CAP

Categories, Algorithms, Programming

2016.02.19

19/02/2016

Sebastian Gutsche

Sebastian Posur

Øystein Skartsæterhagen

Sebastian Gutsche
Email: gutsche@mathematik.uni-kl.de
Homepage: http://wwwb.math.rwth-aachen.de/~gutsche
Address: TODO

Sebastian Posur
Email: sposur@momo.math.rwth-aachen.de
Homepage: http://wwwb.math.rwth-aachen.de/Mitarbeiter/posur.php
Address: Lehrstuhl B für Mathematik RWTH - Aachen
Templergraben 64
52062 Aachen
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 . . . 11

3 Morphisms **13**

3.1 Attributes for the Type of Morphisms . . . 13
3.2 Categorical Properties of Morphisms . . . 13
3.3 Non-Categorical Properties of Morphisms . . . 14
3.4 Equality and Congruence for Morphisms . . . 16
3.5 Basic Operations for Morphisms in Ab-Categories . . . 17
3.6 Subobject and Factorobject Operations . . . 18
3.7 Identity Morphism and Composition of Morphisms . . . 19
3.8 Well-Definedness of Morphisms . . . 20
3.9 Basic Operations for Morphisms in Abelian Categories . . . 21
3.10 Lift/ Colift . . . 21
3.11 Inverses . . . 22
3.12 Tool functions for caches . . . 23

4 Objects **24**

4.1 Attributes for the Type of Objects . . . 24
4.2 Equality for Objects . . . 24
4.3 Categorical Properties of Objects . . . 25
4.4 Tool functions for caches . . . 26
4.5 Well-Definedness of Objects . . . 26

5 Category 2-Cells **27**
5.1 Attributes for the Type of 2-Cells . . . 27
5.2 Identity 2-Cell and Composition of 2-Cells . . . 27
5.3 Well-Definedness for 2-Cells . . . 29

6 Universal Objects **30**
6.1 Kernel . . . 30
6.2 Cokernel . . . 33
6.3 Zero Object . . . 35
6.4 Terminal Object . . . 39
6.5 Initial Object . . . 41
6.6 Direct Sum . . . 42
6.7 Coproduct . . . 49
6.8 Direct Product . . . 53
6.9 Fiber Product . . . 56
6.10 Pushout . . . 61
6.11 Image . . . 66
6.12 Coimage . . . 70
6.13 Convenience Methods . . . 75

7 Tensor Product and Internal Hom **76**
7.1 Monoidal Categories . . . 76
7.2 Braided Monoidal Categories . . . 82
7.3 Symmetric Monoidal Categories . . . 84
7.4 Symmetric Closed Monoidal Categories . . . 84
7.5 Rigid Symmetric Closed Monoidal Categories . . . 96

8 Managing Derived Methods **98**
8.1 Info Class . . . 98
8.2 Derivation Objects . . . 98
8.3 Derivation Graphs . . . 100
8.4 Managing Derivations in a Category . . . 103
8.5 Min Heaps for Strings . . . 105

9 Add Functions **107**
9.1 Functions Installed by Add . . . 107
9.2 Add Method . . . 108
9.3 InstallAdd Function . . . 109
9.4 Install All Adds . . . 110

10 Technical Details **112**
10.1 The Category Cat . . . 112
10.2 Install Functions for IsWellDefined . . . 112
10.3 Universal Objects . . . 114

11 Examples and Tests **117**
11.1 Spectral Sequences 117
11.2 Monoidal Categories 120
11.3 Generalized Morphisms Category 122
11.4 IsWellDefined 129
11.5 Kernel 129
11.6 FiberProduct 131

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 and IsAdditiveElementWithInverse)

▷ `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 cateogry 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 cateogry 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 cateogry 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 cateogry 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 cateogry 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 implemetation 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 an 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 funtor *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

2.5 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 $F: A \to B$ and $G: A \to B$. 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 $N : F \to G$, the arguments should be $F(A), A, G(A)$. The ouptput should be a morphism, $F(A) \to G(A)$.

2.5.4 ApplyNaturalTransformation

▷ `ApplyNaturalTransformation(N, A)` (function)

Returns: a morphism

Given a natural transformation $N: F \to G$ and an object `A`, this function should return the morphism $F(A) \to G(A)$, 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`. $F : \alpha \mapsto \texttt{IsMonomorphism}(\alpha)$.

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`. $F : \alpha \mapsto$ `IsEpimorphism`(α).

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`. $F : \alpha \mapsto$ `IsIsomorphism`(α).

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`. $F : \alpha \mapsto$ `IsSplitMonomorphism`(α).

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`. $F : \alpha \mapsto$ `IsSplitEpimorphism`(α).

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`. $F : \alpha \mapsto$ `IsOne`(α).

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`. $F : \alpha \mapsto$ `IsIdempotent`(α).

3.3 Non-Categorical Properties of Morphisms

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

3.3.1 IsIdenticalToIdentityMorphism (for IsCapCategoryMorphism)

▷ `IsIdenticalToIdentityMorphism(alpha)` (property)

Returns: a boolean

The argument is a morphism $\alpha : a \to b$. The output is `true` if $\alpha = \mathrm{id}_a$, 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`. $F : \alpha \mapsto$ `IsIdenticalToIdentityMorphism`(α).

3.3.3 IsIdenticalToZeroMorphism (for IsCapCategoryMorphism)

▷ `IsIdenticalToZeroMorphism(alpha)` (property)

Returns: a boolean

The argument is a morphism $\alpha : a \to b$. 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`. $F : \alpha \mapsto$ `IsIdenticalToZeroMorphism`(α).

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`. $F : \alpha \mapsto$ `IsEndomorphism`(α).

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`. $F : \alpha \mapsto$ `IsAutomorphism`(α).

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`. $F : (\alpha, \beta) \mapsto$ `IsCongruentForMorphisms`(α, β).

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`. $F : (\alpha, \beta) \mapsto$ `IsEqualForMorphisms`(α, β).

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`. $F : (\alpha, \beta) \mapsto$ `IsEqualForMorphismsOnMor`(α, β).

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`. $F : \alpha \mapsto$ `IsZeroForMorphisms`(α).

3.5.3 AdditionForMorphisms (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

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

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 $\mathrm{Hom}(a,b)$

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 $\mathrm{Hom}(a,b)$

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`. $F : (a,b) \mapsto (0 : a \to b)$.

3.6 Subobject and Factorobject Operations

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

3.6.1 IsEqualAsSubobjects (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

Returns: a boolean

The arguments are two subobjects $\alpha : a \to c$, $\beta : b \to c$. The output is `true` if there exists an isomorphism $\iota : a \to b$ 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`. $F : (\alpha,\beta) \mapsto$ `IsEqualAsSubobjects`(α,β).

3.6.3 IsEqualAsFactorobjects (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

Returns: a boolean

The arguments are two factorobjects $\alpha : c \to a$, $\beta : c \to b$. The output is `true` if there exists an isomorphism $\iota : b \to a$ 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`. $F : (\alpha,\beta) \mapsto$ `IsEqualAsFactorobjects`(α,β).

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 $\alpha : a \to c$, $\beta : b \to c$. The output is `true` if there exists a morphism $\iota : a \to b$ 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`. $F : (\alpha, \beta) \mapsto \texttt{IsDominating}(\alpha, \beta)$.

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 $\alpha : c \to a$, $\beta : c \to b$. The output is `true` if there exists a morphism $\iota : b \to a$ 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`. $F : (\alpha, \beta) \mapsto \texttt{IsCodominating}(\alpha, \beta)$.

3.7 Identity Morphism and Composition of Morphisms

3.7.1 IdentityMorphism (for IsCapCategoryObject)

▷ `IdentityMorphism(a)` (attribute)

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

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`. $F : a \mapsto \mathrm{id}_a$.

3.7.3 PreCompose (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

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

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

3.7.4 PreCompose (for IsList)

▷ `PreCompose(L)` (operation)

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

This is a convenience method. The argument is a list of morphisms $L = (\alpha_1 : a_1 \to a_2, \alpha_2 : a_2 \to a_3, \dots, \alpha_n : a_n \to a_{n+1})$. 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 $\mathrm{Hom}(a, c)$

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

3.7.7 PostCompose (for IsList)

▷ `PostCompose(L)` (operation)

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

This is a convenience method. The argument is a list of morphisms $L = (\alpha_n : a_n \to a_{n+1}, \alpha_{n-1} : a_{n-1} \to a_n, \dots, \alpha_1 : a_1 \to a_2)$. 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`. $F : \alpha \mapsto$ `IsWellDefinedForMorphisms`(α).

3.9 Basic Operations for Morphisms in Abelian Categories

3.9.1 LiftAlongMonomorphism (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

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

The arguments are a monomorphism $\iota : k \hookrightarrow a$ and a morphism $\tau : t \to a$ such that there is a morphism $u : t \to k$ 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 $\mathrm{Hom}(c,t)$

The arguments are an epimorphism $\varepsilon : a \to c$ and a morphism $\tau : a \to t$ such that there is a morphism $u : c \to t$ 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 $\alpha : a \to c$, $\beta : b \to c$, we call each morphism $\alpha/\beta : a \to b$ such that $\beta \circ (\alpha/\beta) \sim_{a,c} \alpha$ a *lift of α along β*.
- For any pair of morphisms $\alpha : a \to c$, $\beta : a \to b$, we call each morphism $\alpha\backslash\beta : c \to b$ such that $(\alpha\backslash\beta) \circ \alpha \sim_{a,b} \beta$ 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 $\mathrm{Hom}(a,b)+\{\mathtt{fail}\}$

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 $\mathrm{Hom}(c,b)+\{\mathtt{fail}\}$

The arguments are two morphisms $\alpha : a \rightarrow c$, $\beta : a \rightarrow b$ such that there is a colift $\alpha\backslash\beta : c \rightarrow b$ of β along α., i.e., a morphism such that $(\alpha\backslash\beta) \circ \alpha \sim_{a,b} \beta$. 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 $\alpha\backslash\beta$ 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 $\alpha \circ \alpha^{-1} \sim_{b,b} \mathrm{id}_b$ and $\alpha^{-1} \circ \alpha \sim_{a,a} \mathrm{id}_a$.

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: northing

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 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`. $F : (a,b) \mapsto \texttt{IsEqualForObjects}(a,b)$.

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`. $F : a \mapsto$ `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`. $F : a \mapsto$ `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`. $F : a \mapsto$ `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`. $F : a \mapsto$ `IsInitial`(a).

4.3.5 IsZeroForObjects (for IsCapCategoryObject)

▷ `IsZeroForObjects(a)` (operation)

Returns: a boolean

The argument is an object a of a category $\mathbf{C}$. The output is `true` if a is isomorphic to the zero object of $\mathbf{C}$, 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`. $F : a \mapsto$ `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: northing

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 $c : \alpha \to \beta$. The output is its source α.

5.1.2 Range (for IsCapCategoryTwoCell)

▷ `Range(c)` (attribute)

Returns: a morphism

The argument is a 2-cell $c : \alpha \to \beta$. 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 $\mathrm{id}_\alpha : \alpha \to \alpha$.

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`. $F : \alpha \mapsto \mathrm{id}_\alpha$.

5.2.3 HorizontalPreCompose (for IsCapCategoryTwoCell, IsCapCategoryTwoCell)

▷ `HorizontalPreCompose(c, d)` (operation)

Returns: a 2-cell

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

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 operations adds the given function F to the category for the basic operation `IsWellDefinedForTwoCells`. $F : c \mapsto$ `IsWellDefinedForMorphisms`(c).

Chapter 6

Universal Objects

6.1 Kernel

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

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

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

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 $\mathrm{Hom}(\mathrm{KernelObject}(\alpha), A)$

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

6.1.3 KernelEmbedding (for IsCapCategoryObject)

▷ `KernelEmbedding(K)` (attribute)

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

This is a convenience method. The argument is an object K that was created as a kernel. The output is the kernel embedding $\iota : K \to A$.

6.1.4 KernelEmbeddingWithGivenKernelObject (for IsCapCategoryMorphism, IsCapCategoryObject)

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

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

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

6.1.5 KernelLift (for IsCapCategoryObject, IsCapCategoryMorphism)

▷ `KernelLift(K, tau)` (operation)

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

This is a convenience method. The arguments are an object K which was created as a kernel, and a test morphism $\tau : T \rightarrow A$ satisfying $\alpha \circ \tau \sim_{T,B} 0$. The output is the morphism $u(\tau) : T \rightarrow K$ given by the universal property of the kernel.

6.1.6 KernelLift (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

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

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

6.1.7 KernelLiftWithGivenKernelObject (for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject)

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

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

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

6.1.8 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`. $F : \alpha \mapsto \mathrm{KernelObject}(\alpha)$.

6.1.9 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`. $F : \alpha \mapsto \iota$.

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

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

6.1.12 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`. $F : (\alpha, \tau, K) \mapsto u$.

6.1.13 KernelObjectFunctorial (for IsList)

▷ `KernelObjectFunctorial(L)` (operation)

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

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 $\mathrm{KernelObject}(\alpha) \rightarrow \mathrm{KernelObject}(\alpha')$ given by the functorality of the kernel.

6.1.14 KernelObjectFunctorial (for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism)

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

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

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

6.1.15 AddKernelObjectFunctorial (for IsCapCategory, IsFunction)

▷ `AddKernelObjectFunctorial(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 `KernelObjectFunctorial`. $F : (\alpha, \mu, \alpha') \mapsto (\mathrm{KernelObject}(\alpha) \rightarrow \mathrm{KernelObject}(\alpha'))$.

6.2 Cokernel

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

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

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

6.2.1 CokernelObject (for IsCapCategoryMorphism)

▷ `CokernelObject(alpha)` (attribute)

Returns: an object

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

6.2.2 CokernelProjection (for IsCapCategoryMorphism)

▷ `CokernelProjection(alpha)` (attribute)

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

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

6.2.3 CokernelProjection (for IsCapCategoryObject)

▷ `CokernelProjection(K)` (attribute)

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

This is a convenience method. The argument is an object K which was created as a cokernel. The output is the cokernel projection $\varepsilon : B \to K$.

6.2.4 CokernelProjectionWithGivenCokernelObject (for IsCapCategoryMorphism, IsCapCategoryObject)

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

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

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

6.2.5 CokernelColift (for IsCapCategoryObject, IsCapCategoryMorphism)

▷ `CokernelColift(K, tau)` (operation)

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

This is a convenience method. The arguments are an object K which was created as a cokernel, and a test morphism $\tau : B \rightarrow T$ satisfying $\tau \circ \alpha \sim_{A,T} 0$. The output is the morphism $u(\tau) : K \rightarrow T$ given by the universal property of the cokernel.

6.2.6 CokernelColift (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

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

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

6.2.7 CokernelColiftWithGivenCokernelObject (for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject)

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

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

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

6.2.8 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`. $F : \alpha \mapsto K$.

6.2.9 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`. $F : \alpha \mapsto \varepsilon$.

6.2.10 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`. $F : (\alpha, K) \mapsto \varepsilon$.

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

6.2.12 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`. $F : (\alpha, \tau, K) \mapsto u(\tau)$.

6.2.13 CokernelFunctorial (for IsList)

▷ `CokernelFunctorial(L)` (operation)

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

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

6.2.14 CokernelFunctorial (for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `CokernelFunctorial(alpha, nu, alpha_prime)` (operation)

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

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

6.2.15 AddCokernelFunctorial (for IsCapCategory, IsFunction)

▷ `AddCokernelFunctorial(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 `CokernelFunctorial`. $F : (\alpha, \nu, \alpha') \mapsto (\mathrm{CokernelObject}(\alpha) \to \mathrm{CokernelObject}(\alpha'))$.

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

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)

▷ `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)

▷ `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 $\mathrm{Hom}(\mathrm{ZeroObject}, A)$

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 $\mathrm{Hom}(A, \mathrm{ZeroObject})$

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 $\mathrm{Hom}(\mathrm{ZeroObject}, A)$

The argument is an object A. The output is the universal morphism $u_{\text{out}} : \mathrm{ZeroObject} \rightarrow A$.

6.3.6 UniversalMorphismFromZeroObjectWithGivenZeroObject (for IsCapCategoryObject, IsCapCategoryObject)

▷ `UniversalMorphismFromZeroObjectWithGivenZeroObject(A, Z)` (operation)

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

The arguments are an object A, and a zero object $Z = \mathrm{ZeroObject}$. 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 $\mathrm{Hom}(A, \mathrm{ZeroObject})$

The argument is an object A. The output is the universal morphism $u_{\mathrm{in}} : A \to \mathrm{ZeroObject}$.

