

# CAP

## Categories, Algorithms, Programming

2024.06-02

10 June 2024

**Sebastian Gutsche**

**Sebastian Posur**

**Øystein Skartsæterhagen**

**Fabian Zickgraf**

**Sebastian Gutsche**

Email: [gutsche@mathematik.uni-siegen.de](mailto:gutsche@mathematik.uni-siegen.de)

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

Address: Department Mathematik

Universität Siegen

Walter-Flex-Straße 3

57068 Siegen

Germany

**Sebastian Posur**

Email: [sebastian.posur@uni-siegen.de](mailto: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](mailto:oysteini@math.ntnu.no)

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

Address: NTNU

Institutt for matematiske fag

7491 Trondheim

Norway

**Fabian Zickgraf**

Email: [fabian.zickgraf@uni-siegen.de](mailto:fabian.zickgraf@uni-siegen.de)

Homepage: <https://github.com/zickgraf/>

Address: Walter-Flex-Str. 3

57068 Siegen

Germany

# Contents

<b>1</b>	<b>CAP Categories</b>	<b>6</b>
1.1	Categories . . . . .	7
1.2	Categorical properties . . . . .	7
1.3	Constructor . . . . .	10
1.4	Internal Attributes . . . . .	10
1.5	Logic switcher . . . . .	12
1.6	Tool functions . . . . .	13
1.7	Well-Definedness of Cells . . . . .	14
1.8	Unpacking data structures . . . . .	14
1.9	Caching . . . . .	15
1.10	Sanity checks . . . . .	15
1.11	Timing statistics . . . . .	16
1.12	Enable automatic calls of Add . . . . .	16
1.13	Performance tweaks . . . . .	16
1.14	LaTeX . . . . .	17
<b>2</b>	<b>Objects</b>	<b>18</b>
2.1	Attributes for the Type of Objects . . . . .	18
2.2	Equalities for Objects . . . . .	18
2.3	Categorical Properties of Objects . . . . .	19
2.4	Random Objects . . . . .	20
2.5	Tool functions for caches . . . . .	20
2.6	Adding Objects to a Category . . . . .	21
2.7	Object constructors . . . . .	22
2.8	Well-Definedness of Objects . . . . .	23
2.9	Projectives . . . . .	23
2.10	Injectives . . . . .	24
2.11	Simplified Objects . . . . .	24
2.12	Dimensions . . . . .	25
<b>3</b>	<b>Morphisms</b>	<b>26</b>
3.1	Attributes for the Type of Morphisms . . . . .	26
3.2	Morphism constructors . . . . .	27
3.3	Categorical Properties of Morphisms . . . . .	27
3.4	Random Morphisms . . . . .	28
3.5	Non-Categorical Properties of Morphisms . . . . .	31

3.6	Adding Morphisms to a Category . . . . .	32
3.7	Equality and Congruence for Morphisms . . . . .	33
3.8	Basic Operations for Morphisms in Ab-Categories . . . . .	34
3.9	Subobject and Factorobject Operations . . . . .	35
3.10	Identity Morphism and Composition of Morphisms . . . . .	36
3.11	Well-Definedness of Morphisms . . . . .	37
3.12	Lift/Colift . . . . .	38
3.13	Inverses . . . . .	40
3.14	Tool functions for caches . . . . .	41
3.15	IsHomSetInhabited . . . . .	41
3.16	Homomorphism structures . . . . .	41
3.17	Simplified Morphisms . . . . .	46
3.18	Reduction by split epi summands . . . . .	49
<b>4</b>	<b>Category 2-Cells</b>	<b>51</b>
4.1	Attributes for the Type of 2-Cells . . . . .	51
4.2	Identity 2-Cell and Composition of 2-Cells . . . . .	51
4.3	Well-Definedness for 2-Cells . . . . .	52
<b>5</b>	<b>Category of Categories</b>	<b>53</b>
5.1	The Category Cat . . . . .	53
5.2	Categories . . . . .	53
5.3	Constructors . . . . .	54
5.4	Functors . . . . .	54
5.5	Natural transformations . . . . .	57
<b>6</b>	<b>Universal Objects</b>	<b>59</b>
6.1	Kernel . . . . .	59
6.2	Cokernel . . . . .	61
6.3	Zero Object . . . . .	64
6.4	Terminal Object . . . . .	66
6.5	Initial Object . . . . .	67
6.6	Direct Sum . . . . .	69
6.7	Coproduct . . . . .	73
6.8	Direct Product . . . . .	76
6.9	Equalizer . . . . .	78
6.10	Coequalizer . . . . .	81
6.11	Fiber Product . . . . .	84
6.12	Pushout . . . . .	87
6.13	Image . . . . .	90
6.14	Coimage . . . . .	93
6.15	Homology objects . . . . .	96
6.16	Projective covers and injective envelopes . . . . .	97

<b>7</b>	<b>Add Functions</b>	<b>99</b>
7.1	Functions Installed by Add . . . . .	99
7.2	Add Method . . . . .	100
7.3	InstallAdd Function . . . . .	101
7.4	Enhancing the method name record . . . . .	104
7.5	Prepare functions . . . . .	104
7.6	Available Add functions . . . . .	105
<b>8</b>	<b>Managing Derived Methods</b>	<b>156</b>
8.1	Info Class . . . . .	156
8.2	Derivation Objects . . . . .	156
8.3	Derivation Graphs . . . . .	158
8.4	Managing Derivations in a Category . . . . .	160
8.5	Min Heaps for Strings . . . . .	162
<b>9</b>	<b>Technical Details</b>	<b>164</b>
9.1	The Category Cat . . . . .	164
9.2	Tools . . . . .	164
<b>10</b>	<b>Limits and Colimits</b>	<b>175</b>
10.1	Specification of Limits and Colimits . . . . .	175
10.2	Enhancing Limit Specifications . . . . .	176
10.3	Functions . . . . .	177
<b>11</b>	<b>The Category Constructor</b>	<b>178</b>
11.1	Info class . . . . .	178
11.2	Constructors . . . . .	178
<b>12</b>	<b>Reinterpretations of categories</b>	<b>181</b>
12.1	Introduction . . . . .	181
12.2	Tutorial . . . . .	181
12.3	Implementation details . . . . .	183
12.4	Relation to CompilerForCAP . . . . .	183
12.5	Attributes . . . . .	184
12.6	Constructors . . . . .	184
<b>13</b>	<b>Create wrapper hulls of a category</b>	<b>186</b>
13.1	GAP categories . . . . .	186
13.2	Attributes . . . . .	186
13.3	Constructors . . . . .	187
<b>14</b>	<b>Dummy implementations</b>	<b>189</b>
14.1	Dummy rings . . . . .	189
14.2	Dummy categories . . . . .	190

<b>15</b>	<b>Examples and Tests</b>	<b>192</b>
15.1	Dummy implementations . . . . .	192
15.2	Functors . . . . .	193
15.3	HandlePrecompiledTowers . . . . .	194
15.4	Homomorphism structure . . . . .	195
15.5	Homology object . . . . .	198
15.6	Liftable . . . . .	199
15.7	Monoidal Categories . . . . .	200
15.8	MorphismFromSourceToPushout and MorphismFromFiberProductToSink . . . . .	202
15.9	Opposite category . . . . .	203
15.10	PreComposeList and PostComposeList . . . . .	203
15.11	Terminal category . . . . .	204
15.12	WrapperCategory . . . . .	209
<b>16</b>	<b>Terminal category</b>	<b>210</b>
16.1	GAP Categories . . . . .	210
16.2	Constructors . . . . .	211
16.3	Attributes . . . . .	211
	<b>Index</b>	<b>213</b>

# 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.

Classically, a category consists of a class of objects, a set of morphisms, identity morphisms, and a composition function satisfying some simple axioms. In CAP, we use a slightly different notion of a category.

