CAP

Categories, Algorithms, Programming

2019.06.07

7 June 2019

Sebastian Gutsche

Sebastian Posur

Øystein Skartsæterhagen

Sebastian Gutsche
Email: gutsche@mathematik.uni-siegen.de
Homepage: http://www.uni-siegen.de/fb6/rmi/
Address: Department Mathematik
Universität Siegen
Walter-Flex-Straße 3
57068 Siegen
Germany

Sebastian Posur
Email: sebastian.posur@uni-siegen.de
Homepage: https://sebastianpos.github.io
Address: Department Mathematik
Universität Siegen
Walter-Flex-Straße 3
57068 Siegen
Germany

Øystein Skartsæterhagen
Email: oysteini@math.ntnu.no
Homepage: http://www.math.ntnu.no/~oysteini/
Address: NTNU
Institutt for matematiske fag
7491 Trondheim
Norway

Contents

1 CAP Categories **5**
- 1.1 Categories . . . 5
- 1.2 Constructor . . . 6
- 1.3 Internal Attributes . . . 6
- 1.4 Logic switcher . . . 7
- 1.5 Tool functions . . . 8
- 1.6 Well-Definedness of Cells . . . 8
- 1.7 Caching . . . 8
- 1.8 Sanity checks . . . 9
- 1.9 Enable automatic calls of `Add` . . . 9
- 1.10 Performance tweaks . . . 9

2 Objects **11**
- 2.1 Attributes for the Type of Objects . . . 11
- 2.2 Equality for Objects . . . 11
- 2.3 Categorical Properties of Objects . . . 12
- 2.4 Random Objects . . . 12
- 2.5 Tool functions for caches . . . 13
- 2.6 Adding Objects to a Category . . . 14
- 2.7 Well-Definedness of Objects . . . 14
- 2.8 Projectives . . . 15
- 2.9 Injectives . . . 16

3 Morphisms **19**
- 3.1 Attributes for the Type of Morphisms . . . 19
- 3.2 Categorical Properties of Morphisms . . . 19
- 3.3 Random Morphisms . . . 20
- 3.4 Non-Categorical Properties of Morphisms . . . 24
- 3.5 Adding Morphisms to a Category . . . 25
- 3.6 Equality and Congruence for Morphisms . . . 26
- 3.7 Basic Operations for Morphisms in Ab-Categories . . . 27
- 3.8 Subobject and Factorobject Operations . . . 28
- 3.9 Identity Morphism and Composition of Morphisms . . . 30
- 3.10 Well-Definedness of Morphisms . . . 31
- 3.11 Lift/Colift . . . 32
- 3.12 Inverses . . . 35

3.13 Tool functions for caches . . . 35
3.14 Homomorphism structures . . . 36

4 Category 2-Cells **39**
4.1 Attributes for the Type of 2-Cells . . . 39
4.2 Identity 2-Cell and Composition of 2-Cells . . . 39
4.3 Well-Definedness for 2-Cells . . . 41

5 Category of Categories **42**
5.1 The Category Cat . . . 42
5.2 Categories . . . 42
5.3 Constructors . . . 43
5.4 Functors . . . 43
5.5 Natural transformations . . . 46

6 Universal Objects **48**
6.1 Kernel . . . 48
6.2 Cokernel . . . 51
6.3 Zero Object . . . 54
6.4 Terminal Object . . . 58
6.5 Initial Object . . . 60
6.6 Direct Sum . . . 61
6.7 Coproduct . . . 70
6.8 Direct Product . . . 73
6.9 Equalizer . . . 76
6.10 Coequalizer . . . 80
6.11 Fiber Product . . . 83
6.12 Pushout . . . 90
6.13 Image . . . 97
6.14 Coimage . . . 101

7 Add Functions **106**
7.1 Functions Installed by Add . . . 106
7.2 Add Method . . . 107
7.3 InstallAdd Function . . . 108
7.4 Install All Adds . . . 110
7.5 Prepare functions . . . 111

8 Managing Derived Methods **112**
8.1 Info Class . . . 112
8.2 Derivation Objects . . . 112
8.3 Derivation Graphs . . . 114
8.4 Managing Derivations in a Category . . . 117
8.5 Min Heaps for Strings . . . 119

9 Technical Details **121**
9.1 The Category Cat . . . 121
9.2 Install Functions for IsWellDefined . . . 121

10 Limits and Colimits **124**
10.1 Specification of Limits and Colimits . . . 124
10.2 Enhancing Limit Specifications . . . 125
10.3 Validating entries of a method name record which are part of a limit or colimit . . . 126

11 Examples and Tests **127**
11.1 Functors . . . 127
11.2 Homomorphism structure . . . 128
11.3 Spectral Sequences . . . 129
11.4 Liftable . . . 132
11.5 Monoidal Categories . . . 133
11.6 Opposite category . . . 135
11.7 Generalized Morphisms Category . . . 135
11.8 IsWellDefined . . . 142
11.9 Kernel . . . 142
11.10 FiberProduct . . . 144

Index **146**

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 IsAttributeStoringRep)

▷ `IsCapCategoryCell(`*object*`)` (filter)

Returns: `true` or `false`

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

1.1.3 IsCapCategoryObject (for IsCapCategoryCell)

▷ `IsCapCategoryObject(`*object*`)` (filter)

Returns: `true` or `false`

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

1.1.4 IsCapCategoryMorphism (for IsCapCategoryCell)

▷ `IsCapCategoryMorphism(object)` (filter)

Returns: `true` or `false`

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

1.1.5 IsCapCategoryTwoCell (for IsCapCategoryCell)

▷ `IsCapCategoryTwoCell(object)` (filter)

Returns: `true` or `false`

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

1.1.6 AddCategoricalProperty

▷ `AddCategoricalProperty(list)` (function)

Adds a categorical property to the list of CAP categorical properties. *list* must be a list containing one entry, if the property is self dual, or two, if the dual property has a different name. If the first entry of the list is empty and the second is a property name, the property is assumed to have no dual.

1.2 Constructor

1.2.1 CreateCapCategory

▷ `CreateCapCategory()` (operation)

Returns: a category

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

1.2.2 CreateCapCategory (for IsString)

▷ `CreateCapCategory(s)` (operation)

Returns: a category

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

1.3 Internal Attributes

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

1.3.1 CategoryFilter (for IsCapCategory)

▷ `CategoryFilter(C)` (attribute)

Returns: a filter

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

1.3.2 CellFilter (for IsCapCategory)

▷ `CellFilter(C)` (attribute)

Returns: a filter

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

1.3.3 ObjectFilter (for IsCapCategory)

▷ `ObjectFilter(C)` (attribute)

Returns: a filter

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

1.3.4 MorphismFilter (for IsCapCategory)

▷ `MorphismFilter(C)` (attribute)

Returns: a filter

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

1.3.5 TwoCellFilter (for IsCapCategory)

▷ `TwoCellFilter(C)` (attribute)

Returns: a filter

The argument is a category C. The output is a filter in which all 2-cells of C shall lie.

1.3.6 CommutativeRingOfLinearCategory (for IsCapCategory)

▷ `CommutativeRingOfLinearCategory(C)` (attribute)

Returns: a ring

The argument is a category C which is expected to lie in the filter `IsLinearCategoryOverCommutativeRing`. The output is a commutative ring over which the category is linear.

1.4 Logic switcher

1.4.1 CapCategorySwitchLogicOn

▷ `CapCategorySwitchLogicOn(C)` (function)

Activates the predicate implication logic for the category `C`.

1.4.2 CapCategorySwitchLogicOff

▷ `CapCategorySwitchLogicOff(C)` (function)

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 basic 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 basic 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 Caching

1.7.1 SetCachingOfCategory

▷ `SetCachingOfCategory(category, type)` (function)

Sets the caching of *category* to *type*.

1.7.2 SetCachingOfCategoryWeak

▷ `SetCachingOfCategoryWeak(category)` (function)
▷ `SetCachingOfCategoryCrisp(category)` (function)
▷ `DeactivateCachingOfCategory(category)` (function)

Sets the caching of *category* to `weak`, `crisp` or `none`, respectively.

1.7.3 SetDefaultCaching

▷ `SetDefaultCaching(type)` (function)
▷ `SetDefaultCachingWeak()` (function)
▷ `SetDefaultCachingCrisp()` (function)

▷ DeactivateDefaultCaching() (function)

Sets the default caching behaviour, all new categories will have their caching set to either weak, crisp, or none. The default at startup is weak.

1.8 Sanity checks

1.8.1 DisableInputSanityChecks

▷ DisableInputSanityChecks(*category*) (function)
▷ DisableOutputSanityChecks(*category*) (function)
▷ EnablePartialInputSanityChecks(*category*) (function)
▷ EnablePartialOutputSanityChecks(*category*) (function)
▷ EnableFullInputSanityChecks(*category*) (function)
▷ EnableFullOutputSanityChecks(*category*) (function)
▷ DisableSanityChecks(*category*) (function)
▷ EnablePartialSanityChecks(*category*) (function)
▷ EnableFullSanityChecks(*category*) (function)

Most operations can perform optional sanity checks on their arguments and results. The checks can either be partial (set by default), full, or disabled. With the following commands you can either enable the full checks, the partial checks or, for performance, disable the checks altogether. You can do this for input checks, output checks or for both at once.

1.9 Enable automatic calls of Add

1.9.1 EnableAddForCategoricalOperations

▷ EnableAddForCategoricalOperations(*C*) (function)
▷ DisableAddForCategoricalOperations(*C*) (function)

Enables/disables the automatic call of Add for the output of primitively added functions for the category *C*. If the automatic call of Add is disabled (default), the output of primitively added functions must belong to the correct category. If the automatic call of Add is enabled, the output of primitively added functions only has to be a GAP object lying in IsAttributeStoringRep (with suitable attributes Source and Range if the output should be a morphism or a twocell).

1.10 Performance tweaks

CAP has several settings which can improve the performance. In the following some of these are listed.

- DeactivateCachingOfCategory: see 1.7. This can either improve or degrade the performance depending on the concrete example.
- CapCategorySwitchLogicOff: see 1.4. This can either improve or degrade the performance depending on the concrete example.

- DisableSanityChecks: see 1.8.
- DisableAddForCategoricalOperations: see 1.9.
- DeactivateToDoList: see the package ToolsForHomalg.
- use ObjectifyObjectForCAPWithAttributes (2.6) instead of AddObject and ObjectifyMorphismForCAPWithAttributes (3.5) instead of AddMorphism.

Chapter 2

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.

2.1 Attributes for the Type of Objects

2.1.1 CapCategory (for IsCapCategoryObject)

▷ `CapCategory(a)` (attribute)

Returns: a category

The argument is an object a. The output is the category $\mathbf{C}$ to which a was added.

2.2 Equality for Objects

2.2.1 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`.

2.2.2 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`. $F : (a,b) \mapsto$ `IsEqualForObjects`(a,b).

2.3 Categorical Properties of Objects

2.3.1 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).

2.3.2 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).

2.3.3 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).

2.3.4 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).

2.3.5 IsZeroForObjects (for IsCapCategoryObject)

▷ `IsZeroForObjects(a)` (property)

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`.

2.3.6 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).

2.4 Random Objects

CAP provides two principal methods to generate random objects:

- *By integers*: The integer is simply a parameter that can be used to create a random object.

- *By lists*: The list is used when creating a random object would need more than one parameter. Lists offer more flexibility at the expense of the genericity of the methods. This happens because lists that are valid as input in some category may be not valid for other categories. Hence, these operations are not thought to be used in generic categorical algorithms.

2.4.1 RandomObjectByInteger (for IsCapCategory, IsInt)

▷ `RandomObjectByInteger(C, n)` (operation)

Returns: an object or `fail`

The arguments are a category C and an integer n. The output is a random object in C or `fail`.

2.4.2 AddRandomObjectByInteger (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `RandomObjectByInteger`. The function F maps (C,n) to `fail` or to a random object in C.

2.4.3 RandomObjectByList (for IsCapCategory, IsList)

▷ `RandomObjectByList(C, L)` (operation)

Returns: an object or `fail`

The arguments are a category C and a list L. The output is a random object in C or `fail`.

2.4.4 AddRandomObjectByList (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `RandomObjectByList`. The function F maps (C,L) to `fail` or to a random object in C.

2.4.5 RandomObject (for IsCapCategory, IsInt)

▷ `RandomObject(C, n)` (operation)

▷ `RandomObject(C, L)` (operation)

These are convenient methods and they, depending on the input, delegate to one of the above methods.

2.5 Tool functions for caches

2.5.1 IsEqualForCacheForObjects (for IsCapCategoryObject, IsCapCategoryObject)

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

Returns: true or false

Compares two objects in the cache

2.5.2 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.

2.6 Adding Objects to a Category

2.6.1 Add (for IsCapCategory, IsCapCategoryObject)

▷ `Add(category, object)` (operation)

Adds *object* as an object to *category*.

2.6.2 AddObject (for IsCapCategory, IsAttributeStoringRep)

▷ `AddObject(category, object)` (operation)

Adds *object* as an object to *category*. If *object* already lies in the filter `IsCapCategoryObject`, the operation `Add` (2.6.1) can be used instead.

2.6.3 AddObjectRepresentation (for IsCapCategory, IsObject)

▷ `AddObjectRepresentation(category, filter)` (operation)

The argument *filter* is used to create an object type for the category *category*, which is then used in `ObjectifyObjectForCAPWithAttributes` to objectify objects for this category.

2.6.4 ObjectifyObjectForCAPWithAttributes

▷ `ObjectifyObjectForCAPWithAttributes(object, category[, attribute1, value1, ...])` (function)

Objectifies the object *object* with the type created for objects in the category *category*. The type is created by passing a representation to `AddObjectRepresentation`. Objects which are objectified using this method do not have to be passed to the `AddObject` function.

2.7 Well-Definedness of Objects

2.7.1 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`.

2.7.2 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).

2.8 Projectives

For a given object A in an abelian category having enough projectives, the following commands allow us to compute some projective object P together with an epimorphism $\pi : P \to A$.

2.8.1 SomeProjectiveObject (for IsCapCategoryObject)

▷ `SomeProjectiveObject(A)` (attribute)

Returns: an object

The argument is an object A. The output is some projective object P for which there exists an epimorphism $\pi : P \to A$.

2.8.2 EpimorphismFromSomeProjectiveObject (for IsCapCategoryObject)

▷ `EpimorphismFromSomeProjectiveObject(A)` (attribute)

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

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

2.8.3 EpimorphismFromSomeProjectiveObjectWithGivenSomeProjectiveObject (for IsCapCategoryObject, IsCapCategoryObject)

▷ `EpimorphismFromSomeProjectiveObjectWithGivenSomeProjectiveObject(A, P)` (operation)

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

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

2.8.4 ProjectiveLift (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `ProjectiveLift(pi, epsilon)` (operation)

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

The arguments are a morphism $\pi : P \to A$ with P a projective, and an epimorphism $\varepsilon : B \to A$. The output is a morphism $\lambda : P \to B$ such that $\varepsilon \circ \lambda = \pi$.

2.8.5 AddSomeProjectiveObject (for IsCapCategory, IsFunction)

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

Returns: nothing

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

2.8.6 AddEpimorphismFromSomeProjectiveObject (for IsCapCategory, IsFunction)

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

Returns: nothing

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

2.8.7 AddEpimorphismFromSomeProjectiveObjectWithGivenSomeProjectiveObject (for IsCapCategory, IsFunction)

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

Returns: nothing

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

2.8.8 AddProjectiveLift (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `ProjectiveLift`. The function F maps a pair (π, ε) to a projective lift λ.

2.9 Injectives

For a given object A in an abelian category having enough injectives, the following commands allow us to compute some injective object I together with a monomorphism $\iota : A \to I$.

2.9.1 SomeInjectiveObject (for IsCapCategoryObject)

▷ `SomeInjectiveObject(A)` (attribute)

Returns: an object

The argument is an object A. The output is some injective object I for which there exists a monomorphism $\iota : A \to I$.

2.9.2 MonomorphismIntoSomeInjectiveObject (for IsCapCategoryObject)

▷ `MonomorphismIntoSomeInjectiveObject(A)` (attribute)

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

The argument is an object A. The output is a monomorphism $\iota : A \to I$ with I an injective object that equals the output of SomeInjectiveObject(A).

2.9.3 MonomorphismIntoSomeInjectiveObjectWithGivenSomeInjectiveObject (for IsCapCategoryObject, IsCapCategoryObject)

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

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

The arguments are an object A and an injective object I that equals the output of SomeInjectiveObject(A). The output is a monomorphism $\iota : A \to I$.

2.9.4 InjectiveColift (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `InjectiveColift(\iota, \beta)` (operation)

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

The arguments are a morphism $\iota : B \to A$ and $\beta : B \to I$ where I is an injective object. The output is a morphism $\lambda : A \to I$ such that $\lambda \circ \iota = \beta$.

2.9.5 AddSomeInjectiveObject (for IsCapCategory, IsFunction)

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

Returns: nothing

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

2.9.6 AddMonomorphismIntoSomeInjectiveObject (for IsCapCategory, IsFunction)

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

Returns: nothing

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

2.9.7 AddMonomorphismIntoSomeInjectiveObjectWithGivenSomeInjectiveObject (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operation adds the given function F to the category for the basic operation `AddMonomorphismIntoSomeInjectiveObjectWithGivenSomeInjectiveObject`. $F : (A,I) \mapsto \pi$.

2.9.8 AddInjectiveColift (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `InjectiveColift`. The function F maps a pair (ι, β) to an injective colift λ if it exists, and to `fail` otherwise.

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 **C** to which α was added.

3.1.2 Source (for IsCapCategoryMorphism)

▷ `Source(`*alpha*`)` (attribute)

Returns: an object

The argument is a morphism $\alpha : a \to b$. The output is its source a.

3.1.3 Range (for IsCapCategoryMorphism)

▷ `Range(`*alpha*`)` (attribute)

Returns: an object

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

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 Random Morphisms

CAP provides two principal methods to generate random morphisms with or without fixed source and range:

- *By integers*: The integer is simply a parameter that can be used to create a random morphism.

- *By lists*: The list is used when creating a random morphism would need more than one parameter. Lists offer more flexibility at the expense of the genericity of the methods. This happens because lists that are valid as input in some category may be not valid for other categories. Hence, these operations are not thought to be used in generic categorical algorithms.

3.3.1 RandomMorphismWithFixedSourceByInteger (for IsCapCategoryObject, IsInt)

▷ `RandomMorphismWithFixedSourceByInteger(a, n)` (operation)

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

The arguments are an object a in a category C and an integer n. The output is a random morphism $\alpha : a \to b$ for some object b in C or `fail`.

3.3.2 AddRandomMorphismWithFixedSourceByInteger (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operation adds the given function F to the category for the basic operation `RandomMorphismWithFixedSourceByInteger`. The function F maps (a,n) to `fail` or to a random morphism in C whose source is a.

3.3.3 RandomMorphismWithFixedSourceByList (for IsCapCategoryObject, IsList)

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

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

The arguments are an object a in a category C and a list L. The output is a random morphism $\alpha : a \to b$ for some object b in C or `fail`.

3.3.4 AddRandomMorphismWithFixedSourceByList (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operation adds the given function F to the category for the basic operation `RandomMorphismWithFixedSourceByList`. The function F maps (a,L) to `fail` or to a random morphism in C whose source is a.

3.3.5 RandomMorphismWithFixedRangeByInteger (for IsCapCategoryObject, IsInt)

▷ `RandomMorphismWithFixedRangeByInteger(b, n)` (operation)

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

The arguments are an object b in a category C and an integer n. The output is a random morphism $\alpha : a \to b$ for some object a in C or `fail`.

3.3.6 AddRandomMorphismWithFixedRangeByInteger (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operation adds the given function F to the category for the basic operation `RandomMorphismWithFixedRangeByInteger`. The function F maps (b,n) to `fail` or to a random morphism in C whose range is b.

3.3.7 RandomMorphismWithFixedRangeByList (for IsCapCategoryObject, IsList)

▷ `RandomMorphismWithFixedRangeByList(b, L)` (operation)

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

The arguments are an object b in a category C and a list L. The output is a random morphism $\alpha : a \to b$ for some object a in C or `fail`.

3.3.8 AddRandomMorphismWithFixedRangeByList (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operation adds the given function F to the category for the basic operation `RandomMorphismWithFixedRangeByList`. The function F maps (b,L) to `fail` or to a random morphism in C whose range is b.

3.3.9 RandomMorphismWithFixedSourceAndRangeByInteger (for IsCapCategoryObject, IsCapCategoryObject, IsInt)

▷ `RandomMorphismWithFixedSourceAndRangeByInteger(a, b, n)` (operation)

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

The arguments are two objects a and b in a category C and an integer n. The output is a random morphism $\alpha : a \to b$ in C or `fail`.

3.3.10 AddRandomMorphismWithFixedSourceAndRangeByInteger (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `RandomMorphismWithFixedSourceAndRangeByInteger`. The function F maps (a,b,n) to `fail` or to a random morphism in C from a to b.

3.3.11 RandomMorphismWithFixedSourceAndRangeByList (for IsCapCategoryObject, IsCapCategoryObject, IsList)

▷ `RandomMorphismWithFixedSourceAndRangeByList(a, b, L)` (operation)

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

This operation is not a CAP basic operation The arguments are two objects a and b in a category C and a list L. The output is a random morphism $\alpha : a \to b$ in C or `fail`.

3.3.12 AddRandomMorphismWithFixedSourceAndRangeByList (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `RandomMorphismWithFixedSourceAndRangeByList`. The function F maps (a,b,L) to `fail` or to a random morphism in C from a to b.

3.3.13 RandomMorphismByInteger (for IsCapCategory, IsInt)

▷ `RandomMorphismByInteger(C, n)` (operation)

Returns: a morphism or `fail`

The arguments are a category C and an integer n. The output is a random morphism in C or `fail`. If the methods `RandomObjectByInteger` and `RandomMorphismWithFixedSourceByInteger`(`RandomMorphismWithFixedRangeByInteger`) are added to the category C, then `RandomMorphismByInteger` can be derived to return a random morphism of complexity n with random source(range) of complexity n.

3.3.14 AddRandomMorphismByInteger (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operation adds the given function F to the category for the basic operation `RandomMorphismByInteger`. The function F maps (C,n) to `fail` or to a random morphism in C.

3.3.15 RandomMorphismByList (for IsCapCategory, IsList)

▷ `RandomMorphismByList(C, L)` (operation)

Returns: a morphism or `fail`

The arguments are a category C and a list L. The output is a random morphism in C or `fail`.

3.3.16 AddRandomMorphismByList (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operation adds the given function F to the category for the basic operation `RandomMorphismByList`. The function F maps (C,L) to `fail` or to a random morphism in C.

3.3.17 RandomMorphismWithFixedSource (for IsCapCategoryObject, IsInt)

▷ `RandomMorphismWithFixedSource(a, n)` (operation)

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

▷ RandomMorphismWithFixedRange(*b*, *n*) (operation)
▷ RandomMorphismWithFixedRange(*b*, *L*) (operation)
▷ RandomMorphismWithFixedSourceAndRange(a, *b*, *n*) (operation)
▷ RandomMorphismWithFixedSourceAndRange(a, *b*, *L*) (operation)
▷ RandomMorphism(*C*, *n*) (operation)
▷ RandomMorphism(*C*, *L*) (operation)

These are convenient methods and they, depending on the input, delegate to one of the above methods.

3.4 Non-Categorical Properties of Morphisms

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

3.4.1 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`.

3.4.2 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).

3.4.3 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`.

3.4.4 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).

3.4.5 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).

3.4.6 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).

3.5 Adding Morphisms to a Category

3.5.1 Add (for IsCapCategory, IsCapCategoryMorphism)

▷ `Add(category, morphism)` (operation)

Adds *morphism* as a morphism to *category*.

3.5.2 AddMorphism (for IsCapCategory, IsAttributeStoringRep)

▷ `AddMorphism(category, morphism)` (operation)

Adds *morphism* as a morphism to *category*. If *morphism* already lies in the filter `IsCapCategoryMorphism`, the operation `Add` (3.5.1) can be used instead.

3.5.3 AddMorphismRepresentation (for IsCapCategory, IsObject)

▷ `AddMorphismRepresentation(category, filter)` (operation)

The argument *filter* is used to create a morphism type for the category *category*, which is then used in `ObjectifyMorphismForCAPWithAttributes` to objectify morphisms for this category.

3.5.4 ObjectifyMorphismForCAPWithAttributes

▷ `ObjectifyMorphismForCAPWithAttributes(morphism, category[, attribute1, value1, ...])` (function)

Objectifies the morphism *morphism* with the type created for morphisms in the category *category*. The type is created by passing a representation to `AddMorphismRepresentation`. Morphisms which are objectified using this method do not have to be passed to the `AddMorphism` function.