6.3.8 UniversalMorphismIntoZeroObjectWithGivenZeroObject (for IsCapCategoryObject, IsCapCategoryObject)

▷ `UniversalMorphismIntoZeroObjectWithGivenZeroObject(A, Z)` (operation)

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

The arguments are an object A, and a zero object $Z = \mathrm{ZeroObject}$. The output is the universal morphism $u_{\mathrm{in}} : A \to Z$.

6.3.9 IsomorphismFromZeroObjectToInitialObject (for IsCapCategory)

▷ `IsomorphismFromZeroObjectToInitialObject(C)` (attribute)

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

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

6.3.10 IsomorphismFromInitialObjectToZeroObject (for IsCapCategory)

▷ `IsomorphismFromInitialObjectToZeroObject(C)` (attribute)

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

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

6.3.11 IsomorphismFromZeroObjectToTerminalObject (for IsCapCategory)

▷ `IsomorphismFromZeroObjectToTerminalObject(C)` (attribute)

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

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

6.3.12 IsomorphismFromTerminalObjectToZeroObject (for IsCapCategory)

▷ `IsomorphismFromTerminalObjectToZeroObject(C)` (attribute)

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

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

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`. $F : () \mapsto \mathrm{ZeroObject}$.

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`. $F : A \mapsto u_{\mathrm{in}}(A)$.

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`. $F : (A,Z) \mapsto u_{\mathrm{in}}(A)$.

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`. $F : A \mapsto u_{\mathrm{out}}(A)$.

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`. $F : (A,Z) \mapsto u_{\mathrm{out}}(A)$.

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`. $F : () \mapsto (\mathrm{ZeroObject} \to \mathrm{InitialObject})$.

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`. $F : () \mapsto (\mathrm{InitialObject} \rightarrow \mathrm{ZeroObject})$.

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`. $F : () \mapsto (\mathrm{ZeroObject} \rightarrow \mathrm{TerminalObject})$.

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`. $F : () \mapsto (\mathrm{TerminalObject} \rightarrow \mathrm{ZeroObject})$.

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 $\mathrm{Hom}(A, \mathrm{TerminalObject})$

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

6.4.4 UniversalMorphismIntoTerminalObjectWithGivenTerminalObject (for IsCapCategoryObject, IsCapCategoryObject)

▷ `UniversalMorphismIntoTerminalObjectWithGivenTerminalObject(A, T)` (operation)

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

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

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 IsCapCategory, 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 $\mathrm{Hom}(\mathrm{TerminalObject}, \mathrm{TerminalObject})$

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

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 Hom(InitialObject $\rightarrow A$).

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

6.5.4 UniversalMorphismFromInitialObjectWithGivenInitialObject (for IsCapCategoryObject, IsCapCategoryObject)

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

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

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

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

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 $u_{\mathrm{in}}(\tau) : T \rightarrow S$ such that $\pi_i \circ u_{\mathrm{in}}(\tau) \sim_{T,S_i} \tau_i$ 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 $u_{\mathrm{out}}(\tau) : S \rightarrow T$ such that $u_{\mathrm{out}}(\tau) \circ \iota_i \sim_{S_i,T} \tau_i$ for all $i = 1, \dots, n$,

such that

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

where $\delta_{i,j} \in \mathrm{Hom}(S_i, S_j)$ is the identity if $i = j$, and 0 otherwise. The 5-tuple $(S, \pi, \iota, u_{\mathrm{in}}, u_{\mathrm{out}})$ 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 $u_{\mathrm{in}}(\tau), u_{\mathrm{out}}(\tau)$ are induced by the *universal property of the direct sum*.
DirectSum is a functorial operation. This means: For $(\mu_i : S_i \to S_i')_{i=1...n}$, we obtain a morphism $\bigoplus_{i=1}^n S_i \to \bigoplus_{i=1}^n S_i'$.

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 $\mathrm{Hom}(\bigoplus_{i=1}^n S_i, S_k)$

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

6.6.3 ProjectionInFactorOfDirectSumOp (for IsList, IsInt, IsCapCategoryObject)

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

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

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 $\pi_k : \bigoplus_{i=1}^n S_i \to S_k$.

6.6.4 ProjectionInFactorOfDirectSumWithGivenDirectSum (for IsList, IsInt, IsCapCategoryObject)

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

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

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 $\pi_k : S \to S_k$.

6.6.5 InjectionOfCofactorOfDirectSum (for IsList, IsInt)

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

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

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

6.6.6 InjectionOfCofactorOfDirectSumOp (for IsList, IsInt, IsCapCategoryObject)

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

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

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 $\iota_k : S_k \to \bigoplus_{i=1}^n S_i$.

6.6.7 InjectionOfCofactorOfDirectSumWithGivenDirectSum (for IsList, IsInt, IsCapCategoryObject)

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

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

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 $\mathrm{Hom}(T, \bigoplus_{i=1}^n S_i)$

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

The output is the morphism $u_{\mathrm{in}}(\tau) : T \rightarrow \bigoplus_{i=1}^n S_i$ 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 $\mathrm{Hom}(T, \bigoplus_{i=1}^n S_i)$

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

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

6.6.11 UniversalMorphismFromDirectSum

▷ `UniversalMorphismFromDirectSum(arg)` (function)

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

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 $\tau = (\tau_i : S_i \rightarrow T)_{i=1\dots n}$.
- The argument is a list of morphisms $\tau = (\tau_i : S_i \rightarrow T)_{i=1\dots n}$.

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

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

6.6.12 UniversalMorphismFromDirectSumOp (for IsList, IsList, IsCapCategoryObject)

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

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

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

6.6.13 UniversalMorphismFromDirectSumWithGivenDirectSum (for IsList, IsList, IsCapCategoryObject)

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

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

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

6.6.14 IsomorphismFromDirectSumToDirectProduct (for IsList)

▷ `IsomorphismFromDirectSumToDirectProduct(D)` (operation)

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

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

6.6.15 IsomorphismFromDirectSumToDirectProductOp (for IsList, IsCapCategoryObject)

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

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

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 $\bigoplus_{i=1}^n S_i \to \prod_{i=1}^n S_i$.

6.6.16 IsomorphismFromDirectProductToDirectSum (for IsList)

▷ `IsomorphismFromDirectProductToDirectSum(D)` (operation)

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

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

6.6.17 IsomorphismFromDirectProductToDirectSumOp (for IsList, IsCapCategoryObject)

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

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

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 $\mathrm{Hom}(\bigoplus_{i=1}^n S_i, \bigsqcup_{i=1}^n S_i)$

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 $\mathrm{Hom}(\bigoplus_{i=1}^n S_i, \bigsqcup_{i=1}^n S_i)$

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 $\mathrm{Hom}(\bigsqcup_{i=1}^n S_i, \bigoplus_{i=1}^n S_i)$

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 $\mathrm{Hom}(\bigsqcup_{i=1}^n S_i, \bigoplus_{i=1}^n S_i)$

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)

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

The argument is a list of lists of morphisms $M = ((\phi_{i,j} : A_i \rightarrow A_j)_{j=1 \dots n})_{i=1 \dots m}$. The output is the morphism $\bigoplus_{i=1}^m A_i \rightarrow \bigoplus_{j=1}^n A_j$ defined by the matrix M.

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

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

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

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 $\phi_{i,j} : A_i \rightarrow A_j$, an integer m, an integer n, and a method selection morphism. The output is the morphism $\bigoplus_{i=1}^m A_i \rightarrow \bigoplus_{j=1}^n A_j$ 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`. $F : (D,\tau) \mapsto u_{\mathrm{in}}(\tau)$.

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`. $F : (D,\tau,S) \mapsto u_{\mathrm{in}}(\tau)$.

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 $\mathrm{Hom}(\bigoplus_{i=1}^n S_i, \bigoplus_{i=1}^n S_i')$

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 functorality of the direct sum.

6.6.38 AddDirectSumFunctorial (for IsCapCategory, IsFunction)

▷ `AddDirectSumFunctorial(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 `DirectSumFunctorial`. $F : ((\mu_1, \dots, \mu_n)) \mapsto (\bigoplus_{i=1}^n S_i \rightarrow \bigoplus_{i=1}^n S_i')$.

6.6.39 DirectSumFunctorialOp (for IsList, IsCapCategoryMorphism)

▷ `DirectSumFunctorialOp(L, method_selection_morphism)` (operation)

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

The arguments are a list of morphisms $L = (\mu_1 : S_1 \rightarrow S_1', \dots, \mu_n : S_n \rightarrow S_n')$ and a method selection morphism. The output is a morphism $\bigoplus_{i=1}^n S_i \rightarrow \bigoplus_{i=1}^n S_i'$ given by the functorality of the direct sum.

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 $u(\tau) \circ \iota_i \sim_{I_i,T} \tau_i$ 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 $I_1 \bigsqcup I_2$.

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 $I_1 \bigsqcup I_2 \bigsqcup I_3$.

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 $\mathrm{Hom}(I_k, \bigsqcup_{i=1}^n I_i)$

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

6.7.6 InjectionOfCofactorOfCoproductOp (for IsList, IsInt, IsCapCategoryObject)

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

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

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 $\iota_k : I_k \to \bigsqcup_{i=1}^n I_i$.

6.7.7 InjectionOfCofactorOfCoproductWithGivenCoproduct (for IsList, IsInt, IsCapCategoryObject)

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

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

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 $\iota_k : I_k \to I$.

6.7.8 UniversalMorphismFromCoproduct

▷ `UniversalMorphismFromCoproduct(arg)` (function)

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

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 $\mathrm{Hom}(\bigsqcup_{i=1}^{n} I_i, T)$

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 $\mathrm{Hom}(I, T)$

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`. $F : ((I_1, \dots, I_n), i) \mapsto \iota_i$.

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`. $F : ((I_1,\dots,I_n),i,I) \mapsto \iota_i$.

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 $\mathrm{Hom}(\bigsqcup_{i=1}^n I_i, \bigsqcup_{i=1}^n I_i')$

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 functorality of the coproduct.

6.7.17 AddCoproductFunctorial (for IsCapCategory, IsFunction)

▷ `AddCoproductFunctorial(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 `CoproductFunctorial`. $F : (\mu_1,\dots,\mu_n) \rightarrow (\bigsqcup_{i=1}^n I_i \rightarrow \bigsqcup_{i=1}^n I_i')$.

6.7.18 CoproductFunctorialOp (for IsList, IsCapCategoryMorphism)

▷ `CoproductFunctorialOp(L, method_selection_morphism)` (operation)

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

The arguments are a list $L = (\mu_1 : I_1 \rightarrow I_1', \dots, \mu_n : I_n \rightarrow I_n')$ and a method selection morphism. The output is a morphism $\bigsqcup_{i=1}^n I_i \rightarrow \bigsqcup_{i=1}^n I_i'$ given by the functorality of the coproduct.

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 $\pi = (\pi_i : P \to P_i)_{i=1\dots n}$
- a dependent function u mapping each list of morphisms $\tau = (\tau_i : T \to P_i)_{i=1,\dots,n}$ to a morphism $u(\tau) : T \to P$ 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 $(\mu_i : P_i \to P_i')_{i=1\dots n}$, we obtain a morphism $\prod_{i=1}^n P_i \to \prod_{i=1}^n P_i'$.

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 $\mathrm{Hom}(\prod_{i=1}^n P_i, P_k)$

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

6.8.3 ProjectionInFactorOfDirectProductOp (for IsList, IsInt, IsCapCategoryObject)

▷ `ProjectionInFactorOfDirectProductOp(D, k, method_selection_object)` (operation)

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

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 $\pi_k : \prod_{i=1}^n P_i \to P_k$.

6.8.4 ProjectionInFactorOfDirectProductWithGivenDirectProduct (for IsList, IsInt, IsCapCategoryObject)

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

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

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 $\pi_k : P \to P_k$.

6.8.5 UniversalMorphismIntoDirectProduct

▷ `UniversalMorphismIntoDirectProduct(arg)` (function)

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

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 $\mathrm{Hom}(T, \prod_{i=1}^{n} P_i)$

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 $\mathrm{Hom}(T, \prod_{i=1}^{n} P_i)$

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 $\mathrm{Hom}(\prod_{i=1}^n P_i, \prod_{i=1}^n P_i')$

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 functorality of the direct product.

6.8.14 AddDirectProductFunctorial (for IsCapCategory, IsFunction)

▷ `AddDirectProductFunctorial(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 `DirectProductFunctorial`. $F : ((\mu_i : P_i \rightarrow P_i')_{i=1\dots n}) \mapsto (\prod_{i=1}^n P_i \rightarrow \prod_{i=1}^n P_i')$

6.8.15 DirectProductFunctorialOp (for IsList, IsCapCategoryMorphism)

▷ `DirectProductFunctorialOp(L, method_selection_morphism)` (operation)

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

The arguments are a list of morphisms $L = (\mu_i : P_i \rightarrow P_i')_{i=1\dots n}$, and a morphism for method selection. The output is a morphism $\prod_{i=1}^n P_i \rightarrow \prod_{i=1}^n P_i'$ given by the functorality of the direct product.

6.9 Fiber Product

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

- an object P,
- a list of morphisms $\pi = (\pi_i : P \to P_i)_{i=1...n}$ 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 $\tau = (\tau_i : T \to P_i)$ such that $\beta_i \circ \tau_i \sim_{T,B} \beta_j \circ \tau_j$ for all pairs i, j to a morphism $u(\tau) : T \to P$ 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 $D' = (\beta_i' : P_i' \to B')_{i=1...n}$ and a natural morphism between pullback diagrams (i.e., a collection of morphisms $(\mu_i : P_i \to P_i')_{i=1...n}$ and $\beta : B \to B'$ 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) $\to$ FiberProduct(D').

6.9.1 IsomorphismFromFiberProductToKernelOfDiagonalDifference (for IsList)

▷ `IsomorphismFromFiberProductToKernelOfDiagonalDifference(D)` (operation)

Returns: a morphism in Hom(FiberProduct(D), Δ)

The argument is a list of morphisms $D = (\beta_i : P_i \to B)_{i=1...n}$. The output is a morphism FiberProduct(D) $\to \Delta$, 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 Hom(FiberProduct(D), Δ)

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

6.9.3 AddIsomorphismFromFiberProductToKernelOfDiagonalDifference (for IsCapCategory, IsFunction)

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

6.9.4 IsomorphismFromKernelOfDiagonalDifferenceToFiberProduct (for IsList)

▷ `IsomorphismFromKernelOfDiagonalDifferenceToFiberProduct(D)` (operation)

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

The argument is a list of morphisms $D = (\beta_i : P_i \to B)_{i=1\dots n}$. The output is a morphism $\Delta \to \mathrm{FiberProduct}(D)$, 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 $\mathrm{Hom}(\Delta, \mathrm{FiberProduct}(D))$

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

6.9.6 AddIsomorphismFromKernelOfDiagonalDifferenceToFiberProduct (for IsCapCategory, IsFunction)

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

6.9.7 DirectSumDiagonalDifference (for IsList)

▷ `DirectSumDiagonalDifference(D)` (operation)

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

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

6.9.8 DirectSumDiagonalDifferenceOp (for IsList, IsCapCategoryMorphism)

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

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

The argument is a list of morphisms $D = (\beta_i : P_i \to B)_{i=1\dots n}$ and a morphism for method selection. The output is a morphism $\bigoplus_{i=1}^n P_i \to B$ 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`. $F : (D) \mapsto \mathrm{DirectSumDiagonalDifference}(D)$

6.9.10 FiberProductEmbeddingInDirectSum (for IsList)

▷ `FiberProductEmbeddingInDirectSum(D)` (operation)

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

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

6.9.11 FiberProductEmbeddingInDirectSumOp (for IsList, IsCapCategoryMorphism)

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

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

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

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

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

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

6.9.14 FiberProductOp (for IsList, IsCapCategoryMorphism)

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

Returns: an object

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

6.9.15 ProjectionInFactorOfFiberProduct (for IsList, IsInt)

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

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

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

6.9.16 ProjectionInFactorOfFiberProductOp (for IsList, IsInt, IsCapCategoryMorphism)

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

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

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 $\pi_k : \mathrm{FiberProduct}(D) \rightarrow P_k$.