A CAP category  $\mathbf{C}$  consists of the following data:

- A set  $\text{Obj}_{\mathbf{C}}$  of *objects*.
- For every pair  $a, b \in \text{Obj}_{\mathbf{C}}$ , a set  $\text{Hom}_{\mathbf{C}}(a, b)$  of *morphisms*.
- For every pair  $a, b \in \text{Obj}_{\mathbf{C}}$ , an equivalence relation  $\sim_{a,b}$  on  $\text{Hom}_{\mathbf{C}}(a, b)$  called *congruence for morphisms*.
- For every  $a \in \text{Obj}_{\mathbf{C}}$ , an *identity morphism*  $\text{id}_a \in \text{Hom}_{\mathbf{C}}(a, a)$ .
- For every triple  $a, b, c \in \text{Obj}_{\mathbf{C}}$ , a *composition function*

$$\circ : \text{Hom}_{\mathbf{C}}(b, c) \times \text{Hom}_{\mathbf{C}}(a, b) \rightarrow \text{Hom}_{\mathbf{C}}(a, c)$$

compatible with the congruence, i.e., if  $\alpha, \alpha' \in \text{Hom}_{\mathbf{C}}(a, b)$ ,  $\beta, \beta' \in \text{Hom}_{\mathbf{C}}(b, c)$ ,  $\alpha \sim_{a,b} \alpha'$  and  $\beta \sim_{b,c} \beta'$ , then  $\beta \circ \alpha \sim_{a,c} \beta' \circ \alpha'$ .

- For all  $a, b \in \text{Obj}_{\mathbf{C}}$ ,  $\alpha \in \text{Hom}_{\mathbf{C}}(a, b)$ , we have

$$(\text{id}_b \circ \alpha) \sim_{a,b} \alpha$$

and

$$\alpha \sim_{a,b} (\alpha \circ \text{id}_a).$$

- For all  $a, b, c, d \in \text{Obj}_{\mathbf{C}}$ ,  $\alpha \in \text{Hom}_{\mathbf{C}}(a, b)$ ,  $\beta \in \text{Hom}_{\mathbf{C}}(b, c)$ ,  $\gamma \in \text{Hom}_{\mathbf{C}}(c, d)$ , we have

$$((\gamma \circ \beta) \circ \alpha) \sim_{a,d} (\gamma \circ (\beta \circ \alpha))$$

## 1.1 Categories

### 1.1.1 IsCapCategory (for IsAttributeStoringRep)

▷ 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.

## 1.2 Categorical properties

### 1.2.1 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.



### 1.2.2 IsEquippedWithHomomorphismStructure (for IsCapCategory)

▷ `IsEquippedWithHomomorphismStructure( $\mathcal{C}$ )` (property)

**Returns:** true or false

The property of the category  $\mathcal{C}$  being equipped with a homomorphism structure.

### 1.2.3 IsCategoryWithDecidableLifts (for IsCapCategory)

▷ `IsCategoryWithDecidableLifts( $\mathcal{C}$ )` (property)

**Returns:** true or false

The property of the category  $\mathcal{C}$  having decidable lifts.

### 1.2.4 IsCategoryWithDecidableColifts (for IsCapCategory)

▷ `IsCategoryWithDecidableColifts( $\mathcal{C}$ )` (property)

**Returns:** true or false

The property of the category  $\mathcal{C}$  having decidable colifts.

### 1.2.5 IsEnrichedOverCommutativeRegularSemigroup (for IsCapCategory)

▷ `IsEnrichedOverCommutativeRegularSemigroup( $\mathcal{C}$ )` (property)

**Returns:** true or false

The property of the category  $\mathcal{C}$  being enriched over a commutative regular semigroup.

### 1.2.6 IsSkeletalCategory (for IsCapCategory)

▷ `IsSkeletalCategory( $\mathcal{C}$ )` (property)

**Returns:** true or false

The property of the category  $\mathcal{C}$  being skeletal, that is, whether `IsEqualForObjects` and `IsIsomorphicForObjects` coincide.

### 1.2.7 IsAbCategory (for IsCapCategory)

▷ `IsAbCategory( $\mathcal{C}$ )` (property)

**Returns:** true or false

The property of the category  $\mathcal{C}$  being preadditive.

### 1.2.8 IsLinearCategoryOverCommutativeRing (for IsCapCategory)

▷ `IsLinearCategoryOverCommutativeRing( $\mathcal{C}$ )` (property)

**Returns:** true or false

The property of the category  $\mathcal{C}$  being linear over a commutative ring.

### 1.2.9 IsLinearCategoryOverCommutativeRingWithFinitelyGeneratedFreeExternalHoms (for IsCapCategory)

▷ `IsLinearCategoryOverCommutativeRingWithFinitelyGeneratedFreeExternalHoms( $\mathcal{C}$ )` (property)

**Returns:** true or false

The property of the category  $\mathcal{C}$  being linear over a commutative ring  $k$  such that all external homs are finitely generated free  $k$ -modules.

### 1.2.10 IsAdditiveCategory (for IsCapCategory)

- ▷ IsAdditiveCategory( $\mathcal{C}$ ) (property)  
**Returns:** true or false  
The property of the category  $\mathcal{C}$  being additive.

### 1.2.11 IsPreAbelianCategory (for IsCapCategory)

- ▷ IsPreAbelianCategory( $\mathcal{C}$ ) (property)  
**Returns:** true or false  
The property of the category  $\mathcal{C}$  being preabelian.

### 1.2.12 IsAbelianCategory (for IsCapCategory)

- ▷ IsAbelianCategory( $\mathcal{C}$ ) (property)  
**Returns:** true or false  
The property of the category  $\mathcal{C}$  being abelian.

### 1.2.13 IsAbelianCategoryWithEnoughProjectives (for IsCapCategory)

- ▷ IsAbelianCategoryWithEnoughProjectives( $\mathcal{C}$ ) (property)  
**Returns:** true or false  
The property of the category  $\mathcal{C}$  being abelian with enough projectives.