Please note that the `Source` and `Range` attribute need to be passed to this function. The values belonging to these attrbutes will not be objectified.

3.6 Equality and Congruence for Morphisms

3.6.1 IsCongruentForMorphisms (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

Returns: a boolean

The arguments are two morphisms $\alpha, \beta : a \to b$. The output is `true` if $\alpha \sim_{a,b} \beta$, otherwise the output is `false`.

3.6.2 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).

3.6.3 IsEqualForMorphisms (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

Returns: a boolean

The arguments are two morphisms $\alpha, \beta : a \to b$. The output is `true` if $\alpha = \beta$, otherwise the output is `false`.

3.6.4 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).

3.6.5 IsEqualForMorphismsOnMor (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

Returns: a boolean

The arguments are two morphisms $\alpha : a \to b, \beta : c \to d$. The output is `true` if $\alpha = \beta$, otherwise the output is `false`.

3.6.6 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).

3.7 Basic Operations for Morphisms in Ab-Categories

3.7.1 IsZeroForMorphisms (for IsCapCategoryMorphism)

▷ `IsZeroForMorphisms(alpha)` (property)

Returns: a boolean

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

3.7.2 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).

3.7.3 AdditionForMorphisms (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

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

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

3.7.4 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`. $F : (\alpha, \beta) \mapsto \alpha + \beta$.

3.7.5 SubtractionForMorphisms (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

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

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

3.7.6 AddSubtractionForMorphisms (for IsCapCategory, IsFunction)

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

Returns: nothing

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

3.7.7 AdditiveInverseForMorphisms (for IsCapCategoryMorphism)

▷ `AdditiveInverseForMorphisms(alpha)` (attribute)

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

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

3.7.8 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`. $F : \alpha \mapsto -\alpha$.

3.7.9 MultiplyWithElementOfCommutativeRingForMorphisms (for IsRingElement, IsCapCategoryMorphism)

▷ `MultiplyWithElementOfCommutativeRingForMorphisms(r, alpha)` (operation)

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

The arguments are an element r of a commutative ring and a morphism $\alpha : a \to b$. The output is the multiplication with the ring element $r \cdot \alpha$.

3.7.10 AddMultiplyWithElementOfCommutativeRingForMorphisms (for IsCapCategory, IsFunction)

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

Returns: nothing

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

3.7.11 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 $0 : a \to b$.

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

3.8 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.

3.8.1 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`.

3.8.2 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).

3.8.3 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`.

3.8.4 AddIsEqualAsFactorobjects (for IsCapCategory, IsFunction)

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

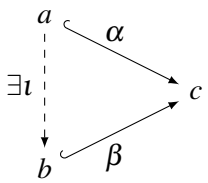
Returns: nothing

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).

3.8.5 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`.

3.8.6 AddIsDominating (for IsCapCategory, IsFunction)

▷ `AddIsDominating(C, F)` (operation)

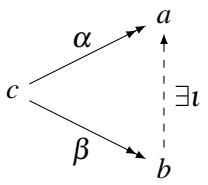
Returns: nothing

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

3.8.7 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`.

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

3.9 Identity Morphism and Composition of Morphisms

3.9.1 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.

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

3.9.3 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 $\beta \circ \alpha : a \to c$.

3.9.4 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 $L = (\alpha_1 : a_1 \to a_2, \alpha_2 : a_2 \to a_3, \ldots, \alpha_n : a_n \to a_{n+1})$. The output is the composition $\alpha_n \circ (\alpha_{n-1} \circ (\ldots (\alpha_2 \circ \alpha_1)))$.

3.9.5 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`. $F : (\alpha, \beta) \mapsto \beta \circ \alpha$.

3.9.6 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 $\beta \circ \alpha : a \to c$.

3.9.7 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 $L = (\alpha_n : a_n \to a_{n+1}, \alpha_{n-1} : a_{n-1} \to a_n, \ldots, \alpha_1 : a_1 \to a_2)$. The output is the composition $((\alpha_n \circ \alpha_{n-1}) \circ \ldots \alpha_2) \circ \alpha_1$.

3.9.8 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`. $F : (\alpha, \beta) \mapsto \alpha \circ \beta$.

3.10 Well-Definedness of Morphisms

3.10.1 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`.

3.10.2 AddIsWellDefinedForMorphisms (for IsCapCategory, IsFunction)

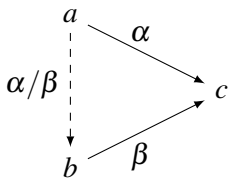
▷ `AddIsWellDefinedForMorphisms(C, F)` (operation)

Returns: nothing

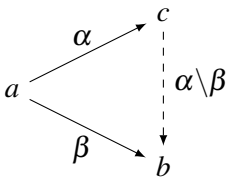
The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `IsWellDefinedForMorphisms`. $F : \alpha \mapsto$ `IsWellDefinedForMorphisms`(α).

3.11 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.

3.11.1 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.

3.11.2 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.

3.11.3 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.

3.11.4 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.

3.11.5 IsLiftableAlongMonomorphism (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

Returns: a boolean

The arguments are a monomorphism $\iota : k \hookrightarrow a$ and a morphism $\tau : t \to a$. The output is `true` if there exists a morphism $u : t \to k$ with $\iota \circ u \sim_{t,a} \tau$. Otherwise, the output is `false`.

3.11.6 AddIsLiftableAlongMonomorphism (for IsCapCategory, IsFunction)

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

Returns: nothing

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

3.11.7 IsColiftableAlongEpimorphism (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

Returns: a boolean

The arguments are an epimorphism $\varepsilon : a \to c$ and a morphism $\tau : a \to t$. The output is `true` if there exists a morphism $u : c \to t$ with $u \circ \varepsilon \sim_{a,t} \tau$. Otherwise, the output is `false`.

3.11.8 AddIsColiftableAlongEpimorphism (for IsCapCategory, IsFunction)

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

Returns: nothing

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

3.11.9 Lift (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `Lift(alpha, beta)` (operation)

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

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

3.11.10 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.

3.11.11 Colift (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `Colift(alpha, beta)` (operation)

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

The arguments are two morphisms $\alpha : a \to c$, $\beta : a \to b$. The output is a colift $\alpha\backslash\beta : c \to b$ of β along α if such a colift exists or `fail` if it doesn't. Recall that a colift $\alpha\backslash\beta : c \to b$ of β along α is a morphism such that $(\alpha\backslash\beta) \circ \alpha \sim_{a,b} \beta$.

3.11.12 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.

3.11.13 IsLiftable (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

Returns: a boolean

The arguments are two morphisms $\alpha : a \to c$, $\beta : b \to c$. The output is `true` if there exists a lift $\alpha/\beta : a \to b$ of α along β, i.e., a morphism such that $\beta \circ (\alpha/\beta) \sim_{a,c} \alpha$. Otherwise, the output is `false`.

3.11.14 AddIsLiftable (for IsCapCategory, IsFunction)

▷ `AddIsLiftable(C, F)` (operation)

Returns: nothing

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

3.11.15 IsColiftable (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

Returns: a boolean

The arguments are two morphisms $\alpha : a \to c$, $\beta : a \to b$. The output is `true` if there exists a colift $\alpha\backslash\beta : c \to b$ of β along α., i.e., a morphism such that $(\alpha\backslash\beta) \circ \alpha \sim_{a,b} \beta$. Otherwise, the output is `false`.

3.11.16 AddIsColiftable (for IsCapCategory, IsFunction)

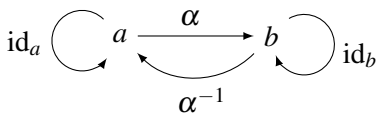
▷ `AddIsColiftable(C, F)` (operation)

Returns: nothing

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

3.12 Inverses

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



3.12.1 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`. $F : \alpha \mapsto \alpha^{-1}$.

3.13 Tool functions for caches

3.13.1 IsEqualForCacheForMorphisms (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

Returns: true or false

Compares two objects in the cache

3.13.2 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.

3.14 Homomorphism structures

Homomorphism structures are way to "oversee" the homomorphisms between two given objects. Let C, D be categories. A D-homomorphism structure for C consists of the following data:

- a functor $H : C^{\mathrm{op}} \times C \to D$ (when C and D are Ab-categories, H is assumed to be bilinear).
- an object $1 \in D$, called the distinguished object,
- a bijection $\nu : \mathrm{Hom}_C(a,b) \simeq \mathrm{Hom}_D(1, H(a,b))$ natural in $a,b \in C$.

3.14.1 HomomorphismStructureOnObjects (for IsCapCategoryObject, IsCapCategoryObject)

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

Returns: an object in D

The arguments are two objects a,b in C. The output is the value of the homomorphism structure on objects $H(a,b)$.

3.14.2 AddHomomorphismStructureOnObjects (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `HomomorphismStructureOnObjects`. $F : (a,b) \mapsto H(a,b)$.

3.14.3 HomomorphismStructureOnMorphisms (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

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

The arguments are two morphisms $\alpha : a \to a', \beta : b \to b'$ in C. The output is the value of the homomorphism structure on morphisms $H(\alpha,\beta)$.

3.14.4 HomomorphismStructureOnMorphismsWithGivenObjects (for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject)

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

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

The arguments are an object $s = H(a',b)$ in D, two morphisms $\alpha : a \to a', \beta : b \to b'$ in C, and an object $r = H(a,b')$ in D. The output is the value of the homomorphism structure on morphisms $H(\alpha,\beta)$.

3.14.5 AddHomomorphismStructureOnMorphismsWithGivenObjects (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `HomomorphismStructureOnMorphismsWithGivenObjects`. $F : (s, \alpha : a \to a', \beta : b \to b', r) \mapsto H(\alpha, \beta)$.

3.14.6 DistinguishedObjectOfHomomorphismStructure (for IsCapCategory)

▷ `DistinguishedObjectOfHomomorphismStructure(C)` (attribute)

Returns: an object in D

The argument is a category C. The output is the distinguished object 1 in D of the homomorphism structure.

3.14.7 AddDistinguishedObjectOfHomomorphismStructure (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `DistinguishedObjectOfHomomorphismStructure`. $F : () \mapsto 1$.

3.14.8 InterpretMorphismAsMorphismFromDinstinguishedObjectToHomomorphismStructure (for IsCapCategoryMorphism)

▷ `InterpretMorphismAsMorphismFromDinstinguishedObjectToHomomorphismStructure(alpha)` (attribute)

Returns: a morphism in $\mathrm{Hom}_D(1, H(a, a'))$

The argument is a morphism $\alpha : a \to a'$ in C. The output is the corresponding morphism $\nu(\alpha) : 1 \to H(a, a')$ in D of the homomorphism structure.

3.14.9 AddInterpretMorphismAsMorphismFromDinstinguishedObjectToHomomorphismStructure (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `InterpretMorphismAsMorphismFromDinstinguishedObjectToHomomorphismStructure`. $F : (\alpha : a \to a') \mapsto (\nu(\alpha) : 1 \to H(a, a'))$.

3.14.10 InterpretMorphismFromDinstinguishedObjectToHomomorphismStructureAsMorphism (for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryMorphism)

▷ `InterpretMorphismFromDinstinguishedObjectToHomomorphismStructureAsMorphism(a, a', iota)` (operation)

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

The arguments are objects a,a' in C and a morphism $\iota : 1 \to H(a,a')$ in D. The output is the corresponding morphism $\nu^{-1}(\iota) : a \to a'$ in C of the homomorphism structure.

3.14.11 AddInterpretMorphismFromDinstinguishedObjectToHomomorphismStructureAsMorphism (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `InterpretMorphismFromDinstinguishedObjectToHomomorphismStructureAsMorphism`.
$F : (a,a',\iota : 1 \to H(a,a')) \mapsto (\nu^{-1}(\iota) : a \to a')$.

3.14.12 SolveLinearSystemInAbCategory (for IsList, IsList, IsList)

▷ `SolveLinearSystemInAbCategory(alpha, beta, gamma)` (operation)

Returns: a list of morphisms $[X_1,\ldots,X_n]$

The arguments are three lists α, β, and γ. The first list α (the left coefficients) is a list of list of morphisms $\alpha_{ij} : A_i \to B_j$, where $i = 1\ldots m$ and $j = 1\ldots n$ for integers $m,n \geq 1$. The second list β (the right coefficients) is a list of list of morphisms $\beta_{ij} : C_j \to D_i$, where $i = 1\ldots m$ and $j = 1\ldots n$. The third list γ (the right side) is a list of morphisms $\gamma_i : A_i \to D_i$, where $i = 1,\ldots,m$. The output is either a list of morphisms $X_j : B_j \to C_j$ for $j = 1\ldots n$ solving the linear system defined by α, β, γ, i.e., $\sum_{j=1}^{n} \alpha_{ij} \cdot X_j \cdot \beta_{ij} = \gamma_i$ for all $i = 1\ldots m$, or `fail` if no such solution exists.

Chapter 4

Category 2-Cells

4.1 Attributes for the Type of 2-Cells

4.1.1 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.

4.1.2 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.

4.2 Identity 2-Cell and Composition of 2-Cells

4.2.1 IdentityTwoCell (for IsCapCategoryMorphism)

▷ `IdentityTwoCell(alpha)` (attribute)

Returns: a 2-cell

The argument is a morphism α. The output is its identity 2-cell $\mathrm{id}_\alpha : \alpha \to \alpha$.

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

4.2.3 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 $d * c : (\gamma \circ \alpha) \to (\delta \circ \beta)$.

4.2.4 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`. $F : (c,d) \mapsto d * c$.

4.2.5 HorizontalPostCompose (for IsCapCategoryTwoCell, IsCapCategoryTwoCell)

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

Returns: a 2-cell

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

4.2.6 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`. $F : (d,c) \mapsto d * c$.

4.2.7 VerticalPreCompose (for IsCapCategoryTwoCell, IsCapCategoryTwoCell)

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

Returns: a 2-cell

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

4.2.8 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`. $F : (c,d) \mapsto d \circ c$.

4.2.9 VerticalPostCompose (for IsCapCategoryTwoCell, IsCapCategoryTwoCell)

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

Returns: a 2-cell

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

4.2.10 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`. $F : (d,c) \mapsto d \circ c$.

4.3 Well-Definedness for 2-Cells

4.3.1 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`.

4.3.2 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 5

Category of Categories

Categories itself with functors as morphisms form a category Cat. So the data structure of CapCategorys 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).

5.1 The Category Cat

5.1.1 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.

5.2 Categories

5.2.1 IsCapCategoryAsCatObject (for IsCapCategoryObject)

▷ IsCapCategoryAsCatObject(*object*) (filter)
Returns: true or false
The GAP category of CAP categories seen as object in Cat.

5.2.2 IsCapFunctor (for IsCapCategoryMorphism)

▷ IsCapFunctor(*object*) (filter)
Returns: true or false
The GAP category of functors.

5.2.3 IsCapNaturalTransformation (for IsCapCategoryTwoCell)

▷ IsCapNaturalTransformation(*object*) (filter)
Returns: true or false

The GAP category of natural transformations.

5.3 Constructors

5.3.1 AsCatObject (for IsCapCategory)

▷ `AsCatObject(C)` (attribute)

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`.

5.3.2 AsCapCategory (for IsCapCategoryAsCatObject)

▷ `AsCapCategory(C)` (attribute)

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.

5.4 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. However, the user can explicitly decide the arity of a functor (which will only have technical implications). Thus, it is for example possible to consider a functor $A \times B \to C$ either as a unary functor with source category $A \times B$ or as a binary functor. Moreover, an object and a morphism function can be added to a functor, to apply it to objects or morphisms in the source category.

5.4.1 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)

These methods construct a unary CAP functor. The first argument is a string for the functor's name. `A` and `B` are the source and target of the functor, and they can be given as objects in `CapCat` or as a CAP-category.

5.4.2 CapFunctor (for IsString, IsList, IsCapCategory)

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

These methods construct a possible multivariate CAP functor. The first argument is a string for the functor's name. The second argument is a list encoding the input signature of the functor. It can be given as a list of pairs $[[A_1,b_1],\ldots,[A_n,b_n]]$ where a pair consists of a category A_i (given as an object in `CapCat` or as a CAP-category) and a boolean b_i for $i=1,\ldots,n$. Instead of a pair $[A_i,b_i]$, you can also give simply A_i, which will be interpreted as the pair $[A_i,$`false`$]$. The third argument is the target `B` of the functor, and it can be given as an object in `CapCat` or as a CAP-category. The output is a functor with source given by the product category $D_1\times\ldots\times D_n$, where $D_i=A_i$ if $b_i=$`false`, and $D_i=A_i^{\mathrm{op}}$ otherwise.

5.4.3 AddObjectFunction (for IsCapFunctor, IsFunction)

▷ `AddObjectFunction(F, f)` (operation)

This operation adds a function f to the functor F which can then be applied to objects in the source. The given function f has to take arguments according to the `InputSignature` of F, i.e., if the input signature is given by $[[A_1,b_1],\ldots,[A_n,b_n]]$, then f must take n arguments, where the i-th argument is an object in the category A_i (the boolean b_i is ignored). The function should return an object in the range of the functor, except when the automatic call of `AddObject` was enabled via `EnableAddForCategoricalOperations`. In this case the output only has to be a GAP object in `IsAttributeStoringRep`, which will be automatically added as an object to the range of the functor.

5.4.4 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.

5.4.5 AddMorphismFunction (for IsCapFunctor, IsFunction)

▷ `AddMorphismFunction(F, f)` (operation)

This operation adds a function f to the functor F which can then be applied to morphisms in the source. The given function f has to take as its first argument an object s that is equal (via `IsEqualForObjects`) to the source of the result of applying F to the input morphisms. The next arguments of f have to morphisms according to the `InputSignature` of F, i.e., if the input signature is given by $[[A_1,b_1],\ldots,[A_n,b_n]]$, then f must take n arguments, where the i-th argument is a morphism in the category A_i (the boolean b_i is ignored). The last argument of f must be an object r that is equal (via `IsEqualForObjects`) to the range of the result of applying F to the input morphisms. The function should return a morphism in the range of the functor, except when the automatic call of `AddMorphism` was enabled via `EnableAddForCategoricalOperations`. In this case the output only has to be a GAP object in `IsAttributeStoringRep` (with attributes `Source` and `Range` containing also GAP objects in `IsAttributeStoringRep`), which will be automatically added as a morphism to the range of the functor.

5.4.6 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.

5.4.7 ApplyFunctor

▷ `ApplyFunctor(func, A)` (function)

Returns: IsCapCategoryCell

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

5.4.8 InputSignature (for IsCapFunctor)

▷ `InputSignature(F)` (attribute)

Returns: IsList

The argument is a functor F. The output is a list of pairs $[[A_1, b_1], \ldots, [A_n, b_n]]$ where a pair consists of a CAP-category A_i and a boolean b_i for $i = 1, \ldots, n$. The source of F is given by the product category $D_1 \times ... \times D_n$, where $D_i = A_i$ if $b_i =$ `false`, and $D_i = A_i^{\mathrm{op}}$ otherwise.

5.4.9 InstallFunctor (for IsCapFunctor, IsString)

▷ `InstallFunctor(F, s)` (operation)

Returns: nothing

The arguments are a functor F and a string s. To simplify the description of this operation, we let $[[A_1, b_1], \ldots, [A_n, b_n]]$ denote the input signature of F. This method tries to install 3 operations: an operation ω_1 with the name s, an operation ω_2 with the name s`OnObjects`, and an operation ω_3 with the name s`OnMorphisms`. The operation ω_1 takes as input either n- objects/morphisms in A_i or a single object/morphism in the source of F, and outputs the result of applying F to this input. ω_2 and ω_3 are the corresponding variants for objects or morphisms only. This function can only be called once for each functor, every further call will be ignored.

5.4.10 IdentityFunctor (for IsCapCategory)

▷ `IdentityFunctor(cat)` (attribute)

Returns: a functor

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

5.4.11 FunctorCanonicalizeZeroObjects (for IsCapCategory)

▷ `FunctorCanonicalizeZeroObjects(cat)` (attribute)

Returns: a functor

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

5.4.12 NaturalIsomorophismFromIdentityToCanonicalizeZeroObjects (for IsCapCategory)

▷ `NaturalIsomorophismFromIdentityToCanonicalizeZeroObjects(cat)` (attribute)

Returns: a natural transformation

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

5.4.13 FunctorCanonicalizeZeroMorphisms (for IsCapCategory)

▷ `FunctorCanonicalizeZeroMorphisms(cat)` (attribute)

Returns: a functor

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

5.4.14 NaturalIsomorophismFromIdentityToCanonicalizeZeroMorphisms (for IsCapCategory)

▷ `NaturalIsomorophismFromIdentityToCanonicalizeZeroMorphisms(cat)` (attribute)

Returns: a natural transformation

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

5.5 Natural transformations

Natural transformations form the 2-cells of Cat. As such, it is possible to compose them vertically and horizontally, see Section 4.2.

5.5.1 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.

5.5.2 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

5.5.3 AddNaturalTransformationFunction (for IsCapNaturalTransformation, IsFunction)

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

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, $F(A) \to G(A)$.

5.5.4 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*.

5.5.5 InstallNaturalTransformation (for IsCapNaturalTransformation, IsString)

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

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

5.5.6 HorizontalPreComposeNaturalTransformationWithFunctor (for IsCapNaturalTransformation, IsCapFunctor)

▷ `HorizontalPreComposeNaturalTransformationWithFunctor(N, F)` (operation)

Returns: a natural transformation

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

5.5.7 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 6

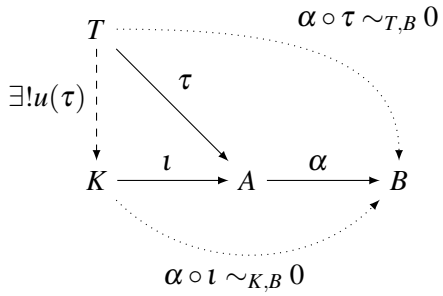
Universal Objects

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 KernelObject(α). 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 KernelObject(α) $\to$ KernelObject(α').



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 Hom(KernelObject(α),A)

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

6.1.3 KernelEmbeddingWithGivenKernelObject (for IsCapCategoryMorphism, IsCapCategoryObject)

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

Returns: a morphism in Hom(K,A)

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

6.1.4 KernelLift (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

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

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

6.1.5 KernelLiftWithGivenKernelObject (for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject)

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

Returns: a morphism in Hom(T,K)

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

6.1.6 AddKernelObject (for IsCapCategory, IsFunction)

▷ `AddKernelObject(C, F)` (operation)

Returns: nothing

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

6.1.7 AddKernelEmbedding (for IsCapCategory, IsFunction)

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

Returns: nothing

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

6.1.8 AddKernelEmbeddingWithGivenKernelObject (for IsCapCategory, IsFunction)

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

Returns: nothing

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

6.1.9 AddKernelLift (for IsCapCategory, IsFunction)

▷ `AddKernelLift(C, F)` (operation)

Returns: nothing

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

6.1.10 AddKernelLiftWithGivenKernelObject (for IsCapCategory, IsFunction)

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

Returns: nothing

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

6.1.11 KernelObjectFunctorial (for IsList)

▷ `KernelObjectFunctorial(L)` (operation)

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

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

6.1.12 KernelObjectFunctorial (for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism)

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

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

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

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

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

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

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

6.1.14 KernelObjectFunctorialWithGivenKernelObjects (for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryMorphism,IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject)

▷ `KernelObjectFunctorialWithGivenKernelObjects(s, alpha, mu, nu, alpha_prime, r)` (operation)

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

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

6.1.15 AddKernelObjectFunctorialWithGivenKernelObjects (for IsCapCategory, IsFunction)

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

Returns: nothing

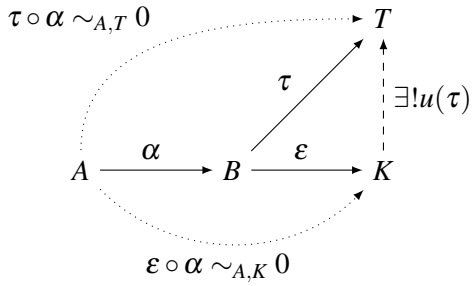
The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `KernelObjectFunctorialWithGivenKernelObjects`. $F : (\text{KernelObject}(\alpha), \alpha, \mu, \alpha', \text{KernelObject}(\alpha')) \mapsto (\text{KernelObject}(\alpha)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

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 $\text{CokernelObject}(\alpha)$. We say that the morphism $u(\tau)$ is induced by the *universal property of the cokernel*.
CokernelObject is a functorial operation. This means: for $\mu : A \to A'$, $\nu : B \to B'$, $\alpha : A \to B$, $\alpha' : A' \to B'$ such that $\nu \circ \alpha \sim_{A,B'} \alpha' \circ \mu$, we obtain a morphism $\text{CokernelObject}(\alpha) \to \text{CokernelObject}(\alpha')$.