6.9.17 ProjectionInFactorOfFiberProductWithGivenFiberProduct (for IsList, IsInt, IsCapCategoryObject)

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

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

The arguments are a list of morphisms $D = (\beta_i : P_i \rightarrow B)_{i=1\dots n}$, an integer k, and an object $P = \mathrm{FiberProduct}(D)$. 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 $u(\tau) : T \rightarrow \mathrm{FiberProduct}(D)$ 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 $u(\tau) : T \rightarrow \mathrm{FiberProduct}(D)$ given by the universal property of the fiber product.
- The arguments are an object P which was created as a pullback from a list $D = (\beta_i : P_i \rightarrow B)_{i=1\dots n}$ 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 $u(\tau) : T \rightarrow P$ 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 $\mathrm{Hom}(T, \mathrm{FiberProduct}(D))$

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 $u(\tau) : T \rightarrow \mathrm{FiberProduct}(D)$ 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 $\mathrm{Hom}(T, P)$

The arguments are a list of morphisms $D = (\beta_i : P_i \rightarrow B)_{i=1...n}$, 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 $P = \mathrm{FiberProduct}(D)$. 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`. $F : ((\beta_i : P_i \rightarrow B)_{i=1...n}) \mapsto P$

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

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

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

6.9.25 AddUniversalMorphismIntoFiberProductWithGivenFiberProduct (for IsCapCategory, IsFunction)

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

Returns: nothing

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

6.9.26 FiberProductFunctorial (for IsList)

▷ `FiberProductFunctorial(L)` (operation)

Returns: a morphism in $\mathrm{Hom}(\mathrm{FiberProduct}((\beta_i)_{i=1...n}), \mathrm{FiberProduct}((\beta_i')_{i=1...n}))$

The argument is a list of triples of morphisms $L = ((\beta_i : P_i \to B, \mu_i : P_i \to P_i', \beta_i' : P_i' \to B')_{i=1...n})$ such that there exists a morphism $\beta : B \to B'$ 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 $\mathrm{FiberProduct}((\beta_i)_{i=1...n}) \to \mathrm{FiberProduct}((\beta_i')_{i=1...n})$ given by the functorality of the fiber product.

6.9.27 AddFiberProductFunctorial (for IsCapCategory, IsFunction)

▷ `AddFiberProductFunctorial(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 `FiberProductFunctorial`. $F : ((\beta_i : P_i \to B, \mu_i : P_i \to P_i', \beta_i' : P_i' \to B')_{i=1...n}) \mapsto (\mathrm{FiberProduct}((\beta_i)_{i=1...n}) \to \mathrm{FiberProduct}((\beta_i')_{i=1...n}))$

6.9.28 FiberProductFunctorialOp (for IsList, IsCapCategoryMorphism)

▷ `FiberProductFunctorialOp(L, method_selection_morphism)` (operation)

Returns: a morphism in $\mathrm{Hom}(\mathrm{FiberProduct}((\beta_i)_{i=1...n}), \mathrm{FiberProduct}((\beta_i')_{i=1...n}))$

The arguments are a list of triples of morphisms $L = ((\beta_i : P_i \to B, \mu_i : P_i \to P_i', \beta_i' : P_i' \to B')_{i=1...n})$ such that there exists a morphism $\beta : B \to B'$ such that $\beta_i' \circ \mu_i \sim_{P_i,B'} \beta \circ \beta_i$ for $i = 1,\dots,n$, and a morphism for method selection. The output is the morphism $\mathrm{FiberProduct}((\beta_i)_{i=1...n}) \to \mathrm{FiberProduct}((\beta_i')_{i=1...n})$ given by the functorality of the fiber product.

6.10 Pushout

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

- an object I,
- a list of morphisms $\iota = (\iota_i : I_i \to I)_{i=1...n}$ 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 $\tau = (\tau_i : I_i \to T)_{i=1...n}$ such that $\tau_i \circ \beta_i \sim_{B,T} \tau_j \circ \beta_j$ to a morphism $u(\tau) : I \to T$ such that $u(\tau) \circ \iota_i \sim_{I_i,T} \tau_i$ 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 $\mathrm{Pushout}(D)$. 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 $D' = (\beta_i' : B' \to I_i')_{i=1...n}$ and a natural morphism between pushout diagrams (i.e., a collection of morphisms $(\mu_i : I_i \to I_i')_{i=1...n}$ and $\beta : B \to B'$ such that $\beta_i' \circ \beta \sim_{B,I_i'} \mu_i \circ \beta_i$ for $i = 1,\dots n$) we obtain a morphism $\mathrm{Pushout}(D) \to \mathrm{Pushout}(D')$.

6.10.1 IsomorphismFromPushoutToCokernelOfDiagonalDifference (for IsList)

▷ `IsomorphismFromPushoutToCokernelOfDiagonalDifference(D)` (operation)

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

The argument is a list of morphisms $D = (\beta_i : B \to I_i)_{i=1...n}$. The output is a morphism $\mathrm{Pushout}(D) \to \Delta$, 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 $\mathrm{Hom}(\mathrm{Pushout}(D), \Delta)$

The argument is a list of morphisms $D = (\beta_i : B \rightarrow I_i)_{i=1...n}$ and a morphism for method selection. The output is a morphism $\mathrm{Pushout}(D) \rightarrow \Delta$, 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`. $F : ((\beta_i : B \rightarrow I_i)_{i=1...n}) \mapsto (\mathrm{Pushout}(D) \rightarrow \Delta)$

6.10.4 IsomorphismFromCokernelOfDiagonalDifferenceToPushout (for IsList)

▷ `IsomorphismFromCokernelOfDiagonalDifferenceToPushout(D)` (operation)

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

The argument is a list of morphisms $D = (\beta_i : B \rightarrow I_i)_{i=1...n}$. The output is a morphism $\Delta \rightarrow \mathrm{Pushout}(D)$, 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 $\mathrm{Hom}(\Delta, \mathrm{Pushout}(D))$

The argument is a list of morphisms $D = (\beta_i : B \rightarrow I_i)_{i=1...n}$ and a morphism for method selection. The output is a morphism $\Delta \rightarrow \mathrm{Pushout}(D)$, 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`. $F : ((\beta_i : B \rightarrow I_i)_{i=1...n}) \mapsto (\Delta \rightarrow \mathrm{Pushout}(D))$

6.10.7 DirectSumCodiagonalDifference (for IsList)

▷ `DirectSumCodiagonalDifference(D)` (operation)

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

The argument is a list of morphisms $D = (\beta_i : B \to I_i)_{i=1\dots n}$. The output is a morphism $B \to \bigoplus_{i=1}^{n} I_i$ 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 $\mathrm{Hom}(B, \bigoplus_{i=1}^{n} I_i)$

The argument is a list of morphisms $D = (\beta_i : B \to I_i)_{i=1\dots n}$ and a morphism for method selection. The output is a morphism $B \to \bigoplus_{i=1}^{n} I_i$ 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`. $F : (D) \mapsto \mathrm{DirectSumCodiagonalDifference}(D)$

6.10.10 DirectSumProjectionInPushout (for IsList)

▷ `DirectSumProjectionInPushout(D)` (operation)

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

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

6.10.11 DirectSumProjectionInPushoutOp (for IsList, IsCapCategoryMorphism)

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

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

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

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`. $F : ((\beta_i : B \to I_i)_{i=1\dots n}) \mapsto (\bigoplus_{i=1}^{n} I_i \to \mathrm{Pushout}(D))$

6.10.13 Pushout (for IsList)

▷ `Pushout(D)` (operation)

Returns: an object

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

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 Pushout(α,β).

6.10.15 PushoutOp (for IsList, IsCapCategoryMorphism)

▷ `PushoutOp(D)` (operation)

Returns: an object

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

6.10.16 InjectionOfCofactorOfPushout (for IsList, IsInt)

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

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

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

6.10.17 InjectionOfCofactorOfPushoutOp (for IsList, IsInt, IsCapCategoryMorphism)

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

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

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

6.10.18 InjectionOfCofactorOfPushoutWithGivenPushout (for IsList, IsInt, IsCapCategoryObject)

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

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

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

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

- The arguments are an object I which was created as a pushout from a list $D = (\beta_i : B \to I_i)_{i=1\dots n}$ and morphisms $\tau_1 : I_1 \to T, \dots, \tau_n : I_n \to T$ such that $\tau_i \circ \beta_i \sim_{B,T} \tau_j \circ \beta_j$. The output is the morphism $u(\tau) : I \to T$ 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 Hom(Pushout(D),T)

The arguments are a list of morphisms $D = (\beta_i : B \to I_i)_{i=1\dots n}$, a list of morphisms $\tau = (\tau_i : I_i \to T)_{i=1\dots n}$ 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 $u(\tau) : \mathrm{Pushout}(D) \to T$ 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 Hom(I,T)

The arguments are a list of morphisms $D = (\beta_i : B \to I_i)_{i=1\dots n}$, a list of morphisms $\tau = (\tau_i : I_i \to T)_{i=1\dots n}$ such that $\tau_i \circ \beta_i \sim_{B,T} \tau_j \circ \beta_j$, and an object $I = \mathrm{Pushout}(D)$. The output is the morphism $u(\tau) : I \to T$ 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`. $F : ((\beta_i : B \to I_i)_{i=1\dots n}) \mapsto I$

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

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

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

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

6.10.27 PushoutFunctorial (for IsList)

▷ `PushoutFunctorial(L)` (operation)

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

The argument is a list $L = ((\beta_i : B \to I_i, \mu_i : I_i \to I_i', \beta_i' : B' \to I_i')_{i=1\dots n})$ such that there exists a morphism $\beta : B \to B'$ 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 $\mathrm{Pushout}((\beta_i)_{i=1}^n) \to \mathrm{Pushout}((\beta_i')_{i=1}^n)$ given by the functorality of the pushout.

6.10.28 AddPushoutFunctorial (for IsCapCategory, IsFunction)

▷ `AddPushoutFunctorial(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`. $F : ((\beta_i : B \to I_i, \mu_i : I_i \to I_i', \beta_i' : B' \to I_i')_{i=1\dots n}) \mapsto (\mathrm{Pushout}((\beta_i)_{i=1}^n) \to \mathrm{Pushout}((\beta_i')_{i=1}^n))$

6.10.29 PushoutFunctorialOp (for IsList, IsCapCategoryMorphism)

▷ `PushoutFunctorialOp(L, method_selection_morphism)` (operation)

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

The argument is a list $L = ((\beta_i : B \to I_i, \mu_i : I_i \to I_i', \beta_i' : B' \to I_i')_{i=1\dots n})$ such that there exists a morphism $\beta : B \to B'$ such that $\beta_i' \circ \beta \sim_{B,I_i'} \mu_i \circ \beta_i$ for $i = 1, \dots n$, and a morphism for method selection. The output is the morphism $\mathrm{Pushout}((\beta_i)_{i=1}^n) \to \mathrm{Pushout}((\beta_i')_{i=1}^n)$ given by the functorality of the pushout.

6.11 Image

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

- an object I,
- a morphism $c : A \to I$,

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

The argument is a morphism α. The output is the canonical morphism $\mathrm{im}(\alpha) \to \mathrm{KernelObject}(\mathrm{CokernelProjection}(\alpha))$.

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`. $F : \alpha \mapsto (\mathrm{im}(\alpha) \to \mathrm{KernelObject}(\mathrm{CokernelProjection}(\alpha)))$

6.11.3 IsomorphismFromKernelOfCokernelToImageObject (for IsCapCategoryMorphism)

▷ `IsomorphismFromKernelOfCokernelToImageObject(alpha)` (attribute)

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

The argument is a morphism α. The output is the canonical morphism $\mathrm{KernelObject}(\mathrm{CokernelProjection}(\alpha)) \to \mathrm{im}(\alpha)$.

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`. $F : \alpha \mapsto (\mathrm{KernelObject}(\mathrm{CokernelProjection}(\alpha)) \to \mathrm{im}(\alpha))$

6.11.5 ImageObject (for IsCapCategoryMorphism)

▷ `ImageObject(alpha)` (attribute)

Returns: an object

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

6.11.6 ImageEmbedding (for IsCapCategoryObject)

▷ `ImageEmbedding(I)` (attribute)

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

This is a convenience method. The argument is an object I which was created as an image object of a morphism $\alpha : A \rightarrow B$. The output is the image embedding $\iota : I \hookrightarrow B$.

6.11.7 ImageEmbedding (for IsCapCategoryMorphism)

▷ `ImageEmbedding(alpha)` (attribute)

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

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

6.11.8 ImageEmbeddingWithGivenImageObject (for IsCapCategoryMorphism, IsCapCategoryObject)

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

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

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

6.11.9 CoastrictionToImage (for IsCapCategoryObject)

▷ `CoastrictionToImage(I)` (attribute)

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

This is a convenience method. The argument is an object I which was created as an image object of a morphism $\alpha : A \rightarrow B$. The output is the coastriction to image $c : A \rightarrow I$.

6.11.10 CoastrictionToImage (for IsCapCategoryMorphism)

▷ `CoastrictionToImage(alpha)` (attribute)

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

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

6.11.11 CoastrictionToImageWithGivenImageObject (for IsCapCategoryMorphism, IsCapCategoryObject)

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

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

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

6.11.12 UniversalMorphismFromImage (for IsCapCategoryMorphism, IsList)

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

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

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

6.11.13 UniversalMorphismFromImageWithGivenImageObject (for IsCapCategoryMorphism, IsList, IsCapCategoryObject)

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

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

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 $I = \mathrm{im}(\alpha)$. The output is the morphism $u(\tau) : \mathrm{im}(\alpha) \rightarrow T$ given by the universal property of the image.

6.11.14 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`. $F : \alpha \mapsto I$.

6.11.15 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`. $F : \alpha \mapsto \iota$.

6.11.16 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`. $F : (\alpha, I) \mapsto \iota$.

6.11.17 AddCoastrictionToImage (for IsCapCategory, IsFunction)

▷ `AddCoastrictionToImage(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 `CoastrictionToImage`. $F : \alpha \mapsto c$.

6.11.18 AddCoastrictionToImageWithGivenImageObject (for IsCapCategory, IsFunction)

▷ `AddCoastrictionToImageWithGivenImageObject(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 `CoastrictionToImageWithGivenImageObject`. $F : (\alpha, I) \mapsto c$.

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

6.11.20 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 $\alpha : A \to B$, a coimage of α consists of four parts:

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

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

6.12.2 MorphismFromCoimageToImageWithGivenObjects (for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryObject)

▷ `MorphismFromCoimageToImageWithGivenObjects(alpha)` (operation)

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

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

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`. $F : (C,\alpha,I) \mapsto (C \to I)$.

6.12.4 InverseMorphismFromCoimageToImage (for IsCapCategoryMorphism)

▷ `InverseMorphismFromCoimageToImage(alpha)` (attribute)

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

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

6.12.5 InverseMorphismFromCoimageToImageWithGivenObjects (for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryObject)

▷ `InverseMorphismFromCoimageToImageWithGivenObjects(alpha)` (operation)

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

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

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`. $F : (C,\alpha,I) \mapsto (I \to C)$.

6.12.7 IsomorphismFromCoimageToCokernelOfKernel (for IsCapCategoryMorphism)

▷ `IsomorphismFromCoimageToCokernelOfKernel(alpha)` (attribute)

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

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

6.12.8 AddIsomorphismFromCoimageToCokernelOfKernel (for IsCapCategory, IsFunction)

▷ `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`. $F : \alpha \mapsto (\mathrm{coim}(\alpha) \to \mathrm{CokernelObject}(\mathrm{KernelEmbedding}(\alpha)))$.

6.12.9 IsomorphismFromCokernelOfKernelToCoimage (for IsCapCategoryMorphism)

▷ `IsomorphismFromCokernelOfKernelToCoimage(alpha)` (attribute)

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

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

6.12.10 AddIsomorphismFromCokernelOfKernelToCoimage (for IsCapCategory, IsFunction)

▷ `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`. $F : \alpha \mapsto (\mathrm{CokernelObject}(\mathrm{KernelEmbedding}(\alpha)) \to \mathrm{coim}(\alpha))$.