### 1.2.14 IsAbelianCategoryWithEnoughInjectives (for IsCapCategory)

- ▷ IsAbelianCategoryWithEnoughInjectives( $\mathcal{C}$ ) (property)  
**Returns:** true or false  
The property of the category  $\mathcal{C}$  being abelian with enough injectives.

### 1.2.15 IsLocallyOfFiniteProjectiveDimension (for IsCapCategory)

- ▷ IsLocallyOfFiniteProjectiveDimension( $\mathcal{C}$ ) (property)  
**Returns:** true or false  
The property of the category  $\mathcal{C}$  being locally of finite projective dimension.

### 1.2.16 IsLocallyOfFiniteInjectiveDimension (for IsCapCategory)

- ▷ IsLocallyOfFiniteInjectiveDimension( $\mathcal{C}$ ) (property)  
**Returns:** true or false  
The property of the category  $\mathcal{C}$  being locally of finite injective dimension.

## 1.3 Constructor

### 1.3.1 CreateCapCategory

- ▷ `CreateCapCategory()` (operation)  
**Returns:** a category  
 Creates a new CAP category from scratch. It gets a generic name.

### 1.3.2 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*.

### 1.3.3 CreateCapCategory (for IsString, IsFunction, IsFunction, IsFunction, IsFunction)

- ▷ `CreateCapCategory(s, category_filter, object_filter, morphism_filter, two_cell_filter)` (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*. The category, its objects, its morphisms, and its two cells will lie in the corresponding given filters.

### 1.3.4 CreateCapCategoryWithDataTypes

- ▷ `CreateCapCategoryWithDataTypes(s, category_filter, object_filter, morphism_filter, two_cell_filter, object_datum_type, morphism_datum_type, two_cell_datum_type)` (function)  
**Returns:** a category  
 The argument is a string *s*. This operation creates a new CAP category from scratch. Its name is set to *s*. The category, its objects, its morphisms, and its two cells will lie in the corresponding given filters. The data types of the object/morphism/two cell datum can be given as described in `CapJitInferredDataTypes` (**CompilerForCAP: CapJitInferredDataTypes**). As a convenience, simply a filter can be given if this suffices to fully determine the data type. If a data type is not specified, pass `fail` instead.

## 1.4 Internal Attributes

### 1.4.1 Name (for IsCapCategory)

- ▷ `Name(C)` (attribute)  
**Returns:** a string  
 The argument is a category *C*. The output is its name.  
 Each category *C* stores various filters. They are used to apply the right functions in the method selection.

### 1.4.2 CategoryFilter (for IsCapCategory)

▷ `CategoryFilter(C)` (attribute)

**Returns:** a filter

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

### 1.4.3 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.

### 1.4.4 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.

### 1.4.5 TwoCellFilter (for IsCapCategory)

▷ `TwoCellFilter(C)` (attribute)

**Returns:** a filter

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

### 1.4.6 ObjectDatumType (for IsCapCategory)

▷ `ObjectDatumType(C)` (attribute)

**Returns:** a data type or fail

The argument is a category *C*. The output is the data type (see `CapJitInferredDataTypes (CompilerForCAP: CapJitInferredDataTypes)`) of object data of *C* (or fail if this data type is not specified).

### 1.4.7 MorphismDatumType (for IsCapCategory)

▷ `MorphismDatumType(C)` (attribute)

**Returns:** a data type or fail

The argument is a category *C*. The output is the data type (see `CapJitInferredDataTypes (CompilerForCAP: CapJitInferredDataTypes)`) of morphism data of *C* (or fail if this data type is not specified).

### 1.4.8 TwoCellDatumType (for IsCapCategory)

▷ `TwoCellDatumType(C)` (attribute)

**Returns:** a data type or fail

The argument is a category *C*. The output is the data type (see `CapJitInferredDataTypes (CompilerForCAP: CapJitInferredDataTypes)`) of two cell data of *C* (or fail if this data type is not specified).

### 1.4.9 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.

### 1.4.10 RangeCategoryOfHomomorphismStructure (for IsCapCategory)

▷ `RangeCategoryOfHomomorphismStructure(C)` (attribute)

**Returns:** a category

The argument is a category  $C$  which is expected to lie in the filter `IsEquippedWithHomomorphismStructure`. The output is the range category  $D$  of the defining functor  $H : C^{\text{op}} \times C \rightarrow D$  of the homomorphism structure.

### 1.4.11 AdditiveGenerators (for IsCapCategory)

▷ `AdditiveGenerators(C)` (attribute)

**Returns:** a list of objects

The argument is an additive category  $C$ . The output is a list  $L$  of objects in  $C$  such that every object in  $C$  is a finite direct sum of objects in  $L$ .

### 1.4.12 IndecomposableProjectiveObjects (for IsCapCategory)

▷ `IndecomposableProjectiveObjects(C)` (attribute)

**Returns:** a list of objects

The argument is an Abelian category  $C$  with enough projectives. The output is the set of indecomposable projective objects in  $C$  up to isomorphism. That is every projective object in  $C$  is isomorphic to a finite direct sum over these objects.

### 1.4.13 IndecomposableInjectiveObjects (for IsCapCategory)

▷ `IndecomposableInjectiveObjects(C)` (attribute)

**Returns:** a list of objects

The argument is an Abelian category  $C$  with enough injectives. The output is the set of indecomposable injective objects in  $C$  up to isomorphism. That is every injective object in  $C$  is isomorphic to a finite direct sum over these objects.

## 1.5 Logic switcher

### 1.5.1 CapCategorySwitchLogicPropagationForObjectsOn

▷ `CapCategorySwitchLogicPropagationForObjectsOn(C)` (function)

Activates the predicate logic propagation between equal objects for the category  $C$ .

### 1.5.2 CapCategorySwitchLogicPropagationForObjectsOff

▷ CapCategorySwitchLogicPropagationForObjectsOff( $\mathcal{C}$ ) (function)

Deactivates the predicate logic propagation between equal objects for the category  $\mathcal{C}$ .

### 1.5.3 CapCategorySwitchLogicPropagationForMorphismsOn

▷ CapCategorySwitchLogicPropagationForMorphismsOn( $\mathcal{C}$ ) (function)

Activates the predicate logic propagation between equal morphisms for the category  $\mathcal{C}$ .

### 1.5.4 CapCategorySwitchLogicPropagationForMorphismsOff

▷ CapCategorySwitchLogicPropagationForMorphismsOff( $\mathcal{C}$ ) (function)

Deactivates the predicate logic propagation between equal morphisms for the category  $\mathcal{C}$ .

### 1.5.5 CapCategorySwitchLogicPropagationOn

▷ CapCategorySwitchLogicPropagationOn( $\mathcal{C}$ ) (function)

Activates the predicate logic propagation between equal cells for the category  $\mathcal{C}$ .