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 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 $\varepsilon : B \to \mathrm{CokernelObject}(\alpha)$.

6.2.4 CokernelColift (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

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

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

6.2.5 CokernelColiftWithGivenCokernelObject (for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject)

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

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

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

6.2.6 AddCokernelObject (for IsCapCategory, IsFunction)

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

Returns: nothing

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

6.2.7 AddCokernelProjection (for IsCapCategory, IsFunction)

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

Returns: nothing

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

6.2.8 AddCokernelProjectionWithGivenCokernelObject (for IsCapCategory, IsFunction)

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

Returns: nothing

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

6.2.9 AddCokernelColift (for IsCapCategory, IsFunction)

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

Returns: nothing

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

6.2.10 AddCokernelColiftWithGivenCokernelObject (for IsCapCategory, IsFunction)

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

Returns: nothing

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

6.2.11 CokernelObjectFunctorial (for IsList)

▷ `CokernelObjectFunctorial(L)` (operation)

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

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 functoriality of the cokernel.

6.2.12 CokernelObjectFunctorial (for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `CokernelObjectFunctorial(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 functoriality of the cokernel.

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

▷ `CokernelObjectFunctorialWithGivenCokernelObjects(s, alpha, nu, alpha_prime, r)` (operation)

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

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

6.2.14 CokernelObjectFunctorialWithGivenCokernelObjects (for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryMorphism,IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject)

▷ `CokernelObjectFunctorialWithGivenCokernelObjects(s, alpha, mu, nu, alpha_prime, r)` (operation)

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

The arguments are an object $s = \mathrm{CokernelObject}(\alpha)$, four morphisms $\alpha : A \to B, \mu : A \to A', \nu : B \to B', \alpha' : A' \to B'$, and an object $r = \mathrm{CokernelObject}(\alpha')$. The output is the morphism $\mathrm{CokernelObject}(\alpha) \to \mathrm{CokernelObject}(\alpha')$ given by the functoriality of the cokernel.

6.2.15 AddCokernelObjectFunctorialWithGivenCokernelObjects (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `CokernelObjectFunctorialWithGivenCokernelObjects`. $F : (\mathrm{CokernelObject}(\alpha), \alpha, \nu, \alpha', \mathrm{CokernelObject}(\alpha')) \mapsto (\mathrm{CokernelObject}(\alpha) \to \mathrm{CokernelObject}(\alpha'))$.

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

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

6.3.6 UniversalMorphismFromZeroObjectWithGivenZeroObject (for IsCapCategoryObject, IsCapCategoryObject)

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

Returns: a morphism in Hom(Z, A)

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

6.3.7 UniversalMorphismIntoZeroObject (for IsCapCategoryObject)

▷ `UniversalMorphismIntoZeroObject(A)` (attribute)

Returns: a morphism in Hom(A, ZeroObject)

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

6.3.8 UniversalMorphismIntoZeroObjectWithGivenZeroObject (for IsCapCategoryObject, IsCapCategoryObject)

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

Returns: a morphism in Hom(A, Z)

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

6.3.9 IsomorphismFromZeroObjectToInitialObject (for IsCapCategory)

▷ `IsomorphismFromZeroObjectToInitialObject(C)` (attribute)

Returns: a morphism in Hom(ZeroObject, InitialObject)

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

6.3.10 IsomorphismFromInitialObjectToZeroObject (for IsCapCategory)

▷ `IsomorphismFromInitialObjectToZeroObject(C)` (attribute)

Returns: a morphism in Hom(InitialObject, ZeroObject)

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

6.3.11 IsomorphismFromZeroObjectToTerminalObject (for IsCapCategory)

▷ `IsomorphismFromZeroObjectToTerminalObject(C)` (attribute)

Returns: a morphism in Hom(ZeroObject, TerminalObject)

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

6.3.12 IsomorphismFromTerminalObjectToZeroObject (for IsCapCategory)

▷ `IsomorphismFromTerminalObjectToZeroObject(C)` (attribute)

Returns: a morphism in Hom(TerminalObject, ZeroObject)

The argument is a category C. The output is the unique isomorphism TerminalObject $\to$ 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`. $F : () \mapsto$ ZeroObject.

6.3.14 AddUniversalMorphismIntoZeroObject (for IsCapCategory, IsFunction)

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

Returns: nothing

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

6.3.15 AddUniversalMorphismIntoZeroObjectWithGivenZeroObject (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `UniversalMorphismIntoZeroObjectWithGivenZeroObject`. $F : (A, Z) \mapsto u_{\text{in}}(A)$.

6.3.16 AddUniversalMorphismFromZeroObject (for IsCapCategory, IsFunction)

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

Returns: nothing

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

6.3.17 AddUniversalMorphismFromZeroObjectWithGivenZeroObject (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `UniversalMorphismFromZeroObjectWithGivenZeroObject`. $F : (A, Z) \mapsto u_{\text{out}}(A)$.

6.3.18 AddIsomorphismFromZeroObjectToInitialObject (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `IsomorphismFromZeroObjectToInitialObject`. $F : () \mapsto$ (ZeroObject $\to$ 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`. $F : () \mapsto$ (InitialObject $\to$ 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`. $F : () \mapsto$ (ZeroObject $\to$ 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`. $F : () \mapsto$ (TerminalObject $\to$ ZeroObject).

6.3.22 ZeroObjectFunctorial (for IsCapCategory)

▷ `ZeroObjectFunctorial(C)` (attribute)

Returns: a morphism in Hom(ZeroObject, ZeroObject)

The argument is a category C. The output is the unique morphism ZeroObject $\to$ ZeroObject.

6.3.23 AddZeroObjectFunctorial (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `ZeroObjectFunctorial`. $F : () \mapsto (T \to T)$.

6.4 Terminal Object

A terminal object consists of two parts:

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

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 $T \to T$.

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 Hom(A, TerminalObject)

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

The argument is a category C. The output is the unique morphism TerminalObject $\to$ 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`. $F : () \mapsto (T \to T)$.

6.5 Initial Object

An initial object consists of two parts:

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

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 $I \to I$.

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 $\to A$).

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

6.5.4 UniversalMorphismFromInitialObjectWithGivenInitialObject (for IsCapCategoryObject, IsCapCategoryObject)

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

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

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

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 Hom(InitialObject, InitialObject)

The argument is a category C. The output is the unique morphism InitialObject $\to$ 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`. $F : () \to (I \to I)$.

6.6 Direct Sum

For an integer $n \geq 1$ and a given list $D = (S_1, \ldots, S_n)$ in an Ab-category, a direct sum consists of five parts:

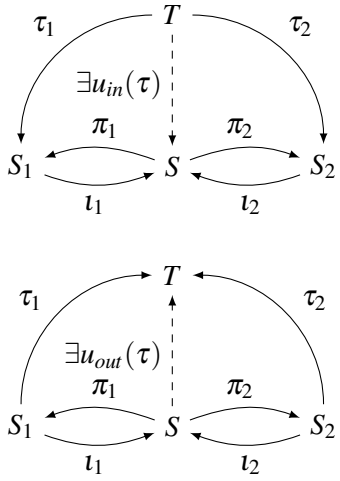
- an object S,
- a list of morphisms $\pi = (\pi_i : S \to S_i)_{i=1\ldots n}$,
- a list of morphisms $\iota = (\iota_i : S_i \to S)_{i=1\ldots n}$,
- a dependent function u_{in} mapping every list $\tau = (\tau_i : T \to S_i)_{i=1\ldots n}$ to a morphism $u_{\text{in}}(\tau) : T \to S$ such that $\pi_i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

such that

- $\sum_{i=1}^{n} \iota_i \circ \pi_i \sim_{S,S} \text{id}_S$,

- $\pi_j \circ \iota_i \sim_{S_i,S_j} \delta_{i,j}$,

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 $(\mu_i : S_i \to S_i')_{i=1\ldots n}$, 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 $D=(S_1,\ldots,S_n)$ 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 $D=(S_1,\ldots,S_n)$ 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 $D=(S_1,\ldots,S_n)$, 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 $D = (S_1, \ldots, S_n)$, 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 $D = (S_1, \ldots, S_n)$ 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 $D = (S_1, \ldots, S_n)$, 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 $D = (S_1, \ldots, S_n)$, an integer k, and an object $S = \bigoplus_{i=1}^{n} S_i$. The output is the k-th injection $\iota_k : S_k \to S$.

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

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

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 $D = (S_1,\ldots,S_n)$, a list of morphisms $\tau = (\tau_i : S_i \to T)_{i=1\ldots n}$, 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 $D = (S_1,\ldots,S_n)$, a list of morphisms $\tau = (\tau_i : S_i \to T)_{i=1\ldots n}$, 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 $D = (S_1,\ldots,S_n)$. 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 $D = (S_1, \ldots, S_n)$ 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 $D = (S_1, \ldots, S_n)$. 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 $D = (S_1, \ldots, S_n)$ and an object for method selection. The output is the canonical isomorphism $\prod_{i=1}^{n} S_i \to \bigoplus_{i=1}^{n} S_i$.

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 $D = (S_1, \ldots, S_n)$. The output is the canonical isomorphism $\bigoplus_{i=1}^{n} S_i \to \bigsqcup_{i=1}^{n} S_i$.

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 $D = (S_1, \ldots, S_n)$ and an object for method selection. The output is the canonical isomorphism $\bigoplus_{i=1}^{n} S_i \to \bigsqcup_{i=1}^{n} S_i$.

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 $D = (S_1, \ldots, S_n)$. The output is the canonical isomorphism $\bigsqcup_{i=1}^{n} S_i \to \bigoplus_{i=1}^{n} S_i$.

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 $D = (S_1, \ldots, S_n)$ and an object for method selection. The output is the canonical isomorphism $\bigsqcup_{i=1}^{n} S_i \to \bigoplus_{i=1}^{n} S_i$.

6.6.22 MorphismBetweenDirectSums (for IsList)

▷ `MorphismBetweenDirectSums(M)` (operation)

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

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

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

6.6.23 AddMorphismBetweenDirectSums (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `MorphismBetweenDirectSums`. $F : (\bigoplus_{i=1}^{m} A_i, M, \bigoplus_{j=1}^{n} B_j) \mapsto (\bigoplus_{i=1}^{m} A_i \to \bigoplus_{j=1}^{n} B_j)$.

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

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

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

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

6.6.25 ComponentOfMorphismIntoDirectSum (for IsCapCategoryMorphism, IsList, IsInt)

▷ `ComponentOfMorphismIntoDirectSum(alpha, D, k)` (operation)

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

The arguments are a morphism $\alpha : A \to S$, a list $D = (S_1, \ldots, S_n)$ of objects with $S = \bigoplus_{j=1}^{n} S_j$, and an integer k. The output is the component morphism $A \to S_k$.

6.6.26 ComponentOfMorphismFromDirectSum (for IsCapCategoryMorphism, IsList, IsInt)

▷ `ComponentOfMorphismFromDirectSum(alpha, D, k)` (operation)

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

The arguments are a morphism $\alpha : S \to A$, a list $D = (S_1, \ldots, S_n)$ of objects with $S = \bigoplus_{j=1}^{n} S_j$, and an integer k. The output is the component morphism $S_k \to A$.

6.6.27 AddComponentOfMorphismIntoDirectSum (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `ComponentOfMorphismIntoDirectSum`. $F : (\alpha : A \to S, D, k) \mapsto (A \to S_k)$.

6.6.28 AddComponentOfMorphismFromDirectSum (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `ComponentOfMorphismFromDirectSum`. $F : (\alpha : S \to A, D, k) \mapsto (S_k \to A)$.

6.6.29 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`. $F : (D, k) \mapsto \pi_k$.

6.6.30 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`. $F : (D, k, S) \mapsto \pi_k$.

6.6.31 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`. $F : (D, k) \mapsto \iota_k$.

6.6.32 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`. $F : (D,k,S) \mapsto \iota_k$.

6.6.33 AddUniversalMorphismIntoDirectSum (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `UniversalMorphismIntoDirectSum`. $F : (D,\tau) \mapsto u_{\text{in}}(\tau)$.

6.6.34 AddUniversalMorphismIntoDirectSumWithGivenDirectSum (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `UniversalMorphismIntoDirectSumWithGivenDirectSum`. $F : (D,\tau,S) \mapsto u_{\text{in}}(\tau)$.

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

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

6.6.37 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`. $F : D \mapsto (\bigoplus_{i=1}^{n} S_i \to \prod_{i=1}^{n} S_i)$.

6.6.38 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`. $F : D \mapsto (\prod_{i=1}^{n} S_i \to \bigoplus_{i=1}^{n} S_i)$.

6.6.39 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`. $F : D \mapsto (\bigoplus_{i=1}^{n} S_i \to \bigsqcup_{i=1}^{n} S_i)$.

6.6.40 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`. $F : D \mapsto (\bigsqcup_{i=1}^{n} S_i \to \bigoplus_{i=1}^{n} S_i)$.

6.6.41 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`. $F : D \mapsto \bigoplus_{i=1}^{n} S_i$.

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

6.6.43 DirectSumFunctorialWithGivenDirectSums (for IsCapCategoryObject, IsList, IsCapCategoryObject)

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

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

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

6.6.44 AddDirectSumFunctorialWithGivenDirectSums (for IsCapCategory, IsFunction)

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

Returns: nothing

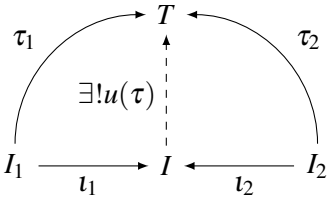
The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `DirectSumFunctorialWithGivenDirectSums`. $F : (\bigoplus_{i=1}^{n} S_i, (\mu_1, \ldots, \mu_n), \bigoplus_{i=1}^{n} S_i') \mapsto (\bigoplus_{i=1}^{n} S_i \to \bigoplus_{i=1}^{n} S_i')$.

6.7 Coproduct

For an integer $n \geq 1$ and a given list of objects $D = (I_1, \ldots, I_n)$, a coproduct of D consists of three parts:

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

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 $(\mu_i : I_i \to I_i')_{i=1\ldots n}$, we obtain a morphism $\bigsqcup_{i=1}^{n} I_i \to \bigsqcup_{i=1}^{n} I_i'$.



6.7.1 Coproduct (for IsList)

▷ `Coproduct(D)` (attribute)

Returns: an object

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

The output is the morphism $u(\tau) : \bigsqcup_{i=1}^{n} I_i \to T$ 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 $D = (I_1, \ldots, I_n)$, a list of morphisms $\tau = (\tau_i : I_i \to T)$, and a method selection object. The output is the morphism $u(\tau) : \bigsqcup_{i=1}^{n} I_i \to T$ 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 $D = (I_1, \ldots, I_n)$, a list of morphisms $\tau = (\tau_i : I_i \to T)$, and an object $I = \bigsqcup_{i=1}^{n} I_i$. The output is the morphism $u(\tau) : I \to T$ 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`. $F : ((I_1, \ldots, I_n)) \mapsto I$.

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

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

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`. $F : ((I_1, \ldots, I_n), \tau) \mapsto u(\tau)$.

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`. $F : ((I_1, \ldots, I_n), \tau, I) \mapsto u(\tau)$.

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

6.7.17 CoproductFunctorialWithGivenCoproducts (for IsCapCategoryObject, IsList, IsCapCategoryObject)

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

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

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

6.7.18 AddCoproductFunctorialWithGivenCoproducts (for IsCapCategory, IsFunction)

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

Returns: nothing

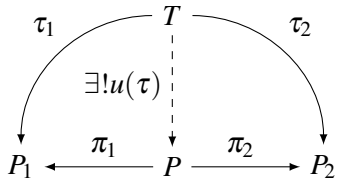
The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `CoproductFunctorialWithGivenCoproducts`. $F : (\bigsqcup_{i=1}^n I_i, (\mu_1, \ldots, \mu_n), \bigsqcup_{i=1}^n I'_i) \to (\bigsqcup_{i=1}^n I_i \to \bigsqcup_{i=1}^n I'_i)$.

6.8 Direct Product

For an integer $n \geq 1$ and a given list of objects $D = (P_1, \ldots, P_n)$, a direct product of D consists of three parts:

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

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 $(\mu_i : P_i \to P'_i)_{i=1\ldots n}$, 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 $D = (P_1, \ldots, P_n)$ 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 $D = (P_1, \ldots, P_n)$ 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 $D = (P_1, \ldots, P_n)$, 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 $D = (P_1, \ldots, P_n)$, 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 $D = (P_1, \ldots, P_n)$ and a list of morphisms $\tau = (\tau_i : T \to P_i)_{i=1,\ldots,n}$.
- The argument is a list of morphisms $\tau = (\tau_i : T \to P_i)_{i=1,\ldots,n}$.
- The arguments are morphisms $\tau_1 : T \to P_1, \ldots, \tau_n : T \to P_n$.

The output is the morphism $u(\tau) : T \to \prod_{i=1}^{n} P_i$ 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 $D = (P_1, \ldots, P_n)$, a list of morphisms $\tau = (\tau_i : T \to P_i)_{i=1,\ldots,n}$, and an object for method selection. The output is the morphism $u(\tau) : T \to \prod_{i=1}^{n} P_i$ 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 $D = (P_1, \ldots, P_n)$, a list of morphisms $\tau = (\tau_i : T \to P_i)_{i=1,\ldots,n}$, and an object $P = \prod_{i=1}^{n} P_i$. The output is the morphism $u(\tau) : T \to \prod_{i=1}^{n} P_i$ 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`. $F : ((P_1, \ldots, P_n)) \mapsto P$

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`. $F : ((P_1, \ldots, P_n), k) \mapsto \pi_k$

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`. $F : ((P_1, \ldots, P_n), k, P) \mapsto \pi_k$

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`. $F : ((P_1, \ldots, P_n), \tau) \mapsto u(\tau)$

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`. $F : ((P_1, \ldots, P_n), \tau, P) \mapsto u(\tau)$

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 $L = (\mu_i : P_i \to P_i')_{i=1\ldots n}$. The output is a morphism $\prod_{i=1}^n P_i \to \prod_{i=1}^n P_i'$ given by the functoriality of the direct product.

6.8.14 DirectProductFunctorialWithGivenDirectProducts (for IsCapCategoryObject, IsList, IsCapCategoryObject)

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

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

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

6.8.15 AddDirectProductFunctorialWithGivenDirectProducts (for IsCapCategory, IsFunction)

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

Returns: nothing

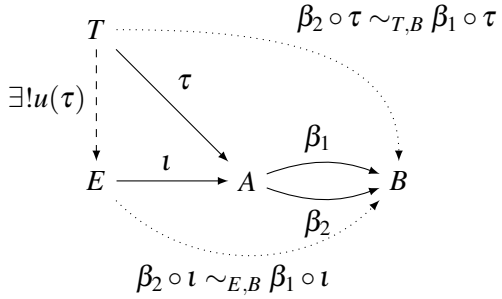
The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `DirectProductFunctorialWithGivenDirectProducts`. $F : (\prod_{i=1}^n P_i, (\mu_i : P_i \to P_i')_{i=1\ldots n}, \prod_{i=1}^n P_i') \mapsto (\prod_{i=1}^n P_i \to \prod_{i=1}^n P_i')$

6.9 Equalizer

For an integer $n \geq 1$ and a given list of morphisms $D = (\beta_i : A \to B)_{i=1\ldots n}$, an equalizer of D consists of three parts:

- an object E,
- a morphism $\iota : E \to A$ such that $\beta_i \circ \iota \sim_{E,B} \beta_j \circ \iota$ for all pairs i, j.
- a dependent function u mapping each morphism $\tau = (\tau : T \to A)$ such that $\beta_i \circ \tau \sim_{T,B} \beta_j \circ \tau$ for all pairs i, j to a morphism $u(\tau) : T \to E$ such that $\iota \circ u(\tau) \sim_{T,A} \tau$.

The triple (E, ι, u) is called an *equalizer* of D if the morphisms $u(\tau)$ are uniquely determined up to congruence of morphisms. We denote the object E of such a triple by Equalizer(D). We say that the morphism $u(\tau)$ is induced by the *universal property of the equalizer*.
Equalizer is a functorial operation. This means: For a second diagram $D' = (\beta'_i : A' \to B')_{i=1\ldots n}$ and a natural morphism between equalizer diagrams (i.e., a collection of morphisms $\mu : A \to A'$ and $\beta : B \to B'$ such that $\beta'_i \circ \mu \sim_{A,B'} \beta \circ \beta_i$ for $i = 1, \ldots, n$) we obtain a morphism Equalizer$(D) \to$ Equalizer(D').



6.9.1 Equalizer

▷ `Equalizer(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 $D = (\beta_i : A \to B)_{i=1\ldots n}$.
- The arguments are morphisms $\beta_1 : A \to B, \ldots, \beta_n : A \to B$.

The output is the equalizer Equalizer(D).

6.9.2 EqualizerOp (for IsList, IsCapCategoryMorphism)

▷ `EqualizerOp(D, method_selection_morphism)` (operation)
Returns: an object
The arguments are a list of morphisms $D = (\beta_i : A \to B)_{i=1\ldots n}$ and a morphism for method selection. The output is the equalizer Equalizer(D).

6.9.3 EmbeddingOfEqualizer (for IsList)

▷ `EmbeddingOfEqualizer(D)` (operation)
Returns: a morphism in Hom(Equalizer$(D), A$)
The arguments are a list of morphisms $D = (\beta_i : A \to B)_{i=1\ldots n}$. The output is the equalizer embedding $\iota :$ Equalizer$(D) \to A$.

6.9.4 EmbeddingOfEqualizerOp (for IsList, IsCapCategoryMorphism)

▷ `EmbeddingOfEqualizerOp(D, method_selection_morphism)` (operation)
Returns: a morphism in Hom(Equalizer$(D), A$)
The arguments are a list of morphisms $D = (\beta_i : A \to B)_{i=1\ldots n}$. and a morphism for method selection. The output is the equalizer embedding $\iota :$ Equalizer$(D) \to A$.

6.9.5 EmbeddingOfEqualizerWithGivenEqualizer (for IsList, IsCapCategoryObject)

▷ `EmbeddingOfEqualizerWithGivenEqualizer(D, E)` (operation)

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

The arguments are a list of morphisms $D = (\beta_i : A \to B)_{i=1\ldots n}$, and an object $E = \mathrm{Equalizer}(D)$. The output is the equalizer embedding $\iota : E \to A$.

6.9.6 UniversalMorphismIntoEqualizer (for IsList, IsCapCategoryMorphism)

▷ `UniversalMorphismIntoEqualizer(D, tau)` (operation)

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

The arguments are a list of morphisms $D = (\beta_i : A \to B)_{i=1\ldots n}$ and a morphism $\tau : T \to A$ such that $\beta_i \circ \tau \sim_{T,B} \beta_j \circ \tau$ for all pairs i, j. The output is the morphism $u(\tau) : T \to \mathrm{Equalizer}(D)$ given by the universal property of the equalizer.

6.9.7 UniversalMorphismIntoEqualizerWithGivenEqualizer (for IsList, IsCapCategoryMorphism, IsCapCategoryObject)

▷ `UniversalMorphismIntoEqualizerWithGivenEqualizer(D, tau, E)` (operation)

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

The arguments are a list of morphisms $D = (\beta_i : A \to B)_{i=1\ldots n}$, a morphism $\tau : T \to A$) such that $\beta_i \circ \tau \sim_{T,B} \beta_j \circ \tau$ for all pairs i, j, and an object $E = \mathrm{Equalizer}(D)$. The output is the morphism $u(\tau) : T \to E$ given by the universal property of the equalizer.