6.12.11 Coimage (for IsCapCategoryMorphism)

▷ `Coimage(alpha)` (attribute)

Returns: an object

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

6.12.12 CoimageProjection (for IsCapCategoryObject)

▷ `CoimageProjection(C)` (attribute)

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

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

6.12.13 CoimageProjection (for IsCapCategoryMorphism)

▷ `CoimageProjection(alpha)` (attribute)

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

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

6.12.14 CoimageProjectionWithGivenCoimage (for IsCapCategoryMorphism, IsCapCategoryObject)

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

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

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

6.12.15 AstrictionToCoimage (for IsCapCategoryObject)

▷ `AstrictionToCoimage(C)` (attribute)

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

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

6.12.16 AstrictionToCoimage (for IsCapCategoryMorphism)

▷ `AstrictionToCoimage(alpha)` (attribute)

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

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

6.12.17 AstrictionToCoimageWithGivenCoimage (for IsCapCategoryMorphism, IsCapCategoryObject)

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

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

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

6.12.18 UniversalMorphismIntoCoimage (for IsCapCategoryMorphism, IsList)

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

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

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

6.12.19 UniversalMorphismIntoCoimageWithGivenCoimage (for IsCapCategoryMorphism, IsList, IsCapCategoryObject)

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

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

The arguments are a morphism $\alpha : A \to B$, a pair of morphisms $\tau = (\tau_1 : A \twoheadrightarrow T, \tau_2 : T \to B)$ where τ_1 is an epimorphism such that $\tau_2 \circ \tau_1 \sim_{A,B} \alpha$, and an object $C = \mathrm{coim}(\alpha)$. The output is the morphism $u(\tau) : T \to C$ 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`. $F : (\alpha, \tau, C) \mapsto u(\tau)$

6.13 Convenience Methods

6.13.1 InjectionOfCofactor

▷ `InjectionOfCofactor(arg)` (function)

Returns:

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

- The arguments are an object S which was created as a direct sum and an integer k. The output is the k-th injection $\iota_k : S_k \to S$.
- The arguments are an object I which was created as a coproduct and an integer k. The output is the k-th injection $\iota_k : I_k \to I$.
- The arguments are an object I which was created as a pushout and an integer k. The output is the k-th injection $\iota_k : I_k \to I$.
- The arguments are a list of objects $D = (I_1, \dots, I_n)$ and an integer k. The output is the k-th injection $\iota_k : I_k \to \bigsqcup_{i=1}^n I_i$.
- The arguments are a list of morphisms $D = (\beta_i : B \to I_i)_{i=1 \dots n}$ and an integer k. The output is the k-th injection $\iota_k : I_k \to \mathrm{Pushout}(D)$.

6.13.2 ProjectionInFactor

▷ `ProjectionInFactor(arg)` (function)

Returns:

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

- The arguments are an object S which was created as a direct sum and an integer k. The output is the k-th projection $\pi_k : S \to S_k$.
- The arguments are an object P which was created as a direct product and an integer k. The output is the k-th projection $\pi_k : P \to P_k$.
- The arguments are an object P which was created as a fiber product and an integer k. The output is the k-th projection $\pi_k : P \to P_k$.
- The arguments are a list of objects $D = (P_1, \dots, P_n)$ and an integer k. The output is the k-th projection $\pi_k : \prod_{i=1}^n P_i \to P_k$.
- The arguments are a list of morphisms $D = (\beta_i : P_i \to B)_{i=1 \dots n}$ and an integer k. The output is the k-th projection $\pi_k : \mathrm{FiberProduct}(D) \to P_k$.

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

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 $\mathrm{Hom}(a \otimes b, a' \otimes b')$

The arguments are two morphisms $\alpha : a \to a', \beta : b \to b'$. 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 $\mathrm{Hom}(a \otimes b, a' \otimes b')$

The arguments are an object $s = a \otimes b$, two morphisms $\alpha : a \to a', \beta : b \to b'$, 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`. $F : (a \otimes b, \alpha : a \to a', \beta : b \to b', a' \otimes b') \mapsto \alpha \otimes \beta$.

7.1.6 AssociatorRightToLeft (for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject)

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

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

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

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

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

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

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 $\alpha_{a,(b,c)} : a \otimes (b \otimes c) \to (a \otimes b) \otimes c$.

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 $\mathrm{Hom}((a \otimes b) \otimes c \to a \otimes (b \otimes c))$.

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

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

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

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

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 $\alpha_{(a,b),c} : (a \otimes b) \otimes c \to a \otimes (b \otimes c)$.

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 $\mathrm{Hom}(1 \otimes a, a)$

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

7.1.15 LeftUnitorWithGivenTensorProduct (for IsCapCategoryObject, IsCapCategoryObject)

▷ `LeftUnitorWithGivenTensorProduct(a, s)` (operation)

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

The arguments are an object a and an object $s = 1 \otimes a$. The output is the left unitor $\lambda_a : 1 \otimes a \to a$.

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 $\mathrm{Hom}(a, 1 \otimes a)$

The argument is an object a. The output is the inverse of the left unitor $\lambda_a^{-1} : a \to 1 \otimes a$.

7.1.18 LeftUnitorInverseWithGivenTensorProduct (for IsCapCategoryObject, IsCapCategoryObject)

▷ `LeftUnitorInverseWithGivenTensorProduct(a, r)` (operation)

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

The argument is an object a and an object $r = 1 \otimes a$. The output is the inverse of the left unitor $\lambda_a^{-1} : a \to 1 \otimes a$.

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 $\mathrm{Hom}(a \otimes 1, a)$

The argument is an object a. The output is the right unitor $\rho_a : a \otimes 1 \to a$.

7.1.21 RightUnitorWithGivenTensorProduct (for IsCapCategoryObject, IsCapCategoryObject)

▷ `RightUnitorWithGivenTensorProduct(a, s)` (operation)

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

The arguments are an object a and an object $s = a \otimes 1$. The output is the right unitor $\rho_a : a \otimes 1 \to a$.

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 $\mathrm{Hom}(a, a \otimes 1)$

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

7.1.24 RightUnitorInverseWithGivenTensorProduct (for IsCapCategoryObject, IsCapCategoryObject)

▷ `RightUnitorInverseWithGivenTensorProduct(a, r)` (operation)

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

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

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 $\mathrm{Hom}(a \otimes (b_1 \oplus \ldots \oplus b_n), (a \otimes b_1) \oplus \ldots \oplus (a \otimes b_n))$

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

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

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

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

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

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`. $F : (a \otimes (b_1 \oplus \ldots \oplus b_n), a, L, (a \otimes b_1) \oplus \ldots \oplus (a \otimes b_n)) \mapsto \mathrm{LeftDistributivityExpandingWithGivenObjects}(a, L)$.

7.1.29 LeftDistributivityFactoring (for IsCapCategoryObject, IsList)

▷ `LeftDistributivityFactoring(a, L)` (operation)

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

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

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

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

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

The arguments are an object $s = (a \otimes b_1) \oplus \ldots \oplus (a \otimes b_n)$, an object a, a list of objects $L = (b_1, \ldots, b_n)$, and an object $r = a \otimes (b_1 \oplus \ldots \oplus b_n)$. 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`. $F : ((a \otimes b_1) \oplus \ldots \oplus (a \otimes b_n), a, L, a \otimes (b_1 \oplus \ldots \oplus b_n)) \mapsto \mathrm{LeftDistributivityFactoringWithGivenObjects}(a, L)$.

7.1.32 RightDistributivityExpanding (for IsList, IsCapCategoryObject)

▷ `RightDistributivityExpanding(L, a)` (operation)

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

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

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

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

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

The arguments are an object $s = (b_1 \oplus \ldots \oplus b_n) \otimes a$, a list of objects $L = (b_1, \ldots, b_n)$, an object a, and an object $r = (b_1 \otimes a) \oplus \ldots \oplus (b_n \otimes a)$. 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`. $F : ((b_1 \oplus \ldots \oplus b_n) \otimes a, L, a, (b_1 \otimes a) \oplus \ldots \oplus (b_n \otimes a)) \mapsto \mathrm{RightDistributivityExpandingWithGivenObjects}(L, a)$.

7.1.35 RightDistributivityFactoring (for IsList, IsCapCategoryObject)

▷ `RightDistributivityFactoring(L, a)` (operation)

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

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

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

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

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

The arguments are an object $s = (b_1 \otimes a) \oplus \ldots \oplus (b_n \otimes a)$, a list of objects $L = (b_1, \ldots, b_n)$, an object a, and an object $r = (b_1 \oplus \ldots \oplus b_n) \otimes a$. 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`. $F : ((b_1 \otimes a) \oplus \ldots \oplus (b_n \otimes a), L, a, (b_1 \oplus \ldots \oplus b_n) \otimes a) \mapsto \mathrm{RightDistributivityFactoringWithGivenObjects}(L, a)$.

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

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

The corresponding GAP property is given by `IsBraidedMonoidalCategory`.

7.2.1 Braiding (for IsCapCategoryObject, IsCapCategoryObject)

▷ `Braiding(a, b)` (operation)

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

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, IsCapCategoryObject)

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

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

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 operations 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 $\mathrm{Hom}(b \otimes a, a \otimes b)$.

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, IsCapCategoryObject)

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

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

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 operations 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 **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 **C** which has for each functor $- \otimes b : \mathbf{C} \to \mathbf{C}$ a right adjoint (denoted by $\underline{\mathrm{Hom}}(b,-)$) 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 $\underline{\mathrm{Hom}}(a,b)$.

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`. $F : (a,b) \mapsto \underline{\mathrm{Hom}}(a,b)$.

7.4.3 InternalHomOnMorphisms (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

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

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

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

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

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

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

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`. $F : (\underline{\mathrm{Hom}}(a',b), \alpha : a \to a', \beta : b \to b', \underline{\mathrm{Hom}}(a,b')) \mapsto \underline{\mathrm{Hom}}(\alpha,\beta)$.

7.4.6 EvaluationMorphism (for IsCapCategoryObject, IsCapCategoryObject)

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

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

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

7.4.7 EvaluationMorphismWithGivenSource (for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject)

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

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

The arguments are two objects a,b and an object $s = \underline{\mathrm{Hom}}(a,b) \otimes a$. The output is the evaluation morphism $\mathrm{ev}_{a,b} : \underline{\mathrm{Hom}}(a,b) \otimes a \to b$, 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`. $F : (a, b, \underline{\mathrm{Hom}}(a,b) \otimes a) \mapsto \mathrm{ev}_{a,b}$.

7.4.9 CoevaluationMorphism (for IsCapCategoryObject, IsCapCategoryObject)

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

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

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

7.4.10 CoevaluationMorphismWithGivenRange (for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject)

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

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

The arguments are two objects a,b and an object $r = \underline{\mathrm{Hom}}(\mathrm{b}, \mathrm{a} \otimes \mathrm{b})$. The output is the coevaluation morphism $\mathrm{coev}_{a,b} : a \to \underline{\mathrm{Hom}}(\mathrm{b}, \mathrm{a} \otimes \mathrm{b})$, 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`. $F : (a, b, \underline{\mathrm{Hom}}(b, a \otimes b)) \mapsto \mathrm{coev}_{a,b}$.

7.4.12 TensorProductToInternalHomAdjunctionMap (for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryMorphism)

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

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

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

7.4.13 AddTensorProductToInternalHomAdjunctionMap (for IsCapCategory, IsFunction)

▷ `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`. $F : (a, b, f : a \otimes b \to c) \mapsto (g : a \to \underline{\mathrm{Hom}}(b, c))$.

7.4.14 InternalHomToTensorProductAdjunctionMap (for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryMorphism)

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

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

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

7.4.15 AddInternalHomToTensorProductAdjunctionMap (for IsCapCategory, IsFunction)

▷ `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`. $F : (b, c, g : a \to \underline{\mathrm{Hom}}(b, c)) \mapsto (g : a \otimes b \to c)$.

7.4.16 MonoidalPreComposeMorphism (for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject)

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

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

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

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

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

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

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

7.4.18 AddMonoidalPreComposeMorphismWithGivenObjects (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `MonoidalPreComposeMorphismWithGivenObjects`. $F : (\underline{\mathrm{Hom}}(a,b) \otimes \underline{\mathrm{Hom}}(b,c), a, b, c, \underline{\mathrm{Hom}}(a,c)) \mapsto \mathrm{MonoidalPreComposeMorphismWithGivenObjects}_{a,b,c}$.

7.4.19 MonoidalPostComposeMorphism (for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject)

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

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

The arguments are three objects a,b,c. The output is the postcomposition morphism $\mathrm{MonoidalPostComposeMorphismWithGivenObjects}_{a,b,c} : \underline{\mathrm{Hom}}(b,c) \otimes \underline{\mathrm{Hom}}(a,b) \rightarrow \underline{\mathrm{Hom}}(a,c)$.

7.4.20 MonoidalPostComposeMorphismWithGivenObjects (for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject)

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

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

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

7.4.21 AddMonoidalPostComposeMorphismWithGivenObjects (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `MonoidalPostComposeMorphismWithGivenObjects`. $F : (\underline{\mathrm{Hom}}(b,c) \otimes \underline{\mathrm{Hom}}(a,b), a, b, c, \underline{\mathrm{Hom}}(a,c)) \mapsto \mathrm{MonoidalPostComposeMorphismWithGivenObjects}_{a,b,c}$.

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 $\mathrm{Hom}(b^\vee, a^\vee)$.

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 $\mathrm{Hom}(b^\vee, a^\vee)$.

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 $\mathrm{Hom}(a^\vee \otimes a, 1)$.

The argument is an object a. The output is the evaluation morphism $\mathrm{ev}_a : a^\vee \otimes a \rightarrow 1$.

7.4.28 EvaluationForDualWithGivenTensorProduct (for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject)

▷ `EvaluationForDualWithGivenTensorProduct(s, a, r)` (operation)

Returns: a morphism in $\mathrm{Hom}(a^\vee \otimes a, 1)$.

The arguments are an object $s = a^\vee \otimes a$, an object a, and an object $r = 1$. The output is the evaluation morphism $\mathrm{ev}_a : a^\vee \otimes a \rightarrow 1$.

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`. $F : (a^\vee \otimes a, a, 1) \mapsto \mathrm{ev}_a$.

7.4.30 CoevaluationForDual (for IsCapCategoryObject)

▷ `CoevaluationForDual(a)` (attribute)

Returns: a morphism in $\mathrm{Hom}(1, a \otimes a^\vee)$.

The argument is an object a. The output is the coevaluation morphism $\mathrm{coev}_a : 1 \to a \otimes a^\vee$.

7.4.31 CoevaluationForDualWithGivenTensorProduct (for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject)

▷ `CoevaluationForDualWithGivenTensorProduct(s, a, r)` (operation)

Returns: a morphism in $\mathrm{Hom}(1, a \otimes a^\vee)$.

The arguments are an object $s = 1$, an object a, and an object $r = a \otimes a^\vee$. The output is the coevaluation morphism $\mathrm{coev}_a : 1 \to a \otimes a^\vee$.

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`. $F : (1, a, a \otimes a^\vee) \mapsto \mathrm{coev}_a$.

7.4.33 MorphismToBidual (for IsCapCategoryObject)

▷ `MorphismToBidual(a)` (attribute)

Returns: a morphism in $\mathrm{Hom}(a, (a^\vee)^\vee)$.

The argument is an object a. The output is the morphism to the bidual $a \to (a^\vee)^\vee$.

7.4.34 MorphismToBidualWithGivenBidual (for IsCapCategoryObject, IsCapCategoryObject)

▷ `MorphismToBidualWithGivenBidual(a, r)` (operation)

Returns: a morphism in $\mathrm{Hom}(a, (a^\vee)^\vee)$.

The arguments are an object a, and an object $r = (a^\vee)^\vee$. The output is the morphism to the bidual $a \to (a^\vee)^\vee$.

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`. $F : (a, (a^\vee)^\vee) \mapsto (a \to (a^\vee)^\vee)$.

7.4.36 TensorProductInternalHomCompatibilityMorphism (for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject)

▷ `TensorProductInternalHomCompatibilityMorphism(a, a', b, b')` (operation)

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

The arguments are four objects a, a', b, b'. The output is the natural morphism $\mathrm{TensorProductInternalHomCompatibilityMorphismWithGivenObjects}_{a,a',b,b'} : \underline{\mathrm{Hom}}(a,a') \otimes \underline{\mathrm{Hom}}(b,b') \to \underline{\mathrm{Hom}}(a \otimes b, a' \otimes b')$.

7.4.37 TensorProductInternalHomCompatibilityMorphismWithGivenObjects (for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsList)

▷ `TensorProductInternalHomCompatibilityMorphismWithGivenObjects(a, a', b, b', L)` (operation)

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

The arguments are four objects a, a', b, b', and a list $L = [\underline{\mathrm{Hom}}(a,a') \otimes \underline{\mathrm{Hom}}(b,b'), \underline{\mathrm{Hom}}(a \otimes b, a' \otimes b')]$. The output is the natural morphism $\mathrm{TensorProductInternalHomCompatibilityMorphismWithGivenObjects}_{a,a',b,b'} : \underline{\mathrm{Hom}}(a,a') \otimes \underline{\mathrm{Hom}}(b,b') \to \underline{\mathrm{Hom}}(a \otimes b, a' \otimes b')$.

