSource
 - for IsCapCategory, IsFunction, 86
- AddFiberProduct
 - for IsCapCategory, IsFunction, 61
- AddFiberProductEmbeddingInDirectSum
 - for IsCapCategory, IsFunction, 59
- AddFiberProductFunctorialWithGivenFiberProducts
 - for IsCapCategory, IsFunction, 62
- AddHorizontalPostCompose
 - for IsCapCategory, IsFunction, 29
- AddHorizontalPreCompose
 - for IsCapCategory, IsFunction, 29
- AddIdentityMorphism
 - for IsCapCategory, IsFunction, 20
- AddIdentityTwoCell
 - for IsCapCategory, IsFunction, 28
- AddImageEmbedding
 - for IsCapCategory, IsFunction, 70
- AddImageEmbeddingWithGivenImageObject
 - for IsCapCategory, IsFunction, 70
- AddImageObject
 - for IsCapCategory, IsFunction, 70
- AddInitialObject
 - for IsCapCategory, IsFunction, 42
- AddInitialObjectFunctorial
 - for IsCapCategory, IsFunction, 43
- AddInjectionOfCofactorOfCoproduct
 - for IsCapCategory, IsFunction, 53
- AddInjectionOfCofactorOfCoproductWithGivenCoproduct
 - for IsCapCategory, IsFunction, 53
- AddInjectionOfCofactorOfDirectSum
 - for IsCapCategory, IsFunction, 48
- AddInjectionOfCofactorOfDirectSumWithGivenDirectSum
 - for IsCapCategory, IsFunction, 48
- AddInjectionOfCofactorOfPushout
 - for IsCapCategory, IsFunction, 66
- AddInjectionOfCofactorOfPushoutWithGivenPushout
 - for IsCapCategory, IsFunction, 67
- AddInternalHomOnMorphismsWithGivenInternalHoms
 - for IsCapCategory, IsFunction, 85
- AddInternalHomOnObjects
 - for IsCapCategory, IsFunction, 85
- AddInternalHomToTensorProductAdjunctionMap
 - for IsCapCategory, IsFunction, 87
- AddInverse
 - for IsCapCategory, IsFunction, 23
- AddInverseMorphismFromCoimageToImageWithGivenObjects
 - for IsCapCategory, IsFunction, 72
- AddIsAutomorphism
 - for IsCapCategory, IsFunction, 16
- AddIsCodominating
 - for IsCapCategory, IsFunction, 20
- AddIsCongruentForMorphisms
 - for IsCapCategory, IsFunction, 17
- AddIsDominating
 - for IsCapCategory, IsFunction, 20
- AddIsEndomorphism
 - for IsCapCategory, IsFunction, 16

- AddIsEpimorphism
 - for IsCapCategory, IsFunction, [15](#)
- AddIsEqualAsFactorobjects
 - for IsCapCategory, IsFunction, [19](#)
- AddIsEqualAsSubobjects
 - for IsCapCategory, IsFunction, [19](#)
- AddIsEqualForCacheForMorphisms
 - for IsCapCategory, IsFunction, [24](#)
- AddIsEqualForCacheForObjects
 - for IsCapCategory, IsFunction, [27](#)
- AddIsEqualForMorphisms
 - for IsCapCategory, IsFunction, [17](#)
- AddIsEqualForMorphismsOnMor
 - for IsCapCategory, IsFunction, [17](#)
- AddIsEqualForObjects
 - for IsCapCategory, IsFunction, [25](#)
- AddIsIdempotent
 - for IsCapCategory, IsFunction, [15](#)
- AddIsIdenticalToIdentityMorphism
 - for IsCapCategory, IsFunction, [16](#)
- AddIsIdenticalToZeroMorphism
 - for IsCapCategory, IsFunction, [16](#)
- AddIsInitial
 - for IsCapCategory, IsFunction, [26](#)
- AddIsInjective
 - for IsCapCategory, IsFunction, [26](#)
- AddIsIsomorphism
 - for IsCapCategory, IsFunction, [15](#)
- AddIsMonomorphism
 - for IsCapCategory, IsFunction, [14](#)
- AddIsomorphismFromCoimageToCokernelOf-
 - Kernel
 - for IsCapCategory, IsFunction, [73](#)
- AddIsomorphismFromCokernelOfDiagonal-
 - DifferenceToPushout
 - for IsCapCategory, IsFunction, [64](#)
- AddIsomorphismFromCokernelOfKernelTo-
 - Coimage
 - for IsCapCategory, IsFunction, [73](#)
- AddIsomorphismFromCoproductToDirectSum
 - for IsCapCategory, IsFunction, [50](#)
- AddIsomorphismFromDirectProductTo-
 - DirectSum
 - for IsCapCategory, IsFunction, [49](#)
- AddIsomorphismFromDirectSumToCoproduct
 - for IsCapCategory, IsFunction, [49](#)
- AddIsomorphismFromDirectSumToDirect-
 - Product
 - for IsCapCategory, IsFunction, [49](#)
- AddIsomorphismFromDualToInternalHom
 - for IsCapCategory, IsFunction, [95](#)
- AddIsomorphismFromFiberProductTo-
 - KernelOfDiagonalDifference
 - for IsCapCategory, IsFunction, [58](#)
- AddIsomorphismFromImageObjectToKernel-
 - OfCokernel
 - for IsCapCategory, IsFunction, [68](#)
- AddIsomorphismFromInitialObjectToZero-
 - Object
 - for IsCapCategory, IsFunction, [39](#)
- AddIsomorphismFromInternalHomToDual
 - for IsCapCategory, IsFunction, [95](#)
- AddIsomorphismFromInternalHomToObject-
 - WithGivenInternalHom
 - for IsCapCategory, IsFunction, [97](#)
- AddIsomorphismFromInternalHomToTensor-
 - Product
 - for IsCapCategory, IsFunction, [94](#)
- AddIsomorphismFromKernelOfCokernelTo-
 - ImageObject
 - for IsCapCategory, IsFunction, [69](#)
- AddIsomorphismFromKernelOfDiagonal-
 - DifferenceToFiberProduct
 - for IsCapCategory, IsFunction, [58](#)
- AddIsomorphismFromObjectToInternalHom-
 - WithGivenInternalHom
 - for IsCapCategory, IsFunction, [96](#)
- AddIsomorphismFromPushoutToCokernelOf-
 - DiagonalDifference