6.9.8 AddEqualizer (for IsCapCategory, IsFunction)

▷ `AddEqualizer(C, F)` (operation)

Returns: nothing

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

6.9.9 AddEmbeddingOfEqualizer (for IsCapCategory, IsFunction)

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

Returns: nothing

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

6.9.10 AddEmbeddingOfEqualizerWithGivenEqualizer (for IsCapCategory, IsFunction)

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

Returns: nothing

The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `EmbeddingOfEqualizerWithGivenEqualizer`. $F : ((\beta_i : A \to B)_{i=1\ldots n}, E) \mapsto \iota$

6.9.11 AddUniversalMorphismIntoEqualizer (for IsCapCategory, IsFunction)

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

Returns: nothing

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

6.9.12 AddUniversalMorphismIntoEqualizerWithGivenEqualizer (for IsCapCategory, IsFunction)

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

Returns: nothing

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

6.9.13 EqualizerFunctorial (for IsList, IsCapCategoryMorphism, IsList)

▷ `EqualizerFunctorial(Ls, mu, Lr)` (operation)

Returns: a morphism in $\mathrm{Hom}(\mathrm{Equalizer}((\beta_i)_{i=1\ldots n}), \mathrm{Equalizer}((\beta'_i)_{i=1\ldots n}))$

The arguments are a list of morphisms $L_s = (\beta_i : A \to B)_{i=1\ldots n}$, a morphism $\mu : A \to A'$, and a list of morphisms $L_r = (\beta'_i : A' \to B')_{i=1\ldots n}$ such that there exists a morphism $\beta : B \to B'$ such that $\beta'_i \circ \mu \sim_{A,B'} \beta \circ \beta_i$ for $i = 1, \ldots, n$. The output is the morphism $\mathrm{Equalizer}((\beta_i)_{i=1\ldots n}) \to \mathrm{Equalizer}((\beta'_i)_{i=1\ldots n})$ given by the functorality of the equalizer.

6.9.14 EqualizerFunctorialWithGivenEqualizers (for IsCapCategoryObject, IsList, IsCapCategoryMorphism, IsList, IsCapCategoryObject)

▷ `EqualizerFunctorialWithGivenEqualizers(s, Ls, mu, Lr, r)` (operation)

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

The arguments are an object $s = \mathrm{Equalizer}((\beta_i)_{i=1\ldots n})$, a list of morphisms $L_s = (\beta_i : A \to B)_{i=1\ldots n}$, a morphism $\mu : A \to A'$, and a list of morphisms $L_r = (\beta'_i : A' \to B')_{i=1\ldots n}$ such that there exists a morphism $\beta : B \to B'$ such that $\beta'_i \circ \mu \sim_{A,B'} \beta \circ \beta_i$ for $i = 1, \ldots, n$, and an object $r = \mathrm{Equalizer}((\beta'_i)_{i=1\ldots n})$. The output is the morphism $s \to r$ given by the functorality of the equalizer.

6.9.15 AddEqualizerFunctorialWithGivenEqualizers (for IsCapCategory, IsFunction)

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

Returns: nothing

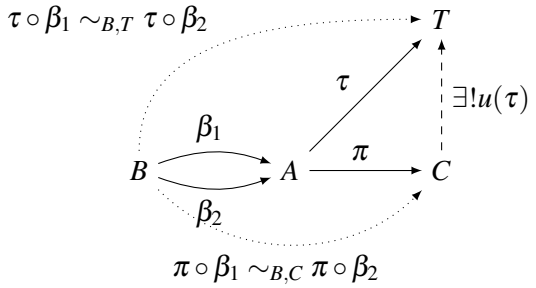
The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `EqualizerFunctorialWithGivenEqualizers`. $F : (\mathrm{Equalizer}((\beta_i)_{i=1\ldots n}), (\beta_i : A \to B)_{i=1\ldots n}, \mu : A \to A', (\beta'_i : A' \to B')_{i=1\ldots n}, \mathrm{Equalizer}((\beta'_i)_{i=1\ldots n})) \mapsto (\mathrm{Equalizer}((\beta_i)_{i=1\ldots n}) \to \mathrm{Equalizer}((\beta'_i)_{i=1\ldots n}))$

6.10 Coequalizer

For an integer $n \geq 1$ and a given list of morphisms $D = (\beta_i : B \to A)_{i=1\ldots n}$, a coequalizer of D consists of three parts:

- an object C,
- a morphism $\pi : A \to C$ such that $\pi \circ \beta_i \sim_{B,C} \pi \circ \beta_j$ for all pairs i, j,
- a dependent function u mapping the morphism $\tau : A \to T$ such that $\tau \circ \beta_i \sim_{B,T} \tau \circ \beta_j$ to a morphism $u(\tau) : C \to T$ such that $u(\tau) \circ \pi \sim_{A,T} \tau$.

The triple (C, π, u) is called a *coequalizer* of D if the morphisms $u(\tau)$ are uniquely determined up to congruence of morphisms. We denote the object C of such a triple by Coequalizer(D). We say that the morphism $u(\tau)$ is induced by the *universal property of the coequalizer*.
Coequalizer is a functorial operation. This means: For a second diagram $D' = (\beta'_i : B' \to A')_{i=1\ldots n}$ and a natural morphism between coequalizer diagrams (i.e., a collection of morphisms $\mu : A \to A'$ and $\beta : B \to B'$ such that $\beta'_i \circ \beta \sim_{B,A'} \mu \circ \beta_i$ for $i = 1, \ldots n$) we obtain a morphism Coequalizer$(D) \to$ Coequalizer(D').



6.10.1 Coequalizer

▷ `Coequalizer(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 $D = (\beta_i : B \to A)_{i=1\ldots n}$.
- The arguments are morphisms $\beta_1 : B \to A, \ldots, \beta_n : B \to A$.

The output is the coequalizer Coequalizer(D).

6.10.2 CoequalizerOp (for IsList, IsCapCategoryMorphism)

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

Returns: an object

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

6.10.3 ProjectionOntoCoequalizer (for IsList)

▷ `ProjectionOntoCoequalizer(D)` (operation)

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

The arguments are a list of morphisms $D = (\beta_i : B \to A)_{i=1\ldots n}$. The output is the projection $\pi : A \to \mathrm{Coequalizer}(D)$.

6.10.4 ProjectionOntoCoequalizerOp (for IsList, IsCapCategoryMorphism)

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

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

The arguments are a list of morphisms $D = (\beta_i : B \to A)_{i=1\ldots n}$, and a morphism for method selection. The output is the projection $\pi : A \to \mathrm{Coequalizer}(D)$.

6.10.5 ProjectionOntoCoequalizerWithGivenCoequalizer (for IsList, IsCapCategoryObject)

▷ `ProjectionOntoCoequalizerWithGivenCoequalizer(D, C)` (operation)

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

The arguments are a list of morphisms $D = (\beta_i : B \to A)_{i=1\ldots n}$, and an object $C = \mathrm{Coequalizer}(D)$. The output is the projection $\pi : A \to C$.

6.10.6 UniversalMorphismFromCoequalizer (for IsList, IsCapCategoryMorphism)

▷ `UniversalMorphismFromCoequalizer(D, tau)` (operation)

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

The arguments are a list of morphisms $D = (\beta_i : B \to A)_{i=1\ldots n}$ and a morphism $\tau : A \to T$ such that $\tau \circ \beta_i \sim_{B,T} \tau \circ \beta_j$ for all pairs i, j. The output is the morphism $u(\tau) : \mathrm{Coequalizer}(D) \to T$ given by the universal property of the coequalizer.

6.10.7 UniversalMorphismFromCoequalizerWithGivenCoequalizer (for IsList, IsCapCategoryMorphism, IsCapCategoryObject)

▷ `UniversalMorphismFromCoequalizerWithGivenCoequalizer(D, tau, C)` (operation)

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

The arguments are a list of morphisms $D = (\beta_i : B \to A)_{i=1\ldots n}$, a morphism $\tau : A \to T$ such that $\tau \circ \beta_i \sim_{B,T} \tau \circ \beta_j$, and an object $C = \mathrm{Coequalizer}(D)$. The output is the morphism $u(\tau) : C \to T$ given by the universal property of the coequalizer.

6.10.8 AddCoequalizer (for IsCapCategory, IsFunction)

▷ `AddCoequalizer(C, F)` (operation)

Returns: nothing

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

6.10.9 AddProjectionOntoCoequalizer (for IsCapCategory, IsFunction)

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

Returns: nothing

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

6.10.10 AddProjectionOntoCoequalizerWithGivenCoequalizer (for IsCapCategory, IsFunction)

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

Returns: nothing

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

6.10.11 AddUniversalMorphismFromCoequalizer (for IsCapCategory, IsFunction)

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

Returns: nothing

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

6.10.12 AddUniversalMorphismFromCoequalizerWithGivenCoequalizer (for IsCapCategory, IsFunction)

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

Returns: nothing

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

6.10.13 CoequalizerFunctorial (for IsList, IsCapCategoryMorphism, IsList)

▷ `CoequalizerFunctorial(Ls, mu, Lr)` (operation)

Returns: a morphism in $\mathrm{Hom}(\mathrm{Coequalizer}((\beta_i)_{i=1\ldots n}), \mathrm{Coequalizer}((\beta'_i)_{i=1\ldots n}))$

The arguments are a list of morphisms $L_s = (\beta_i : B \to A)_{i=1\ldots n}$, a morphism $\mu : A \to A'$, and a list of morphisms $L_r = (\beta'_i : B' \to A')_{i=1\ldots n}$ such that there exists a morphism $\beta : B \to B'$ such that $\beta'_i \circ \beta \sim_{B,A'} \mu \circ \beta_i$ for $i = 1, \ldots n$. The output is the morphism $\mathrm{Coequalizer}((\beta_i)_{i=1}^n) \to \mathrm{Coequalizer}((\beta'_i)_{i=1}^n)$ given by the functorality of the coequalizer.

6.10.14 CoequalizerFunctorialWithGivenCoequalizers (for IsCapCategoryObject, IsList, IsCapCategoryMorphism, IsList, IsCapCategoryObject)

▷ `CoequalizerFunctorialWithGivenCoequalizers(s, Ls, mu, Lr, r)` (operation)

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

The arguments are an object $s = \text{Coequalizer}((\beta_i)_{i=1}^n)$, a list of morphisms $L_s = (\beta_i : B \to A)_{i=1\ldots n}$, a morphism $\mu : A \to A'$, and a list of morphisms $L_r = (\beta_i' : B' \to A')_{i=1\ldots n}$ such that there exists a morphism $\beta : B \to B'$ such that $\beta_i' \circ \beta \sim_{B,A'} \mu \circ \beta_i$ for $i = 1, \ldots n$, and an object $r = \text{Coequalizer}((\beta_i')_{i=1}^n)$. The output is the morphism $s \to r$ given by the functorality of the coequalizer.

6.10.15 AddCoequalizerFunctorialWithGivenCoequalizers (for IsCapCategory, IsFunction)

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

Returns: nothing

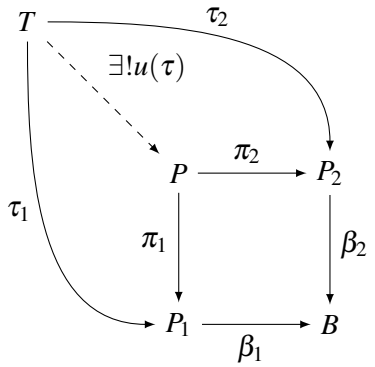
The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `CoequalizerFunctorialWithGivenCoequalizers`. $F : (\text{Coequalizer}((\beta_i)_{i=1}^n), (\beta_i : B \to A)_{i=1\ldots n}, \mu : A \to A', (\beta_i' : B' \to A')_{i=1\ldots n}, \text{Coequalizer}((\beta_i')_{i=1}^n)) \mapsto (\text{Coequalizer}((\beta_i)_{i=1}^n) \to \text{Coequalizer}((\beta_i')_{i=1}^n))$

6.11 Fiber Product

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

- an object P,
- a list of morphisms $\pi = (\pi_i : P \to P_i)_{i=1\ldots n}$ 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 $i = 1, \ldots, n$.

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 $D' = (\beta_i' : P_i' \to B')_{i=1\ldots n}$ and a natural morphism between pullback diagrams (i.e., a collection of morphisms $(\mu_i : P_i \to P_i')_{i=1\ldots n}$ and $\beta : B \to B'$ such that $\beta_i' \circ \mu_i \sim_{P_i,B'} \beta \circ \beta_i$ for $i = 1, \ldots, n$) we obtain a morphism FiberProduct$(D) \to$ FiberProduct(D').



6.11.1 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.

6.11.2 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.

6.11.3 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) $\to \Delta$

6.11.4 IsomorphismFromKernelOfDiagonalDifferenceToFiberProduct (for IsList)

▷ `IsomorphismFromKernelOfDiagonalDifferenceToFiberProduct(D)` (operation)

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

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

6.11.5 IsomorphismFromKernelOfDiagonalDifferenceToFiberProductOp (for IsList, IsCapCategoryMorphism)

▷ `IsomorphismFromKernelOfDiagonalDifferenceToFiberProductOp(D)` (operation)

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

The argument is 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 $\Delta \to$ FiberProduct(D), where Δ denotes the kernel object equalizing the morphisms β_i.

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

6.11.7 IsomorphismFromFiberProductToEqualizerOfDirectProductDiagram (for IsList)

▷ `IsomorphismFromFiberProductToEqualizerOfDirectProductDiagram(D)` (operation)

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

The argument is a list of morphisms $D = (\beta_i : P_i \to B)_{i=1...n}$. The output is a morphism $\text{FiberProduct}(D) \to \Delta$, where Δ denotes the equalizer of the product diagram of the morphisms β_i.

6.11.8 IsomorphismFromFiberProductToEqualizerOfDirectProductDiagramOp (for IsList, IsCapCategoryMorphism)

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

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

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

6.11.9 AddIsomorphismFromFiberProductToEqualizerOfDirectProductDiagram (for IsCapCategory, IsFunction)

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

Returns: nothing

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

6.11.10 IsomorphismFromEqualizerOfDirectProductDiagramToFiberProduct (for IsList)

▷ `IsomorphismFromEqualizerOfDirectProductDiagramToFiberProduct(D)` (operation)

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

The argument is a list of morphisms $D = (\beta_i : P_i \to B)_{i=1...n}$. The output is a morphism $\Delta \to \text{FiberProduct}(D)$, where Δ denotes the equalizer of the product diagram of the morphisms β_i.

6.11.11 IsomorphismFromEqualizerOfDirectProductDiagramToFiberProductOp (for IsList, IsCapCategoryMorphism)

▷ `IsomorphismFromEqualizerOfDirectProductDiagramToFiberProductOp(D)` (operation)

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

The argument is 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 $\Delta \to \text{FiberProduct}(D)$, where Δ denotes the equalizer of the product diagram of the morphisms β_i.

6.11.12 AddIsomorphismFromEqualizerOfDirectProductDiagramToFiberProduct (for IsCapCategory, IsFunction)

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

Returns: nothing

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

6.11.13 DirectSumDiagonalDifference (for IsList)

▷ `DirectSumDiagonalDifference(D)` (operation)

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

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

6.11.14 DirectSumDiagonalDifferenceOp (for IsList, IsCapCategoryMorphism)

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

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

The argument is a list of morphisms $D = (\beta_i : P_i \to B)_{i=1...n}$ 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.

6.11.15 AddDirectSumDiagonalDifference (for IsCapCategory, IsFunction)

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

Returns: nothing

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

6.11.16 FiberProductEmbeddingInDirectSum (for IsList)

▷ `FiberProductEmbeddingInDirectSum(D)` (operation)

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

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

6.11.17 FiberProductEmbeddingInDirectSumOp (for IsList, IsCapCategoryMorphism)

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

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

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

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

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

The output is the fiber product FiberProduct(D).

6.11.20 FiberProductOp (for IsList, IsCapCategoryMorphism)

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

Returns: an object

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

6.11.21 ProjectionInFactorOfFiberProduct (for IsList, IsInt)

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

Returns: a morphism in Hom(FiberProduct$(D), P_k$)

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

6.11.22 ProjectionInFactorOfFiberProductOp (for IsList, IsInt, IsCapCategoryMorphism)

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

Returns: a morphism in Hom(FiberProduct$(D), P_k$)

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

6.11.23 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 $D = (\beta_i : P_i \to B)_{i=1\ldots n}$, an integer k, and an object $P = \mathrm{FiberProduct}(D)$. The output is the k-th projection $\pi_k : P \to P_k$.

6.11.24 UniversalMorphismIntoFiberProduct

▷ `UniversalMorphismIntoFiberProduct(arg)` (function)

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

- The arguments are a list of morphisms $D = (\beta_i : P_i \to B)_{i=1\ldots n}$ and a 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. The output is the morphism $u(\tau) : T \to \mathrm{FiberProduct}(D)$ given by the universal property of the fiber product.
- The arguments are a list of morphisms $D = (\beta_i : P_i \to B)_{i=1\ldots n}$ and morphisms $\tau_1 : T \to P_1, \ldots, \tau_n : T \to P_n$ 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 $u(\tau) : T \to \mathrm{FiberProduct}(D)$ given by the universal property of the fiber product.

6.11.25 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 $D = (\beta_i : P_i \to B)_{i=1\ldots n}$, a 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, and a morphism for method selection. The output is the morphism $u(\tau) : T \to \mathrm{FiberProduct}(D)$ given by the universal property of the fiber product.

6.11.26 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 $D = (\beta_i : P_i \to B)_{i=1\ldots n}$, a 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, and an object $P = \mathrm{FiberProduct}(D)$. The output is the morphism $u(\tau) : T \to P$ given by the universal property of the fiber product.

6.11.27 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`. $F : ((\beta_i : P_i \to B)_{i=1\ldots n}) \mapsto P$

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

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

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

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

6.11.32 FiberProductFunctorial (for IsList, IsList, IsList)

▷ `FiberProductFunctorial(Ls, Lm, Lr)` (operation)

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

The arguments are three lists of morphisms $L_s = (\beta_i : P_i \to B)_{i=1\ldots n}$, $L_m = (\mu_i : P_i \to P'_i)_{i=1\ldots n}$, $L_r = (\beta'_i : P'_i \to B')_{i=1\ldots n}$ having the same length 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 $i = 1, \ldots, n$. The output is the morphism $\mathrm{FiberProduct}((\beta_i)_{i=1\ldots n}) \to \mathrm{FiberProduct}((\beta'_i)_{i=1\ldots n})$ given by the functoriality of the fiber product.

6.11.33 FiberProductFunctorialWithGivenFiberProducts (for IsCapCategoryObject, IsList, IsList, IsList, IsCapCategoryObject)

▷ `FiberProductFunctorialWithGivenFiberProducts(s, Ls, Lm, Lr, r)` (operation)

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

The arguments are an object $s = \mathrm{FiberProduct}((\beta_i)_{i=1\ldots n})$, three lists of morphisms $L_s = (\beta_i : P_i \to B)_{i=1\ldots n}$, $L_m = (\mu_i : P_i \to P_i')_{i=1\ldots n}$, $L_r = (\beta_i' : P_i' \to B')_{i=1\ldots n}$ having the same length 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 $i = 1,\ldots,n$, and an object $r = \mathrm{FiberProduct}((\beta_i')_{i=1\ldots n})$. The output is the morphism $s \to r$ given by the functoriality of the fiber product.

6.11.34 AddFiberProductFunctorialWithGivenFiberProducts (for IsCapCategory, IsFunction)

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

Returns: nothing

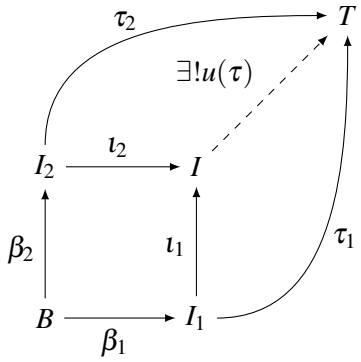
The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `FiberProductFunctorialWithGivenFiberProducts`. $F : (\mathrm{FiberProduct}((\beta_i)_{i=1\ldots n}), (\beta_i : P_i \to B)_{i=1\ldots n}, (\mu_i : P_i \to P_i')_{i=1\ldots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

6.12 Pushout

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

- an object I,
- a list of morphisms $\iota = (\iota_i : I_i \to I)_{i=1\ldots n}$ 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 $\tau = (\tau_i : I_i \to T)_{i=1\ldots n}$ 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 $i = 1,\ldots,n$.

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 $D' = (\beta_i' : B' \to I_i')_{i=1\ldots n}$ and a natural morphism between pushout diagrams (i.e., a collection of morphisms $(\mu_i : I_i \to I_i')_{i=1\ldots n}$ and $\beta : B \to B'$ such that $\beta_i' \circ \beta \sim_{B,I_i'} \mu_i \circ \beta_i$ for $i = 1,\ldots n$) we obtain a morphism $\mathrm{Pushout}(D) \to \mathrm{Pushout}(D')$.



6.12.1 IsomorphismFromPushoutToCokernelOfDiagonalDifference (for IsList)

▷ `IsomorphismFromPushoutToCokernelOfDiagonalDifference(`*D*`)` (operation)

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

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

6.12.2 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 $D = (\beta_i : B \to I_i)_{i=1\ldots n}$ and a morphism for method selection. The output is a morphism $\mathrm{Pushout}(D) \to \Delta$, where Δ denotes the cokernel object coequalizing the morphisms β_i.

6.12.3 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`. $F : ((\beta_i : B \to I_i)_{i=1\ldots n}) \mapsto (\mathrm{Pushout}(D) \to \Delta)$

6.12.4 IsomorphismFromCokernelOfDiagonalDifferenceToPushout (for IsList)

▷ `IsomorphismFromCokernelOfDiagonalDifferenceToPushout(`*D*`)` (operation)

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

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

6.12.5 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 $D = (\beta_i : B \to I_i)_{i=1\ldots n}$ and a morphism for method selection. The output is a morphism $\Delta \to \mathrm{Pushout}(D)$, where Δ denotes the cokernel object coequalizing the morphisms β_i.

6.12.6 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`. $F : ((\beta_i : B \to I_i)_{i=1\ldots n}) \mapsto (\Delta \to \mathrm{Pushout}(D))$

6.12.7 IsomorphismFromPushoutToCoequalizerOfCoproductDiagram (for IsList)

▷ `IsomorphismFromPushoutToCoequalizerOfCoproductDiagram(D)` (operation)

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

The argument is a list of morphisms $D = (\beta_i : B \to I_i)_{i=1\ldots n}$. The output is a morphism $\mathrm{Pushout}(D) \to \Delta$, where Δ denotes the coequalizer of the coproduct diagram of the morphisms β_i.

6.12.8 IsomorphismFromPushoutToCoequalizerOfCoproductDiagramOp (for IsList, IsCapCategoryMorphism)

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

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

The argument is a list of morphisms $D = (\beta_i : B \to I_i)_{i=1\ldots n}$ and a morphism for method selection. The output is a morphism $\mathrm{Pushout}(D) \to \Delta$, where Δ denotes the coequalizer of the coproduct diagram of the morphisms β_i.

6.12.9 AddIsomorphismFromPushoutToCoequalizerOfCoproductDiagram (for IsCapCategory, IsFunction)

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

Returns: nothing

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

6.12.10 IsomorphismFromCoequalizerOfCoproductDiagramToPushout (for IsList)

▷ `IsomorphismFromCoequalizerOfCoproductDiagramToPushout(D)` (operation)

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

The argument is a list of morphisms $D = (\beta_i : B \to I_i)_{i=1\ldots n}$. The output is a morphism $\Delta \to \mathrm{Pushout}(D)$, where Δ denotes the coequalizer of the coproduct diagram of the morphisms β_i.

6.12.11 IsomorphismFromCoequalizerOfCoproductDiagramToPushoutOp (for IsList, IsCapCategoryMorphism)

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

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

The argument is a list of morphisms $D = (\beta_i : B \to I_i)_{i=1\ldots n}$ and a morphism for method selection. The output is a morphism $\Delta \to \mathrm{Pushout}(D)$, where Δ denotes the coequalizer of the coproduct diagram of the morphisms β_i.

6.12.12 AddIsomorphismFromCoequalizerOfCoproductDiagramToPushout (for IsCapCategory, IsFunction)

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

Returns: nothing

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

6.12.13 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 $D = (\beta_i : B \to I_i)_{i=1\ldots n}$. The output is a morphism $B \to \bigoplus_{i=1}^{n} I_i$ such that its cokernel coequalizes the β_i.

6.12.14 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 $D = (\beta_i : B \to I_i)_{i=1\ldots n}$ 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.

6.12.15 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`. $F : (D) \mapsto \mathrm{DirectSumCodiagonalDifference}(D)$

6.12.16 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 $D = (\beta_i : B \to I_i)_{i=1\ldots n}$. The output is the natural projection $\bigoplus_{i=1}^{n} I_i \to \mathrm{Pushout}(D)$.

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

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

6.12.19 Pushout (for IsList)

▷ `Pushout(D)` (operation)

Returns: an object

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

6.12.20 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 $\mathrm{Pushout}(\alpha, \beta)$.