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 `TensorProductInternalHomCompatibilityMorphismWithGivenObjects`. $F : (a, a', b, b', [\underline{\mathrm{Hom}}(a,a') \otimes \underline{\mathrm{Hom}}(b,b'), \underline{\mathrm{Hom}}(a \otimes b, a' \otimes b')]) \mapsto \mathrm{TensorProductInternalHomCompatibilityMorphismWithGivenObjects}_{a,a',b,b'}$.

7.4.39 TensorProductDualityCompatibilityMorphism (for IsCapCategoryObject, IsCapCategoryObject)

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

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

The arguments are two objects a, b. The output is the natural morphism $\mathrm{TensorProductDualityCompatibilityMorphismWithGivenObjects} : a^\vee \otimes b^\vee \to (a \otimes b)^\vee$.

7.4.40 TensorProductDualityCompatibilityMorphismWithGivenObjects (for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject)

▷ `TensorProductDualityCompatibilityMorphismWithGivenObjects(s, a, b, r)` (operation)

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

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 $\mathrm{TensorProductDualityCompatibilityMorphismWithGivenObjects}_{a,b} : a^\vee \otimes b^\vee \rightarrow (a \otimes b)^\vee$.

7.4.41 AddTensorProductDualityCompatibilityMorphismWithGivenObjects (for IsCapCategory, IsFunction)

▷ `AddTensorProductDualityCompatibilityMorphismWithGivenObjects(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 `TensorProductDualityCompatibilityMorphismWithGivenObjects`. $F : (a^\vee \otimes b^\vee, a, b, (a \otimes b)^\vee) \mapsto \mathrm{TensorProductDualityCompatibilityMorphismWithGivenObjects}_{a,b}$.

7.4.42 MorphismFromTensorProductToInternalHom (for IsCapCategoryObject, IsCapCategoryObject)

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

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

The arguments are two objects a,b. The output is the natural morphism $\mathrm{MorphismFromTensorProductToInternalHomWithGivenObjects}_{a,b} : a^\vee \otimes b \rightarrow \underline{\mathrm{Hom}}(a,b)$.

7.4.43 MorphismFromTensorProductToInternalHomWithGivenObjects (for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject)

▷ `MorphismFromTensorProductToInternalHomWithGivenObjects(s, a, b, r)` (operation)

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

The arguments are an object $s = a^\vee \otimes b$, two objects a,b, and an object $r = \underline{\mathrm{Hom}}(a,b)$. The output is the natural morphism $\mathrm{MorphismFromTensorProductToInternalHomWithGivenObjects}_{a,b} : a^\vee \otimes b \rightarrow \underline{\mathrm{Hom}}(a,b)$.

7.4.44 AddMorphismFromTensorProductToInternalHomWithGivenObjects (for IsCapCategory, IsFunction)

▷ `AddMorphismFromTensorProductToInternalHomWithGivenObjects(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 `MorphismFromTensorProductToInternalHomWithGivenObjects`. $F : (a^\vee \otimes b, a, b, \underline{\mathrm{Hom}}(a,b)) \mapsto \mathrm{MorphismFromTensorProductToInternalHomWithGivenObjects}_{a,b}$.

7.4.45 IsomorphismFromTensorProductToInternalHom (for IsCapCategoryObject, IsCapCategoryObject)

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

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

The arguments are two objects a,b. The output is the natural morphism $\mathrm{IsomorphismFromTensorProductToInternalHom}_{a,b} : a^\vee \otimes b \to \underline{\mathrm{Hom}}(a,b)$.

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 `IsomorphismFromTensorProductToInternalHom`. $F : (a,b) \mapsto \mathrm{IsomorphismFromTensorProductToInternalHom}_{a,b}$.

7.4.47 MorphismFromInternalHomToTensorProduct (for IsCapCategoryObject, IsCapCategoryObject)

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

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

The arguments are two objects a,b. The output is the inverse of MorphismFromTensorProductToInternalHomWithGivenObjects, namely $\mathrm{MorphismFromInternalHomToTensorProductWithGivenObjects}_{a,b} : \underline{\mathrm{Hom}}(a,b) \to a^\vee \otimes b$.

7.4.48 MorphismFromInternalHomToTensorProductWithGivenObjects (for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject)

▷ `MorphismFromInternalHomToTensorProductWithGivenObjects(s, a, b, r)` (operation)

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

The arguments are an object $s = \underline{\mathrm{Hom}}(a,b)$, two objects a,b, and an object $r = a^\vee \otimes b$. The output is the inverse of MorphismFromTensorProductToInternalHomWithGivenObjects, namely $\mathrm{MorphismFromInternalHomToTensorProductWithGivenObjects}_{a,b} : \underline{\mathrm{Hom}}(a,b) \to a^\vee \otimes b$.

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 `MorphismFromInternalHomToTensorProductWithGivenObjects`. $F : (\underline{\mathrm{Hom}}(a,b), a, b, a^\vee \otimes b) \mapsto \mathrm{MorphismFromInternalHomToTensorProductWithGivenObjects}_{a,b}$.

7.4.50 IsomorphismFromInternalHomToTensorProduct (for IsCapCategoryObject, IsCapCategoryObject)

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

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

The arguments are two objects a,b. The output is the inverse of IsomorphismFromTensorProductToInternalHom, namely $\mathrm{IsomorphismFromInternalHomToTensorProduct}_{a,b} : \underline{\mathrm{Hom}}(a,b) \to a^\vee \otimes b$.

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`. $F : (a,b) \mapsto \mathrm{IsomorphismFromInternalHomToTensorProduct}_{a,b}$.

7.4.52 TraceMap (for IsCapCategoryMorphism)

▷ `TraceMap(alpha)` (attribute)

Returns: a morphism in $\mathrm{Hom}(1,1)$.

The argument is a morphism α. The output is the trace morphism $\mathrm{trace}_\alpha : 1 \to 1$.

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`. $F : \alpha \mapsto \mathrm{trace}_\alpha$

7.4.54 RankMorphism (for IsCapCategoryObject)

▷ `RankMorphism(a)` (attribute)

Returns: a morphism in $\mathrm{Hom}(1,1)$.

The argument is an object a. The output is the rank morphism $\mathrm{rank}_a : 1 \to 1$.

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`. $F : a \mapsto \mathrm{rank}_a$

7.4.56 IsomorphismFromDualToInternalHom (for IsCapCategoryObject)

▷ `IsomorphismFromDualToInternalHom(a)` (attribute)

Returns: a morphism in $\mathrm{Hom}(a^\vee, \mathrm{Hom}(a,1))$.

The argument is an object a. The output is the isomorphism $\mathrm{IsomorphismFromDualToInternalHom}_a : a^\vee \to \mathrm{Hom}(a,1)$.

7.4.57 AddIsomorphismFromDualToInternalHom (for IsCapCategory, IsFunction)

▷ `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 `IsomorphismFromDualToInternalHom`. $F : a \mapsto \mathrm{IsomorphismFromDualToInternalHom}_a$

7.4.58 IsomorphismFromInternalHomToDual (for IsCapCategoryObject)

▷ `IsomorphismFromInternalHomToDual(a)` (attribute)

Returns: a morphism in $\mathrm{Hom}(\mathrm{Hom}(a,1),a^\vee)$.

The argument is an object a. The output is the isomorphism $\mathrm{IsomorphismFromInternalHomToDual}_a : \mathrm{Hom}(a,1) \to a^\vee$.

7.4.59 AddIsomorphismFromInternalHomToDual (for IsCapCategory, IsFunction)

▷ `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 `IsomorphismFromInternalHomToDual`. $F : a \mapsto \mathrm{IsomorphismFromInternalHomToDual}_a$

7.4.60 UniversalPropertyOfDual (for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryMorphism)

▷ `UniversalPropertyOfDual(t, a, alpha)` (operation)

Returns: a morphism in $\mathrm{Hom}(t,a^\vee)$.

The arguments are two objects t,a, and a morphism $\alpha : t \otimes a \to 1$. The output is the morphism $t \to a^\vee$ given by the universal property of $a^\vee$.

7.4.61 AddUniversalPropertyOfDual (for IsCapCategory, IsFunction)

▷ `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 `UniversalPropertyOfDual`. $F : (t,a,\alpha : t \otimes a \to 1) \mapsto (t \to a^\vee)$.

7.4.62 LambdaIntroduction (for IsCapCategoryMorphism)

▷ `LambdaIntroduction(alpha)` (attribute)

Returns: a morphism in $\mathrm{Hom}(1,\underline{\mathrm{Hom}}(a,b))$.

The argument is a morphism $\alpha : a \to b$. The output is the corresponding morphism $1 \to \underline{\mathrm{Hom}}(a,b)$ 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`. $F : (\alpha : a \rightarrow b) \mapsto (1 \rightarrow \underline{\mathrm{Hom}}(a,b))$.

7.4.64 LambdaElimination (for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryMorphism)

▷ `LambdaElimination(a, b, alpha)` (operation)

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

The arguments are two objects a,b, and a morphism $\alpha : 1 \rightarrow \underline{\mathrm{Hom}}(a,b)$. 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`. $F : (a,b,\alpha : 1 \rightarrow \underline{\mathrm{Hom}}(a,b)) \mapsto (a \rightarrow b)$.

7.4.66 IsomorphismFromObjectToInternalHom (for IsCapCategoryObject)

▷ `IsomorphismFromObjectToInternalHom(a)` (attribute)

Returns: a morphism in $\mathrm{Hom}(a, \underline{\mathrm{Hom}}(1,a))$.

The argument is an object a. The output is the natural isomorphism $a \rightarrow \underline{\mathrm{Hom}}(1,a)$.

7.4.67 IsomorphismFromObjectToInternalHomWithGivenInternalHom (for IsCapCategoryObject, IsCapCategoryObject)

▷ `IsomorphismFromObjectToInternalHomWithGivenInternalHom(a, r)` (operation)

Returns: a morphism in $\mathrm{Hom}(a, \underline{\mathrm{Hom}}(1,a))$.

The argument is an object a, and an object $r = \underline{\mathrm{Hom}}(1,a)$. The output is the natural isomorphism $a \rightarrow \underline{\mathrm{Hom}}(1,a)$.

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`. $F : (a, \underline{\mathrm{Hom}}(1,a)) \mapsto (a \rightarrow \underline{\mathrm{Hom}}(1,a))$.

7.4.69 IsomorphismFromInternalHomToObject (for IsCapCategoryObject)

▷ `IsomorphismFromInternalHomToObject(a)` (attribute)

Returns: a morphism in $\mathrm{Hom}(\underline{\mathrm{Hom}}(1,a),a)$.

The argument is an object a. The output is the natural isomorphism $\underline{\mathrm{Hom}}(1,a) \to a$.

7.4.70 IsomorphismFromInternalHomToObjectWithGivenInternalHom (for IsCapCategoryObject, IsCapCategoryObject)

▷ `IsomorphismFromInternalHomToObjectWithGivenInternalHom(a, s)` (operation)

Returns: a morphism in $\mathrm{Hom}(\underline{\mathrm{Hom}}(1,a),a)$.

The argument is an object a, and an object $s = \underline{\mathrm{Hom}}(1,a)$. The output is the natural isomorphism $\underline{\mathrm{Hom}}(1,a) \to a$.

7.4.71 AddIsomorphismFromInternalHomToObjectWithGivenInternalHom (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `IsomorphismFromInternalHomToObjectWithGivenInternalHom`. $F : (a, \underline{\mathrm{Hom}}(1,a)) \mapsto (\underline{\mathrm{Hom}}(1,a) \to a)$.

7.5 Rigid Symmetric Closed Monoidal Categories

A symmetric closed monoidal category $\mathbf{C}$ satisfying

- the natural morphism $\underline{\mathrm{Hom}}(a_1,b_1) \otimes \underline{\mathrm{Hom}}(a_2,b_2) \to \underline{\mathrm{Hom}}(a_1 \otimes a_2, b_1 \otimes b_2)$ is an isomorphism,
- the natural morphism $a \to \underline{\mathrm{Hom}}(\underline{\mathrm{Hom}}(a,1),1)$ 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 $\mathrm{Hom}(\underline{\mathrm{Hom}}(a \otimes b, a' \otimes b'), \underline{\mathrm{Hom}}(a,a') \otimes \underline{\mathrm{Hom}}(b,b'))$.

The arguments are four objects a, a', b, b'. The output is the natural morphism $\mathrm{TensorProductInternalHomCompatibilityMorphismInverseWithGivenObjects}_{a,a',b,b'} : \underline{\mathrm{Hom}}(a \otimes b, a' \otimes b') \to \underline{\mathrm{Hom}}(a,a') \otimes \underline{\mathrm{Hom}}(b,b')$.

7.5.2 TensorProductInternalHomCompatibilityMorphismInverseWithGivenObjects (for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsList)

▷ `TensorProductInternalHomCompatibilityMorphismInverseWithGivenObjects(a, a', b, b', L)` (operation)

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

The arguments are four objects a,a',b,b', and a list $L = [\underline{\mathrm{Hom}}(a,a') \otimes \underline{\mathrm{Hom}}(b,b'), \underline{\mathrm{Hom}}(a \otimes b, a' \otimes b')]$. The output is the natural morphism $\mathrm{TensorProductInternalHomCompatibilityMorphismInverseWithGivenObjects}_{a,a',b,b'} : \underline{\mathrm{Hom}}(a \otimes b, a' \otimes b') \to \underline{\mathrm{Hom}}(a,a') \otimes \underline{\mathrm{Hom}}(b,b')$.

7.5.3 AddTensorProductInternalHomCompatibilityMorphismInverseWithGivenObjects (for IsCapCategory, IsFunction)

▷ `AddTensorProductInternalHomCompatibilityMorphismInverseWithGivenObjects(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 `TensorProductInternalHomCompatibilityMorphismInverseWithGivenObjects`. $F : (a,a',b,b',[\underline{\mathrm{Hom}}(a,a') \otimes \underline{\mathrm{Hom}}(b,b'), \underline{\mathrm{Hom}}(a \otimes b, a' \otimes b')]) \mapsto \mathrm{TensorProductInternalHomCompatibilityMorphismInverseWithGivenObjects}_{a,a',b,b'}$.

7.5.4 MorphismFromBidual (for IsCapCategoryObject)

▷ `MorphismFromBidual(a)` (attribute)

Returns: a morphism in $\mathrm{Hom}((a^{\vee})^{\vee}, a)$.

The argument is an object a. The output is the inverse of the morphism to the bidual $(a^{\vee})^{\vee} \to a$.

7.5.5 MorphismFromBidualWithGivenBidual (for IsCapCategoryObject, IsCapCategoryObject)

▷ `MorphismFromBidualWithGivenBidual(a, s)` (operation)

Returns: a morphism in $\mathrm{Hom}((a^{\vee})^{\vee}, a)$.

The argument is an object a, and an object $s = (a^{\vee})^{\vee}$. The output is the inverse of the morphism to the bidual $(a^{\vee})^{\vee} \to a$.

7.5.6 AddMorphismFromBidualWithGivenBidual (for IsCapCategory, IsFunction)

▷ `AddMorphismFromBidualWithGivenBidual(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 `MorphismFromBidualWithGivenBidual`. $F : (a,(a^{\vee})^{\vee}) \mapsto ((a^{\vee})^{\vee} \to a)$.

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 IsMomomorphism). 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 categorys 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 wether range and source of the morphims coincide). Generally, the prefunction is defined in the method record and only depend 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 KernelEmbed-dingedding, 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 wether 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 availible in the category. Also, some derivations, which are triggered by the setting of the primitive value, might be availible.

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 wether 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 contructions. If no argument list is given, all arguments are used.
- 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 wether the function which is to be installed 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.

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

Retuns 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

Retuns 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 themselfes, 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.

10.3 Universal Objects

10.3.1 WasCreatedAsKernelObject

▷ `WasCreatedAsKernelObject` (filter)

When created, this filter is set to true for a kernel object. Note that we chose `WasCreatedAsKernelObject` to be a filter rather than a property, because by default, a filter is set to false.

10.3.2 WasCreatedAsCokernelObject

▷ `WasCreatedAsCokernelObject` (filter)

When created, this filter is set to true for a cokernel object. Note that we chose `WasCreatedAsCokernelObject` to be a filter rather than a property, because by default, a filter is set to false.