 - for IsCapCategory, IsFunction, [63](#)
- AddIsomorphismFromTensorProductTo-
 - InternalHom
 - for IsCapCategory, IsFunction, [93](#)
- AddIsomorphismFromTerminalObjectTo-
 - ZeroObject
 - for IsCapCategory, IsFunction, [39](#)
- AddIsomorphismFromZeroObjectToInitial-
 - Object
 - for IsCapCategory, IsFunction, [39](#)
- AddIsomorphismFromZeroObjectTo-
 - TerminalObject
 - for IsCapCategory, IsFunction, [39](#)
- AddIsOne

- for IsCapCategory, IsFunction, 15
- AddIsProjective
 - for IsCapCategory, IsFunction, 26
- AddIsSplitEpimorphism
 - for IsCapCategory, IsFunction, 15
- AddIsSplitMonomorphism
 - for IsCapCategory, IsFunction, 15
- AddIsTerminal
 - for IsCapCategory, IsFunction, 26
- AddIsWellDefinedForMorphisms
 - for IsCapCategory, IsFunction, 21
- AddIsWellDefinedForObjects
 - for IsCapCategory, IsFunction, 27
- AddIsWellDefinedForTwoCells
 - for IsCapCategory, IsFunction, 30
- AddIsZeroForMorphisms
 - for IsCapCategory, IsFunction, 18
- AddIsZeroForObjects
 - for IsCapCategory, IsFunction, 26
- AdditionForMorphisms
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, 18
- AdditiveInverseForMorphisms
 - for IsCapCategoryMorphism, 18
- AddKernelEmbedding
 - for IsCapCategory, IsFunction, 32
- AddKernelEmbeddingWithGivenKernelObject
 - for IsCapCategory, IsFunction, 32
- AddKernelLift
 - for IsCapCategory, IsFunction, 32
- AddKernelLiftWithGivenKernelObject
 - for IsCapCategory, IsFunction, 33
- AddKernelObject
 - for IsCapCategory, IsFunction, 32
- AddKernelObjectFunctorialWithGivenKernelObjects
 - for IsCapCategory, IsFunction, 33
- AddLambdaElimination
 - for IsCapCategory, IsFunction, 96
- AddLambdaIntroduction
 - for IsCapCategory, IsFunction, 96
- AddLeftDistributivityExpandingWithGivenObjects
 - for IsCapCategory, IsFunction, 82
- AddLeftDistributivityFactoringWithGivenObjects
 - for IsCapCategory, IsFunction, 82
- AddLeftUnitorInverseWithGivenTensorProduct
 - for IsCapCategory, IsFunction, 80
- AddLeftUnitorWithGivenTensorProduct
 - for IsCapCategory, IsFunction, 80
- AddLift
 - for IsCapCategory, IsFunction, 23
- AddLiftAlongMonomorphism
 - for IsCapCategory, IsFunction, 22
- AddMonoidalPostComposeMorphismWithGivenObjects
 - for IsCapCategory, IsFunction, 88
- AddMonoidalPreComposeMorphismWithGivenObjects
 - for IsCapCategory, IsFunction, 88
- AddMorphismFromBidualWithGivenBidual
 - for IsCapCategory, IsFunction, 98
- AddMorphismFromCoimageToImageWithGivenObjects
 - for IsCapCategory, IsFunction, 72
- AddMorphismFromInternalHomToTensorProductWithGivenObjects
 - for IsCapCategory, IsFunction, 93
- AddMorphismFromTensorProductToInternalHomWithGivenObjects
 - for IsCapCategory, IsFunction, 92
- AddMorphismFunction
 - for IsCapFunctor, IsFunction, 11
- AddMorphismToBidualWithGivenBidual
 - for IsCapCategory, IsFunction, 91
- AddNaturalTransformationFunction
 - for IsCapNaturalTransformation, IsFunction, 13
- AddObjectFunction
 - for IsCapFunctor, IsFunction, 11
- AddOperationsToDerivationGraph
 - for IsDerivedMethodGraph, IsDenseList, 102
- AddPostCompose
 - for IsCapCategory, IsFunction, 21
- AddPreCompose
 - for IsCapCategory, IsFunction, 21
- AddPrimitiveOperation
 - for IsOperationWeightList, IsString, IsInt, 105
- AddProjectionInFactorOfDirectProduct

- for IsCapCategory, IsFunction, [56](#)
- AddProjectionInFactorOfDirectProduct-
WithGivenDirectProduct
for IsCapCategory, IsFunction, [56](#)
- AddProjectionInFactorOfDirectSum
for IsCapCategory, IsFunction, [48](#)
- AddProjectionInFactorOfDirectSumWith-
GivenDirectSum
for IsCapCategory, IsFunction, [48](#)
- AddProjectionInFactorOfFiberProduct
for IsCapCategory, IsFunction, [61](#)
- AddProjectionInFactorOfFiberProduct-
WithGivenFiberProduct
for IsCapCategory, IsFunction, [61](#)
- AddPushout
for IsCapCategory, IsFunction, [66](#)
- AddPushoutFunctorialWithGivenPushouts
for IsCapCategory, IsFunction, [67](#)
- AddRankMorphism
for IsCapCategory, IsFunction, [94](#)
- AddRightDistributivityExpandingWith-
GivenObjects
for IsCapCategory, IsFunction, [83](#)
- AddRightDistributivityFactoringWith-
GivenObjects
for IsCapCategory, IsFunction, [83](#)
- AddRightUnitorInverseWithGivenTensor-
Product
for IsCapCategory, IsFunction, [81](#)
- AddRightUnitorWithGivenTensorProduct
for IsCapCategory, IsFunction, [81](#)
- AddTensorProductDualityCompatibility-
MorphismWithGivenObjects
for IsCapCategory, IsFunction, [92](#)
- AddTensorProductInternalHom-
CompatibilityMorphismInverse-
WithGivenObjects
for IsCapCategory, IsFunction, [98](#)
- AddTensorProductInternalHom-
CompatibilityMorphismWith-
GivenObjects
for IsCapCategory, IsFunction, [91](#)
- AddTensorProductOnMorphismsWithGiven-
TensorProducts