6.12.21 PushoutOp (for IsList, IsCapCategoryMorphism)

▷ `PushoutOp(D)` (operation)

Returns: an object

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

6.12.22 InjectionOfCofactorOfPushout (for IsList, IsInt)

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

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

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

6.12.23 InjectionOfCofactorOfPushoutOp (for IsList, IsInt, IsCapCategoryMorphism)

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

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

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

6.12.24 InjectionOfCofactorOfPushoutWithGivenPushout (for IsList, IsInt, IsCapCategoryObject)

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

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

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

6.12.25 UniversalMorphismFromPushout

▷ `UniversalMorphismFromPushout(arg)` (function)

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

- The arguments are a list of morphisms $D = (\beta_i : B \to I_i)_{i=1\ldots n}$ and a list of morphisms $\tau = (\tau_i : I_i \to T)_{i=1\ldots n}$ 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 $D = (\beta_i : B \to I_i)_{i=1\ldots n}$ and morphisms $\tau_1 : I_1 \to T, \ldots, \tau_n : I_n \to T$ 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.

6.12.26 UniversalMorphismFromPushoutOp (for IsList, IsList, IsCapCategoryMorphism)

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

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

The arguments are a list of morphisms $D = (\beta_i : B \to I_i)_{i=1\ldots n}$, a list of morphisms $\tau = (\tau_i : I_i \to T)_{i=1\ldots n}$ 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.

6.12.27 UniversalMorphismFromPushoutWithGivenPushout (for IsList, IsList, IsCapCategoryObject)

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

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

The arguments are a list of morphisms $D = (\beta_i : B \to I_i)_{i=1\ldots n}$, a list of morphisms $\tau = (\tau_i : I_i \to T)_{i=1\ldots n}$ 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.

6.12.28 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`. $F : ((\beta_i : B \to I_i)_{i=1\ldots n}) \mapsto I$

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

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

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

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

6.12.33 PushoutFunctorial (for IsList, IsList, IsList)

▷ `PushoutFunctorial(Ls, Lm, Lr)` (operation)

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

The arguments are three lists of morphisms $L_s = (\beta_i : B \to I_i)_{i=1\ldots n}$, $L_m = (\mu_i : I_i \to I_i')_{i=1\ldots n}$, $L_r = (\beta_i' : B' \to I_i')_{i=1\ldots n}$ having the same length 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 $i = 1, \ldots n$. The output is the morphism $\mathrm{Pushout}((\beta_i)_{i=1}^n) \to \mathrm{Pushout}((\beta_i')_{i=1}^n)$ given by the functoriality of the pushout.

6.12.34 PushoutFunctorialWithGivenPushouts (for IsCapCategoryObject, IsList, IsList, IsList, IsCapCategoryObject)

▷ `PushoutFunctorialWithGivenPushouts(s, Ls, Lm, Lr, r)` (operation)

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

The arguments are an object $s = \mathrm{Pushout}((\beta_i)_{i=1}^n)$, three lists of morphisms $L_s = (\beta_i : B \to I_i)_{i=1\ldots n}$, $L_m = (\mu_i : I_i \to I_i')_{i=1\ldots n}$, $L_r = (\beta_i' : B' \to I_i')_{i=1\ldots n}$ having the same length 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 $i = 1,\ldots n$, and an object $r = \mathrm{Pushout}((\beta_i')_{i=1}^n)$. The output is the morphism $s \to r$ given by the functoriality of the pushout.

6.12.35 AddPushoutFunctorialWithGivenPushouts (for IsCapCategory, IsFunction)

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

Returns: nothing

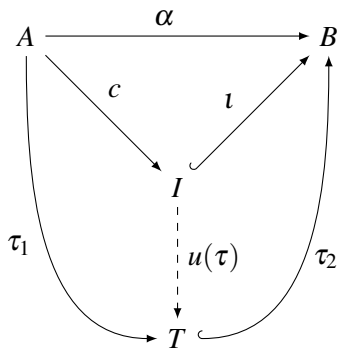
The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `PushoutFunctorial`. $F : (\mathrm{Pushout}((\beta_i)_{i=1}^n), (\beta_i : B \to I_i)_{i=1\ldots n}, (\mu_i : I_i \to I_i')_{i=1\ldots n}, (\beta_i' : B' \to I_i')_{i=1\ldots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

6.13 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*.



6.13.1 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 $\mathrm{im}(\alpha) \to \mathrm{KernelObject}(\mathrm{CokernelProjection}(\alpha))$.

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

6.13.3 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 $\mathrm{KernelObject}(\mathrm{CokernelProjection}(\alpha)) \to \mathrm{im}(\alpha)$.

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

6.13.5 ImageObject (for IsCapCategoryMorphism)

▷ `ImageObject(alpha)` (attribute)

Returns: an object

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

6.13.6 ImageEmbedding (for IsCapCategoryMorphism)

▷ `ImageEmbedding(alpha)` (attribute)

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

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

6.13.7 ImageEmbeddingWithGivenImageObject (for IsCapCategoryMorphism, IsCapCategoryObject)

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

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

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

6.13.8 CoastrictionToImage (for IsCapCategoryMorphism)

▷ `CoastrictionToImage(alpha)` (attribute)

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

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

6.13.9 CoastrictionToImageWithGivenImageObject (for IsCapCategoryMorphism, IsCapCategoryObject)

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

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

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

6.13.10 UniversalMorphismFromImage (for IsCapCategoryMorphism, IsList)

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

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

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

6.13.11 UniversalMorphismFromImageWithGivenImageObject (for IsCapCategoryMorphism, IsList, IsCapCategoryObject)

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

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

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

6.13.12 AddImageObject (for IsCapCategory, IsFunction)

▷ `AddImageObject(C, F)` (operation)

Returns: nothing

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

6.13.13 AddImageEmbedding (for IsCapCategory, IsFunction)

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

Returns: nothing

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

6.13.14 AddImageEmbeddingWithGivenImageObject (for IsCapCategory, IsFunction)

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

Returns: nothing

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

6.13.15 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`. $F : \alpha \mapsto c$.

6.13.16 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`. $F : (\alpha, I) \mapsto c$.

6.13.17 AddUniversalMorphismFromImage (for IsCapCategory, IsFunction)

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

Returns: nothing

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

6.13.18 AddUniversalMorphismFromImageWithGivenImageObject (for IsCapCategory, IsFunction)

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

Returns: nothing

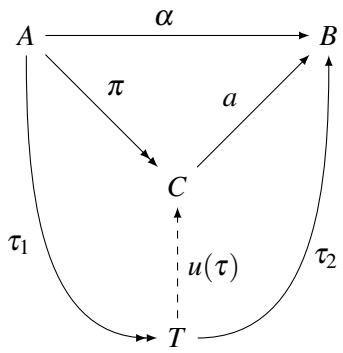
The arguments are a category C and a function F. This operations adds the given function F to the category for the basic operation `UniversalMorphismFromImageWithGivenImageObject`. $F : (\alpha, \tau, I) \mapsto u(\tau)$.

6.14 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*.



6.14.1 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) $\mathrm{coim}(\alpha) \to \mathrm{im}(\alpha)$.

6.14.2 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) $C \to I$.

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

6.14.4 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) $\mathrm{im}(\alpha) \to \mathrm{coim}(\alpha)$.

6.14.5 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) $I \to C$.

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

6.14.7 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 $\mathrm{coim}(\alpha) \to \mathrm{CokernelObject}(\mathrm{KernelEmbedding}(\alpha))$.

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

6.14.9 IsomorphismFromCokernelOfKernelToCoimage (for IsCapCategoryMorphism)

▷ `IsomorphismFromCokernelOfKernelToCoimage(alpha)` (attribute)

Returns: a morphism in Hom(CokernelObject(KernelEmbedding(α)), coim(α)).

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

6.14.10 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`. $F : \alpha \mapsto$ (CokernelObject(KernelEmbedding(α)) $\to$ coim(α)).

6.14.11 Coimage (for IsCapCategoryMorphism)

▷ `Coimage(alpha)` (attribute)

Returns: an object

The argument is a morphism α. The output is the coimage coim(α).

6.14.12 CoimageProjection (for IsCapCategoryObject)

▷ `CoimageProjection(C)` (attribute)

Returns: a morphism in 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 $\pi : A \twoheadrightarrow C$.

6.14.13 CoimageProjection (for IsCapCategoryMorphism)

▷ `CoimageProjection(alpha)` (attribute)

Returns: a morphism in Hom(A, coim(α))

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

6.14.14 CoimageProjectionWithGivenCoimage (for IsCapCategoryMorphism, IsCapCategoryObject)

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

Returns: a morphism in Hom(A,C)

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

6.14.15 AstrictionToCoimage (for IsCapCategoryObject)

▷ `AstrictionToCoimage(C)` (attribute)

Returns: a morphism in 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 $a : C \to B$.

6.14.16 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 $a : \mathrm{coim}(\alpha) \to B$.

6.14.17 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 $a : C \to B$.

6.14.18 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.

6.14.19 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.

6.14.20 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`. $F : \alpha \mapsto C$

6.14.21 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`. $F : \alpha \mapsto \pi$

6.14.22 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`. $F : (\alpha, C) \mapsto \pi$

6.14.23 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`. $F : \alpha \mapsto a$

6.14.24 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`. $F : (\alpha, C) \mapsto a$

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

6.14.26 AddUniversalMorphismIntoCoimageWithGivenCoimage (for IsCapCategory, IsFunction)

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

Returns: nothing

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

Chapter 7

Add Functions

This section describes the overall structure of Add-functions and the functions installed by them.

7.1 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.

- Full prefunction: The full prefuction has the same semantics as the prefunction, but can perform additional, very costly checks. They are disabled by default.

- 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 KernelEmbeddingedding, 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.

7.2 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 available in the category. Also, some derivations, which are triggered by the setting of the primitive value, might be available.

7.3 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. Please note that if you have a method selection argument for your function, you need to give the argument_list to explicitly state which argument is the method selection argument.

- return_type (optional): The return type can be one of the following:

  - `object` or `object_or_fail`,
  - `morphism` or `morphism_or_fail`,
  - `twocell`,
  - `bool`,
  - `other_object`,
  - `other_morphism`.

  If it is one of the first three options, 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.

- functorial (optional): If an object has a corresponding functorial function, e.g., KernelObject and KernelObjectFunctorial, the name of the functorial is stored as a string.

- number_of_diagram_arguments: Specifies how many of the arguments (counting from the first argument) of the function specify the diagram of the universal object. Default value is 1.

- dual_arguments_reversed: Boolean, marks whether for the call of the dual operation all arguments have to be given in reversed order.

- dual_preprocessor_func: let f be an operation with dual operation g. For the automatic installation of g from f, the arguments given to g are preprocessed by this given function.

- dual_postprocessor_func: let f be an operation with dual operation g. For the automatic installation of g from f, the computed value of f is postprocessed by the given function.

- zero_arguments_for_add_method: the add method of this operation should get a function without arguments

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.

7.4 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.

7.5 Prepare functions

7.5.1 CAPOperationPrepareFunction

▷ `CAPOperationPrepareFunction(`*`prepare_function, category, func`*`)` (function)

Returns: a function

Given a non-CAP-conform function for any of the categorical operations, i.e., a function that computes the direct sum of two objects instead of a list of objects, this function wraps the function with a wrapper function to fit in the CAP context. For the mentioned binary direct sum one can call this function with `"BinaryDirectSumToDirectSum"` as *`prepare_function`*, the category, and the binary direct sum function. The function then returns a function that can be used for the direct sum categorical operation.

Note that *`func`* is not handled by the CAP caching mechanism and that the use of prepare functions is incompatible with `WithGiven` operations. Thus, one has to ensure manually that the equality and typing specifications are fulfilled.

7.5.2 CAPAddPrepareFunction

▷ `CAPAddPrepareFunction(`*`prepare_function, name, doc_string[, precondition_list]`*`)` (function)

Adds a prepare function to the list of CAP's prepare functions. The first argument is the prepare function itself. It should always be a function that takes a category and a function and returns a function. The argument *`name`* is the name of the prepare function, which is used in `CAPOperationPrepareFunction`. The argument *`doc_string`* should be a short string describing the functions. The optional argument *`precondition_list`* can describe preconditions for the prepare function to work, i.e., if the category does need to have PreCompose computable. This information is also recovered automatically from the prepare function itself, so the *`precondition_list`* is only necessary if the function needed is not explicitly used in the prepare function, e.g., if you use `+` instead of `AdditionForMorphisms`.

7.5.3 ListCAPPrepareFunctions

▷ `ListCAPPrepareFunctions(`*`arg`*`)` (function)

Lists all prepare functions.

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)

8.1.3 DeactivateDerivationInfo

▷ `DeactivateDerivationInfo(`*arg*`)` (function)

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)

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`. The Option `ConditionsListComplete` indicates if the manually given list of preconditions for this derivation is complete or should be extended by looking for categorical operations in the function body. The value `false` indicates it is not complete, every other value that it is complete. Default is `false`.

8.2.3 DerivationName (for IsDerivedMethod)

▷ `DerivationName(`*d*`)` (attribute)

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)

Extra weight for the derivation.

8.2.5 DerivationFunctionsWithExtraFilters (for IsDerivedMethod)

▷ `DerivationFunctionsWithExtraFilters(`*d*`)` (attribute)

The implementation(s) of the derivation, together with lists of extra filters for each implementation.

8.2.6 CategoryFilter (for IsDerivedMethod)

▷ `CategoryFilter(`*d*`)` (attribute)

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)

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)

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)

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)

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 basic 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)

Add a derivation to a derivation graph.

8.3.5 AddDerivation (for IsDerivedMethodGraph, IsFunction, IsDenseList, IsObject)

▷ AddDerivation(*arg1, arg2, arg3, arg4*) (operation)

8.3.6 AddDerivation (for IsDerivedMethodGraph, IsFunction, IsDenseList)

▷ AddDerivation(*arg1, arg2, arg3*) (operation)

8.3.7 AddDerivation (for IsDerivedMethodGraph, IsFunction, IsFunction)

▷ AddDerivation(*arg1, arg2, arg3*) (operation)

8.3.8 AddDerivationPair (for IsDerivedMethodGraph, IsFunction, IsFunction, IsDenseList, IsDenseList, IsDenseList)

▷ AddDerivationPair(*arg1, arg2, arg3, arg4, arg5, arg6*) (operation)

8.3.9 AddDerivationPair (for IsDerivedMethodGraph, IsFunction, IsFunction, IsDenseList, IsDenseList)

▷ AddDerivationPair(*arg1, arg2, arg3, arg4, arg5*) (operation)

8.3.10 AddDerivationPair (for IsDerivedMethodGraph, IsFunction, IsFunction, IsDenseList, IsFunction, IsFunction)

▷ AddDerivationPair(*arg1, arg2, arg3, arg4, arg5, arg6*) (operation)

8.3.11 AddDerivationPair (for IsDerivedMethodGraph, IsFunction, IsFunction, IsFunction, IsFunction, IsFunction)

▷ AddDerivationPair(*arg1, arg2, arg3, arg4, arg5*) (operation)

8.3.12 AddDerivationToCAP

▷ AddDerivationToCAP(*arg*) (function)

8.3.13 AddDerivationPairToCAP

▷ AddDerivationPairToCAP(*arg*) (function)

8.3.14 AddWithGivenDerivationPairToCAP

▷ AddWithGivenDerivationPairToCAP(*arg*) (function)

8.3.15 Operations (for IsDerivedMethodGraph)

▷ Operations(*G*) (attribute)

Gives the operations in the graph *G*, as a list of strings.

8.3.16 DerivationsUsingOperation (for IsDerivedMethodGraph, IsString)

▷ DerivationsUsingOperation(*G, op_name*) (operation)

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)

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)

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 the derivation graph used by the operation weight list `owl`.

8.4.4 CategoryOfOperationWeightList (for IsOperationWeightList)

▷ `CategoryOfOperationWeightList(owl)` (attribute)

Returns the CAP category associated to the operation weight list `owl`.

8.4.5 CurrentOperationWeight (for IsOperationWeightList, IsString)

▷ `CurrentOperationWeight(owl, op_name)` (operation)

Returns the current weight of the operation named `op_name`.

8.4.6 OperationWeightUsingDerivation (for IsOperationWeightList, IsDerivedMethod)

▷ `OperationWeightUsingDerivation(owl, d)` (operation)

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 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)

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)

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 Saturate (for IsOperationWeightList)

▷ `Saturate(`*`owl`*`)` (operation)

Saturates the derivation graph, i.e., calls reevaluate until no more changes in the derivation graph occur.

8.4.11 AddPrimitiveOperation (for IsOperationWeightList, IsString, IsInt)

▷ `AddPrimitiveOperation(`*`owl, op_name, weight`*`)` (operation)

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).

8.4.12 PrintDerivationTree (for IsOperationWeightList, IsString)

▷ `PrintDerivationTree(`*`owl, op_name`*`)` (operation)

Print a tree representation of the way the operation named *`op_name`* is implemented in the category of the operation weight list *`owl`*.

8.4.13 PrintTree (for IsObject, IsFunction, IsFunction)

▷ `PrintTree(arg1, arg2, arg3)` (operation)

Prints a tree structure.

8.4.14 PrintTreeRec (for IsObject, IsFunction, IsFunction, IsInt)

▷ `PrintTreeRec(arg1, arg2, arg3, arg4)` (operation)

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)

Create an empty string min heap.

8.5.3 Add (for IsStringMinHeap, IsString, IsInt)

▷ `Add(H, string, key)` (operation)

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)

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)

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 `true` if the heap *`H`* is empty, `false` otherwise.

8.5.7 HeapSize (for IsStringMinHeap)

▷ `HeapSize(`*`H`*`)` (operation)

Returns the number of nodes in the heap *`H`*.

8.5.8 Contains (for IsStringMinHeap, IsString)

▷ `Contains(`*`H, string`*`)` (operation)

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)

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)

Heapify the heap *`H`*, starting from index *`i`*. This is an internal function.

Chapter 9

Technical Details

9.1 The Category Cat

9.1.1 ObjectCache (for IsCapFunctor)

▷ `ObjectCache(`*`functor`*`)` (attribute)

Returns: IsCachingObject

Retuns the caching object which stores the results of the functor *`functor`* applied to objects.

9.1.2 MorphismCache (for IsCapFunctor)

▷ `MorphismCache(`*`functor`*`)` (attribute)

Returns: IsCachingObject

Retuns the caching object which stores the results of the functor *`functor`* applied to morphisms.

9.2 Install Functions for IsWellDefined

9.2.1 InstallMethodWithToDoForIsWellDefined

▷ `InstallMethodWithToDoForIsWellDefined(`*`arg`*`)` (function)

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.

9.2.2 InstallSetWithToDoForIsWellDefined (for IsObject, IsString, IsList)

▷ InstallSetWithToDoForIsWellDefined(*arg1, arg2, arg3*) (operation)

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.

9.2.3 DeclareAttributeWithToDoForIsWellDefined

▷ DeclareAttributeWithToDoForIsWellDefined(*arg*) (function)

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.

9.2.4 DeclareFamilyProperty

▷ DeclareFamilyProperty(*arg*) (function)

9.2.5 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.

9.2.6 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.

9.2.7 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.

9.2.8 CAP_INTERNAL_FIND_APPEARANCE_OF_SYMBOL_IN_FUNCTION

▷ CAP_INTERNAL_FIND_APPEARANCE_OF_SYMBOL_IN_FUNCTION(*function, symbol_list, loop_multiple, replacement_record*) (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. Moreover, if appearances of found strings should be replaced by collections of other strings, then these can be specified in the replacement record.

9.2.9 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.

9.2.10 CachingStatistic

▷ CachingStatistic(*category[, operation]*) (function)

Prints statistics for all caches in *category*. If *operation* is given (as a string), only statistics for the given operation cache is stored.

9.2.11 BrowseCachingStatistic

▷ BrowseCachingStatistic(*category*) (function)

Displays statistics for all caches in *category*. in a Browse window. Here "status" indicates if the cache is weak, strong, or inactive, "hits" is the number of successful cache accesses, "misses" the number of unsuccessful cache accesses, and "stored" the number of objects currently stored in the cache.

Chapter 10

Limits and Colimits

This section describes the support for limits and colimits in CAP. All notions defined in the following are considered with regard to limits, not colimits, except if explicitly stated otherwise. In particular, the diagram specification specifies a diagram over which the limit is taken. The colimit in turn is taken over the opposite diagram.

10.1 Specification of Limits and Colimits

A record specifying a limit in CAP has the following entries:

- object_specification: see below
- morphism_specifiation: see below
- limit_object_name: the name of the method returning the limit object, e.g. `DirectProduct` or `KernelObject`
- limit_projection_name (optional): the name of the method returning the projection(s) from the limit object, e.g. `ProjectionInFactorOfDirectProduct` or `KernelEmbedding`. Defaults to `Concatenation( "ProjectionInFactorOf", limit_object_name )`.
- limit_universal_morphism_name (optional): the name of the method returning the universal morphism into the limit object, e.g. `UniversalMorphismIntoDirectProduct` or `KernelLift`. Defaults to `Concatenation( "UniversalMorphismInto", limit_object_name )`.
- colimit_object_name: the name of the method returning the colimit object, e.g. `Coproduct` or `CokernelObject`
- colimit_injection_name (optional): the name of the method returning the injection(s) into the colimit object, e.g. `InjectionOfCofactorOfCoproduct` or `CokernelProjection`. Defaults to `Concatenation( "InjectionOfCofactorOf", colimit_object_name )`.
- colimit_universal_morphism_name (optional): the name of the method returning the universal morphism from the colimit object, e.g. `UniversalMorphismFromCoproduct` or `CokernelColift`. Defaults to `Concatenation( "UniversalMorphismFrom", colimit_object_name )`.

limit_object_name and colimit_object_name can be the same, e.g. for `DirectSum` or `ZeroObject`.

The object_specification and morphism_specification together specify the shape of the diagram defining the limit or colimit. The syntax is the following:

- object_specification is a list of strings. Only the strings "fixedobject" and "varobject" are allowed as entries of the list. These are called "types" in the following.
- morphism_specification is a list of triples. The first and third entry of a triple are integers greater or equal to 1 and less or equal to `Length( object_specification )`. The second entry is one of the following strings: "fixedmorphism", "varmorphism", "zeromorphism". This entry is called "type" in the following.

Semantics is given as follows:

- The type "fixedobject" specifies a single object. The type "varobject" specifies arbitrarily many objects.
- The first and the third entry of a triple specify the source and range of a morphism (or multiple morphisms) encoded by the position in object_specification respectively. The type "fixedmorphism" specifies a single morphism. In this case, source and range can only be of type "fixedobject", not of type "varobject". The type "varmorphism" specifies arbitrarily many morphisms. In this case, if the source (resp. range) is of type "fixedobject" all the morphisms must have the same source (resp. range). On the contrary, if the source (resp. range) is of the type "varobject", the objects correspond one-to-one to the sources (resp. ranges) of the morphisms. The type "zeromorphism" is currently ignored but will be endowed with semantics in the future.

For example, a FiberProduct diagram consists of arbitrarily many morphisms which have arbitrary sources but the same common range. This can be expressed as follows:

Code

```
rec(
  object_specification := [ "fixedobject", "varobject" ],
  morphism_specification := [ [ 2, "varmorphism", 1 ] ],
  limit_object_name := "FiberProduct",
  colimit_object_name := "Pushout",
)
```

Note that not all diagrams which can be expressed with the above are actually supported. For now, at most one unbound object (see below for the definition of "unbound") may be of type "varobject", and if there is such an unbound object it must be the last one among the unbound objects. Similarly, at most one unbound morphism may be of type "varmorphism", and if there is such an unbound morphism it must be the last one among the unbound morphisms.

10.2 Enhancing Limit Specifications

The function CAP_INTERNAL_ENHANCE_NAME_RECORD_LIMITS takes a list of limits (given by records as explained above), and computes some additional properties. For example, the number of so-called unbound objects, unbound morphisms and targets is computed. The term "unbound" signifies that for creating a concrete diagram, these objects or morphisms have to be specified by the user because they cannot be derived by CAP:

- Unbound morphisms are the triples which are of type "fixedmorphism" or "varmorphism".
- Unbound objects are the objects which are not source or range of an unbound morphism.

Finally, targets are the objects which are not the range of a morphism. These are of interest for the following reason: for limits, only projections into targets are relevant because the projections into other objects can simply be computed by composition. Similarly, one only has to give morphisms into these targets to compute a universal morphism.

The number of unbound objects, unbound morphisms and targets is expressed by the integers 0, 1 and 2:

- 0: no such object/morphism/target exists
- 1: there exists exactly one such object/target of type "fixedobject" respectively exactly one such morphism of type "fixedmorphism"
- 2: else

10.3 Validating entries of a method name record which are part of a limit or colimit

The function CAP_INTERNAL_VALIDATE_LIMITS_IN_NAME_RECORD takes a method name record and a list of enhanced limits, and validates the entries of the method name record. Prefunctions, full prefunctions and postfunctions are excluded from the validation.