10.3.3 WasCreatedAsZeroObject

▷ `WasCreatedAsZeroObject` (filter)

When created, this filter is set to true for a zero object. Note that we chose `WasCreatedAsZeroObject` to be a filter rather than a property, because by default, a filter is set to false.

10.3.4 WasCreatedAsTerminalObject

▷ `WasCreatedAsTerminalObject` (filter)

When created, this filter is set to true for a terminal object. Note that we chose `WasCreatedAsTerminalObject` to be a filter rather than a property, because by default, a filter is set to false.

10.3.5 WasCreatedAsInitialObject

▷ `WasCreatedAsInitialObject` (filter)

When created, this filter is set to true for an initial object. Note that we chose `WasCreatedAsInitialObject` to be a filter rather than a property, because by default, a filter is set to false.

10.3.6 WasCreatedAsDirectSum

▷ `WasCreatedAsDirectSum` (filter)

When created, this filter is set to true for a direct sum object. Note that we chose `WasCreatedAsDirectSum` to be a filter rather than a property, because by default, a filter is set to false.

10.3.7 WasCreatedAsCoproduct

▷ `WasCreatedAsCoproduct` (filter)

When created, this filter is set to true for a coproduct object. Note that we chose `WasCreatedAsCoprodcut` to be a filter rather than a property, because by default, a filter is set to false.

10.3.8 WasCreatedAsDirectProduct

▷ `WasCreatedAsDirectProduct` (filter)

When created, this filter is set to true for a terminal object. Note that we chose `WasCreatedAsDirectProduct` to be a filter rather than a property, because by default, a filter is set to false.

10.3.9 WasCreatedAsFiberProduct

▷ `WasCreatedAsFiberProduct` (filter)

When created, this filter is set to true for a pullback. Note that we chose `WasCreatedAsFiberProduct` to be a filter rather than a property, because by default, a filter is set to false.

10.3.10 WasCreatedAsPushout

▷ `WasCreatedAsPushout` (filter)

When created, this filter is set to true for a pushout. Note that we chose `WasCreatedAsPushout` to be a filter rather than a property, because by default, a filter is set to false.

10.3.11 WasCreatedAsImageObject

▷ `WasCreatedAsImageObject` (filter)

When created, this filter is set to true for an image. Note that we chose `WasCreatedAsImageObject` to be a filter rather than a property, because by default, a filter is set to false.

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( [ InjectionOfCofactor( DirectSum( 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( [ InjectionOfCofactor( DirectSum( 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 lef
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 ) );
```

```
[ [  1,  2 ],
  [  0,  3 ] ]
gap> generator_l1 := StandardGeneratorMorphism( IntHoml, 1 );
<A morphism in Category of left presentations of Z>
gap> morphism_l1 := LambdaElimination( DirectSum( Ml, Ul ), Nl, generator_l1 );
<A morphism in Category of left presentations of Z>
gap> Display( UnderlyingMatrix( morphism_l1 ) );
[ [  -3 ],
  [   2 ] ]
gap> generator_l2 := StandardGeneratorMorphism( IntHoml, 2 );
<A morphism in Category of left presentations of Z>
gap> morphism_l2 := LambdaElimination( DirectSum( Ml, Ul ), Nl, generator_l2 );
<A morphism in Category of left presentations of Z>
gap> Display( UnderlyingMatrix( morphism_l2 ) );
[ [   0 ],
  [  -1 ] ]
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 ) );
[ [  1,  0 ],
  [  2,  3 ] ]
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 ) );
[ [  -3,   2 ] ]
gap> generator_r2 := StandardGeneratorMorphism( IntHoml, 2 );
<A morphism in Category of left presentations of Z>
gap> morphism_r2 := LambdaElimination( DirectSum( Ml, Ul ), Nl, generator_r2 );
<A morphism in Category of left presentations of Z>
gap> Display( UnderlyingMatrix( morphism_r2 ) );
[ [   0 ],
  [  -1 ] ]
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_ran
<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( k, tau );
A rational vector space homomorphism with matrix:
[ [  2 ],
  [  2 ] ]

gap> HasKernelEmbedding( k );
false
gap> KernelEmbedding( k );
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( k, tau );
A rational vector space homomorphism with matrix:
[ [  2 ],
  [  2 ] ]

gap> HasKernelEmbedding( k );
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( k );
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 ] ]

gap> P := FiberProduct( AtoC, BtoC );
<A rational vector space of dimension 1>
gap> p1 := ProjectionInFactor( P, 1 );
A rational vector space homomorphism with matrix:
[ [  1/2 ] ]