### 1.5.6 CapCategorySwitchLogicPropagationOff

▷ CapCategorySwitchLogicPropagationOff( $\mathcal{C}$ ) (function)

Deactivates the predicate logic propagation between equal cells for the category  $\mathcal{C}$ .

### 1.5.7 CapCategorySwitchLogicOn

▷ CapCategorySwitchLogicOn( $\mathcal{C}$ ) (function)

Activates the predicate implication logic for the category  $\mathcal{C}$ .

### 1.5.8 CapCategorySwitchLogicOff

▷ CapCategorySwitchLogicOff( $\mathcal{C}$ ) (function)

Deactivates the predicate implication logic for the category  $\mathcal{C}$ .

## 1.6 Tool functions

### 1.6.1 CanCompute (for IsCapCategory, IsString)

▷ CanCompute( $\mathcal{C}$ , *string*) (operation)

▷ CanCompute( $\mathcal{C}$ , *operation*) (operation)

**Returns:** true or false

The argument is a category  $C$  and a string *string*, which should be the name of a CAP 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 CAP operation, an error is raised. For debugging purposes one can also pass the CAP operation instead of its name.

### 1.6.2 MissingOperationsForConstructivenessOfCategory (for IsCapCategory, Is-String)

▷ `MissingOperationsForConstructivenessOfCategory(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 CAP operations which are missing in  $C$  to have the categorical property constructively. If  $s$  is not a categorical property, an error is raised.

## 1.7 Well-Definedness of Cells

### 1.7.1 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`.

## 1.8 Unpacking data structures

### 1.8.1 Down (for IsObject)

▷ `Down(x)` (attribute)

**Returns:** a GAP object

The argument is a GAP object  $x$ . If  $x$  is an object in a CAP category, the output consists of data which are needed to reconstruct  $x$  (e.g., by passing them to an appropriate constructor). If  $x$  is a morphism in a CAP category, the output consists of a triple whose first entry is the source of  $x$ , the third entry is the range of  $x$ , and the second entry consists of data which are needed to reconstruct  $x$  (e.g., by passing them to an appropriate constructor, possibly together with the source and range of  $x$ ).

### 1.8.2 DownOnlyMorphismData (for IsCapCategoryMorphism)

▷ `DownOnlyMorphismData(x)` (attribute)

**Returns:** a GAP object

The argument is a morphism in a CAP category, the output consists of data which are needed to reconstruct  $x$  (e.g., by passing it to an appropriate constructor, possibly together with its source and range).

### 1.8.3 DownToBottom (for IsObject)

▷ `DownToBottom(x)` (attribute)

**Returns:** a GAP object

The argument is a GAP object  $x$ . This function iteratively calls `Down` until it becomes stable.

## 1.9 Caching

### 1.9.1 SetCachingOfCategory

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

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

### 1.9.2 SetCachingOfCategoryWeak

▷ `SetCachingOfCategoryWeak(category)` (function)

▷ `SetCachingOfCategoryCrisp(category)` (function)

▷ `DeactivateCachingOfCategory(category)` (function)

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

### 1.9.3 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.

## 1.10 Sanity checks

### 1.10.1 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.



## 1.11 Timing statistics

### 1.11.1 EnableTimingStatistics

▷ EnableTimingStatistics( <i>category</i> )	(function)
▷ DisableTimingStatistics( <i>category</i> )	(function)
▷ ResetTimingStatistics( <i>category</i> )	(function)
▷ DisplayTimingStatistics( <i>category</i> )	(function)
▷ BrowseTimingStatistics( <i>category</i> )	(function)

Enable, disable, reset, display, or browse timing statistics of the primitive operations of *category*. Caution: If a primitive operation calls another primitive operation, the runtime of the later (including sanity checks etc.) is also included in the runtime of the former.

## 1.12 Enable automatic calls of Add

### 1.12.1 EnableAddForCategoricalOperations

▷ EnableAddForCategoricalOperations( <i>C</i> )	(function)
▷ DisableAddForCategoricalOperations( <i>C</i> )	(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).

## 1.13 Performance tweaks

For finding performance issues in primitive operations, you can collect timing statistics, see 1.11. You can use the package CompilerForCAP to compile your code. Additionally, CAP has several settings which can improve the performance. In the following some of these are listed.

- DeactivateCachingOfCategory or DeactivateDefaultCaching: see 1.9. This can either improve or degrade the performance depending on the concrete example.
- CapCategorySwitchLogicOff (on by default) or CapCategorySwitchLogicPropagationOff (off by default): see 1.5. This can either improve or degrade the performance depending on the concrete example.
- DisableSanityChecks: see 1.10.
- DisableAddForCategoricalOperations: see 1.12.
- DeactivateToDoList: see the package ToolsForHomalg.
- Use CreateCapCategoryObjectWithAttributes (2.6) instead of AddObject and CreateCapCategoryMorphismWithAttributes (3.6) instead of AddMorphism.

- Pass the option `overhead := false` to `CreateCapCategory`. Note: this may have unintended effects. Use with care!

## 1.14 LaTeX

### 1.14.1 LaTeXOutput (for IsCapCategoryCell)

▷ `LaTeXOutput(c)` (operation)

**Returns:** a string

The argument is a cell  $c$ . The output is a LaTeX string  $s$  (without enclosing dollar signs) that may be used to print out  $c$  nicely.

### 1.14.2 LaTeXOutput (for IsCapCategory)

▷ `LaTeXOutput(C)` (operation)

**Returns:** a string

The argument is a category  $C$ . The output is a LaTeX string  $s$  (without enclosing dollar signs) that may be used to print out  $C$  nicely.

## 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  $C$  to which  $a$  was added.

### 2.2 Equalities 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 `IsIsomorphicForObjects` (for `IsCapCategoryObject`, `IsCapCategoryObject`)

- ▷ `IsIsomorphicForObjects(a, b)` (operation)  
**Returns:** a boolean  
The arguments are two objects  $a$  and  $b$ . The output is `true` if  $a$  and  $b$  are isomorphic, that is, if there exists an isomorphism  $a \rightarrow b$ , otherwise the output is `false`.

#### 2.2.3 `SomeIsomorphismBetweenObjects` (for `IsCapCategoryObject`, `IsCapCategoryObject`)

- ▷ `SomeIsomorphismBetweenObjects(a, b)` (operation)  
**Returns:** an isomorphism in  $\text{Hom}(a, b)$

The arguments are two isomorphic objects  $a$  and  $b$ . The output is an isomorphism  $a \rightarrow b$ .

## 2.3 Categorical Properties of Objects

### 2.3.1 IsBijjectiveObject (for IsCapCategoryObject)

▷ `IsBijjectiveObject(a)` (property)  
**Returns:** a boolean  
 The argument is an object  $a$ . The output is `true` if  $a$  is a bijective object, otherwise the output is `false`.