for IsCapCategory, IsFunction, [78](#)
- AddTensorProductOnObjects
for IsCapCategory, IsFunction, [77](#)
- AddTensorProductToInternalHom-
AdjunctionMap
for IsCapCategory, IsFunction, [87](#)
- AddTensorUnit
for IsCapCategory, IsFunction, [79](#)
- AddTerminalObject
for IsCapCategory, IsFunction, [41](#)
- AddTerminalObjectFunctorial
for IsCapCategory, IsFunction, [41](#)
- AddTraceMap
for IsCapCategory, IsFunction, [94](#)
- AddUniversalMorphismFromCoproduct
for IsCapCategory, IsFunction, [53](#)
- AddUniversalMorphismFromCoproductWith-
GivenCoproduct
for IsCapCategory, IsFunction, [53](#)
- AddUniversalMorphismFromDirectSum
for IsCapCategory, IsFunction, [49](#)
- AddUniversalMorphismFromDirectSumWith-
GivenDirectSum
for IsCapCategory, IsFunction, [49](#)
- AddUniversalMorphismFromImage
for IsCapCategory, IsFunction, [71](#)
- AddUniversalMorphismFromImageWith-
GivenImageObject
for IsCapCategory, IsFunction, [71](#)
- AddUniversalMorphismFromInitialObject
for IsCapCategory, IsFunction, [43](#)
- AddUniversalMorphismFromInitialObject-
WithGivenInitialObject
for IsCapCategory, IsFunction, [43](#)
- AddUniversalMorphismFromPushout
for IsCapCategory, IsFunction, [67](#)
- AddUniversalMorphismFromPushoutWith-
GivenPushout
for IsCapCategory, IsFunction, [67](#)
- AddUniversalMorphismFromZeroObject
for IsCapCategory, IsFunction, [38](#)
- AddUniversalMorphismFromZeroObject-
WithGivenZeroObject
for IsCapCategory, IsFunction, [39](#)
- AddUniversalMorphismIntoCoimage
for IsCapCategory, IsFunction, [75](#)
- AddUniversalMorphismIntoCoimageWith-
GivenCoimage
for IsCapCategory, IsFunction, [75](#)
- AddUniversalMorphismIntoDirectProduct

- for IsCapCategory, IsFunction, 56
- AddUniversalMorphismIntoDirectProduct-
WithGivenDirectProduct
for IsCapCategory, IsFunction, 56
- AddUniversalMorphismIntoDirectSum
for IsCapCategory, IsFunction, 48
- AddUniversalMorphismIntoDirectSumWith-
GivenDirectSum
for IsCapCategory, IsFunction, 48
- AddUniversalMorphismIntoFiberProduct
for IsCapCategory, IsFunction, 61
- AddUniversalMorphismIntoFiberProduct-
WithGivenFiberProduct
for IsCapCategory, IsFunction, 62
- AddUniversalMorphismIntoTerminalObject
for IsCapCategory, IsFunction, 41
- AddUniversalMorphismIntoTerminal-
ObjectWithGivenTerminalObject
for IsCapCategory, IsFunction, 41
- AddUniversalMorphismIntoZeroObject
for IsCapCategory, IsFunction, 38
- AddUniversalMorphismIntoZeroObject-
WithGivenZeroObject
for IsCapCategory, IsFunction, 38
- AddUniversalPropertyOfDual
for IsCapCategory, IsFunction, 95
- AddVerticalPostCompose
for IsCapCategory, IsFunction, 29
- AddVerticalPreCompose
for IsCapCategory, IsFunction, 29
- AddWithGivenDerivationPairToCAP, 103
- AddZeroMorphism
for IsCapCategory, IsFunction, 19
- AddZeroObject
for IsCapCategory, IsFunction, 38
- AddZeroObjectFunctorial
for IsCapCategory, IsFunction, 40
- ApplyFunctor, 11
- ApplyNaturalTransformation, 13
- AsCapCategory
for IsCapCategoryAsCatObject, 10
- AsCatObject
for IsCapCategory, 10
- AssociatorLeftToRight
for IsCapCategoryObject, IsCapCategory-
Object, IsCapCategoryObject, 79
- AssociatorLeftToRightWithGivenTensor-
Products
for IsCapCategoryObject, IsCapCategory-
Object, IsCapCategoryObject, IsCap-
CategoryObject, IsCapCategoryObject, 79
- AssociatorRightToLeft
for IsCapCategoryObject, IsCapCategory-
Object, IsCapCategoryObject, 78
- AssociatorRightToLeftWithGivenTensor-
Products
for IsCapCategoryObject, IsCapCategory-
Object, IsCapCategoryObject, IsCap-
CategoryObject, IsCapCategoryObject, 78
- AstrictionToCoimage
for IsCapCategoryMorphism, 74
for IsCapCategoryObject, 74
- AstrictionToCoimageWithGivenCoimage
for IsCapCategoryMorphism, IsCapCatego-
ryObject, 74
- Braiding
for IsCapCategoryObject, IsCapCategory-
Object, 84
- BraidingInverse
for IsCapCategoryObject, IsCapCategory-
Object, 84
- BraidingInverseWithGivenTensorProducts
for IsCapCategoryObject, IsCapCategory-
Object, IsCapCategoryObject, IsCap-
CategoryObject, 84
- BraidingWithGivenTensorProducts
for IsCapCategoryObject, IsCapCategory-
Object, IsCapCategoryObject, IsCap-
CategoryObject, 84
- CanCompute
for IsCapCategory, IsString, 7
- CapCat, 9
- CapCategory
for IsCapCategoryMorphism, 14
for IsCapCategoryObject, 25
- CapCategorySwitchLogicOff, 7
- CapCategorySwitchLogicOn, 7
- CapFunctor

- for IsString, IsCapCategory, IsCapCategory, 10
- for IsString, IsCapCategory, IsCapCategoryAsCatObject, 10
- for IsString, IsCapCategoryAsCatObject, IsCapCategory, 10
- for IsString, IsCapCategoryAsCatObject, IsCapCategoryAsCatObject, 10
- for IsString, IsList, IsCapCategory, 10
- for IsString, IsList, IsCapCategoryAsCatObject, 10