Chapter 11

Examples and Tests

11.1 Functors

We create a binary functor F with one covariant and one contravariant component in two ways. Here is the first way to model a binary functor:

Example

```
gap> field := HomalgFieldOfRationals( );;
gap> vec := LeftPresentations( field );;
gap> F := CapFunctor( "CohomForVec", [ vec, [ vec, true ] ], vec );;
gap> obj_func := function( A, B ) return TensorProductOnObjects( A, DualOnObjects( B ) ); end;;
gap> mor_func := function( source, alpha, beta, range ) return TensorProductOnMorphismsWithGiven
gap> AddObjectFunction( F, obj_func );;
gap> AddMorphismFunction( F, mor_func );;
```

CAP regards F as a binary functor on a technical level, as we can see by looking at its input signature:

Example

```
gap> InputSignature( F );
[ [ Category of left presentations of Q, false ], [ Category of left presentations of Q, true ]
```

We can see that `ApplyFunctor` works both on two arguments and on one argument (in the product category).

Example

```
gap> V1 := TensorUnit( vec );;
gap> V3 := DirectSum( V1, V1, V1 );;
gap> pi1 := ProjectionInFactorOfDirectSum( [ V1, V1 ], 1 );;
gap> pi2 := ProjectionInFactorOfDirectSum( [ V3, V1 ], 1 );;
gap> value1 := ApplyFunctor( F, pi1, pi2 );;
gap> input := Product( pi1, Opposite( pi2 ) );;
gap> value2 := ApplyFunctor( F, input );;
gap> IsCongruentForMorphisms( value1, value2 );
true
```

Here is the second way to model a binary functor:

Example

```
gap> F2 := CapFunctor( "CohomForVec2", Product( vec, Opposite( vec ) ), vec );;
gap> AddObjectFunction( F2, a -> obj_func( a[1], Opposite( a[2] ) ) );;
gap> AddMorphismFunction( F2, function( source, datum, range ) return mor_func( source, datum[1]
```

```
gap> value3 := ApplyFunctor( F2,input );;
gap> IsCongruentForMorphisms( value1, value3 );
true
```

CAP regards $F2$ as a unary functor on a technical level, as we can see by looking at its input signature:

Example

```
gap> InputSignature( F2 );
[ [ Product of: Category of left presentations of Q, Opposite of Category of left presentations
```

Installation of the first functor as a GAP-operation. It will be installed both as a unary and binary version.

Example

```
gap> InstallFunctor( F, "F_installation" );;
gap> F_installation( pi1, pi2 );;
gap> F_installation( input );;
gap> F_installationOnObjects( V1, V1 );;
gap> F_installationOnObjects( Product( V1, Opposite( V1 ) ) );;
gap> F_installationOnMorphisms( pi1, pi2 );;
gap> F_installationOnMorphisms( input );;
```

Installation of the second functor as a GAP-operation. It will be installed only as a unary version.

Example

```
gap> InstallFunctor( F2, "F_installation2" );;
gap> F_installation2( input );;
gap> F_installation2OnObjects( Product( V1, Opposite( V1 ) ) );;
gap> F_installation2OnMorphisms( input );;
```

11.2 Homomorphism structure

Example

```
gap> ReadPackage( "CAP", "examples/testfiles/FieldAsCategory.gi" );;
gap> Q := HomalgFieldOfRationals();;
gap> Qoid := FieldAsCategory( Q );;
gap> a := FieldAsCategoryMorphism( 1/2, Qoid );;
gap> b := FieldAsCategoryMorphism( -2/3, Qoid );;
gap> u := FieldAsCategoryUniqueObject( Qoid );;
gap> IsCongruentForMorphisms( a,
>     InterpretMorphismFromDinstinguishedObjectToHomomorphismStructureAsMorphism(
>         u,u,
>         InterpretMorphismAsMorphismFromDinstinguishedObjectToHomomorphismStructure(
>             a
>         )
>     )
> );
true
gap> c := FieldAsCategoryMorphism( 3, Qoid );;
gap> d := FieldAsCategoryMorphism( 0, Qoid );;
gap> left_coeffs := [ [ a, b ], [ c, d ] ];;
gap> right_coeffs := [ [ PreCompose( a, b ), PreCompose( b, c ) ], [ c, PreCompose( a, a ) ] ];;
gap> right_side := [ a, b ];;
gap> solution :=
```

```
>     SolveLinearSystemInAbCategory(
>     left_coeffs,
>     right_coeffs,
>     right_side
> );;
gap> ForAll( [ 1, 2 ], i ->
>     IsCongruentForMorphisms(
>         Sum( List( [ 1, 2 ], j -> PreCompose( [ left_coeffs[i][j], solution[j], right_coeffs[i
>         right_side[i]
>     )
> );
true
gap> Lift( c, d );
fail
gap> Lift( d, c );
0
gap> Colift( c, d );
0
gap> Colift( d, c );
fail
```

11.3 Spectral Sequences

Example

```
gap> ZZ := HomalgRingOfIntegersInSingular( );
Z
gap> C1 := FreeLeftPresentation( 1, ZZ );
<An object in Category of left presentations of Z>
gap> C2 := FreeLeftPresentation( 2, ZZ );
<An object in Category of left presentations of Z>
gap> h1 := PresentationMorphism( C2, HomalgMatrix( [ [ 0 ], [ 4 ] ], ZZ ), C1 );
<A morphism in Category of left presentations of Z>
gap> h2 := PresentationMorphism( C2, HomalgMatrix( [ [ 0 ], [ 2 ] ], ZZ ), C1 );
<A morphism in Category of left presentations of Z>
gap> v1 := PresentationMorphism( C2, HomalgMatrix( [ [ 2, 0 ], [ 1, 2 ] ], ZZ ), C2 );
<A morphism in Category of left presentations of Z>
gap> v2 := PresentationMorphism( C1, HomalgMatrix( [ [ 4 ] ], ZZ ), C1 );
<A morphism in Category of left presentations of Z>
gap> cocomplex_h1 := CocomplexFromMorphismList( [ h1 ] );
<An object in Cocomplex category of Category of left presentations of Z>
gap> cocomplex_h2 := CocomplexFromMorphismList( [ h2 ] );
<An object in Cocomplex category of Category of left presentations of Z>
gap> cocomplex_mor := CochainMap( cocomplex_h2, [ v1, v2 ], cocomplex_h1 );
<A morphism in Cocomplex category of Category of left presentations of Z>
gap> Zmod := CapCategory( C1 );
Category of left presentations of Z
gap> CH0 := CohomologyFunctor( Zmod, 0 );
0-th cohomology functor of Category of left presentations of Z
gap> cmor0 := ApplyFunctor( CH0, cocomplex_mor );
<A morphism in Category of left presentations of Z>
gap> Display( UnderlyingMatrix( cmor0 ) );
2
```

```
gap> CH1 := CohomologyFunctor( Zmod, 1 );
1-th cohomology functor of Category of left presentations of Z
gap> cmor1 := ApplyFunctor( CH1, cocomplex_mor );
<A morphism in Category of left presentations of Z>
gap> Display( UnderlyingMatrix( cmor1 ) );
4
gap> ToComplex := CocomplexToComplexFunctor( Zmod );
Cocomplex to complex functor of Category of left presentations of Z
gap> complex_mor := ApplyFunctor( ToComplex, cocomplex_mor );
<A morphism in Complex category of Category of left presentations of Z>
gap> H0 := HomologyFunctor( Zmod, 0 );
0-th homology functor of Category of left presentations of Z
gap> mor0 := ApplyFunctor( H0, complex_mor );
<A morphism in Category of left presentations of Z>
gap> Display( UnderlyingMatrix( mor0 ) );
2
gap> Hm1 := HomologyFunctor( Zmod, -1 );
-1-th homology functor of Category of left presentations of Z
gap> mor1 := ApplyFunctor( Hm1, complex_mor );
<A morphism in Category of left presentations of Z>
gap> Display( UnderlyingMatrix( mor1 ) );
4
```

Example

```
gap> QQ := HomalgFieldOfRationalsInSingular( );;
gap> R := QQ * "x,y";
Q[x,y]
gap> SetRecursionTrapInterval( 10000 );
gap> category := LeftPresentations( R );
Category of left presentations of Q[x,y]
gap> S := FreeLeftPresentation( 1, R );
<An object in Category of left presentations of Q[x,y]>
gap> object_func := function( i ) return S; end;
function( i ) ... end
gap> morphism_func := function( i ) return IdentityMorphism( S ); end;
function( i ) ... end
gap> C0 := ZFunctorObjectExtendedByInitialAndIdentity( object_func, morphism_func, category, 0,
<An object in Functors from integers into Category of left presentations of Q[x,y]>
gap> S2 := FreeLeftPresentation( 2, R );
<An object in Category of left presentations of Q[x,y]>
gap> C1 := ZFunctorObjectFromMorphismList( [ InjectionOfCofactorOfDirectSum( [ S2, S ], 1 ) ], 2
<An object in Functors from integers into Category of left presentations of Q[x,y]>
gap> C1 := ZFunctorObjectExtendedByInitialAndIdentity( C1, 2, 3 );
<An object in Functors from integers into Category of left presentations of Q[x,y]>
gap> C2 := ZFunctorObjectFromMorphismList( [ InjectionOfCofactorOfDirectSum( [ S, S ], 1 ) ], 3
<An object in Functors from integers into Category of left presentations of Q[x,y]>
gap> C2 := ZFunctorObjectExtendedByInitialAndIdentity( C2, 3, 4 );
<An object in Functors from integers into Category of left presentations of Q[x,y]>
gap> delta_1_3 := PresentationMorphism( C1[3], HomalgMatrix( [ [ "x^2" ], [ "xy" ], [ "y^3"] ],
<A morphism in Category of left presentations of Q[x,y]>
gap> delta_1_2 := PresentationMorphism( C1[2], HomalgMatrix( [ [ "x^2" ], [ "xy" ] ], 2, 1, R ),
<A morphism in Category of left presentations of Q[x,y]>
gap> delta1 := ZFunctorMorphism( C1, [ UniversalMorphismFromInitialObject( C0[1] ), UniversalMor
<A morphism in Functors from integers into Category of left presentations of Q[x,y]>
```

```
gap> delta1 := ZFunctorMorphismExtendedByInitialAndIdentity( delta1, 0, 3 );
<A morphism in Functors from integers into Category of left presentations of Q[x,y]>
gap> delta1 := AsAscendingFilteredMorphism( delta1 );
<A morphism in Ascending filtered object category of Category of left presentations of Q[x,y]>
gap> delta_2_3 := PresentationMorphism( C2[3], HomalgMatrix( [ [ "y", "-x", "0" ] ], 1, 3, R ),
<A morphism in Category of left presentations of Q[x,y]>
gap> delta_2_4 := PresentationMorphism( C2[4], HomalgMatrix( [ [ "y", "-x", "0" ], [ "0", "y^2",
<A morphism in Category of left presentations of Q[x,y]>
gap> delta2 := ZFunctorMorphism( C2, [  UniversalMorphismFromInitialObject( C1[2] ), delta_2_3,
<A morphism in Functors from integers into Category of left presentations of Q[x,y]>
gap> delta2 := ZFunctorMorphismExtendedByInitialAndIdentity( delta2, 2, 4 );
<A morphism in Functors from integers into Category of left presentations of Q[x,y]>
gap> delta2 := AsAscendingFilteredMorphism( delta2 );
<A morphism in Ascending filtered object category of Category of left presentations of Q[x,y]>
gap> SetIsAdditiveCategory( CategoryOfAscendingFilteredObjects( category ), true );
gap> complex := ZFunctorObjectFromMorphismList( [ delta2, delta1 ], -2 );
<An object in Functors from integers into Ascending filtered object category of Category of left
gap> complex := AsComplex( complex );
<An object in Complex category of Ascending filtered object category of Category of left present
gap> LessGenFunctor := FunctorLessGeneratorsLeft( R );
Less generators for Category of left presentations of Q[x,y]
gap> s := SpectralSequenceEntryOfAscendingFilteredComplex( complex, 0, 0, 0 );
<A morphism in Generalized morphism category of Category of left presentations of Q[x,y]>
gap> Display( UnderlyingMatrix( ApplyFunctor( LessGenFunctor, UnderlyingHonestObject( Source( s
(an empty 0 x 1 matrix)
gap> s := SpectralSequenceEntryOfAscendingFilteredComplex( complex, 1, 0, 0 );
<A morphism in Generalized morphism category of Category of left presentations of Q[x,y]>
gap> Display( UnderlyingMatrix( ApplyFunctor( LessGenFunctor, UnderlyingHonestObject( Source( s
(an empty 0 x 1 matrix)
gap> s := SpectralSequenceEntryOfAscendingFilteredComplex( complex, 2, 0, 0 );
<A morphism in Generalized morphism category of Category of left presentations of Q[x,y]>
gap> Display( UnderlyingMatrix( ApplyFunctor( LessGenFunctor, UnderlyingHonestObject( Source( s
(an empty 0 x 1 matrix)
gap> s := SpectralSequenceEntryOfAscendingFilteredComplex( complex, 3, 0, 0 );
<A morphism in Generalized morphism category of Category of left presentations of Q[x,y]>
gap> Display( UnderlyingMatrix( ApplyFunctor( LessGenFunctor, UnderlyingHonestObject( Source( s
x*y,
x^2
gap> s := SpectralSequenceEntryOfAscendingFilteredComplex( complex, 4, 0, 0 );
<A morphism in Generalized morphism category of Category of left presentations of Q[x,y]>
gap> Display( UnderlyingMatrix( ApplyFunctor( LessGenFunctor, UnderlyingHonestObject( Source( s
x*y,
x^2,
y^3
gap> s := SpectralSequenceEntryOfAscendingFilteredComplex( complex, 5, 0, 0 );
<A morphism in Generalized morphism category of Category of left presentations of Q[x,y]>
gap> Display( UnderlyingMatrix( ApplyFunctor( LessGenFunctor, UnderlyingHonestObject( Source( s
x*y,
x^2,
y^3
gap> s := SpectralSequenceDifferentialOfAscendingFilteredComplex( complex, 3, 3, -2 );
<A morphism in Category of left presentations of Q[x,y]>
gap> Display( UnderlyingMatrix( ApplyFunctor( LessGenFunctor, s ) ) );
```

```
y^3
gap> AscToDescFunctor := AscendingToDescendingFilteredObjectFunctor( category );
Ascending to descending filtered object functor of Category of left presentations of Q[x,y]
gap> cocomplex := ZFunctorObjectFromMorphismList( [ ApplyFunctor( AscToDescFunctor, delta2 ), Ap
<An object in Functors from integers into Descending filtered object category of Category of 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
```

11.4 Liftable

Example

```
gap> field := HomalgFieldOfRationals( );;
gap> V := VectorSpaceObject( 1, field );;
gap> W := VectorSpaceObject( 2, field );;
gap> alpha := VectorSpaceMorphism( V, HomalgMatrix( [ [ 1, -1 ] ], 1, 2, field ), W );;
gap> beta := VectorSpaceMorphism( W, HomalgMatrix( [ [ 1, 2 ], [ 3, 4 ] ], 2, 2, field ), W );;
gap> IsLiftable( alpha, beta );
true
gap> IsLiftable( beta, alpha );
false
gap> IsLiftableAlongMonomorphism( beta, alpha );
true
gap> gamma := VectorSpaceMorphism( W, HomalgMatrix( [ [ 1 ], [ 1 ] ], 2, 1, field ), V );;
gap> IsColiftable( beta, gamma );
true
gap> IsColiftable( gamma, beta );
false
gap> IsColiftableAlongEpimorphism( beta, gamma );
true
```

11.5 Monoidal Categories

Example

```
gap> ZZ := HomalgRingOfIntegers();;
gap> Ml := AsLeftPresentation( HomalgMatrix( [ [ 2 ] ], 1, 1, ZZ ) );
<An object in Category of left presentations of Z>
gap> Nl := AsLeftPresentation( HomalgMatrix( [ [ 3 ] ], 1, 1, ZZ ) );
<An object in Category of left presentations of Z>
gap> Tl := TensorProductOnObjects( Ml, Nl );
<An object in Category of left presentations of Z>
gap> Display( UnderlyingMatrix( Tl ) );
[ [  3 ],
  [  2 ] ]
gap> IsZeroForObjects( Tl );
true
gap> Bl := Braiding( DirectSum( Ml, Nl ), DirectSum( Ml, Ml ) );
<A morphism in Category of left presentations of Z>
gap> Display( UnderlyingMatrix( Bl ) );
[ [  1,  0,  0,  0 ],
  [  0,  0,  1,  0 ],
  [  0,  1,  0,  0 ],
  [  0,  0,  0,  1 ] ]
gap> IsWellDefined( Bl );
true
gap> Ul := TensorUnit( CapCategory( Ml ) );
<An object in Category of left presentations of Z>
gap> IntHoml := InternalHomOnObjects( DirectSum( Ml, Ul ), Nl );
<An object in Category of left presentations of Z>
gap> Display( UnderlyingMatrix( IntHoml ) );
[ [  -2,  -1 ],
  [   1,  -1 ] ]
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 ) );
[ [  0 ],
  [  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 ],
  [  2 ] ]
gap> IsEqualForMorphisms( LambdaIntroduction( morphism_l1 ), generator_l1 );
false
gap> IsCongruentForMorphisms( LambdaIntroduction( morphism_l1 ), generator_l1 );
true
gap> IsEqualForMorphisms( LambdaIntroduction( morphism_l2 ), generator_l2 );
false
gap> IsCongruentForMorphisms( LambdaIntroduction( morphism_l2 ), generator_l2 );
true
```

```
gap> Mr := AsRightPresentation( HomalgMatrix( [ [ 2 ] ], 1, 1, ZZ ) );
<An object in Category of right presentations of Z>
gap> Nr := AsRightPresentation( HomalgMatrix( [ [ 3 ] ], 1, 1, ZZ ) );
<An object in Category of right presentations of Z>
gap> Tr := TensorProductOnObjects( Mr, Nr );
<An object in Category of right presentations of Z>
gap> Display( UnderlyingMatrix( Tr ) );
[ [  3,  2 ] ]
gap> IsZeroForObjects( Tr );
true
gap> Br := Braiding( DirectSum( Mr, Nr ), DirectSum( Mr, Mr ) );
<A morphism in Category of right presentations of Z>
gap> Display( UnderlyingMatrix( Br ) );
[ [  1,  0,  0,  0 ],
  [  0,  0,  1,  0 ],
  [  0,  1,  0,  0 ],
  [  0,  0,  0,  1 ] ]
gap> IsWellDefined( Br );
true
gap> Ur := TensorUnit( CapCategory( Mr ) );
<An object in Category of right presentations of Z>
gap> IntHomr := InternalHomOnObjects( DirectSum( Mr, Ur ), Nr );
<An object in Category of right presentations of Z>
gap> Display( UnderlyingMatrix( IntHomr ) );
[ [  -2,   1 ],
  [  -1,  -1 ] ]
gap> generator_r1 := StandardGeneratorMorphism( IntHomr, 1 );
<A morphism in Category of right presentations of Z>
gap> morphism_r1 := LambdaElimination( DirectSum( Mr, Ur ), Nr, generator_r1 );
<A morphism in Category of right presentations of Z>
gap> Display( UnderlyingMatrix( morphism_r1 ) );
[ [  0,   2 ] ]
gap> generator_r2 := StandardGeneratorMorphism( 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 ],
  [   2 ] ]
gap> IsEqualForMorphisms( LambdaIntroduction( morphism_r1 ), generator_r1 );
false
gap> IsCongruentForMorphisms( LambdaIntroduction( morphism_r1 ), generator_r1 );
true
gap> IsEqualForMorphisms( LambdaIntroduction( morphism_r2 ), generator_r2 );
false
gap> IsCongruentForMorphisms( LambdaIntroduction( morphism_r2 ), generator_r2 );
true
```

11.6 Opposite category

Example

```
gap> QQ := HomalgFieldOfRationals();;
gap> vec := MatrixCategory( QQ );;
gap> V1 := Opposite( TensorUnit( vec ) );;
gap> V2 := DirectSum( V1, V1 );;
gap> V3 := DirectSum( V1, V2 );;
gap> V4 := DirectSum( V1, V3 );;
gap> V5 := DirectSum( V1, V4 );;
gap> alpha13 := InjectionOfCofactorOfDirectSum( [ V1, V2 ], 1 );;
gap> alpha14 := InjectionOfCofactorOfDirectSum( [ V1, V2, V1 ], 3 );;
gap> alpha15 := InjectionOfCofactorOfDirectSum( [ V2, V1, V2 ], 2 );;
gap> alpha23 := InjectionOfCofactorOfDirectSum( [ V2, V1 ], 1 );;
gap> alpha24 := InjectionOfCofactorOfDirectSum( [ V1, V2, V1 ], 2 );;
gap> alpha25 := InjectionOfCofactorOfDirectSum( [ V2, V2, V1 ], 1 );;
gap> mat := [
>     [ alpha13, alpha14, alpha15 ],
>     [ alpha23, alpha24, alpha25 ]
> ];;
gap> mor := MorphismBetweenDirectSums( mat );;
gap> IsWellDefined( mor );
true
gap> IsWellDefined( Opposite( mor ) );
true
gap> IsOne( UniversalMorphismFromImage( mor, [ CoastrictionToImage( mor ), ImageEmbedding( mor ) ] ) * ImageEmbedding( mor ) );
true
```

11.7 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
```

11.8 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
```

11.9 Kernel

Example

```
gap> vecspaces := CreateCapCategory( "VectorSpaces01" );
VectorSpaces01
gap> ReadPackage( "CAP", "examples/testfiles/VectorSpacesAddKernel01.gi" );
true
gap> V := QVectorSpace( 2 );
<A rational vector space of dimension 2>
gap> W := QVectorSpace( 3 );
<A rational vector space of dimension 3>
gap> alpha := VectorSpaceMorphism( V, [ [ 1, 1, 1 ], [ -1, -1, -1 ] ], W );
```

```
A rational vector space homomorphism with matrix:
[ [   1,   1,   1 ],
  [  -1,  -1,  -1 ] ]

gap> k := KernelObject( alpha );
<A rational vector space of dimension 1>
gap> T := QVectorSpace( 2 );
<A rational vector space of dimension 2>
gap> tau := VectorSpaceMorphism( T, [ [ 2, 2 ], [ 2, 2 ] ], V );
A rational vector space homomorphism with matrix:
[ [  2,  2 ],
  [  2,  2 ] ]

gap> k_lift := KernelLift( alpha, tau );
A rational vector space homomorphism with matrix:
[ [  2 ],
  [  2 ] ]

gap> HasKernelEmbedding( alpha );
false
gap> KernelEmbedding( alpha );
A rational vector space homomorphism with matrix:
[ [  1,  1 ] ]
```

Example

```
gap> vecspaces := CreateCapCategory( "VectorSpaces02" );
VectorSpaces02
gap> ReadPackage( "CAP", "examples/testfiles/VectorSpacesAddKernel02.gi" );
true
gap> V := QVectorSpace( 2 );
<A rational vector space of dimension 2>
gap> W := QVectorSpace( 3 );
<A rational vector space of dimension 3>
gap> alpha := VectorSpaceMorphism( V, [ [ 1, 1, 1 ], [ -1, -1, -1 ] ], W );
A rational vector space homomorphism with matrix:
[ [   1,   1,   1 ],
  [  -1,  -1,  -1 ] ]

gap> k := KernelObject( alpha );
<A rational vector space of dimension 1>
gap> T := QVectorSpace( 2 );
<A rational vector space of dimension 2>
gap> tau := VectorSpaceMorphism( T, [ [ 2, 2 ], [ 2, 2 ] ], V );
A rational vector space homomorphism with matrix:
[ [  2,  2 ],
  [  2,  2 ] ]

gap> k_lift := KernelLift( alpha, tau );
A rational vector space homomorphism with matrix:
[ [  2 ],
  [  2 ] ]

gap> HasKernelEmbedding( alpha );
```

```
false
```