### 2.3.2 IsProjective (for IsCapCategoryObject)

▷ `IsProjective(a)` (property)  
**Returns:** a boolean  
 The argument is an object  $a$ . The output is `true` if  $a$  is a projective object, otherwise the output is `false`.

### 2.3.3 IsInjective (for IsCapCategoryObject)

▷ `IsInjective(a)` (property)  
**Returns:** a boolean  
 The argument is an object  $a$ . The output is `true` if  $a$  is an injective object, otherwise the output is `false`.

### 2.3.4 IsTerminal (for IsCapCategoryObject)

▷ `IsTerminal(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 terminal object of  $\mathbf{C}$ , otherwise the output is `false`.

### 2.3.5 IsInitial (for IsCapCategoryObject)

▷ `IsInitial(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 initial object of  $\mathbf{C}$ , otherwise the output is `false`.

### 2.3.6 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`.

### 2.3.7 IsZero (for IsCapCategoryObject)

▷ `IsZero(a)` (property)

**Returns:** a boolean

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

## 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 in  $C$

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

### 2.4.2 RandomObjectByList (for IsCapCategory, IsList)

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

**Returns:** an object in  $C$

The arguments are a category  $C$  and a list  $L$ . The output is a random object in  $C$ .

### 2.4.3 RandomObject (for IsCapCategory, IsInt)

▷ `RandomObject(C, n)` (operation)

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

### 2.4.4 RandomObject (for IsCapCategory, IsList)

▷ `RandomObject(C, L)` (operation)

## 2.5 Tool functions for caches

### 2.5.1 IsEqualForCacheForObjects (for IsCapCategoryObject, IsCapCategoryObject)

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

**Returns:** true or false

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, `IsEqualForCacheForObjects` is used. By default, `IsEqualForCacheForObjects` falls back to `IsEqualForCache` (see `ToolsForHomalg`), which in turn defaults to recursive comparison for lists and `IsIdenticalObj` in all other cases. If you add a function via `AddIsEqualForCacheForObjects`, that function is used instead. A function  $F : a, b \mapsto \text{bool}$  is expected there. 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)

*Deprecated:* use `CreateCapCategoryWithDataTypes` (1.3.4) instead. 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. *filter* must imply `IsCapCategoryObject`.

### 2.6.4 ObjectifyObjectForCAPWithAttributes

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

**Returns:** an object

*Deprecated:* use `CreateCapCategoryObjectWithAttributes` (2.6.5) instead.

### 2.6.5 CreateCapCategoryObjectWithAttributes

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

**Returns:** an object

Creates an object in *category* with the given attributes.

### 2.6.6 AsCapCategoryObject

▷ `AsCapCategoryObject(category, value)` (function)

**Returns:** an object

EXPERIMENTAL: This specification might change any time without prior notice. Views *value* as an object in *category*.

### 2.6.7 AsPrimitiveValue (for IsCapCategoryObject)

▷ `AsPrimitiveValue(object)` (attribute)

▷ `AsInteger(object)` (attribute)

▷ `AsHomalgMatrix(object)` (attribute)

**Returns:** a value

EXPERIMENTAL: This specification might change any time without prior notice. Views an object obtained via `AsCapCategoryObject` (2.6.6) as a primitive value again. Here, the word *primitive* means *primitive from the perspective of the category*. For example, from the perspective of an opposite category, objects of the underlying category are primitive values. The attribute is chosen according to the object datum type:

- For `IsInt`, the attribute `AsInteger` is used.
- For `IsHomalgMatrix`, the attribute `AsHomalgMatrix` is used.

In all other cases or if no object datum type is given, the attribute `AsPrimitiveValue` is used.

## 2.7 Object constructors

### 2.7.1 ObjectConstructor (for IsCapCategory, IsObject)

▷ `ObjectConstructor(C, a)` (operation)

**Returns:** an object

The arguments are a category *C* and an object datum *a* (type and semantics of the object datum depend on the category). The output is an object of *C* defined by *a*. Note that by default this CAP operation is not cached. You can change this behaviour by calling `SetCachingToWeak( C, "ObjectConstructor" )` resp. `SetCachingToCrisp( C, "ObjectConstructor" )`.

### 2.7.2 / (for IsObject, IsCapCategory)

▷ `/(a, C)` (operation)

**Returns:** an object

Shorthand for `ObjectConstructor( C, a )`.

### 2.7.3 ObjectDatum (for IsCapCategoryObject)

▷ `ObjectDatum(obj)` (attribute)

**Returns:** depends on the category

The argument is a CAP category object *obj*. The output is a datum which can be used to construct *obj*, that is, `IsEqualForObjects( obj, ObjectConstructor( CapCategory( obj ), ObjectDatum( obj ) ) )`. Note that by default this CAP operation is not cached. You can change

this behaviour by calling `SetCachingToWeak( C, "ObjectDatum" )` resp. `SetCachingToCrisp( C, "ObjectDatum" )`.

## 2.8 Well-Definedness of Objects

### 2.8.1 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`.

## 2.9 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  $\pi : P \rightarrow A$ .

### 2.9.1 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  $\pi : P \rightarrow A$ .

### 2.9.2 EpimorphismFromSomeProjectiveObject (for IsCapCategoryObject)

▷ `EpimorphismFromSomeProjectiveObject(A)` (attribute)

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

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

### 2.9.3 EpimorphismFromSomeProjectiveObjectWithGivenSomeProjectiveObject (for IsCapCategoryObject, IsCapCategoryObject)

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

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

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

### 2.9.4 ProjectiveLift (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

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

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



## 2.10 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  $\iota : A \rightarrow I$ .

### 2.10.1 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  $\iota : A \rightarrow I$ .

### 2.10.2 MonomorphismIntoSomeInjectiveObject (for IsCapCategoryObject)

▷ `MonomorphismIntoSomeInjectiveObject(A)` (attribute)

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

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

### 2.10.3 MonomorphismIntoSomeInjectiveObjectWithGivenSomeInjectiveObject (for IsCapCategoryObject, IsCapCategoryObject)

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

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

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

### 2.10.4 InjectiveColift (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

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

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

## 2.11 Simplified Objects

Let  $i$  be a positive integer or  $\infty$ . For a given object  $A$ , an  $i$ -th simplified object of  $A$  consists of

- an object  $A_i$ ,
- an isomorphism  $\iota_A^i : A \rightarrow A_i$ .

The idea is that the greater the  $i$ , the "simpler" the  $A_i$  (but this could mean the harder the computation) with  $\infty$  as a possible value.

### 2.11.1 Simplify (for IsCapCategoryObject)

▷ `Simplify(A)` (attribute)

**Returns:** an object

The argument is an object  $A$ . The output is a simplified object  $A_\infty$ .