- CAP_INTERNAL_FIND_APPEARANCE_OF_SYMBOL_IN_FUNCTION, 114
- CAP_INTERNAL_MERGE_FILTER_LISTS, 114
- CAP_INTERNAL_MERGE_PRECONDITIONS_LIST, 115
- CAP_INTERNAL_REPLACE_STRINGS_WITH_FILTERS, 114
- CAP_INTERNAL_RETURN_OPTION_OR_DEFAULT, 114
- CategoryFilter
 - for IsCapCategory, 6
 - for IsDerivedMethod, 100
- CategoryOfOperationWeightList
 - for IsOperationWeightList, 104
- CellFilter
 - for IsCapCategory, 6
- CheckConstructivenessOfCategory
 - for IsCapCategory, IsString, 7
- CostrictionToImage
 - for IsCapCategoryMorphism, 69
- CostrictionToImageWithGivenImageObject
 - for IsCapCategoryMorphism, IsCapCategoryObject, 69
- CoevaluationForDual
 - for IsCapCategoryObject, 90
- CoevaluationForDualWithGivenTensorProduct
 - for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, 90
- CoevaluationMorphism
 - for IsCapCategoryObject, IsCapCategoryObject, 86
- CoevaluationMorphismWithGivenRange
 - for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, 86
- Coimage
 - for IsCapCategoryMorphism, 73
- CoimageProjection
 - for IsCapCategoryMorphism, 73
 - for IsCapCategoryObject, 73
- CoimageProjectionWithGivenCoimage
 - for IsCapCategoryMorphism, IsCapCategoryObject, 74
- CokernelColift
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, 34
- CokernelColiftWithGivenCokernelObject
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject, 34
- CokernelObject
 - for IsCapCategoryMorphism, 34
- CokernelObjectFunctorial
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, 35
 - for IsList, 35
- CokernelObjectFunctorialWithGivenCokernelObjects
 - for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject, 36
- CokernelProjection
 - for IsCapCategoryMorphism, 34
- CokernelProjectionWithGivenCokernelObject
 - for IsCapCategoryMorphism, IsCapCategoryObject, 34
- Colift
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, 23
- ColiftAlongEpimorphism
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, 22
- Contains
 - for IsStringMinHeap, IsString, 107
- Coproduct
 - for IsCapCategoryObject, IsCapCategoryObject, 51
 - for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, 51
 - for IsList, 51

- CoproductFunctorial
 - for IsList, [53](#)
- CoproductFunctorialWithGivenCoproducts
 - for IsCapCategoryObject, IsList, IsCapCategoryObject, [53](#)
- CoproductOp
 - for IsList, IsCapCategoryObject, [51](#)
- CreateCapCategory, [6](#)
 - for IsString, [6](#)
- CurrentOperationWeight
 - for IsOperationWeightList, IsString, [104](#)
- DeactivateDerivationInfo, [99](#)
- DeclareAttributeWithToDoForIsWell-Defined, [114](#)
- DeclareFamilyProperty, [114](#)
- DecreaseKey
 - for IsStringMinHeap, IsString, IsInt, [106](#)
- DerivationFunctionsWithExtraFilters
 - for IsDerivedMethod, [100](#)
- DerivationGraph
 - for IsOperationWeightList, [104](#)
- DerivationInfo, [99](#)
- DerivationName
 - for IsDerivedMethod, [100](#)
- DerivationOfOperation
 - for IsOperationWeightList, IsString, [105](#)
- DerivationResultWeight
 - for IsDerivedMethod, IsDenseList, [101](#)
- DerivationsOfOperation
 - for IsDerivedMethodGraph, IsString, [103](#)
- DerivationsUsingOperation
 - for IsDerivedMethodGraph, IsString, [103](#)
- DerivationWeight
 - for IsDerivedMethod, [100](#)
- DirectProductFunctorial
 - for IsList, [56](#)
- DirectProductFunctorialWithGiven-DirectProducts
 - for IsCapCategoryObject, IsList, IsCapCategoryObject, [56](#)
- DirectProductOp
 - for IsList, IsCapCategoryObject, [54](#)
- DirectSumCodiagonalDifference
 - for IsList, [64](#)
- DirectSumCodiagonalDifferenceOp
 - for IsList, IsCapCategoryMorphism, [64](#)
- DirectSumDiagonalDifference
 - for IsList, [58](#)
- DirectSumDiagonalDifferenceOp
 - for IsList, IsCapCategoryMorphism, [58](#)
- DirectSumFunctorial
 - for IsList, [50](#)
- DirectSumFunctorialWithGivenDirectSums
 - for IsCapCategoryObject, IsList, IsCapCategoryObject, [50](#)
- DirectSumOp
 - for IsList, IsCapCategoryObject, [44](#)
- DirectSumProjectionInPushout
 - for IsList, [64](#)
- DirectSumProjectionInPushoutOp
 - for IsList, IsCapCategoryMorphism, [64](#)
- DisableBasicOperationTypeCheck, [8](#)
- DualOnMorphisms
 - for IsCapCategoryMorphism, [89](#)
- DualOnMorphismsWithGivenDuals
 - for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryObject, [89](#)
- DualOnObjects
 - for IsCapCategoryObject, [89](#)
- EnableBasicOperationTypeCheck, [8](#)
- EvaluationForDual
 - for IsCapCategoryObject, [89](#)
- EvaluationForDualWithGivenTensor-Product
 - for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, [89](#)
- EvaluationMorphism
 - for IsCapCategoryObject, IsCapCategoryObject, [86](#)
- EvaluationMorphismWithGivenSource
 - for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, [86](#)