gap> p2 := ProjectionInFactor( P, 2 );
A rational vector space homomorphism with matrix:
[ [  1/2,    1 ] ]
```

Index

CAP, 5

ActivateDerivationInfo, 98
Add
for IsStringMinHeap, IsString, IsInt, 105
AddAdditionForMorphisms
for IsCapCategory, IsFunction, 17
AddAdditiveInverseForMorphisms
for IsCapCategory, IsFunction, 17
AddAssociatorLeftToRightWithGiven-TensorProducts
for IsCapCategory, IsFunction, 78
AddAssociatorRightToLeftWithGiven-TensorProducts
for IsCapCategory, IsFunction, 77
AddAstrictionToCoimage
for IsCapCategory, IsFunction, 74
AddAstrictionToCoimageWithGivenCoimage
for IsCapCategory, IsFunction, 74
AddBraidingInverseWithGivenTensor-Products
for IsCapCategory, IsFunction, 83
AddBraidingWithGivenTensorProducts
for IsCapCategory, IsFunction, 83
AddCoastrictionToImage
for IsCapCategory, IsFunction, 69
AddCoastrictionToImageWithGivenImage-Object
for IsCapCategory, IsFunction, 69
AddCoevaluationForDualWithGivenTensor-Product
for IsCapCategory, IsFunction, 89
AddCoevaluationMorphismWithGivenRange
for IsCapCategory, IsFunction, 85
AddCoimage
for IsCapCategory, IsFunction, 73
AddCoimageProjection
for IsCapCategory, IsFunction, 74
AddCoimageProjectionWithGivenCoimage
for IsCapCategory, IsFunction, 74
AddCokernelColift
for IsCapCategory, IsFunction, 35
AddCokernelColiftWithGivenCokernel-Object
for IsCapCategory, IsFunction, 35
AddCokernelFunctorial
for IsCapCategory, IsFunction, 35
AddCokernelObject
for IsCapCategory, IsFunction, 34
AddCokernelProjection
for IsCapCategory, IsFunction, 34
AddCokernelProjectionWithGiven-CokernelObject
for IsCapCategory, IsFunction, 34
AddColift
for IsCapCategory, IsFunction, 22
AddColiftAlongEpimorphism
for IsCapCategory, IsFunction, 21
AddCoproduct
for IsCapCategory, IsFunction, 51
AddCoproductFunctorial
for IsCapCategory, IsFunction, 52
AddDerivation
for IsDerivedMethodGraph, IsDerived-Method, 101
for IsDerivedMethodGraph, IsFunction, Is-DenseList, 101
for IsDerivedMethodGraph, IsFunction, Is-DenseList, IsObject, 101
for IsDerivedMethodGraph, IsFunction, Is-Function, 101
AddDerivationPair
for IsDerivedMethodGraph, IsFunction, Is-Function, IsDenseList, IsDenseList, 101
for IsDerivedMethodGraph, IsFunction, Is-Function, IsDenseList, IsDenseList, Is-DenseList, 101
for IsDerivedMethodGraph, IsFunction, Is-

Function, IsDenseList, IsFunction, IsFunction, 102
for IsDerivedMethodGraph, IsFunction, IsFunction, IsFunction, IsFunction, 102
AddDerivationPairToCAP, 102
AddDerivationToCAP, 102
AddDirectProduct
for IsCapCategory, IsFunction, 54
AddDirectProductFunctorial
for IsCapCategory, IsFunction, 55
AddDirectSum
for IsCapCategory, IsFunction, 49
AddDirectSumCodiagonalDifference
for IsCapCategory, IsFunction, 63
AddDirectSumDiagonalDifference
for IsCapCategory, IsFunction, 57
AddDirectSumFunctorial
for IsCapCategory, IsFunction, 49
AddDirectSumProjectionInPushout
for IsCapCategory, IsFunction, 63
AddDualOnMorphismsWithGivenDuals
for IsCapCategory, IsFunction, 88
AddDualOnObjects
for IsCapCategory, IsFunction, 88
AddEvaluationForDualWithGivenTensorProduct
for IsCapCategory, IsFunction, 89
AddEvaluationMorphismWithGivenSource
for IsCapCategory, IsFunction, 85
AddFiberProduct
for IsCapCategory, IsFunction, 60
AddFiberProductEmbeddingInDirectSum
for IsCapCategory, IsFunction, 58
AddFiberProductFunctorial
for IsCapCategory, IsFunction, 61
AddHorizontalPostCompose
for IsCapCategory, IsFunction, 28
AddHorizontalPreCompose
for IsCapCategory, IsFunction, 28
AddIdentityMorphism
for IsCapCategory, IsFunction, 19
AddIdentityTwoCell
for IsCapCategory, IsFunction, 27
AddImageEmbedding
for IsCapCategory, IsFunction, 69
AddImageEmbeddingWithGivenImageObject
for IsCapCategory, IsFunction, 69
AddImageObject
for IsCapCategory, IsFunction, 69
AddInitialObject
for IsCapCategory, IsFunction, 41
AddInitialObjectFunctorial
for IsCapCategory, IsFunction, 42
AddInjectionOfCofactorOfCoproduct
for IsCapCategory, IsFunction, 51
AddInjectionOfCofactorOfCoproductWithGivenCoproduct
for IsCapCategory, IsFunction, 52
AddInjectionOfCofactorOfDirectSum
for IsCapCategory, IsFunction, 47
AddInjectionOfCofactorOfDirectSumWithGivenDirectSum
for IsCapCategory, IsFunction, 47
AddInjectionOfCofactorOfPushout
for IsCapCategory, IsFunction, 65
AddInjectionOfCofactorOfPushoutWithGivenPushout
for IsCapCategory, IsFunction, 65
AddInternalHomOnMorphismsWithGivenInternalHoms
for IsCapCategory, IsFunction, 84
AddInternalHomOnObjects
for IsCapCategory, IsFunction, 84
AddInternalHomToTensorProductAdjunctionMap
for IsCapCategory, IsFunction, 86
AddInverse
for IsCapCategory, IsFunction, 22
AddInverseMorphismFromCoimageToImageWithGivenObjects
for IsCapCategory, IsFunction, 71
AddIsAutomorphism
for IsCapCategory, IsFunction, 15
AddIsCodominating
for IsCapCategory, IsFunction, 19
AddIsCongruentForMorphisms
for IsCapCategory, IsFunction, 16
AddIsDominating
for IsCapCategory, IsFunction, 19
AddIsEndomorphism
for IsCapCategory, IsFunction, 15
AddIsEpimorphism
for IsCapCategory, IsFunction, 14
AddIsEqualAsFactorobjects

for IsCapCategory, IsFunction, 18
AddIsEqualAsSubobjects
for IsCapCategory, IsFunction, 18
AddIsEqualForCacheForMorphisms
for IsCapCategory, IsFunction, 23
AddIsEqualForCacheForObjects
for IsCapCategory, IsFunction, 26
AddIsEqualForMorphisms
for IsCapCategory, IsFunction, 16
AddIsEqualForMorphismsOnMor
for IsCapCategory, IsFunction, 16
AddIsEqualForObjects
for IsCapCategory, IsFunction, 24
AddIsIdempotent
for IsCapCategory, IsFunction, 14
AddIsIdenticalToIdentityMorphism
for IsCapCategory, IsFunction, 15
AddIsIdenticalToZeroMorphism
for IsCapCategory, IsFunction, 15
AddIsInitial
for IsCapCategory, IsFunction, 25
AddIsInjective
for IsCapCategory, IsFunction, 25
AddIsIsomorphism
for IsCapCategory, IsFunction, 14
AddIsMonomorphism
for IsCapCategory, IsFunction, 13
AddIsomorphismFromCoimageToCokernelOf-Kernel
for IsCapCategory, IsFunction, 71
AddIsomorphismFromCokernelOfDiagonal-DifferenceToPushout
for IsCapCategory, IsFunction, 62
AddIsomorphismFromCokernelOfKernelTo-Coimage
for IsCapCategory, IsFunction, 72
AddIsomorphismFromCoproductToDirectSum
for IsCapCategory, IsFunction, 48
AddIsomorphismFromDirectProductTo-DirectSum
for IsCapCategory, IsFunction, 48
AddIsomorphismFromDirectSumToCoproduct
for IsCapCategory, IsFunction, 48
AddIsomorphismFromDirectSumToDirect-Product
for IsCapCategory, IsFunction, 48
AddIsomorphismFromDualToInternalHom
for IsCapCategory, IsFunction, 94
AddIsomorphismFromFiberProductTo-KernelOfDiagonalDifference
for IsCapCategory, IsFunction, 56
AddIsomorphismFromImageObjectToKernel-OfCokernel
for IsCapCategory, IsFunction, 67
AddIsomorphismFromInitialObjectToZero-Object
for IsCapCategory, IsFunction, 38
AddIsomorphismFromInternalHomToDual
for IsCapCategory, IsFunction, 94
AddIsomorphismFromInternalHomToObject-WithGivenInternalHom
for IsCapCategory, IsFunction, 96
AddIsomorphismFromInternalHomToTensor-Product
for IsCapCategory, IsFunction, 93
AddIsomorphismFromKernelOfCokernelTo-ImageObject
for IsCapCategory, IsFunction, 67
AddIsomorphismFromKernelOfDiagonal-DifferenceToFiberProduct
for IsCapCategory, IsFunction, 57
AddIsomorphismFromObjectToInternalHom-WithGivenInternalHom
for IsCapCategory, IsFunction, 95
AddIsomorphismFromPushoutToCokernelOf-DiagonalDifference
for IsCapCategory, IsFunction, 62
AddIsomorphismFromTensorProductTo-InternalHom
for IsCapCategory, IsFunction, 92
AddIsomorphismFromTerminalObjectTo-ZeroObject
for IsCapCategory, IsFunction, 39
AddIsomorphismFromZeroObjectToInitial-Object
for IsCapCategory, IsFunction, 38
AddIsomorphismFromZeroObjectTo-TerminalObject
for IsCapCategory, IsFunction, 39
AddIsOne
for IsCapCategory, IsFunction, 14
AddIsProjective
for IsCapCategory, IsFunction, 25
AddIsSplitEpimorphism

for IsCapCategory, IsFunction, 14
AddIsSplitMonomorphism
for IsCapCategory, IsFunction, 14
AddIsTerminal
for IsCapCategory, IsFunction, 25
AddIsWellDefinedForMorphisms
for IsCapCategory, IsFunction, 20
AddIsWellDefinedForObjects
for IsCapCategory, IsFunction, 26
AddIsWellDefinedForTwoCells
for IsCapCategory, IsFunction, 29
AddIsZeroForMorphisms
for IsCapCategory, IsFunction, 17
AddIsZeroForObjects
for IsCapCategory, IsFunction, 25
AdditionForMorphisms
for IsCapCategoryMorphism, IsCapCategoryMorphism, 17
AdditiveInverseForMorphisms
for IsCapCategoryMorphism, 17
AddKernelEmbedding
for IsCapCategory, IsFunction, 31
AddKernelEmbeddingWithGivenKernelObject
for IsCapCategory, IsFunction, 32
AddKernelLift
for IsCapCategory, IsFunction, 32
AddKernelLiftWithGivenKernelObject
for IsCapCategory, IsFunction, 32
AddKernelObject
for IsCapCategory, IsFunction, 31
AddKernelObjectFunctorial
for IsCapCategory, IsFunction, 32
AddLambdaElimination
for IsCapCategory, IsFunction, 95
AddLambdaIntroduction
for IsCapCategory, IsFunction, 95
AddLeftDistributivityExpandingWithGivenObjects
for IsCapCategory, IsFunction, 81
AddLeftDistributivityFactoringWithGivenObjects
for IsCapCategory, IsFunction, 81
AddLeftUnitorInverseWithGivenTensorProduct
for IsCapCategory, IsFunction, 79
AddLeftUnitorWithGivenTensorProduct
for IsCapCategory, IsFunction, 79
AddLift
for IsCapCategory, IsFunction, 22
AddLiftAlongMonomorphism
for IsCapCategory, IsFunction, 21
AddMonoidalPostComposeMorphismWithGivenObjects
for IsCapCategory, IsFunction, 87
AddMonoidalPreComposeMorphismWithGivenObjects
for IsCapCategory, IsFunction, 87
AddMorphismFromBidualWithGivenBidual
for IsCapCategory, IsFunction, 97
AddMorphismFromCoimageToImageWithGivenObjects
for IsCapCategory, IsFunction, 71
AddMorphismFromInternalHomToTensorProductWithGivenObjects
for IsCapCategory, IsFunction, 92
AddMorphismFromTensorProductToInternalHomWithGivenObjects
for IsCapCategory, IsFunction, 91
AddMorphismFunction
for IsCapFunctor, IsFunction, 11
AddMorphismToBidualWithGivenBidual
for IsCapCategory, IsFunction, 90
AddNaturalTransformationFunction
for IsCapNaturalTransformation, IsFunction, 12
AddObjectFunction
for IsCapFunctor, IsFunction, 11
AddOperationsToDerivationGraph
for IsDerivedMethodGraph, IsDenseList, 101
AddPostCompose
for IsCapCategory, IsFunction, 20
AddPreCompose
for IsCapCategory, IsFunction, 20
AddPrimitiveOperation
for IsOperationWeightList, IsString, IsInt, 104
AddProjectionInFactorOfDirectProduct
for IsCapCategory, IsFunction, 54
AddProjectionInFactorOfDirectProductWithGivenDirectProduct
for IsCapCategory, IsFunction, 55
AddProjectionInFactorOfDirectSum

for IsCapCategory, IsFunction, 47
AddProjectionInFactorOfDirectSumWithGivenDirectSum
for IsCapCategory, IsFunction, 47
AddProjectionInFactorOfFiberProduct
for IsCapCategory, IsFunction, 60
AddProjectionInFactorOfFiberProductWithGivenFiberProduct
for IsCapCategory, IsFunction, 60
AddPushout
for IsCapCategory, IsFunction, 65
AddPushoutFunctorial
for IsCapCategory, IsFunction, 66
AddRankMorphism
for IsCapCategory, IsFunction, 93
AddRightDistributivityExpandingWithGivenObjects
for IsCapCategory, IsFunction, 82
AddRightDistributivityFactoringWithGivenObjects
for IsCapCategory, IsFunction, 82
AddRightUnitorInverseWithGivenTensorProduct
for IsCapCategory, IsFunction, 80
AddRightUnitorWithGivenTensorProduct
for IsCapCategory, IsFunction, 80
AddTensorProductDualityCompatibilityMorphismWithGivenObjects
for IsCapCategory, IsFunction, 91
AddTensorProductInternalHomCompatibilityMorphismInverseWithGivenObjects
for IsCapCategory, IsFunction, 97
AddTensorProductInternalHomCompatibilityMorphismWithGivenObjects
for IsCapCategory, IsFunction, 90
AddTensorProductOnMorphismsWithGivenTensorProducts
for IsCapCategory, IsFunction, 77
AddTensorProductOnObjects
for IsCapCategory, IsFunction, 76
AddTensorProductToInternalHomAdjunctionMap
for IsCapCategory, IsFunction, 86
AddTensorUnit
for IsCapCategory, IsFunction, 78
AddTerminalObject
for IsCapCategory, IsFunction, 40
AddTerminalObjectFunctorial
for IsCapCategory, IsFunction, 40
AddTraceMap
for IsCapCategory, IsFunction, 93
AddUniversalMorphismFromCoproduct
for IsCapCategory, IsFunction, 52
AddUniversalMorphismFromCoproductWithGivenCoproduct
for IsCapCategory, IsFunction, 52
AddUniversalMorphismFromDirectSum
for IsCapCategory, IsFunction, 48
AddUniversalMorphismFromDirectSumWithGivenDirectSum
for IsCapCategory, IsFunction, 48
AddUniversalMorphismFromImage
for IsCapCategory, IsFunction, 70
AddUniversalMorphismFromImageWithGivenImageObject
for IsCapCategory, IsFunction, 70
AddUniversalMorphismFromInitialObject
for IsCapCategory, IsFunction, 42
AddUniversalMorphismFromInitialObjectWithGivenInitialObject
for IsCapCategory, IsFunction, 42
AddUniversalMorphismFromPushout
for IsCapCategory, IsFunction, 66
AddUniversalMorphismFromPushoutWithGivenPushout
for IsCapCategory, IsFunction, 66
AddUniversalMorphismFromZeroObject
for IsCapCategory, IsFunction, 38
AddUniversalMorphismFromZeroObjectWithGivenZeroObject
for IsCapCategory, IsFunction, 38
AddUniversalMorphismIntoCoimage
for IsCapCategory, IsFunction, 74
AddUniversalMorphismIntoCoimageWithGivenCoimage
for IsCapCategory, IsFunction, 74
AddUniversalMorphismIntoDirectProduct
for IsCapCategory, IsFunction, 55
AddUniversalMorphismIntoDirectProductWithGivenDirectProduct
for IsCapCategory, IsFunction, 55
AddUniversalMorphismIntoDirectSum

for IsCapCategory, IsFunction, 47
AddUniversalMorphismIntoDirectSumWithGivenDirectSum
for IsCapCategory, IsFunction, 47
AddUniversalMorphismIntoFiberProduct
for IsCapCategory, IsFunction, 60
AddUniversalMorphismIntoFiberProductWithGivenFiberProduct
for IsCapCategory, IsFunction, 60
AddUniversalMorphismIntoTerminalObject
for IsCapCategory, IsFunction, 40
AddUniversalMorphismIntoTerminalObjectWithGivenTerminalObject
for IsCapCategory, IsFunction, 40
AddUniversalMorphismIntoZeroObject
for IsCapCategory, IsFunction, 38
AddUniversalMorphismIntoZeroObjectWithGivenZeroObject
for IsCapCategory, IsFunction, 38
AddUniversalPropertyOfDual
for IsCapCategory, IsFunction, 94
AddVerticalPostCompose
for IsCapCategory, IsFunction, 28
AddVerticalPreCompose
for IsCapCategory, IsFunction, 28
AddWithGivenDerivationPairToCAP, 102
AddZeroMorphism
for IsCapCategory, IsFunction, 18
AddZeroObject
for IsCapCategory, IsFunction, 37
ApplyFunctor, 11
ApplyNaturalTransformation, 12
AsCapCategory
for IsCapCategoryAsCatObject, 10
AsCatObject
for IsCapCategory, 10
AssociatorLeftToRight
for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, 78
AssociatorLeftToRightWithGivenTensorProducts
for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, 78
AssociatorRightToLeft
for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, 77
AssociatorRightToLeftWithGivenTensorProducts
for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, 77
AstrictionToCoimage
for IsCapCategoryMorphism, 73
for IsCapCategoryObject, 73
AstrictionToCoimageWithGivenCoimage
for IsCapCategoryMorphism, IsCapCategoryObject, 73

Braiding
for IsCapCategoryObject, IsCapCategoryObject, 83
BraidingInverse
for IsCapCategoryObject, IsCapCategoryObject, 83
BraidingInverseWithGivenTensorProducts
for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, 83
BraidingWithGivenTensorProducts
for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, 83

CanCompute
for IsCapCategory, IsString, 7
CapCat, 9
CapCategory
for IsCapCategoryMorphism, 13
for IsCapCategoryObject, 24
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_MERGE_FILTER_LISTS, 113
CategoryFilter
for IsCapCategory, 6
for IsDerivedMethod, 99
CategoryOfOperationWeightList
for IsOperationWeightList, 103
CellFilter
for IsCapCategory, 6
CheckConstructivenessOfCategory
for IsCapCategory, IsString, 7
CoastrictionToImage
for IsCapCategoryMorphism, 68
for IsCapCategoryObject, 68
CoastrictionToImageWithGivenImageObject
for IsCapCategoryMorphism, IsCapCategoryObject, 68
CoevaluationForDual
for IsCapCategoryObject, 89
CoevaluationForDualWithGivenTensorProduct
for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, 89
CoevaluationMorphism
for IsCapCategoryObject, IsCapCategoryObject, 85
CoevaluationMorphismWithGivenRange
for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, 85
Coimage
for IsCapCategoryMorphism, 72
CoimageProjection
for IsCapCategoryMorphism, 72
for IsCapCategoryObject, 72
CoimageProjectionWithGivenCoimage
for IsCapCategoryMorphism, IsCapCategoryObject, 72
CokernelColift
for IsCapCategoryMorphism, IsCapCategoryMorphism, 34
for IsCapCategoryObject, IsCapCategoryMorphism, 34
CokernelColiftWithGivenCokernelObject
for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject, 34
CokernelFunctorial
for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, 35
for IsList, 35
CokernelObject
for IsCapCategoryMorphism, 33
CokernelProjection
for IsCapCategoryMorphism, 33
for IsCapCategoryObject, 33
CokernelProjectionWithGivenCokernelObject
for IsCapCategoryMorphism, IsCapCategoryObject, 33
Colift
for IsCapCategoryMorphism, IsCapCategoryMorphism, 22
ColiftAlongEpimorphism
for IsCapCategoryMorphism, IsCapCategoryMorphism, 21
Contains
for IsStringMinHeap, IsString, 106
Coproduct
for IsCapCategoryObject, IsCapCategoryObject, 50
for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, 50
for IsList, 50
CoproductFunctorial
for IsList, 52
CoproductFunctorialOp
for IsList, IsCapCategoryMorphism, 52
CoproductOp
for IsList, IsCapCategoryObject, 50
CreateCapCategory, 6
for IsString, 6
CurrentOperationWeight
for IsOperationWeightList, IsString, 103

DeactivateDerivationInfo, 98
DeclareAttributeWithToDoForIsWellDefined, 113
DeclareFamilyProperty, 113
DecreaseKey
for IsStringMinHeap, IsString, IsInt, 105

DerivationFunctionsWithExtraFilters
for IsDerivedMethod, 99
DerivationGraph
for IsOperationWeightList, 103
DerivationInfo, 98
DerivationName
for IsDerivedMethod, 99
DerivationOfOperation
for IsOperationWeightList, IsString, 104
DerivationResultWeight
for IsDerivedMethod, IsDenseList, 100
DerivationsOfOperation
for IsDerivedMethodGraph, IsString, 102
DerivationsUsingOperation
for IsDerivedMethodGraph, IsString, 102
DerivationWeight
for IsDerivedMethod, 99
DirectProductFunctorial
for IsList, 55
DirectProductFunctorialOp
for IsList, IsCapCategoryMorphism, 55
DirectProductOp
for IsList, IsCapCategoryObject, 53
DirectSumCodiagonalDifference
for IsList, 63
DirectSumCodiagonalDifferenceOp