### 2.11.2 SimplifyObject (for IsCapCategoryObject, IsObject)

▷ `SimplifyObject( $A, i$ )` (operation)

**Returns:** an object

The arguments are an object  $A$  and a positive integer  $i$  or infinity. The output is a simplified object  $A_i$ .

### 2.11.3 SimplifyObject\_IsoFromInputObject (for IsCapCategoryObject, IsObject)

▷ `SimplifyObject_IsoFromInputObject( $A, i$ )` (operation)

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

The arguments are an object  $A$  and a positive integer  $i$  or infinity. The output is an isomorphism to a simplified object  $\iota_A^i : A \rightarrow A_i$ .

### 2.11.4 SimplifyObject\_IsoToInputObject (for IsCapCategoryObject, IsObject)

▷ `SimplifyObject_IsoToInputObject( $A, i$ )` (operation)

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

The arguments are an object  $A$  and a positive integer  $i$  or infinity. The output is an isomorphism from a simplified object  $(\iota_A^i)^{-1} : A_i \rightarrow A$  which is the inverse of the output of `SimplifyObject_IsoFromInputObject`.

## 2.12 Dimensions

### 2.12.1 ProjectiveDimension (for IsCapCategoryObject)

▷ `ProjectiveDimension( $A$ )` (attribute)

**Returns:** a nonnegative integer or infinity

The argument is an object  $A$ . The output is the projective dimension of  $A$ .

### 2.12.2 InjectiveDimension (for IsCapCategoryObject)

▷ `InjectiveDimension( $A$ )` (attribute)

**Returns:** a nonnegative integer or infinity

The argument is an object  $A$ . The output is the injective dimension of  $A$ .

## 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  $\alpha$ . The output is the category  $\mathbf{C}$  to which  $\alpha$  was added.

#### 3.1.2 `Source` (for `IsCapCategoryMorphism`)

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

#### 3.1.3 `Range` (for `IsCapCategoryMorphism`)

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

#### 3.1.4 `Target` (for `IsCapCategoryMorphism`)

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

## 3.2 Morphism constructors

### 3.2.1 MorphismConstructor (for IsCapCategoryObject, IsObject, IsCapCategoryObject)

▷ `MorphismConstructor( $S$ ,  $a$ ,  $T$ )` (operation)

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

The arguments are two objects  $S$  and  $T$  in a category, and a morphism datum  $a$  (type and semantics of the morphism datum depend on the category). The output is a morphism in  $\text{Hom}(S, T)$  defined by  $a$ . Note that by default this CAP operation is not cached. You can change this behaviour by calling `SetCachingToWeak( C, "MorphismConstructor" )` resp. `SetCachingToCrisp( C, "MorphismConstructor" )`.

### 3.2.2 MorphismDatum (for IsCapCategoryMorphism)

▷ `MorphismDatum( $mor$ )` (attribute)

**Returns:** depends on the category

The argument is a CAP category morphism  $mor$ . The output is a datum which can be used to construct  $mor$ , that is, `IsEqualForMorphisms(  $mor$ , MorphismConstructor( Source(  $mor$  ), MorphismDatum(  $mor$  ), Range(  $mor$  ) ) )). Note that by default this CAP operation is not cached. You can change this behaviour by calling SetCachingToWeak( C, "MorphismDatum" ) resp. SetCachingToCrisp( C, "MorphismDatum" ).`

## 3.3 Categorical Properties of Morphisms

### 3.3.1 IsMonomorphism (for IsCapCategoryMorphism)

▷ `IsMonomorphism( $\alpha$ )` (property)

**Returns:** a boolean

The argument is a morphism  $\alpha$ . The output is true if  $\alpha$  is a monomorphism, otherwise the output is false.

### 3.3.2 IsEpimorphism (for IsCapCategoryMorphism)

▷ `IsEpimorphism( $\alpha$ )` (property)

**Returns:** a boolean

The argument is a morphism  $\alpha$ . The output is true if  $\alpha$  is an epimorphism, otherwise the output is false.

### 3.3.3 IsIsomorphism (for IsCapCategoryMorphism)

▷ `IsIsomorphism( $\alpha$ )` (property)

**Returns:** a boolean

The argument is a morphism  $\alpha$ . The output is true if  $\alpha$  is an isomorphism, otherwise the output is false.

### 3.3.4 IsSplitMonomorphism (for IsCapCategoryMorphism)

▷ `IsSplitMonomorphism(alpha)` (property)

**Returns:** a boolean

The argument is a morphism  $\alpha$ . The output is `true` if  $\alpha$  is a split monomorphism, otherwise the output is `false`.

### 3.3.5 IsSplitEpimorphism (for IsCapCategoryMorphism)

▷ `IsSplitEpimorphism(alpha)` (property)

**Returns:** a boolean

The argument is a morphism  $\alpha$ . The output is `true` if  $\alpha$  is a split epimorphism, otherwise the output is `false`.

### 3.3.6 IsOne (for IsCapCategoryMorphism)

▷ `IsOne(alpha)` (property)

**Returns:** a boolean

The argument is a morphism  $\alpha : a \rightarrow a$ . The output is `true` if  $\alpha$  is congruent to the identity of  $a$ , otherwise the output is `false`.

### 3.3.7 IsIdempotent (for IsCapCategoryMorphism)

▷ `IsIdempotent(alpha)` (property)

**Returns:** a boolean

The argument is a morphism  $\alpha : a \rightarrow a$ . The output is `true` if  $\alpha^2 \sim_{a,a} \alpha$ , otherwise the output is `false`.

## 3.4 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.

### 3.4.1 RandomMorphismWithFixedSourceByInteger (for IsCapCategoryObject, IsInt)

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

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

The arguments are an object  $a$  in a category  $C$  and an integer  $n$ . The output is a random morphism  $\alpha : a \rightarrow b$  for some object  $b$  in  $C$ . If  $C$  is equipped with the methods `RandomObjectByInteger` and `RandomMorphismWithFixedSourceAndRangeByInteger` and  $C$  is

an Ab-category, then  $\text{RandomMorphismWithFixedSourceByInteger}(C, a, n)$  can be derived as  $\text{RandomMorphismWithFixedSourceAndRangeByInteger}(C, a, b, 1 + \text{Log2Int}(n))$  where  $b$  is computed via  $\text{RandomObjectByInteger}(C, n)$ .

### 3.4.2 RandomMorphismWithFixedSourceByList (for IsCapCategoryObject, IsList)

▷  $\text{RandomMorphismWithFixedSourceByList}(a, L)$  (operation)

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

The arguments are an object  $a$  in a category  $C$  and a list  $L$ . The output is a random morphism  $\alpha : a \rightarrow b$  for some object  $b$  in  $C$ . If  $C$  is equipped with the methods  $\text{RandomObjectByList}$  and  $\text{RandomMorphismWithFixedSourceAndRangeByList}$  and  $C$  is an Ab-category, then  $\text{RandomMorphismWithFixedSourceByList}(C, a, L)$  can be derived as  $\text{RandomMorphismWithFixedSourceAndRangeByList}(C, a, b, L[2])$  where  $b$  is computed via  $\text{RandomObjectByList}(C, L[1])$ .