- ExtractMin
 - for IsStringMinHeap, [106](#)
- FiberProduct, [59](#)
- FiberProductEmbeddingInDirectSum
 - for IsList, [59](#)
- FiberProductEmbeddingInDirectSumOp
 - for IsList, IsCapCategoryMorphism, [59](#)
- FiberProductFunctorial
 - for IsList, [62](#)

- FiberProductFunctorialWithGivenFiberProducts
 - for IsCapCategoryObject, IsList, IsCapCategoryObject, 62
- FiberProductOp
 - for IsList, IsCapCategoryMorphism, 60
- FunctorCanonicalizeZeroMorphisms
 - for IsCapCategory, 12
- FunctorCanonicalizeZeroObjects
 - for IsCapCategory, 12
- FunctorMorphismOperation
 - for IsCapFunctor, 11
- FunctorObjectOperation
 - for IsCapFunctor, 11
- Heapify
 - for IsStringMinHeap, IsPosInt, 107
- HeapSize
 - for IsStringMinHeap, 107
- HorizontalPostCompose
 - for IsCapCategoryTwoCell, IsCapCategoryTwoCell, 29
- HorizontalPreCompose
 - for IsCapCategoryTwoCell, IsCapCategoryTwoCell, 28
- HorizontalPreComposeFunctorWithNaturalTransformation
 - for IsCapFunctor, IsCapNaturalTransformation, 13
- HorizontalPreComposeNaturalTransformationWithFunctor
 - for IsCapNaturalTransformation, IsCapFunctor, 13
- IdentityFunctor
 - for IsCapCategory, 11
- IdentityMorphism
 - for IsCapCategoryObject, 20
- IdentityTwoCell
 - for IsCapCategoryMorphism, 28
- ImageEmbedding
 - for IsCapCategoryMorphism, 69
- ImageEmbeddingWithGivenImageObject
 - for IsCapCategoryMorphism, IsCapCategoryObject, 69
- ImageObject
 - for IsCapCategoryMorphism, 69
- InitialObject
 - for IsCapCategory, 42
 - for IsCapCategoryCell, 42
- InitialObjectFunctorial
 - for IsCapCategory, 43
- InjectionOfCofactorOfCoproduct
 - for IsList, IsInt, 51
- InjectionOfCofactorOfCoproductOp
 - for IsList, IsInt, IsCapCategoryObject, 51
- InjectionOfCofactorOfCoproductWithGivenCoproduct
 - for IsList, IsInt, IsCapCategoryObject, 52
- InjectionOfCofactorOfDirectSum
 - for IsList, IsInt, 44
- InjectionOfCofactorOfDirectSumOp
 - for IsList, IsInt, IsCapCategoryObject, 44
- InjectionOfCofactorOfDirectSumWithGivenDirectSum
 - for IsList, IsInt, IsCapCategoryObject, 45
- InjectionOfCofactorOfPushout
 - for IsList, IsInt, 65
- InjectionOfCofactorOfPushoutOp
 - for IsList, IsInt, IsCapCategoryMorphism, 65
- InjectionOfCofactorOfPushoutWithGivenPushout
 - for IsList, IsInt, IsCapCategoryObject, 65
- InstallDerivationForCategory
 - for IsDerivedMethod, IsPosInt, IsCapCategory, 101
- InstallDerivationsUsingOperation
 - for IsOperationWeightList, IsString, 105
- InstallFunctor
 - for IsCapFunctor, IsString, 11
- InstallMethodWithToDoForIsWellDefined,
 - 113
- InstallNaturalTransformation
 - for IsCapNaturalTransformation, IsString, 13
- InstallSetWithToDoForIsWellDefined
 - for IsObject, IsString, IsList, 114
- InternalHomOnMorphisms
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, 85
- InternalHomOnMorphismsWithGivenInternalHoms

- for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject, 85
- InternalHomOnObjects
 - for IsCapCategoryObject, IsCapCategoryObject, 85
- InternalHomToTensorProductAdjunctionMap
 - for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryMorphism, 87
- InverseMorphismFromCoimageToImage
 - for IsCapCategoryMorphism, 72
- InverseMorphismFromCoimageToImageWithGivenObjects
 - for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryObject, 72
- IsApplicableToCategory
 - for IsDerivedMethod, IsCapCategory, 100
- IsCapCategory
 - for IsObject, 5
- IsCapCategoryAsCatObject
 - for IsCapCategoryObject, 9
- IsCapCategoryCell
 - for IsObject, 5
- IsCapCategoryMorphism
 - for IsCapCategoryCell, 6
- IsCapCategoryObject
 - for IsCapCategoryCell, 5
- IsCapCategoryTwoCell
 - for IsCapCategoryCell, 6
- IsCapFunctor
 - for IsCapCategoryMorphism, 9
- IsCapNaturalTransformation
 - for IsCapCategoryTwoCell, 9
- IsCodominating
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, 20
- IsCongruentForMorphisms
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, 17
- IsDerivedMethod
 - for IsObject, 99
- IsDerivedMethodGraph
 - for IsObject, 101
- IsDominating
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, 19
- IsEmptyHeap
 - for IsStringMinHeap, 106
- IsEqualAsFactorobjects
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, 19
- IsEqualAsSubobjects
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, 19
- IsEqualForCacheForMorphisms
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, 24
- IsEqualForCacheForObjects
 - for IsCapCategoryObject, IsCapCategoryObject, 27
- IsEqualForMorphisms
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, 17
- IsEqualForMorphismsOnMor
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, 17
- IsEqualForObjects
 - for IsCapCategoryObject, IsCapCategoryObject, 25