for IsList, IsCapCategoryMorphism, 63
DirectSumDiagonalDifference
for IsList, 57
DirectSumDiagonalDifferenceOp
for IsList, IsCapCategoryMorphism, 57
DirectSumFunctorial
for IsList, 49
DirectSumFunctorialOp
for IsList, IsCapCategoryMorphism, 49
DirectSumOp
for IsList, IsCapCategoryObject, 43
DirectSumProjectionInPushout
for IsList, 63
DirectSumProjectionInPushoutOp
for IsList, IsCapCategoryMorphism, 63
DisableBasicOperationTypeCheck, 8
DualOnMorphisms
for IsCapCategoryMorphism, 88
DualOnMorphismsWithGivenDuals
for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryObject, 88
DualOnObjects
for IsCapCategoryObject, 88

EnableBasicOperationTypeCheck, 8
EvaluationForDual
for IsCapCategoryObject, 88
EvaluationForDualWithGivenTensorProduct
for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, 88
EvaluationMorphism
for IsCapCategoryObject, IsCapCategoryObject, 85
EvaluationMorphismWithGivenSource
for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, 85
ExtractMin
for IsStringMinHeap, 105

FiberProduct, 58
FiberProductEmbeddingInDirectSum
for IsList, 58
FiberProductEmbeddingInDirectSumOp
for IsList, IsCapCategoryMorphism, 58
FiberProductFunctorial
for IsList, 61
FiberProductFunctorialOp
for IsList, IsCapCategoryMorphism, 61
FiberProductOp
for IsList, IsCapCategoryMorphism, 58
FunctorMorphismOperation
for IsCapFunctor, 11
FunctorObjectOperation
for IsCapFunctor, 11

Heapify
for IsStringMinHeap, IsPosInt, 106
HeapSize
for IsStringMinHeap, 106
HorizontalPostCompose
for IsCapCategoryTwoCell, IsCapCategoryTwoCell, 28
HorizontalPreCompose
for IsCapCategoryTwoCell, IsCapCategoryTwoCell, 27
HorizontalPreComposeFunctorWithNaturalTransformation

for IsCapFunctor, IsCapNaturalTransformation, 12
HorizontalPreComposeNaturalTransformationWithFunctor
for IsCapNaturalTransformation, IsCapFunctor, 12

IdentityMorphism
for IsCapCategoryObject, 19
IdentityTwoCell
for IsCapCategoryMorphism, 27
ImageEmbedding
for IsCapCategoryMorphism, 68
for IsCapCategoryObject, 68
ImageEmbeddingWithGivenImageObject
for IsCapCategoryMorphism, IsCapCategoryObject, 68
ImageObject
for IsCapCategoryMorphism, 67
InitialObject
for IsCapCategory, 41
for IsCapCategoryCell, 41
InitialObjectFunctorial
for IsCapCategory, 42
InjectionOfCofactor, 75
InjectionOfCofactorOfCoproduct
for IsList, IsInt, 50
InjectionOfCofactorOfCoproductOp
for IsList, IsInt, IsCapCategoryObject, 50
InjectionOfCofactorOfCoproductWithGivenCoproduct
for IsList, IsInt, IsCapCategoryObject, 50
InjectionOfCofactorOfDirectSum
for IsList, IsInt, 43
InjectionOfCofactorOfDirectSumOp
for IsList, IsInt, IsCapCategoryObject, 43
InjectionOfCofactorOfDirectSumWithGivenDirectSum
for IsList, IsInt, IsCapCategoryObject, 44
InjectionOfCofactorOfPushout
for IsList, IsInt, 64
InjectionOfCofactorOfPushoutOp
for IsList, IsInt, IsCapCategoryMorphism, 64
InjectionOfCofactorOfPushoutWithGivenPushout
for IsList, IsInt, IsCapCategoryObject, 64
InstallDerivationForCategory
for IsDerivedMethod, IsPosInt, IsCapCategory, 100
InstallDerivationsUsingOperation
for IsOperationWeightList, IsString, 104
InstallFunctor
for IsCapFunctor, IsString, 11
InstallMethodWithToDoForIsWellDefined, 112
InstallNaturalTransformation
for IsCapNaturalTransformation, IsString, 12
InstallSetWithToDoForIsWellDefined
for IsObject, IsString, IsList, 113
InternalHomOnMorphisms
for IsCapCategoryMorphism, IsCapCategoryMorphism, 84
InternalHomOnMorphismsWithGivenInternalHoms
for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject, 84
InternalHomOnObjects
for IsCapCategoryObject, IsCapCategoryObject, 84
InternalHomToTensorProductAdjunctionMap
for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryMorphism, 86
InverseMorphismFromCoimageToImage
for IsCapCategoryMorphism, 71
InverseMorphismFromCoimageToImageWithGivenObjects
for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryObject, 71
IsApplicableToCategory
for IsDerivedMethod, IsCapCategory, 99
IsCapCategory
for IsObject, 5
IsCapCategoryAsCatObject
for IsCapCategoryObject, 9
IsCapCategoryCell
for IsObject, 5
IsCapCategoryMorphism
for IsCapCategoryCell and IsAdditiveElementWithInverse, 6
IsCapCategoryObject

for IsCapCategoryCell, 5
IsCapCategoryTwoCell
for IsCapCategoryCell, 6
IsCapFunctor
for IsCapCategoryMorphism, 9
IsCapNaturalTransformation
for IsCapCategoryTwoCell, 9
IsCodominating
for IsCapCategoryMorphism, IsCapCategoryMorphism, 19
IsCongruentForMorphisms
for IsCapCategoryMorphism, IsCapCategoryMorphism, 16
IsDerivedMethod
for IsObject, 98
IsDerivedMethodGraph
for IsObject, 100
IsDominating
for IsCapCategoryMorphism, IsCapCategoryMorphism, 18
IsEmptyHeap
for IsStringMinHeap, 105
IsEqualAsFactorobjects
for IsCapCategoryMorphism, IsCapCategoryMorphism, 18
IsEqualAsSubobjects
for IsCapCategoryMorphism, IsCapCategoryMorphism, 18
IsEqualForCacheForMorphisms
for IsCapCategoryMorphism, IsCapCategoryMorphism, 23
IsEqualForCacheForObjects
for IsCapCategoryObject, IsCapCategoryObject, 26
IsEqualForMorphisms
for IsCapCategoryMorphism, IsCapCategoryMorphism, 16
IsEqualForMorphismsOnMor
for IsCapCategoryMorphism, IsCapCategoryMorphism, 16
IsEqualForObjects
for IsCapCategoryObject, IsCapCategoryObject, 24
IsIdenticalToIdentityMorphism
for IsCapCategoryMorphism, 15
IsIdenticalToZeroMorphism
for IsCapCategoryMorphism, 15
IsomorphismFromCoimageToCokernelOfKernel
for IsCapCategoryMorphism, 71
IsomorphismFromCokernelOfDiagonalDifferenceToPushout
for IsList, 62
IsomorphismFromCokernelOfDiagonalDifferenceToPushoutOp
for IsList, IsCapCategoryMorphism, 62
IsomorphismFromCokernelOfKernelToCoimage
for IsCapCategoryMorphism, 72
IsomorphismFromCoproductToDirectSum
for IsList, 46
IsomorphismFromCoproductToDirectSumOp
for IsList, IsCapCategoryObject, 46
IsomorphismFromDirectProductToDirectSum
for IsList, 45
IsomorphismFromDirectProductToDirectSumOp
for IsList, IsCapCategoryObject, 45
IsomorphismFromDirectSumToCoproduct
for IsList, 46
IsomorphismFromDirectSumToCoproductOp
for IsList, IsCapCategoryObject, 46
IsomorphismFromDirectSumToDirectProduct
for IsList, 45
IsomorphismFromDirectSumToDirectProductOp
for IsList, IsCapCategoryObject, 45
IsomorphismFromDualToInternalHom
for IsCapCategoryObject, 93
IsomorphismFromFiberProductToKernelOfDiagonalDifference
for IsList, 56
IsomorphismFromFiberProductToKernelOfDiagonalDifferenceOp
for IsList, IsCapCategoryMorphism, 56
IsomorphismFromImageObjectToKernelOfCokernel
for IsCapCategoryMorphism, 67
IsomorphismFromInitialObjectToZeroObject
for IsCapCategory, 37
IsomorphismFromInternalHomToDual

for IsCapCategoryObject, 94
IsomorphismFromInternalHomToObject
for IsCapCategoryObject, 96
IsomorphismFromInternalHomToObject-WithGivenInternalHom
for IsCapCategoryObject, IsCapCategory-Object, 96
IsomorphismFromInternalHomToTensor-Product
for IsCapCategoryObject, IsCapCategory-Object, 93
IsomorphismFromKernelOfCokernelTo-ImageObject
for IsCapCategoryMorphism, 67
IsomorphismFromKernelOfDiagonal-DifferenceToFiberProduct
for IsList, 57
IsomorphismFromKernelOfDiagonal-DifferenceToFiberProductOp
for IsList, IsCapCategoryMorphism, 57
IsomorphismFromObjectToInternalHom
for IsCapCategoryObject, 95
IsomorphismFromObjectToInternalHom-WithGivenInternalHom
for IsCapCategoryObject, IsCapCategory-Object, 95
IsomorphismFromPushoutToCokernelOf-DiagonalDifference
for IsList, 61
IsomorphismFromPushoutToCokernelOf-DiagonalDifferenceOp
for IsList, IsCapCategoryMorphism, 62
IsomorphismFromTensorProductTo-InternalHom
for IsCapCategoryObject, IsCapCategory-Object, 92
IsomorphismFromTerminalObjectToZero-Object
for IsCapCategory, 37
IsomorphismFromZeroObjectToInitial-Object
for IsCapCategory, 37
IsomorphismFromZeroObjectToTerminal-Object
for IsCapCategory, 37
IsOperationWeightList
for IsObject, 103
IsStringMinHeap
for IsObject, 105
IsWellDefined
for IsCapCategoryCell, 8
IsWellDefinedForMorphisms
for IsCapCategoryMorphism, 20
IsWellDefinedForObjects
for IsCapCategoryObject, 26
IsWellDefinedForTwoCells
for IsCapCategoryTwoCell, 29
IsZeroForMorphisms
for IsCapCategoryMorphism, 17
IsZeroForObjects
for IsCapCategoryObject, 25

KernelEmbedding
for IsCapCategoryMorphism, 30
for IsCapCategoryObject, 30
KernelEmbeddingWithGivenKernelObject
for IsCapCategoryMorphism, IsCapCategoryObject, 31
KernelLift
for IsCapCategoryMorphism, IsCapCategoryMorphism, 31
for IsCapCategoryObject, IsCapCategory-Morphism, 31
KernelLiftWithGivenKernelObject
for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject, 31
KernelObject
for IsCapCategoryMorphism, 30
KernelObjectFunctorial
for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, 32
for IsList, 32

LambdaElimination
for IsCapCategoryObject, IsCapCategory-Object, IsCapCategoryMorphism, 95
LambdaIntroduction
for IsCapCategoryMorphism, 94
LeftDistributivityExpanding
for IsCapCategoryObject, IsList, 80
LeftDistributivityExpandingWithGiven-Objects

for IsCapCategoryObject, IsCapCategoryObject, IsList, IsCapCategoryObject, 80
LeftDistributivityFactoring
for IsCapCategoryObject, IsList, 81
LeftDistributivityFactoringWithGivenObjects
for IsCapCategoryObject, IsCapCategoryObject, IsList, IsCapCategoryObject, 81
LeftUnitor
for IsCapCategoryObject, 78
LeftUnitorInverse
for IsCapCategoryObject, 79
LeftUnitorInverseWithGivenTensorProduct
for IsCapCategoryObject, IsCapCategoryObject, 79
LeftUnitorWithGivenTensorProduct
for IsCapCategoryObject, IsCapCategoryObject, 79
Lift
for IsCapCategoryMorphism, IsCapCategoryMorphism, 22
LiftAlongMonomorphism
for IsCapCategoryMorphism, IsCapCategoryMorphism, 21

MakeDerivation
for IsString, IsFunction, IsDenseList,IsPosInt, IsDenseList, IsFunction, 98
MakeDerivationGraph
for IsDenseList, 101
MakeOperationWeightList
for IsCapCategory, IsDerivedMethodGraph, 103
MonoidalPostComposeMorphism
for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, 87
MonoidalPostComposeMorphismWithGivenObjects
for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, 87
MonoidalPreComposeMorphism
for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, 86
MonoidalPreComposeMorphismWithGivenObjects
for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, 87
MorphismBetweenDirectSums
for IsList, 46
MorphismBetweenDirectSumsOp
for IsList, IsInt, IsInt, IsCapCategoryMorphism, 46
MorphismCache
for IsCapFunctor, 112
MorphismFilter
for IsCapCategory, 7
MorphismFromBidual
for IsCapCategoryObject, 97
MorphismFromBidualWithGivenBidual
for IsCapCategoryObject, IsCapCategoryObject, 97
MorphismFromCoimageToImage
for IsCapCategoryMorphism, 70
MorphismFromCoimageToImageWithGivenObjects
for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryObject, 70
MorphismFromInternalHomToTensorProduct
for IsCapCategoryObject, IsCapCategoryObject, 92
MorphismFromInternalHomToTensorProductWithGivenObjects
for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, 92
MorphismFromTensorProductToInternalHom
for IsCapCategoryObject, IsCapCategoryObject, 91
MorphismFromTensorProductToInternalHomWithGivenObjects
for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, 91
MorphismFromZeroObject
for IsCapCategoryObject, 36
MorphismIntoZeroObject
for IsCapCategoryObject, 36
MorphismToBidual

for IsCapCategoryObject, 89
MorphismToBidualWithGivenBidual
for IsCapCategoryObject, IsCapCategoryObject, 89

Name
for IsCapNaturalTransformation, 11
NaturalTransformation
for IsCapFunctor, IsCapFunctor, 12

ObjectCache
for IsCapFunctor, 112
ObjectFilter
for IsCapCategory, 7
Operations
for IsDerivedMethodGraph, 102
OperationWeightUsingDerivation
for IsOperationWeightList, IsDerivedMethod, 103

PostCompose
for IsCapCategoryMorphism, IsCapCategoryMorphism, 20
for IsList, 20
PreCompose
for IsCapCategoryMorphism, IsCapCategoryMorphism, 19
for IsList, 20
PrintDerivationTree
for IsOperationWeightList, IsString, 104
PrintTree
for IsObject, IsFunction, IsFunction, 104
PrintTreeRec
for IsObject, IsFunction, IsFunction, IsInt, 105
ProjectionInFactor, 75
ProjectionInFactorOfDirectProduct
for IsList, IsInt, 53
ProjectionInFactorOfDirectProductOp
for IsList, IsInt, IsCapCategoryObject, 53
ProjectionInFactorOfDirectProductWithGivenDirectProduct
for IsList, IsInt, IsCapCategoryObject, 53
ProjectionInFactorOfDirectSum
for IsList, IsInt, 43
ProjectionInFactorOfDirectSumOp
for IsList, IsInt, IsCapCategoryObject, 43
ProjectionInFactorOfDirectSumWithGivenDirectSum
for IsList, IsInt, IsCapCategoryObject, 43
ProjectionInFactorOfFiberProduct
for IsList, IsInt, 58
ProjectionInFactorOfFiberProductOp
for IsList, IsInt, IsCapCategoryMorphism, 59
ProjectionInFactorOfFiberProductWithGivenFiberProduct
for IsList, IsInt, IsCapCategoryObject, 59
Pushout
for IsCapCategoryMorphism, IsCapCategoryMorphism, 64
for IsList, 63
PushoutFunctorial
for IsList, 66
PushoutFunctorialOp
for IsList, IsCapCategoryMorphism, 66
PushoutOp
for IsList, IsCapCategoryMorphism, 64

Range
for IsCapCategoryMorphism, 13
for IsCapCategoryTwoCell, 27
RankMorphism
for IsCapCategoryObject, 93
Reevaluate
for IsOperationWeightList, 104
RightDistributivityExpanding
for IsList, IsCapCategoryObject, 81
RightDistributivityExpandingWithGivenObjects
for IsCapCategoryObject, IsList, IsCapCategoryObject, IsCapCategoryObject, 81
RightDistributivityFactoring
for IsList, IsCapCategoryObject, 82
RightDistributivityFactoringWithGivenObjects
for IsCapCategoryObject, IsList, IsCapCategoryObject, IsCapCategoryObject, 82
RightUnitor
for IsCapCategoryObject, 79
RightUnitorInverse
for IsCapCategoryObject, 80
RightUnitorInverseWithGivenTensorProduct

for IsCapCategoryObject, IsCapCategoryObject, 80
RightUnitorWithGivenTensorProduct
for IsCapCategoryObject, IsCapCategoryObject, 79

Source
for IsCapCategoryMorphism, 13
for IsCapCategoryTwoCell, 27
StringMinHeap, 105
Swap
for IsStringMinHeap, IsPosInt, IsPosInt, 106

TargetOperation
for IsDerivedMethod, 100
TensorProductDualityCompatibilityMorphism
for IsCapCategoryObject, IsCapCategoryObject, 90
TensorProductDualityCompatibilityMorphismWithGivenObjects
for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, 91
TensorProductInternalHomCompatibilityMorphism
for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, 90
TensorProductInternalHomCompatibilityMorphismInverse
for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, 96
TensorProductInternalHomCompatibilityMorphismInverseWithGivenObjects
for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsList, 96
TensorProductInternalHomCompatibilityMorphismWithGivenObjects
for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, IsList, 90
TensorProductOnMorphisms
for IsCapCategoryMorphism, IsCapCategoryMorphism, 77
TensorProductOnMorphismsWithGivenTensorProducts
for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject, 77
TensorProductOnObjects
for IsCapCategoryObject, IsCapCategoryObject, 76
TensorProductToInternalHomAdjunctionMap
for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryMorphism, 86
TensorUnit
for IsCapCategory, 78
TerminalObject
for IsCapCategory, 39
for IsCapCategoryCell, 39
TerminalObjectFunctorial
for IsCapCategory, 40
TraceMap
for IsCapCategoryMorphism, 93
TwoCellFilter
for IsCapCategory, 7

UniversalMorphismFromCoproduct, 51
UniversalMorphismFromCoproductOp
for IsList, IsList, IsCapCategoryObject, 51
UniversalMorphismFromCoproductWithGivenCoproduct
for IsList, IsList, IsCapCategoryObject, 51
UniversalMorphismFromDirectSum, 44
UniversalMorphismFromDirectSumOp
for IsList, IsList, IsCapCategoryObject, 45
UniversalMorphismFromDirectSumWithGivenDirectSum
for IsList, IsList, IsCapCategoryObject, 45
UniversalMorphismFromImage
for IsCapCategoryMorphism, IsList, 68
UniversalMorphismFromImageWithGivenImageObject
for IsCapCategoryMorphism, IsList, IsCapCategoryObject, 69
UniversalMorphismFromInitialObject
for IsCapCategoryObject, 41
UniversalMorphismFromInitialObjectWithGivenInitialObject

for IsCapCategoryObject, IsCapCategoryObject, 41
UniversalMorphismFromPushout, 64
UniversalMorphismFromPushoutOp
for IsList, IsList, IsCapCategoryMorphism, 65
UniversalMorphismFromPushoutWithGivenPushout
for IsList, IsList, IsCapCategoryObject, 65
UniversalMorphismFromZeroObject
for IsCapCategoryObject, 36
UniversalMorphismFromZeroObjectWithGivenZeroObject
for IsCapCategoryObject, IsCapCategoryObject, 36
UniversalMorphismIntoCoimage
for IsCapCategoryMorphism, IsList, 73
UniversalMorphismIntoCoimageWithGivenCoimage
for IsCapCategoryMorphism, IsList, IsCapCategoryObject, 73
UniversalMorphismIntoDirectProduct, 54
UniversalMorphismIntoDirectProductOp
for IsList, IsList, IsCapCategoryObject, 54
UniversalMorphismIntoDirectProductWithGivenDirectProduct
for IsList, IsList, IsCapCategoryObject, 54
UniversalMorphismIntoDirectSum, 44
UniversalMorphismIntoDirectSumOp
for IsList, IsList, IsCapCategoryObject, 44
UniversalMorphismIntoDirectSumWithGivenDirectSum
for IsList, IsList, IsCapCategoryObject, 44
UniversalMorphismIntoFiberProduct, 59
UniversalMorphismIntoFiberProductOp
for IsList, IsList, IsCapCategoryMorphism, 59
UniversalMorphismIntoFiberProductWithGivenFiberProduct
for IsList, IsList, IsCapCategoryObject, 59
UniversalMorphismIntoTerminalObject
for IsCapCategoryObject, 40
UniversalMorphismIntoTerminalObjectWithGivenTerminalObject
for IsCapCategoryObject, IsCapCategoryObject, 40
UniversalMorphismIntoZeroObject
for IsCapCategoryObject, 37
UniversalMorphismIntoZeroObjectWithGivenZeroObject
for IsCapCategoryObject, IsCapCategoryObject, 37
UniversalPropertyOfDual
for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryMorphism, 94
UsedOperationMultiples
for IsDerivedMethod, 100
UsedOperations
for IsDerivedMethod, 100
UsedOperationsWithMultiples
for IsDerivedMethod, 100

VerticalPostCompose
for IsCapCategoryTwoCell, IsCapCategoryTwoCell, 28
VerticalPreCompose
for IsCapCategoryTwoCell, IsCapCategoryTwoCell, 28

WasCreatedAsCokernelObject, 114
WasCreatedAsCoproduct, 115
WasCreatedAsDirectProduct, 115
WasCreatedAsDirectSum, 115
WasCreatedAsFiberProduct, 115
WasCreatedAsImageObject, 116
WasCreatedAsInitialObject, 115
WasCreatedAsKernelObject, 114
WasCreatedAsPushout, 115
WasCreatedAsTerminalObject, 114
WasCreatedAsZeroObject, 114

ZeroMorphism
for IsCapCategoryObject, IsCapCategoryObject, 17
ZeroObject
for IsCapCategory, 36
for IsCapCategoryCell, 36