### 3.4.3 RandomMorphismWithFixedRangeByInteger (for IsCapCategoryObject, IsInt)

▷  $\text{RandomMorphismWithFixedRangeByInteger}(b, n)$  (operation)

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

The arguments are an object  $b$  in a category  $C$  and an integer  $n$ . The output is a random morphism  $\alpha : a \rightarrow b$  for some object  $a$  in  $C$ . If  $C$  is equipped with the methods  $\text{RandomObjectByInteger}$  and  $\text{RandomMorphismWithFixedSourceAndRangeByInteger}$  and  $C$  is an Ab-category, then  $\text{RandomMorphismWithFixedRangeByInteger}(C, b, n)$  can be derived as  $\text{RandomMorphismWithFixedSourceAndRangeByInteger}(C, a, b, 1 + \text{Log2Int}(n))$  where  $a$  is computed via  $\text{RandomObjectByInteger}(C, n)$ .

### 3.4.4 RandomMorphismWithFixedRangeByList (for IsCapCategoryObject, IsList)

▷  $\text{RandomMorphismWithFixedRangeByList}(b, L)$  (operation)

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

The arguments are an object  $b$  in a category  $C$  and a list  $L$ . The output is a random morphism  $\alpha : a \rightarrow b$  for some object  $a$  in  $C$ . If  $C$  is equipped with the methods  $\text{RandomObjectByList}$  and  $\text{RandomMorphismWithFixedSourceAndRangeByList}$  and  $C$  is an Ab-category, then  $\text{RandomMorphismWithFixedRangeByList}(C, b, L)$  can be derived as  $\text{RandomMorphismWithFixedSourceAndRangeByList}(C, a, b, L[2])$  where  $a$  is computed via  $\text{RandomObjectByList}(C, L[1])$ .

### 3.4.5 RandomMorphismWithFixedSourceAndRangeByInteger (for IsCapCategory-Object, IsCapCategoryObject, IsInt)

▷  $\text{RandomMorphismWithFixedSourceAndRangeByInteger}(a, b, n)$  (operation)

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

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

### 3.4.6 RandomMorphismWithFixedSourceAndRangeByList (for IsCapCategoryObject, IsCapCategoryObject, IsList)

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

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

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  $\alpha : a \rightarrow b$  in  $C$ .

### 3.4.7 RandomMorphismByInteger (for IsCapCategory, IsInt)

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

**Returns:** a morphism in  $C$

The arguments are a category  $C$  and an integer  $n$ . The output is a random morphism in  $C$ . The operation can be derived in three different ways:

- If  $C$  is equipped with the methods RandomObjectByInteger and RandomMorphismWithFixedSourceAndRangeByInteger and  $C$  is an Ab-category, then RandomMorphism( $C, n$ ) can be derived as RandomMorphismWithFixedSourceAndRangeByInteger( $C, a, b, 1 + \text{Log2Int}(n)$ ) where  $a$  and  $b$  are computed via RandomObjectByInteger( $C, n$ ).
- If  $C$  is equipped with the methods RandomObjectByInteger and RandomMorphismWithFixedSourceByInteger, then RandomMorphism( $C, n$ ) can be derived as RandomMorphismWithFixedSourceByInteger( $C, a, 1 + \text{Log2Int}(n)$ ) where  $a$  is computed via RandomObjectByInteger( $C, n$ ).
- If  $C$  is equipped with the methods RandomObjectByInteger and RandomMorphismWithFixedRangeByInteger, then RandomMorphism( $C, n$ ) can be derived as RandomMorphismWithFixedRangeByInteger( $C, b, 1 + \text{Log2Int}(n)$ ) where  $b$  is computed via RandomObjectByInteger( $C, n$ ).

### 3.4.8 RandomMorphismByList (for IsCapCategory, IsList)

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

**Returns:** a morphism in  $C$

The arguments are a category  $C$  and a list  $L$ . The output is a random morphism in  $C$ . The operation can be derived in three different ways:

- If  $C$  is equipped with the methods RandomObjectByList and RandomMorphismWithFixedSourceAndRangeByList and  $C$  is an Ab-category, then RandomMorphism( $C, L$ ) can be derived as RandomMorphismWithFixedSourceAndRangeByList( $C, a, b, L[3]$ ) where  $a$  and  $b$  are computed via RandomObjectByList( $C, L[i]$ ) for  $i = 1, 2$  respectively.
- If  $C$  is equipped with the methods RandomObjectByList and RandomMorphismWithFixedSourceByList, then RandomMorphism( $C, L$ ) can be derived as RandomMorphismWithFixedSourceByList( $C, a, L[2]$ ) where  $a$  is computed via RandomObjectByList( $C, L[1]$ ).

- If  $C$  is equipped with the methods `RandomObjectByList` and `RandomMorphismWithFixedRangeByList`, then `RandomMorphism( $C, L$ )` can be derived as `RandomMorphismWithFixedRangeByList( $C, b, L[2]$ )` where  $b$  is computed via `RandomObjectByList( $C, L[1]$ )`.

### 3.4.9 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( $a, b, n$ )` (operation)
- ▷ `RandomMorphism( $a, b, L$ )` (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.

## 3.5 Non-Categorical Properties of Morphisms

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

### 3.5.1 IsEqualToIdentityMorphism (for IsCapCategoryMorphism)

- ▷ `IsEqualToIdentityMorphism( $\alpha$ )` (property)

**Returns:** a boolean

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

### 3.5.2 IsEqualToZeroMorphism (for IsCapCategoryMorphism)

- ▷ `IsEqualToZeroMorphism( $\alpha$ )` (property)

**Returns:** a boolean

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

### 3.5.3 IsEndomorphism (for IsCapCategoryMorphism)

- ▷ `IsEndomorphism( $\alpha$ )` (property)

**Returns:** a boolean

The argument is a morphism  $\alpha$ . The output is true if  $\alpha$  is an endomorphism, otherwise the output is false.



### 3.5.4 IsAutomorphism (for IsCapCategoryMorphism)

▷ `IsAutomorphism(alpha)` (property)

**Returns:** a boolean

The argument is a morphism  $\alpha$ . The output is true if  $\alpha$  is an automorphism, otherwise the output is false.

## 3.6 Adding Morphisms to a Category

### 3.6.1 Add (for IsCapCategory, IsCapCategoryMorphism)

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

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

### 3.6.2 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` (3.6.1) can be used instead.

### 3.6.3 AddMorphismRepresentation (for IsCapCategory, IsObject)

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

*Deprecated:* use `CreateCapCategoryWithDataTypes` (1.3.4) instead. The argument *filter* is used to create a morphism type for the category *category*, which is then used in `ObjectifyMorphismWithSourceAndRangeForCAPWithAttributes` to objectify morphisms for this category. *filter* must imply `IsCapCategoryMorphism`.