- IsIdenticalToIdentityMorphism
 - for IsCapCategoryMorphism, 16
- IsIdenticalToZeroMorphism
 - for IsCapCategoryMorphism, 16
- IsomorphismFromCoimageToCokernelOfKernel
 - for IsCapCategoryMorphism, 72
- IsomorphismFromCokernelOfDiagonalDifferenceToPushout
 - for IsList, 63
- IsomorphismFromCokernelOfDiagonalDifferenceToPushoutOp
 - for IsList, IsCapCategoryMorphism, 63
- IsomorphismFromCokernelOfKernelToCoimage
 - for IsCapCategoryMorphism, 73
- IsomorphismFromCoproductToDirectSum
 - for IsList, 47
- IsomorphismFromCoproductToDirectSumOp
 - for IsList, IsCapCategoryObject, 47
- IsomorphismFromDirectProductToDirectSum
 - for IsList, 46

- IsomorphismFromDirectProductToDirectSumOp
 - for IsList, IsCapCategoryObject, 46
- IsomorphismFromDirectSumToCoproduct
 - for IsList, 47
- IsomorphismFromDirectSumToCoproductOp
 - for IsList, IsCapCategoryObject, 47
- IsomorphismFromDirectSumToDirectProduct
 - for IsList, 46
- IsomorphismFromDirectSumToDirectProductOp
 - for IsList, IsCapCategoryObject, 46
- IsomorphismFromDualToInternalHom
 - for IsCapCategoryObject, 94
- IsomorphismFromFiberProductToKernelOfDiagonalDifference
 - for IsList, 57
- IsomorphismFromFiberProductToKernelOfDiagonalDifferenceOp
 - for IsList, IsCapCategoryMorphism, 57
- IsomorphismFromImageObjectToKernelOfCokernel
 - for IsCapCategoryMorphism, 68
- IsomorphismFromInitialObjectToZeroObject
 - for IsCapCategory, 38
- IsomorphismFromInternalHomToDual
 - for IsCapCategoryObject, 95
- IsomorphismFromInternalHomToObject
 - for IsCapCategoryObject, 97
- IsomorphismFromInternalHomToObjectWithGivenInternalHom
 - for IsCapCategoryObject, IsCapCategoryObject, 97
- IsomorphismFromInternalHomToTensorProduct
 - for IsCapCategoryObject, IsCapCategoryObject, 94
- IsomorphismFromKernelOfCokernelToImageObject
 - for IsCapCategoryMorphism, 68
- IsomorphismFromKernelOfDiagonalDifferenceToFiberProduct
 - for IsList, 58
- IsomorphismFromKernelOfDiagonalDifferenceToFiberProductOp
 - for IsList, IsCapCategoryMorphism, 58
- IsomorphismFromObjectToInternalHom
 - for IsCapCategoryObject, 96
- IsomorphismFromObjectToInternalHomWithGivenInternalHom
 - for IsCapCategoryObject, IsCapCategoryObject, 96
- IsomorphismFromPushoutToCokernelOfDiagonalDifference
 - for IsList, 63
- IsomorphismFromPushoutToCokernelOfDiagonalDifferenceOp
 - for IsList, IsCapCategoryMorphism, 63
- IsomorphismFromTensorProductToInternalHom
 - for IsCapCategoryObject, IsCapCategoryObject, 93
- IsomorphismFromTerminalObjectToZeroObject
 - for IsCapCategory, 38
- IsomorphismFromZeroObjectToInitialObject
 - for IsCapCategory, 37
- IsomorphismFromZeroObjectToTerminalObject
 - for IsCapCategory, 38
- IsOperationWeightList
 - for IsObject, 104
- IsStringMinHeap
 - for IsObject, 106
- IsWellDefined
 - for IsCapCategoryCell, 8
- IsWellDefinedForMorphisms
 - for IsCapCategoryMorphism, 21
- IsWellDefinedForObjects
 - for IsCapCategoryObject, 27
- IsWellDefinedForTwoCells
 - for IsCapCategoryTwoCell, 30
- IsZeroForMorphisms
 - for IsCapCategoryMorphism, 18
- IsZeroForObjects
 - for IsCapCategoryObject, 26
- KernelEmbedding
 - for IsCapCategoryMorphism, 31
- KernelEmbeddingWithGivenKernelObject

- for IsCapCategoryMorphism, IsCapCategoryObject, 31
- KernelLift
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, 32
- KernelLiftWithGivenKernelObject
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject, 32
- KernelObject
 - for IsCapCategoryMorphism, 31
- KernelObjectFunctorial
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, 33
 - for IsList, 33
- KernelObjectFunctorialWithGivenKernelObjects
 - for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject, 33
- LambdaElimination
 - for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryMorphism, 96
- LambdaIntroduction
 - for IsCapCategoryMorphism, 95
- LeftDistributivityExpanding
 - for IsCapCategoryObject, IsList, 81
- LeftDistributivityExpandingWithGivenObjects
 - for IsCapCategoryObject, IsCapCategoryObject, IsList, IsCapCategoryObject, 81
- LeftDistributivityFactoring
 - for IsCapCategoryObject, IsList, 82
- LeftDistributivityFactoringWithGivenObjects
 - for IsCapCategoryObject, IsCapCategoryObject, IsList, IsCapCategoryObject, 82
- LeftUnitor
 - for IsCapCategoryObject, 79
- LeftUnitorInverse
 - for IsCapCategoryObject, 80
- LeftUnitorInverseWithGivenTensorProduct
 - for IsCapCategoryObject, IsCapCategoryObject, 80
- LeftUnitorWithGivenTensorProduct
 - for IsCapCategoryObject, IsCapCategoryObject, 80
- Lift
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, 23
- LiftAlongMonomorphism
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, 22
- MakeDerivation