Example

```
gap> vecspaces := CreateCapCategory( "VectorSpaces03" );
VectorSpaces03
gap> ReadPackage( "CAP", "examples/testfiles/VectorSpacesAddKernel03.gi" );
true
gap> V := QVectorSpace( 2 );
<A rational vector space of dimension 2>
gap> W := QVectorSpace( 3 );
<A rational vector space of dimension 3>
gap> alpha := VectorSpaceMorphism( V, [ [ 1, 1, 1 ], [ -1, -1, -1 ] ], W );
A rational vector space homomorphism with matrix:
[ [   1,   1,   1 ],
  [  -1,  -1,  -1 ] ]

gap> k := KernelObject( alpha );
<A rational vector space of dimension 1>
gap> k_emb := KernelEmbedding( alpha );
A rational vector space homomorphism with matrix:
[ [  1,  1 ] ]

gap> IsIdenticalObj( Source( k_emb ), k );
true
gap> V := QVectorSpace( 2 );
<A rational vector space of dimension 2>
gap> W := QVectorSpace( 3 );
<A rational vector space of dimension 3>
gap> beta := VectorSpaceMorphism( V, [ [ 1, 1, 1 ], [ -1, -1, -1 ] ], W );
A rational vector space homomorphism with matrix:
[ [   1,   1,   1 ],
  [  -1,  -1,  -1 ] ]

gap> k_emb := KernelEmbedding( beta );
A rational vector space homomorphism with matrix:
[ [  1,  1 ] ]

gap> IsIdenticalObj( Source( k_emb ), KernelObject( beta ) );
true
```

11.10 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 := ProjectionInFactorOfFiberProduct( [ AtoC, BtoC ], 1 );
A rational vector space homomorphism with matrix:
[ [  1/2 ] ]

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

Index

ActivateDerivationInfo, 112
Add
    for IsCapCategory, IsCapCategoryMorphism, 25
    for IsCapCategory, IsCapCategoryObject, 14
    for IsStringMinHeap, IsString, IsInt, 119
AddAdditionForMorphisms
    for IsCapCategory, IsFunction, 27
AddAdditiveInverseForMorphisms
    for IsCapCategory, IsFunction, 28
AddAstrictionToCoimage
    for IsCapCategory, IsFunction, 105
AddAstrictionToCoimageWithGivenCoimage
    for IsCapCategory, IsFunction, 105
AddCategoricalProperty, 6
AddCoastrictionToImage
    for IsCapCategory, IsFunction, 100
AddCoastrictionToImageWithGivenImageObject
    for IsCapCategory, IsFunction, 100
AddCoequalizer
    for IsCapCategory, IsFunction, 81
AddCoequalizerFunctorialWithGivenCoequalizers
    for IsCapCategory, IsFunction, 83
AddCoimage
    for IsCapCategory, IsFunction, 104
AddCoimageProjection
    for IsCapCategory, IsFunction, 104
AddCoimageProjectionWithGivenCoimage
    for IsCapCategory, IsFunction, 105
AddCokernelColift
    for IsCapCategory, IsFunction, 53
AddCokernelColiftWithGivenCokernelObject
    for IsCapCategory, IsFunction, 53
AddCokernelObject
    for IsCapCategory, IsFunction, 52
AddCokernelObjectFunctorialWithGivenCokernelObjects
    for IsCapCategory, IsFunction, 54
AddCokernelProjection
    for IsCapCategory, IsFunction, 52
AddCokernelProjectionWithGivenCokernelObject
    for IsCapCategory, IsFunction, 53
AddColift
    for IsCapCategory, IsFunction, 34
AddColiftAlongEpimorphism
    for IsCapCategory, IsFunction, 33
AddComponentOfMorphismFromDirectSum
    for IsCapCategory, IsFunction, 67
AddComponentOfMorphismIntoDirectSum
    for IsCapCategory, IsFunction, 67
AddCoproduct
    for IsCapCategory, IsFunction, 72
AddCoproductFunctorialWithGivenCoproducts
    for IsCapCategory, IsFunction, 73
AddDerivation
    for IsDerivedMethodGraph, IsDerivedMethod, 115
    for IsDerivedMethodGraph, IsFunction, IsDenseList, 115
    for IsDerivedMethodGraph, IsFunction, IsDenseList, IsObject, 115
    for IsDerivedMethodGraph, IsFunction, IsFunction, 115
AddDerivationPair
    for IsDerivedMethodGraph, IsFunction, IsFunction, IsDenseList, IsDenseList, 115
    for IsDerivedMethodGraph, IsFunction, IsFunction, IsDenseList, IsDenseList, IsDenseList, 115
    for IsDerivedMethodGraph, IsFunction, IsFunction, IsDenseList, IsFunction, IsFunction, 116

for IsDerivedMethodGraph, IsFunction, IsFunction, IsFunction, IsFunction, 116
AddDerivationPairToCAP, 116
AddDerivationToCAP, 116
AddDirectProduct
for IsCapCategory, IsFunction, 75
AddDirectProductFunctorialWithGivenDirectProducts
for IsCapCategory, IsFunction, 76
AddDirectSum
for IsCapCategory, IsFunction, 69
AddDirectSumCodiagonalDifference
for IsCapCategory, IsFunction, 93
AddDirectSumDiagonalDifference
for IsCapCategory, IsFunction, 86
AddDirectSumFunctorialWithGivenDirectSums
for IsCapCategory, IsFunction, 70
AddDirectSumProjectionInPushout
for IsCapCategory, IsFunction, 94
AddDistinguishedObjectOfHomomorphismStructure
for IsCapCategory, IsFunction, 37
AddEmbeddingOfEqualizer
for IsCapCategory, IsFunction, 78
AddEmbeddingOfEqualizerWithGivenEqualizer
for IsCapCategory, IsFunction, 78
AddEpimorphismFromSomeProjectiveObject
for IsCapCategory, IsFunction, 16
AddEpimorphismFromSomeProjectiveObjectWithGivenSomeProjectiveObject
for IsCapCategory, IsFunction, 16
AddEqualizer
for IsCapCategory, IsFunction, 78
AddEqualizerFunctorialWithGivenEqualizers
for IsCapCategory, IsFunction, 79
AddFiberProduct
for IsCapCategory, IsFunction, 88
AddFiberProductEmbeddingInDirectSum
for IsCapCategory, IsFunction, 87
AddFiberProductFunctorialWithGivenFiberProducts
for IsCapCategory, IsFunction, 90
AddHomomorphismStructureOnMorphismsWithGivenObjects
for IsCapCategory, IsFunction, 37
AddHomomorphismStructureOnObjects
for IsCapCategory, IsFunction, 36
AddHorizontalPostCompose
for IsCapCategory, IsFunction, 40
AddHorizontalPreCompose
for IsCapCategory, IsFunction, 40
AddIdentityMorphism
for IsCapCategory, IsFunction, 30
AddIdentityTwoCell
for IsCapCategory, IsFunction, 39
AddImageEmbedding
for IsCapCategory, IsFunction, 100
AddImageEmbeddingWithGivenImageObject
for IsCapCategory, IsFunction, 100
AddImageObject
for IsCapCategory, IsFunction, 99
AddInitialObject
for IsCapCategory, IsFunction, 60
AddInitialObjectFunctorial
for IsCapCategory, IsFunction, 61
AddInjectionOfCofactorOfCoproduct
for IsCapCategory, IsFunction, 72
AddInjectionOfCofactorOfCoproductWithGivenCoproduct
for IsCapCategory, IsFunction, 72
AddInjectionOfCofactorOfDirectSum
for IsCapCategory, IsFunction, 67
AddInjectionOfCofactorOfDirectSumWithGivenDirectSum
for IsCapCategory, IsFunction, 67
AddInjectionOfCofactorOfPushout
for IsCapCategory, IsFunction, 96
AddInjectionOfCofactorOfPushoutWithGivenPushout
for IsCapCategory, IsFunction, 96
AddInjectiveColift
for IsCapCategory, IsFunction, 18
AddInterpretMorphismAsMorphismFromDinstinguishedObjectToHomomorphismStructure
for IsCapCategory, IsFunction, 37
AddInterpretMorphismFromDinstinguishedObjectToHomomorphismStructureAsMorphism

for IsCapCategory, IsFunction, 38
AddInverse
for IsCapCategory, IsFunction, 35
AddInverseMorphismFromCoimageToImage-
WithGivenObjects
for IsCapCategory, IsFunction, 102
AddIsAutomorphism
for IsCapCategory, IsFunction, 25
AddIsCodominating
for IsCapCategory, IsFunction, 30
AddIsColiftable
for IsCapCategory, IsFunction, 35
AddIsColiftableAlongEpimorphism
for IsCapCategory, IsFunction, 33
AddIsCongruentForMorphisms
for IsCapCategory, IsFunction, 26
AddIsDominating
for IsCapCategory, IsFunction, 29
AddIsEndomorphism
for IsCapCategory, IsFunction, 25
AddIsEpimorphism
for IsCapCategory, IsFunction, 20
AddIsEqualAsFactorobjects
for IsCapCategory, IsFunction, 29
AddIsEqualAsSubobjects
for IsCapCategory, IsFunction, 29
AddIsEqualForCacheForMorphisms
for IsCapCategory, IsFunction, 35
AddIsEqualForCacheForObjects
for IsCapCategory, IsFunction, 14
AddIsEqualForMorphisms
for IsCapCategory, IsFunction, 26
AddIsEqualForMorphismsOnMor
for IsCapCategory, IsFunction, 26
AddIsEqualForObjects
for IsCapCategory, IsFunction, 11
AddIsIdempotent
for IsCapCategory, IsFunction, 20
AddIsIdenticalToIdentityMorphism
for IsCapCategory, IsFunction, 24
AddIsIdenticalToZeroMorphism
for IsCapCategory, IsFunction, 24
AddIsInitial
for IsCapCategory, IsFunction, 12
AddIsInjective
for IsCapCategory, IsFunction, 12
AddIsIsomorphism
for IsCapCategory, IsFunction, 20
AddIsLiftable
for IsCapCategory, IsFunction, 34
AddIsLiftableAlongMonomorphism
for IsCapCategory, IsFunction, 33
AddIsMonomorphism
for IsCapCategory, IsFunction, 19
AddIsomorphismFromCoequalizerOf-
CoproductDiagramToPushout
for IsCapCategory, IsFunction, 93
AddIsomorphismFromCoimageToCokernelOf-
Kernel
for IsCapCategory, IsFunction, 102
AddIsomorphismFromCokernelOfDiagonal-
DifferenceToPushout
for IsCapCategory, IsFunction, 92
AddIsomorphismFromCokernelOfKernelTo-
Coimage
for IsCapCategory, IsFunction, 103
AddIsomorphismFromCoproductToDirectSum
for IsCapCategory, IsFunction, 69
AddIsomorphismFromDirectProductTo-
DirectSum
for IsCapCategory, IsFunction, 69
AddIsomorphismFromDirectSumToCoproduct
for IsCapCategory, IsFunction, 69
AddIsomorphismFromDirectSumToDirect-
Product
for IsCapCategory, IsFunction, 68
AddIsomorphismFromEqualizerOfDirect-
ProductDiagramToFiberProduct
for IsCapCategory, IsFunction, 86
AddIsomorphismFromFiberProductTo-
EqualizerOfDirectProductDiagram
for IsCapCategory, IsFunction, 85
AddIsomorphismFromFiberProductTo-
KernelOfDiagonalDifference
for IsCapCategory, IsFunction, 84
AddIsomorphismFromImageObjectToKernel-
OfCokernel
for IsCapCategory, IsFunction, 98
AddIsomorphismFromInitialObjectToZero-
Object
for IsCapCategory, IsFunction, 57
AddIsomorphismFromKernelOfCokernelTo-
ImageObject
for IsCapCategory, IsFunction, 98

AddIsomorphismFromKernelOfDiagonal-
DifferenceToFiberProduct
for IsCapCategory, IsFunction, 84
AddIsomorphismFromPushoutTo-
CoequalizerOfCoproductDiagram
for IsCapCategory, IsFunction, 92
AddIsomorphismFromPushoutToCokernelOf-
DiagonalDifference
for IsCapCategory, IsFunction, 91
AddIsomorphismFromTerminalObjectTo-
ZeroObject
for IsCapCategory, IsFunction, 57
AddIsomorphismFromZeroObjectToInitial-
Object
for IsCapCategory, IsFunction, 57
AddIsomorphismFromZeroObjectTo-
TerminalObject
for IsCapCategory, IsFunction, 57
AddIsOne
for IsCapCategory, IsFunction, 20
AddIsProjective
for IsCapCategory, IsFunction, 12
AddIsSplitEpimorphism
for IsCapCategory, IsFunction, 20
AddIsSplitMonomorphism
for IsCapCategory, IsFunction, 20
AddIsTerminal
for IsCapCategory, IsFunction, 12
AddIsWellDefinedForMorphisms
for IsCapCategory, IsFunction, 31
AddIsWellDefinedForObjects
for IsCapCategory, IsFunction, 15
AddIsWellDefinedForTwoCells
for IsCapCategory, IsFunction, 41
AddIsZeroForMorphisms
for IsCapCategory, IsFunction, 27
AddIsZeroForObjects
for IsCapCategory, IsFunction, 12
AdditionForMorphisms
for IsCapCategoryMorphism, IsCapCatego-
ryMorphism, 27
AdditiveInverseForMorphisms
for IsCapCategoryMorphism, 27
AddKernelEmbedding
for IsCapCategory, IsFunction, 49
AddKernelEmbeddingWithGivenKernel-
Object
for IsCapCategory, IsFunction, 49
AddKernelLift
for IsCapCategory, IsFunction, 50
AddKernelLiftWithGivenKernelObject
for IsCapCategory, IsFunction, 50
AddKernelObject
for IsCapCategory, IsFunction, 49
AddKernelObjectFunctorialWithGiven-
KernelObjects
for IsCapCategory, IsFunction, 51
AddLift
for IsCapCategory, IsFunction, 34
AddLiftAlongMonomorphism
for IsCapCategory, IsFunction, 32
AddMonomorphismIntoSomeInjectiveObject
for IsCapCategory, IsFunction, 17
AddMonomorphismIntoSomeInjective-
ObjectWithGivenSomeInjective-
Object
for IsCapCategory, IsFunction, 17
AddMorphism
for IsCapCategory, IsAttributeStoringRep,
25
AddMorphismBetweenDirectSums
for IsCapCategory, IsFunction, 66
AddMorphismFromCoimageToImageWith-
GivenObjects
for IsCapCategory, IsFunction, 101
AddMorphismFunction
for IsCapFunctor, IsFunction, 44
AddMorphismRepresentation
for IsCapCategory, IsObject, 25
AddMultiplyWithElementOfCommutative-
RingForMorphisms
for IsCapCategory, IsFunction, 28
AddNaturalTransformationFunction
for IsCapNaturalTransformation, IsFunction,
46
AddObject
for IsCapCategory, IsAttributeStoringRep,
14
AddObjectFunction
for IsCapFunctor, IsFunction, 44
AddObjectRepresentation
for IsCapCategory, IsObject, 14
AddOperationsToDerivationGraph

for IsDerivedMethodGraph, IsDenseList, 115
AddPostCompose
for IsCapCategory, IsFunction, 31
AddPreCompose
for IsCapCategory, IsFunction, 31
AddPrimitiveOperation
for IsOperationWeightList, IsString, IsInt, 118
AddProjectionInFactorOfDirectProduct
for IsCapCategory, IsFunction, 75
AddProjectionInFactorOfDirectProductWithGivenDirectProduct
for IsCapCategory, IsFunction, 75
AddProjectionInFactorOfDirectSum
for IsCapCategory, IsFunction, 67
AddProjectionInFactorOfDirectSumWithGivenDirectSum
for IsCapCategory, IsFunction, 67
AddProjectionInFactorOfFiberProduct
for IsCapCategory, IsFunction, 89
AddProjectionInFactorOfFiberProductWithGivenFiberProduct
for IsCapCategory, IsFunction, 89
AddProjectionOntoCoequalizer
for IsCapCategory, IsFunction, 82
AddProjectionOntoCoequalizerWithGivenCoequalizer
for IsCapCategory, IsFunction, 82
AddProjectiveLift
for IsCapCategory, IsFunction, 16
AddPushout
for IsCapCategory, IsFunction, 96
AddPushoutFunctorialWithGivenPushouts
for IsCapCategory, IsFunction, 97
AddRandomMorphismByInteger
for IsCapCategory, IsFunction, 23
AddRandomMorphismByList
for IsCapCategory, IsFunction, 23
AddRandomMorphismWithFixedRangeByInteger
for IsCapCategory, IsFunction, 22
AddRandomMorphismWithFixedRangeByList
for IsCapCategory, IsFunction, 22
AddRandomMorphismWithFixedSourceAndRangeByInteger
for IsCapCategory, IsFunction, 22
AddRandomMorphismWithFixedSourceAndRangeByList
for IsCapCategory, IsFunction, 23
AddRandomMorphismWithFixedSourceByInteger
for IsCapCategory, IsFunction, 21
AddRandomMorphismWithFixedSourceByList
for IsCapCategory, IsFunction, 21
AddRandomObjectByInteger
for IsCapCategory, IsFunction, 13
AddRandomObjectByList
for IsCapCategory, IsFunction, 13
AddSomeInjectiveObject
for IsCapCategory, IsFunction, 17
AddSomeProjectiveObject
for IsCapCategory, IsFunction, 16
AddSubtractionForMorphisms
for IsCapCategory, IsFunction, 27
AddTerminalObject
for IsCapCategory, IsFunction, 59
AddTerminalObjectFunctorial
for IsCapCategory, IsFunction, 59
AddUniversalMorphismFromCoequalizer
for IsCapCategory, IsFunction, 82
AddUniversalMorphismFromCoequalizerWithGivenCoequalizer
for IsCapCategory, IsFunction, 82
AddUniversalMorphismFromCoproduct
for IsCapCategory, IsFunction, 72
AddUniversalMorphismFromCoproductWithGivenCoproduct
for IsCapCategory, IsFunction, 73
AddUniversalMorphismFromDirectSum
for IsCapCategory, IsFunction, 68
AddUniversalMorphismFromDirectSumWithGivenDirectSum
for IsCapCategory, IsFunction, 68
AddUniversalMorphismFromImage
for IsCapCategory, IsFunction, 100
AddUniversalMorphismFromImageWithGivenImageObject
for IsCapCategory, IsFunction, 100
AddUniversalMorphismFromInitialObject
for IsCapCategory, IsFunction, 61
AddUniversalMorphismFromInitialObjectWithGivenInitialObject
for IsCapCategory, IsFunction, 61

AddUniversalMorphismFromPushout
for IsCapCategory, IsFunction, 96
AddUniversalMorphismFromPushoutWithGivenPushout
for IsCapCategory, IsFunction, 96
AddUniversalMorphismFromZeroObject
for IsCapCategory, IsFunction, 56
AddUniversalMorphismFromZeroObjectWithGivenZeroObject
for IsCapCategory, IsFunction, 57
AddUniversalMorphismIntoCoimage
for IsCapCategory, IsFunction, 105
AddUniversalMorphismIntoCoimageWithGivenCoimage
for IsCapCategory, IsFunction, 105
AddUniversalMorphismIntoDirectProduct
for IsCapCategory, IsFunction, 75
AddUniversalMorphismIntoDirectProductWithGivenDirectProduct
for IsCapCategory, IsFunction, 76
AddUniversalMorphismIntoDirectSum
for IsCapCategory, IsFunction, 68
AddUniversalMorphismIntoDirectSumWithGivenDirectSum
for IsCapCategory, IsFunction, 68
AddUniversalMorphismIntoEqualizer
for IsCapCategory, IsFunction, 79
AddUniversalMorphismIntoEqualizerWithGivenEqualizer
for IsCapCategory, IsFunction, 79
AddUniversalMorphismIntoFiberProduct
for IsCapCategory, IsFunction, 89
AddUniversalMorphismIntoFiberProductWithGivenFiberProduct
for IsCapCategory, IsFunction, 89
AddUniversalMorphismIntoTerminalObject
for IsCapCategory, IsFunction, 59
AddUniversalMorphismIntoTerminalObjectWithGivenTerminalObject
for IsCapCategory, IsFunction, 59
AddUniversalMorphismIntoZeroObject
for IsCapCategory, IsFunction, 56
AddUniversalMorphismIntoZeroObjectWithGivenZeroObject
for IsCapCategory, IsFunction, 56
AddVerticalPostCompose
for IsCapCategory, IsFunction, 40
AddVerticalPreCompose
for IsCapCategory, IsFunction, 40
AddWithGivenDerivationPairToCAP, 116
AddZeroMorphism
for IsCapCategory, IsFunction, 28
AddZeroObject
for IsCapCategory, IsFunction, 56
AddZeroObjectFunctorial
for IsCapCategory, IsFunction, 58
ApplyFunctor, 45
ApplyNaturalTransformation, 47
AsCapCategory
for IsCapCategoryAsCatObject, 43
AsCatObject
for IsCapCategory, 43
AstrictionToCoimage
for IsCapCategoryMorphism, 104
for IsCapCategoryObject, 103
AstrictionToCoimageWithGivenCoimage
for IsCapCategoryMorphism, IsCapCategoryObject, 104

BrowseCachingStatistic, 123

CachingStatistic, 123
CanCompute
for IsCapCategory, IsString, 8
CAPAddPrepareFunction, 111
CapCat, 42
CapCategory
for IsCapCategoryMorphism, 19
for IsCapCategoryObject, 11
CapCategorySwitchLogicOff, 7
CapCategorySwitchLogicOn, 7
CapFunctor
for IsString, IsCapCategory, IsCapCategory, 43
for IsString, IsCapCategory, IsCapCategoryAsCatObject, 43
for IsString, IsCapCategoryAsCatObject, IsCapCategory, 43
for IsString, IsCapCategoryAsCatObject, IsCapCategoryAsCatObject, 43
for IsString, IsList, IsCapCategory, 43
for IsString, IsList, IsCapCategoryAsCatObject, 43

CAP_INTERNAL_FIND_APPEARANCE_OF_SYMBOL_IN_FUNCTION, 122
CAP_INTERNAL_MERGE_FILTER_LISTS, 122
CAP_INTERNAL_MERGE_PRECONDITIONS_LIST, 123
CAP_INTERNAL_REPLACE_STRINGS_WITH_FILTERS, 122
CAP_INTERNAL_RETURN_OPTION_OR_DEFAULT, 122
CAPOperationPrepareFunction, 111
CategoryFilter
for IsCapCategory, 6
for IsDerivedMethod, 113
CategoryOfOperationWeightList
for IsOperationWeightList, 117
CellFilter
for IsCapCategory, 7
CheckConstructivenessOfCategory
for IsCapCategory, IsString, 8
CoastrictionToImage
for IsCapCategoryMorphism, 99
CoastrictionToImageWithGivenImageObject
for IsCapCategoryMorphism, IsCapCategoryObject, 99
Coequalizer, 80
CoequalizerFunctorial
for IsList, IsCapCategoryMorphism, IsList, 82
CoequalizerFunctorialWithGivenCoequalizers
for IsCapCategoryObject, IsList, IsCapCategoryMorphism, IsList, IsCapCategoryObject, 82
CoequalizerOp
for IsList, IsCapCategoryMorphism, 80
Coimage
for IsCapCategoryMorphism, 103
CoimageProjection
for IsCapCategoryMorphism, 103
for IsCapCategoryObject, 103
CoimageProjectionWithGivenCoimage
for IsCapCategoryMorphism, IsCapCategoryObject, 103
CokernelColift
for IsCapCategoryMorphism, IsCapCategoryMorphism, 52
CokernelColiftWithGivenCokernelObject
for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject, 52
CokernelObject
for IsCapCategoryMorphism, 51
CokernelObjectFunctorial
for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, 53
for IsList, 53
CokernelObjectFunctorialWithGivenCokernelObjects
for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryMorphism,IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject, 54
for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryMorphism,IsCapCategoryMorphism, IsCapCategoryObject, 53
CokernelProjection
for IsCapCategoryMorphism, 52
CokernelProjectionWithGivenCokernelObject
for IsCapCategoryMorphism, IsCapCategoryObject, 52
Colift
for IsCapCategoryMorphism, IsCapCategoryMorphism, 34
ColiftAlongEpimorphism
for IsCapCategoryMorphism, IsCapCategoryMorphism, 33
CommutativeRingOfLinearCategory
for IsCapCategory, 7
ComponentOfMorphismFromDirectSum
for IsCapCategoryMorphism, IsList, IsInt, 66
ComponentOfMorphismIntoDirectSum
for IsCapCategoryMorphism, IsList, IsInt, 66
Contains
for IsStringMinHeap, IsString, 120
Coproduct
for IsCapCategoryObject, IsCapCategoryObject, 70
for IsCapCategoryObject, IsCapCategory-

Object, IsCapCategoryObject, 71
for IsList, 70
CoproductFunctorial
for IsList, 73
CoproductFunctorialWithGivenCoproducts
for IsCapCategoryObject, IsList, IsCapCategoryObject, 73
CoproductOp
for IsList, IsCapCategoryObject, 71
CreateCapCategory, 6
for IsString, 6
CurrentOperationWeight
for IsOperationWeightList, IsString, 117

DeactivateCachingOfCategory, 8
DeactivateDefaultCaching, 9
DeactivateDerivationInfo, 112
DeclareAttributeWithToDoForIsWellDefined, 122
DeclareFamilyProperty, 122
DecreaseKey
for IsStringMinHeap, IsString, IsInt, 119
DerivationFunctionsWithExtraFilters
for IsDerivedMethod, 113
DerivationGraph
for IsOperationWeightList, 117
DerivationInfo, 112
DerivationName
for IsDerivedMethod, 113
DerivationOfOperation
for IsOperationWeightList, IsString, 118
DerivationResultWeight
for IsDerivedMethod, IsDenseList, 114
DerivationsOfOperation
for IsDerivedMethodGraph, IsString, 116
DerivationsUsingOperation
for IsDerivedMethodGraph, IsString, 116
DerivationWeight
for IsDerivedMethod, 113
DirectProductFunctorial
for IsList, 76
DirectProductFunctorialWithGivenDirectProducts
for IsCapCategoryObject, IsList, IsCapCategoryObject, 76
DirectProductOp
for IsList, IsCapCategoryObject, 74
DirectSumCodiagonalDifference
for IsList, 93
DirectSumCodiagonalDifferenceOp
for IsList, IsCapCategoryMorphism, 93
DirectSumDiagonalDifference
for IsList, 86
DirectSumDiagonalDifferenceOp
for IsList, IsCapCategoryMorphism, 86
DirectSumFunctorial
for IsList, 69
DirectSumFunctorialWithGivenDirectSums
for IsCapCategoryObject, IsList, IsCapCategoryObject, 69
DirectSumOp
for IsList, IsCapCategoryObject, 62
DirectSumProjectionInPushout
for IsList, 94
DirectSumProjectionInPushoutOp
for IsList, IsCapCategoryMorphism, 94
DisableAddForCategoricalOperations, 9
DisableInputSanityChecks, 9
DisableOutputSanityChecks, 9
DisableSanityChecks, 9
DistinguishedObjectOfHomomorphismStructure
for IsCapCategory, 37

EmbeddingOfEqualizer
for IsList, 77
EmbeddingOfEqualizerOp
for IsList, IsCapCategoryMorphism, 77
EmbeddingOfEqualizerWithGivenEqualizer
for IsList, IsCapCategoryObject, 78
EnableAddForCategoricalOperations, 9
EnableFullInputSanityChecks, 9
EnableFullOutputSanityChecks, 9
EnableFullSanityChecks, 9
EnablePartialInputSanityChecks, 9
EnablePartialOutputSanityChecks, 9
EnablePartialSanityChecks, 9
EpimorphismFromSomeProjectiveObject
for IsCapCategoryObject, 15
EpimorphismFromSomeProjectiveObjectWithGivenSomeProjectiveObject
for IsCapCategoryObject, IsCapCategoryObject, 15
Equalizer, 77

EqualizerFunctorial
for IsList, IsCapCategoryMorphism, IsList, 79
EqualizerFunctorialWithGivenEqualizers
for IsCapCategoryObject, IsList, IsCapCategoryMorphism, IsList, IsCapCategoryObject, 79
EqualizerOp
for IsList, IsCapCategoryMorphism, 77
ExtractMin
for IsStringMinHeap, 119

FiberProduct, 87
FiberProductEmbeddingInDirectSum
for IsList, 86
FiberProductEmbeddingInDirectSumOp
for IsList, IsCapCategoryMorphism, 86
FiberProductFunctorial
for IsList, IsList, IsList, 89
FiberProductFunctorialWithGivenFiberProducts
for IsCapCategoryObject, IsList, IsList, IsList, IsCapCategoryObject, 90
FiberProductOp
for IsList, IsCapCategoryMorphism, 87
FunctorCanonicalizeZeroMorphisms
for IsCapCategory, 46
FunctorCanonicalizeZeroObjects
for IsCapCategory, 45
FunctorMorphismOperation
for IsCapFunctor, 45
FunctorObjectOperation
for IsCapFunctor, 44

Heapify
for IsStringMinHeap, IsPosInt, 120
HeapSize
for IsStringMinHeap, 120
HomomorphismStructureOnMorphisms
for IsCapCategoryMorphism, IsCapCategoryMorphism, 36
HomomorphismStructureOnMorphismsWithGivenObjects
for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject, 36
HomomorphismStructureOnObjects
for IsCapCategoryObject, IsCapCategoryObject, 36
HorizontalPostCompose
for IsCapCategoryTwoCell, IsCapCategoryTwoCell, 40
HorizontalPreCompose
for IsCapCategoryTwoCell, IsCapCategoryTwoCell, 39
HorizontalPreComposeFunctorWithNaturalTransformation
for IsCapFunctor, IsCapNaturalTransformation, 47
HorizontalPreComposeNaturalTransformationWithFunctor
for IsCapNaturalTransformation, IsCapFunctor, 47

IdentityFunctor
for IsCapCategory, 45
IdentityMorphism
for IsCapCategoryObject, 30
IdentityTwoCell
for IsCapCategoryMorphism, 39
ImageEmbedding
for IsCapCategoryMorphism, 98
ImageEmbeddingWithGivenImageObject
for IsCapCategoryMorphism, IsCapCategoryObject, 99
ImageObject
for IsCapCategoryMorphism, 98
InitialObject
for IsCapCategory, 60
for IsCapCategoryCell, 60
InitialObjectFunctorial
for IsCapCategory, 61
InjectionOfCofactorOfCoproduct
for IsList, IsInt, 71
InjectionOfCofactorOfCoproductOp
for IsList, IsInt, IsCapCategoryObject, 71
InjectionOfCofactorOfCoproductWithGivenCoproduct
for IsList, IsInt, IsCapCategoryObject, 71
InjectionOfCofactorOfDirectSum
for IsList, IsInt, 63
InjectionOfCofactorOfDirectSumOp
for IsList, IsInt, IsCapCategoryObject, 63

InjectionOfCofactorOfDirectSumWithGivenDirectSum
for IsList, IsInt, IsCapCategoryObject, 63
InjectionOfCofactorOfPushout
for IsList, IsInt, 94
InjectionOfCofactorOfPushoutOp
for IsList, IsInt, IsCapCategoryMorphism, 95
InjectionOfCofactorOfPushoutWithGivenPushout
for IsList, IsInt, IsCapCategoryObject, 95
InjectiveColift
for IsCapCategoryMorphism, IsCapCategoryMorphism, 17
InputSignature
for IsCapFunctor, 45
InstallDerivationForCategory
for IsDerivedMethod, IsPosInt, IsCapCategory, 114
InstallDerivationsUsingOperation
for IsOperationWeightList, IsString, 118
InstallFunctor
for IsCapFunctor, IsString, 45
InstallMethodWithToDoForIsWellDefined, 121
InstallNaturalTransformation
for IsCapNaturalTransformation, IsString, 47
InstallSetWithToDoForIsWellDefined
for IsObject, IsString, IsList, 122
InterpretMorphismAsMorphismFromDinstinguishedObjectToHomomorphismStructure
for IsCapCategoryMorphism, 37
InterpretMorphismFromDinstinguishedObjectToHomomorphismStructureAsMorphism
for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryMorphism, 38
InverseMorphismFromCoimageToImage
for IsCapCategoryMorphism, 102
InverseMorphismFromCoimageToImageWithGivenObjects
for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryObject, 102
IsApplicableToCategory
for IsDerivedMethod, IsCapCategory, 113
IsCapCategory
for IsObject, 5
IsCapCategoryAsCatObject
for IsCapCategoryObject, 42
IsCapCategoryCell
for IsAttributeStoringRep, 5
IsCapCategoryMorphism
for IsCapCategoryCell, 6
IsCapCategoryObject
for IsCapCategoryCell, 5
IsCapCategoryTwoCell
for IsCapCategoryCell, 6
IsCapFunctor
for IsCapCategoryMorphism, 42
IsCapNaturalTransformation
for IsCapCategoryTwoCell, 42
IsCodominating
for IsCapCategoryMorphism, IsCapCategoryMorphism, 30
IsColiftable
for IsCapCategoryMorphism, IsCapCategoryMorphism, 35
IsColiftableAlongEpimorphism
for IsCapCategoryMorphism, IsCapCategoryMorphism, 33
IsCongruentForMorphisms
for IsCapCategoryMorphism, IsCapCategoryMorphism, 26
IsDerivedMethod
for IsObject, 112
IsDerivedMethodGraph
for IsObject, 114
IsDominating
for IsCapCategoryMorphism, IsCapCategoryMorphism, 29
IsEmptyHeap
for IsStringMinHeap, 120
IsEqualAsFactorobjects
for IsCapCategoryMorphism, IsCapCategoryMorphism, 29
IsEqualAsSubobjects
for IsCapCategoryMorphism, IsCapCategoryMorphism, 28
IsEqualForCacheForMorphisms
for IsCapCategoryMorphism, IsCapCategoryMorphism, 35
IsEqualForCacheForObjects

for IsCapCategoryObject, IsCapCategoryObject, 13
IsEqualForMorphisms
for IsCapCategoryMorphism, IsCapCategoryMorphism, 26
IsEqualForMorphismsOnMor
for IsCapCategoryMorphism, IsCapCategoryMorphism, 26
IsEqualForObjects
for IsCapCategoryObject, IsCapCategoryObject, 11
IsIdenticalToIdentityMorphism
for IsCapCategoryMorphism, 24
IsIdenticalToZeroMorphism
for IsCapCategoryMorphism, 24
IsLiftable
for IsCapCategoryMorphism, IsCapCategoryMorphism, 34
IsLiftableAlongMonomorphism
for IsCapCategoryMorphism, IsCapCategoryMorphism, 33
IsomorphismFromCoequalizerOfCoproductDiagramToPushout
for IsList, 93
IsomorphismFromCoequalizerOfCoproductDiagramToPushoutOp
for IsList, IsCapCategoryMorphism, 93
IsomorphismFromCoimageToCokernelOfKernel
for IsCapCategoryMorphism, 102
IsomorphismFromCokernelOfDiagonalDifferenceToPushout
for IsList, 91
IsomorphismFromCokernelOfDiagonalDifferenceToPushoutOp
for IsList, IsCapCategoryMorphism, 92
IsomorphismFromCokernelOfKernelToCoimage
for IsCapCategoryMorphism, 103
IsomorphismFromCoproductToDirectSum
for IsList, 65
IsomorphismFromCoproductToDirectSumOp
for IsList, IsCapCategoryObject, 66
IsomorphismFromDirectProductToDirectSum
for IsList, 65
IsomorphismFromDirectProductToDirectSumOp
for IsList, IsCapCategoryObject, 65
IsomorphismFromDirectSumToCoproduct
for IsList, 65
IsomorphismFromDirectSumToCoproductOp
for IsList, IsCapCategoryObject, 65
IsomorphismFromDirectSumToDirectProduct
for IsList, 64
IsomorphismFromDirectSumToDirectProductOp
for IsList, IsCapCategoryObject, 65
IsomorphismFromEqualizerOfDirectProductDiagramToFiberProduct
for IsList, 85
IsomorphismFromEqualizerOfDirectProductDiagramToFiberProductOp
for IsList, IsCapCategoryMorphism, 85
IsomorphismFromFiberProductToEqualizerOfDirectProductDiagram
for IsList, 85
IsomorphismFromFiberProductToEqualizerOfDirectProductDiagramOp
for IsList, IsCapCategoryMorphism, 85
IsomorphismFromFiberProductToKernelOfDiagonalDifference
for IsList, 84
IsomorphismFromFiberProductToKernelOfDiagonalDifferenceOp
for IsList, IsCapCategoryMorphism, 84
IsomorphismFromImageObjectToKernelOfCokernel
for IsCapCategoryMorphism, 98
IsomorphismFromInitialObjectToZeroObject
for IsCapCategory, 56
IsomorphismFromKernelOfCokernelToImageObject
for IsCapCategoryMorphism, 98
IsomorphismFromKernelOfDiagonalDifferenceToFiberProduct
for IsList, 84
IsomorphismFromKernelOfDiagonalDifferenceToFiberProductOp
for IsList, IsCapCategoryMorphism, 84

IsomorphismFromPushoutToCoequalizerOfCoproductDiagram
for IsList, 92
IsomorphismFromPushoutToCoequalizerOfCoproductDiagramOp
for IsList, IsCapCategoryMorphism, 92
IsomorphismFromPushoutToCokernelOfDiagonalDifference
for IsList, 91
IsomorphismFromPushoutToCokernelOfDiagonalDifferenceOp
for IsList, IsCapCategoryMorphism, 91
IsomorphismFromTerminalObjectToZeroObject
for IsCapCategory, 56
IsomorphismFromZeroObjectToInitialObject
for IsCapCategory, 55
IsomorphismFromZeroObjectToTerminalObject
for IsCapCategory, 56
IsOperationWeightList
for IsObject, 117
IsStringMinHeap
for IsObject, 119
IsWellDefined
for IsCapCategoryCell, 8
IsWellDefinedForMorphisms
for IsCapCategoryMorphism, 31
IsWellDefinedForObjects
for IsCapCategoryObject, 14
IsWellDefinedForTwoCells
for IsCapCategoryTwoCell, 41
IsZeroForMorphisms
for IsCapCategoryMorphism, 27
IsZeroForObjects
for IsCapCategoryObject, 12

KernelEmbedding
for IsCapCategoryMorphism, 48
KernelEmbeddingWithGivenKernelObject
for IsCapCategoryMorphism, IsCapCategoryObject, 49
KernelLift
for IsCapCategoryMorphism, IsCapCategoryMorphism, 49
KernelLiftWithGivenKernelObject
for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject, 49
KernelObject
for IsCapCategoryMorphism, 48
KernelObjectFunctorial
for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, 50
for IsList, 50
KernelObjectFunctorialWithGivenKernelObjects
for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryMorphism,IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject, 50
for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryMorphism,IsCapCategoryMorphism, IsCapCategoryObject, 50

Lift
for IsCapCategoryMorphism, IsCapCategoryMorphism, 34
LiftAlongMonomorphism
for IsCapCategoryMorphism, IsCapCategoryMorphism, 32
ListCAPPrepareFunctions, 111

MakeDerivation
for IsString, IsFunction, IsDenseList,IsPosInt, IsDenseList, IsFunction, 112
MakeDerivationGraph
for IsDenseList, 115
MakeOperationWeightList
for IsCapCategory, IsDerivedMethodGraph, 117
MonomorphismIntoSomeInjectiveObject
for IsCapCategoryObject, 17
MonomorphismIntoSomeInjectiveObjectWithGivenSomeInjectiveObject
for IsCapCategoryObject, IsCapCategoryObject, 17
MorphismBetweenDirectSums
for IsCapCategoryObject, IsList, IsCapCategoryObject, 66

for IsList, 66
MorphismBetweenDirectSumsOp
for IsList, IsInt, IsInt, IsCapCategoryMorphism, 66
MorphismCache
for IsCapFunctor, 121
MorphismFilter
for IsCapCategory, 7
MorphismFromCoimageToImage
for IsCapCategoryMorphism, 101
MorphismFromCoimageToImageWithGivenObjects
for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryObject, 101
MorphismFromZeroObject
for IsCapCategoryObject, 55
MorphismIntoZeroObject
for IsCapCategoryObject, 55
MultiplyWithElementOfCommutativeRingForMorphisms
for IsRingElement, IsCapCategoryMorphism, 28

Name
for IsCapNaturalTransformation, 46
NaturalIsomorophismFromIdentityToCanonicalizeZeroMorphisms
for IsCapCategory, 46
NaturalIsomorophismFromIdentityToCanonicalizeZeroObjects
for IsCapCategory, 46
NaturalTransformation
for IsCapFunctor, IsCapFunctor, 46

ObjectCache
for IsCapFunctor, 121
ObjectFilter
for IsCapCategory, 7
ObjectifyMorphismForCAPWithAttributes, 25
ObjectifyObjectForCAPWithAttributes, 14
Operations
for IsDerivedMethodGraph, 116
OperationWeightUsingDerivation
for IsOperationWeightList, IsDerivedMethod, 117

PostCompose
for IsCapCategoryMorphism, IsCapCategoryMorphism, 31
for IsList, 31
PreCompose
for IsCapCategoryMorphism, IsCapCategoryMorphism, 30
for IsList, 31
PrintDerivationTree
for IsOperationWeightList, IsString, 118
PrintTree
for IsObject, IsFunction, IsFunction, 119
PrintTreeRec
for IsObject, IsFunction, IsFunction, IsInt, 119
ProjectionInFactorOfDirectProduct
for IsList, IsInt, 74
ProjectionInFactorOfDirectProductOp
for IsList, IsInt, IsCapCategoryObject, 74
ProjectionInFactorOfDirectProductWithGivenDirectProduct
for IsList, IsInt, IsCapCategoryObject, 74
ProjectionInFactorOfDirectSum
for IsList, IsInt, 62
ProjectionInFactorOfDirectSumOp
for IsList, IsInt, IsCapCategoryObject, 62
ProjectionInFactorOfDirectSumWithGivenDirectSum
for IsList, IsInt, IsCapCategoryObject, 62
ProjectionInFactorOfFiberProduct
for IsList, IsInt, 87
ProjectionInFactorOfFiberProductOp
for IsList, IsInt, IsCapCategoryMorphism, 87
ProjectionInFactorOfFiberProductWithGivenFiberProduct
for IsList, IsInt, IsCapCategoryObject, 88
ProjectionOntoCoequalizer
for IsList, 81
ProjectionOntoCoequalizerOp
for IsList, IsCapCategoryMorphism, 81
ProjectionOntoCoequalizerWithGivenCoequalizer
for IsList, IsCapCategoryObject, 81
ProjectiveLift
for IsCapCategoryMorphism, IsCapCategoryMorphism, 15
Pushout

for IsCapCategoryMorphism, IsCapCategoryMorphism, 94
for IsList, 94
PushoutFunctorial
for IsList, IsList, IsList, 96
PushoutFunctorialWithGivenPushouts
for IsCapCategoryObject, IsList, IsList, IsList, IsCapCategoryObject, 97
PushoutOp
for IsList, IsCapCategoryMorphism, 94

RandomMorphism
for IsCapCategory, IsInt, 24
for IsCapCategory, IsList, 24
RandomMorphismByInteger
for IsCapCategory, IsInt, 23
RandomMorphismByList
for IsCapCategory, IsList, 23
RandomMorphismWithFixedRange
for IsCapCategoryObject, IsInt, 24
for IsCapCategoryObject, IsList, 24
RandomMorphismWithFixedRangeByInteger
for IsCapCategoryObject, IsInt, 21
RandomMorphismWithFixedRangeByList
for IsCapCategoryObject, IsList, 22
RandomMorphismWithFixedSource
for IsCapCategoryObject, IsInt, 23
for IsCapCategoryObject, IsList, 23
RandomMorphismWithFixedSourceAndRange
for IsCapCategoryObject, IsCapCategoryObject, IsInt, 24
for IsCapCategoryObject, IsCapCategoryObject, IsList, 24
RandomMorphismWithFixedSourceAndRangeByInteger
for IsCapCategoryObject, IsCapCategoryObject, IsInt, 22
RandomMorphismWithFixedSourceAndRangeByList
for IsCapCategoryObject, IsCapCategoryObject, IsList, 22
RandomMorphismWithFixedSourceByInteger
for IsCapCategoryObject, IsInt, 21
RandomMorphismWithFixedSourceByList
for IsCapCategoryObject, IsList, 21
RandomObject
for IsCapCategory, IsInt, 13
for IsCapCategory, IsList, 13
RandomObjectByInteger
for IsCapCategory, IsInt, 13
RandomObjectByList
for IsCapCategory, IsList, 13
Range
for IsCapCategoryMorphism, 19
for IsCapCategoryTwoCell, 39
Reevaluate
for IsOperationWeightList, 118

Saturate
for IsOperationWeightList, 118
SetCachingOfCategory, 8
SetCachingOfCategoryCrisp, 8
SetCachingOfCategoryWeak, 8
SetDefaultCaching, 8
SetDefaultCachingCrisp, 8
SetDefaultCachingWeak, 8
SolveLinearSystemInAbCategory
for IsList, IsList, IsList, 38
SomeInjectiveObject
for IsCapCategoryObject, 16
SomeProjectiveObject
for IsCapCategoryObject, 15
Source
for IsCapCategoryMorphism, 19
for IsCapCategoryTwoCell, 39
StringMinHeap, 119
SubtractionForMorphisms
for IsCapCategoryMorphism, IsCapCategoryMorphism, 27
Swap
for IsStringMinHeap, IsPosInt, IsPosInt, 120

TargetOperation
for IsDerivedMethod, 114
TerminalObject
for IsCapCategory, 58
for IsCapCategoryCell, 58
TerminalObjectFunctorial
for IsCapCategory, 59
TwoCellFilter
for IsCapCategory, 7

UniversalMorphismFromCoequalizer
for IsList, IsCapCategoryMorphism, 81

UniversalMorphismFromCoequalizerWithGivenCoequalizer
for IsList, IsCapCategoryMorphism, IsCapCategoryObject, 81
UniversalMorphismFromCoproduct, 71
UniversalMorphismFromCoproductOp
for IsList, IsList, IsCapCategoryObject, 72
UniversalMorphismFromCoproductWithGivenCoproduct
for IsList, IsList, IsCapCategoryObject, 72
UniversalMorphismFromDirectSum, 64
UniversalMorphismFromDirectSumOp
for IsList, IsList, IsCapCategoryObject, 64
UniversalMorphismFromDirectSumWithGivenDirectSum
for IsList, IsList, IsCapCategoryObject, 64
UniversalMorphismFromImage
for IsCapCategoryMorphism, IsList, 99
UniversalMorphismFromImageWithGivenImageObject
for IsCapCategoryMorphism, IsList, IsCapCategoryObject, 99
UniversalMorphismFromInitialObject
for IsCapCategoryObject, 60
UniversalMorphismFromInitialObjectWithGivenInitialObject
for IsCapCategoryObject, IsCapCategoryObject, 60
UniversalMorphismFromPushout, 95
UniversalMorphismFromPushoutOp
for IsList, IsList, IsCapCategoryMorphism, 95
UniversalMorphismFromPushoutWithGivenPushout
for IsList, IsList, IsCapCategoryObject, 95
UniversalMorphismFromZeroObject
for IsCapCategoryObject, 55
UniversalMorphismFromZeroObjectWithGivenZeroObject
for IsCapCategoryObject, IsCapCategoryObject, 55
UniversalMorphismIntoCoimage
for IsCapCategoryMorphism, IsList, 104
UniversalMorphismIntoCoimageWithGivenCoimage
for IsCapCategoryMorphism, IsList, IsCapCategoryObject, 104
UniversalMorphismIntoDirectProduct, 74
UniversalMorphismIntoDirectProductOp
for IsList, IsList, IsCapCategoryObject, 75
UniversalMorphismIntoDirectProductWithGivenDirectProduct
for IsList, IsList, IsCapCategoryObject, 75
UniversalMorphismIntoDirectSum, 63
UniversalMorphismIntoDirectSumOp
for IsList, IsList, IsCapCategoryObject, 63
UniversalMorphismIntoDirectSumWithGivenDirectSum
for IsList, IsList, IsCapCategoryObject, 64
UniversalMorphismIntoEqualizer
for IsList, IsCapCategoryMorphism, 78
UniversalMorphismIntoEqualizerWithGivenEqualizer
for IsList, IsCapCategoryMorphism, IsCapCategoryObject, 78
UniversalMorphismIntoFiberProduct, 88
UniversalMorphismIntoFiberProductOp
for IsList, IsList, IsCapCategoryMorphism, 88
UniversalMorphismIntoFiberProductWithGivenFiberProduct
for IsList, IsList, IsCapCategoryObject, 88
UniversalMorphismIntoTerminalObject
for IsCapCategoryObject, 58
UniversalMorphismIntoTerminalObjectWithGivenTerminalObject
for IsCapCategoryObject, IsCapCategoryObject, 59
UniversalMorphismIntoZeroObject
for IsCapCategoryObject, 55
UniversalMorphismIntoZeroObjectWithGivenZeroObject
for IsCapCategoryObject, IsCapCategoryObject, 55
UsedOperationMultiples
for IsDerivedMethod, 114
UsedOperations
for IsDerivedMethod, 114
UsedOperationsWithMultiples
for IsDerivedMethod, 114

VerticalPostCompose
for IsCapCategoryTwoCell, IsCapCategoryTwoCell, 40

VerticalPreCompose
for IsCapCategoryTwoCell, IsCapCategoryTwoCell, 40

ZeroMorphism
for IsCapCategoryObject, IsCapCategoryObject, 28
ZeroObject
for IsCapCategory, 54
for IsCapCategoryCell, 54
ZeroObjectFunctorial
for IsCapCategory, 58