 - for IsString, IsFunction, IsDenseList, IsPosInt, IsDenseList, IsFunction, 99
- MakeDerivationGraph
 - for IsDenseList, 102
- MakeOperationWeightList
 - for IsCapCategory, IsDerivedMethodGraph, 104
- MonoidalPostComposeMorphism
 - for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, 88
- MonoidalPostComposeMorphismWithGivenObjects
 - for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, 88
- MonoidalPreComposeMorphism
 - for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, 87
- MonoidalPreComposeMorphismWithGivenObjects
 - for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, 88
- MorphismBetweenDirectSums
 - for IsCapCategoryObject, IsList, IsCapCategoryObject, 47
 - for IsList, 47
- MorphismBetweenDirectSumsOp
 - for IsList, IsInt, IsInt, IsCapCategoryMorphism, 47
- MorphismCache
 - for IsCapFunctor, 113
- MorphismFilter

- for IsCapCategory, 7
- MorphismFromBidual
 - for IsCapCategoryObject, 98
- MorphismFromBidualWithGivenBidual
 - for IsCapCategoryObject, IsCapCategory-Object, 98
- MorphismFromCoimageToImage
 - for IsCapCategoryMorphism, 71
- MorphismFromCoimageToImageWithGiven-Objects
 - for IsCapCategoryObject, IsCapCategory-Object, IsCapCategoryMorphism, IsCapCategoryObject, 71
- MorphismFromInternalHomToTensorProduct
 - for IsCapCategoryObject, IsCapCategory-Object, 93
- MorphismFromInternalHomToTensor-ProductWithGivenObjects
 - for IsCapCategoryObject, IsCapCategory-Object, IsCapCategoryObject, IsCap-CategoryObject, 93
- MorphismFromTensorProductToInternalHom
 - for IsCapCategoryObject, IsCapCategory-Object, 92
- MorphismFromTensorProductToInternal-HomWithGivenObjects
 - for IsCapCategoryObject, IsCapCategory-Object, IsCapCategoryObject, IsCap-CategoryObject, 92
- MorphismFromZeroObject
 - for IsCapCategoryObject, 37
- MorphismIntoZeroObject
 - for IsCapCategoryObject, 37
- MorphismToBidual
 - for IsCapCategoryObject, 90
- MorphismToBidualWithGivenBidual
 - for IsCapCategoryObject, IsCapCategory-Object, 90
- Name
 - for IsCapNaturalTransformation, 12
- NaturalIsomorphismFromIdentityTo-CanonicalizeZeroMorphisms
 - for IsCapCategory, 12
- NaturalIsomorphismFromIdentityTo-CanonicalizeZeroObjects
 - for IsCapCategory, 12
- NaturalTransformation
 - for IsCapFunctor, IsCapFunctor, 12
- ObjectCache
 - for IsCapFunctor, 113
- ObjectFilter
 - for IsCapCategory, 7
- Operations
 - for IsDerivedMethodGraph, 103
- OperationWeightUsingDerivation
 - for IsOperationWeightList, IsDerived-Method, 104
- PostCompose
 - for IsCapCategoryMorphism, IsCapCatego-ryMorphism, 21
 - for IsList, 21
- PreCompose
 - for IsCapCategoryMorphism, IsCapCatego-ryMorphism, 20
 - for IsList, 21
- PrintDerivationTree
 - for IsOperationWeightList, IsString, 105
- PrintTree
 - for IsObject, IsFunction, IsFunction, 105
- PrintTreeRec
 - for IsObject, IsFunction, IsFunction, IsInt, 106
- ProjectionInFactorOfDirectProduct
 - for IsList, IsInt, 54
- ProjectionInFactorOfDirectProductOp
 - for IsList, IsInt, IsCapCategoryObject, 54
- ProjectionInFactorOfDirectProductWith-GivenDirectProduct
 - for IsList, IsInt, IsCapCategoryObject, 55
- ProjectionInFactorOfDirectSum
 - for IsList, IsInt, 44
- ProjectionInFactorOfDirectSumOp
 - for IsList, IsInt, IsCapCategoryObject, 44
- ProjectionInFactorOfDirectSumWith-GivenDirectSum
 - for IsList, IsInt, IsCapCategoryObject, 44
- ProjectionInFactorOfFiberProduct
 - for IsList, IsInt, 60
- ProjectionInFactorOfFiberProductOp
 - for IsList, IsInt, IsCapCategoryMorphism, 60

- ProjectionInFactorOfFiberProductWithGivenFiberProduct
 - for IsList, IsInt, IsCapCategoryObject, 60
- Pushout
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, 65
 - for IsList, 65
- PushoutFunctorial
 - for IsList, 67
- PushoutFunctorialWithGivenPushouts
 - for IsCapCategoryObject, IsList, IsCapCategoryObject, 67
- PushoutOp
 - for IsList, IsCapCategoryMorphism, 65
- Range
 - for IsCapCategoryMorphism, 14
 - for IsCapCategoryTwoCell, 28
- RankMorphism
 - for IsCapCategoryObject, 94
- Reevaluate
 - for IsOperationWeightList, 105
- RightDistributivityExpanding
 - for IsList, IsCapCategoryObject, 82
- RightDistributivityExpandingWithGivenObjects
 - for IsCapCategoryObject, IsList, IsCapCategoryObject, IsCapCategoryObject, 82
- RightDistributivityFactoring
 - for IsList, IsCapCategoryObject, 83
- RightDistributivityFactoringWithGivenObjects
 - for IsCapCategoryObject, IsList, IsCapCategoryObject, IsCapCategoryObject, 83
- RightUnitor
 - for IsCapCategoryObject, 80
- RightUnitorInverse
 - for IsCapCategoryObject, 81
- RightUnitorInverseWithGivenTensorProduct
 - for IsCapCategoryObject, IsCapCategoryObject, 81
- RightUnitorWithGivenTensorProduct
 - for IsCapCategoryObject, IsCapCategoryObject, 80
- Source
 - for IsCapCategoryMorphism, 14
 - for IsCapCategoryTwoCell, 28
- StringMinHeap, 106
- Swap
 - for IsStringMinHeap, IsPosInt, IsPosInt, 107
- TargetOperation
 - for IsDerivedMethod, 101
- TensorProductDualityCompatibilityMorphism
 - for IsCapCategoryObject, IsCapCategoryObject, 91
- TensorProductDualityCompatibilityMorphismWithGivenObjects
 - for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, 92
- TensorProductInternalHomCompatibilityMorphism
 - for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, 91
- TensorProductInternalHomCompatibilityMorphismInverse
 - for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, 97
- TensorProductInternalHomCompatibilityMorphismInverseWithGivenObjects
 - for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsList, 97
- TensorProductInternalHomCompatibilityMorphismWithGivenObjects
 - for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsList, 91
- TensorProductOnMorphisms
 - for IsCapCategoryMorphism, IsCapCategoryMorphism, 78
- TensorProductOnMorphismsWithGivenTensorProducts
 - for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject, 78
- TensorProductOnObjects

- for IsCapCategoryObject, IsCapCategory-Object, 77
- TensorProductToInternalHom-AdjunctionMap
 - for IsCapCategoryObject, IsCapCategory-Object, IsCapCategoryMorphism, 87
- TensorUnit
 - for IsCapCategory, 79
- TerminalObject
 - for IsCapCategory, 40
 - for IsCapCategoryCell, 40
- TerminalObjectFunctorial
 - for IsCapCategory, 41
- TraceMap
 - for IsCapCategoryMorphism, 94
- TwoCellFilter
 - for IsCapCategory, 7
- UniversalMorphismFromCoproduct, 52
- UniversalMorphismFromCoproductOp
 - for IsList, IsList, IsCapCategoryObject, 52
- UniversalMorphismFromCoproductWith-GivenCoproduct
 - for IsList, IsList, IsCapCategoryObject, 52
- UniversalMorphismFromDirectSum, 45
- UniversalMorphismFromDirectSumOp
 - for IsList, IsList, IsCapCategoryObject, 46
- UniversalMorphismFromDirectSumWith-GivenDirectSum
 - for IsList, IsList, IsCapCategoryObject, 46
- UniversalMorphismFromImage
 - for IsCapCategoryMorphism, IsList, 69
- UniversalMorphismFromImageWithGiven-ImageObject
 - for IsCapCategoryMorphism, IsList, IsCap-CategoryObject, 70
- UniversalMorphismFromInitialObject
 - for IsCapCategoryObject, 42
- UniversalMorphismFromInitialObject-WithGivenInitialObject
 - for IsCapCategoryObject, IsCapCategory-Object, 42
- UniversalMorphismFromPushout, 66
- UniversalMorphismFromPushoutOp
 - for IsList, IsList, IsCapCategoryMorphism, 66
- UniversalMorphismFromPushoutWithGiven-Pushout
 - for IsList, IsList, IsCapCategoryObject, 66
- UniversalMorphismFromZeroObject
 - for IsCapCategoryObject, 37
- UniversalMorphismFromZeroObjectWith-GivenZeroObject
 - for IsCapCategoryObject, IsCapCategory-Object, 37
- UniversalMorphismIntoCoimage
 - for IsCapCategoryMorphism, IsList, 74
- UniversalMorphismIntoCoimageWithGiven-Coimage
 - for IsCapCategoryMorphism, IsList, IsCap-CategoryObject, 74
- UniversalMorphismIntoDirectProduct, 55
- UniversalMorphismIntoDirectProductOp
 - for IsList, IsList, IsCapCategoryObject, 55
- UniversalMorphismIntoDirectProduct-WithGivenDirectProduct
 - for IsList, IsList, IsCapCategoryObject, 55
- UniversalMorphismIntoDirectSum, 45
- UniversalMorphismIntoDirectSumOp
 - for IsList, IsList, IsCapCategoryObject, 45
- UniversalMorphismIntoDirectSumWith-GivenDirectSum
 - for IsList, IsList, IsCapCategoryObject, 45
- UniversalMorphismIntoFiberProduct, 60
- UniversalMorphismIntoFiberProductOp
 - for IsList, IsList, IsCapCategoryMorphism, 60
- UniversalMorphismIntoFiberProductWith-GivenFiberProduct
 - for IsList, IsList, IsCapCategoryObject, 61
- UniversalMorphismIntoTerminalObject
 - for IsCapCategoryObject, 40
- UniversalMorphismIntoTerminalObject-WithGivenTerminalObject
 - for IsCapCategoryObject, IsCapCategory-Object, 41
- UniversalMorphismIntoZeroObject
 - for IsCapCategoryObject, 37
- UniversalMorphismIntoZeroObjectWith-GivenZeroObject
 - for IsCapCategoryObject, IsCapCategory-Object, 37
- UniversalPropertyOfDual

- for IsCapCategoryObject, IsCapCategory-Object, IsCapCategoryMorphism, [95](#)
- UsedOperationMultiples
 - for IsDerivedMethod, [101](#)
- UsedOperations
 - for IsDerivedMethod, [101](#)
- UsedOperationsWithMultiples
 - for IsDerivedMethod, [101](#)
- VerticalPostCompose
 - for IsCapCategoryTwoCell, IsCapCategoryTwoCell, [29](#)
- VerticalPreCompose
 - for IsCapCategoryTwoCell, IsCapCategoryTwoCell, [29](#)
- ZeroMorphism
 - for IsCapCategoryObject, IsCapCategory-Object, [18](#)
- ZeroObject
 - for IsCapCategory, [36](#)
 - for IsCapCategoryCell, [36](#)
- ZeroObjectFunctorial
 - for IsCapCategory, [40](#)