### 3.6.4 ObjectifyMorphismWithSourceAndRangeForCAPWithAttributes

▷ `ObjectifyMorphismWithSourceAndRangeForCAPWithAttributes(morphism, category, source, range[, attr1, val1, attr2, val2, ...])` (function)

**Returns:** a morphism

*Deprecated:* use `CreateCapCategoryMorphismWithAttributes` (3.6.5) instead.

### 3.6.5 CreateCapCategoryMorphismWithAttributes

▷ `CreateCapCategoryMorphismWithAttributes(category, source, range[, attr1, val1, attr2, val2, ...])` (function)

**Returns:** a morphism

Creates a morphism in *category* with the given attributes.

### 3.6.6 AsCapCategoryMorphism

▷ `AsCapCategoryMorphism(category, source, value, range)` (function)

**Returns:** a morphism

EXPERIMENTAL: This specification might change any time without prior notice. Views *value* as a morphism from *source* to *range* in *category*.

### 3.6.7 AsPrimitiveValue (for IsCapCategoryMorphism)

▷ `AsPrimitiveValue(morphism)` (attribute)

▷ `AsInteger(morphism)` (attribute)

▷ `AsHomalgMatrix(morphism)` (attribute)

**Returns:** a value

EXPERIMENTAL: This specification might change any time without prior notice. Views a morphism obtained via `AsCapCategoryMorphism` (3.6.6) as a primitive value again. Here, the word *primitive* means *primitive from the perspective of the category*. For example, from the perspective of an opposite category, morphisms of the underlying category are primitive values. The attribute is chosen according to the morphism datum type:

- For `IsInt`, the attribute `AsInteger` is used.
- For `IsHomalgMatrix`, the attribute `AsHomalgMatrix` is used.

In all other cases or if no morphism datum type is given, the attribute `AsPrimitiveValue` is used.

## 3.7 Equality and Congruence for Morphisms

### 3.7.1 IsCongruentForMorphisms (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

**Returns:** a boolean

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

### 3.7.2 IsEqualForMorphisms (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

**Returns:** a boolean

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

### 3.7.3 IsEqualForMorphismsOnMor (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

**Returns:** a boolean

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

## 3.8 Basic Operations for Morphisms in Ab-Categories

### 3.8.1 IsZeroForMorphisms (for IsCapCategoryMorphism)

- ▷ `IsZeroForMorphisms(alpha)` (property)  
**Returns:** a boolean  
 The argument is a morphism  $\alpha : a \rightarrow b$ . The output is true if  $\alpha \sim_{a,b} 0$ , otherwise the output is false.

### 3.8.2 AdditionForMorphisms (for IsCapCategoryMorphism, IsCapCategoryMorphism)

- ▷ `AdditionForMorphisms(alpha, beta)` (operation)  
**Returns:** a morphism in  $\text{Hom}(a, b)$   
 The arguments are two morphisms  $\alpha, \beta : a \rightarrow b$ . The output is the addition  $\alpha + \beta$ . Note: The addition has to be compatible with the congruence of morphisms.

### 3.8.3 SubtractionForMorphisms (for IsCapCategoryMorphism, IsCapCategoryMorphism)

- ▷ `SubtractionForMorphisms(alpha, beta)` (operation)  
**Returns:** a morphism in  $\text{Hom}(a, b)$   
 The arguments are two morphisms  $\alpha, \beta : a \rightarrow b$ . The output is the addition  $\alpha - \beta$ . Note: The addition has to be compatible with the congruence of morphisms.

### 3.8.4 AdditiveInverseForMorphisms (for IsCapCategoryMorphism)

- ▷ `AdditiveInverseForMorphisms(alpha)` (attribute)  
**Returns:** a morphism in  $\text{Hom}(a, b)$   
 The argument is a morphism  $\alpha : a \rightarrow b$ . The output is its additive inverse  $-\alpha$ . Note: The addition has to be compatible with the congruence of morphisms.

### 3.8.5 MultiplyWithElementOfCommutativeRingForMorphisms (for IsRingElement, IsCapCategoryMorphism)

- ▷ `MultiplyWithElementOfCommutativeRingForMorphisms(r, alpha)` (operation)  
**Returns:** a morphism in  $\text{Hom}(a, b)$   
 The arguments are an element  $r$  of a commutative ring and a morphism  $\alpha : a \rightarrow b$ . The output is the multiplication with the ring element  $r \cdot \alpha$ . Note: The multiplication has to be compatible with the congruence of morphisms.

### 3.8.6 \* (for IsRingElement, IsCapCategoryMorphism)

- ▷ `*(r, alpha)` (operation)  
**Returns:** a morphism in  $\text{Hom}(a, b)$   
 This is a convenience method. It has two arguments. The first argument is either a rational number  $q$  or an element  $r$  of a commutative ring  $R$ . The second argument is a morphism  $\alpha : a \rightarrow b$  in a linear category over the commutative ring  $R$ . In the case where the first element is a rational number,

this method tries to interpret  $q$  as an element  $r$  of  $R$  via  $R!.interpret\_rationals\_func$ . If no such interpretation exists, this method throws an error. The output is the multiplication with the ring element  $r \cdot \alpha$ .

### 3.8.7 ZeroMorphism (for IsCapCategoryObject, IsCapCategoryObject)

▷ ZeroMorphism( $a$ ,  $b$ ) (operation)

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

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

## 3.9 Subobject and Factorobject Operations

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

### 3.9.1 IsEqualAsSubobjects (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

**Returns:** a boolean

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

### 3.9.2 IsEqualAsFactorobjects (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

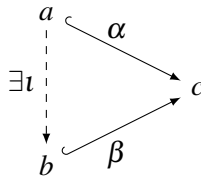
**Returns:** a boolean

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

### 3.9.3 IsDominating (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

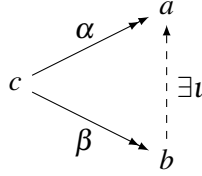
**Returns:** a boolean



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

### 3.9.4 IsCodomining (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `IsCodomining(alpha, beta)` (operation)  
**Returns:** a boolean



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

## 3.10 Identity Morphism and Composition of Morphisms

### 3.10.1 IdentityMorphism (for IsCapCategoryObject)

▷ `IdentityMorphism(a)` (attribute)  
**Returns:** a morphism in  $\text{Hom}(a, a)$   
 The argument is an object  $a$ . The output is its identity morphism  $\text{id}_a$ .

### 3.10.2 PreCompose (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `PreCompose(alpha, beta)` (operation)  
**Returns:** a morphism in  $\text{Hom}(a, c)$   
 The arguments are two morphisms  $\alpha : a \rightarrow b$ ,  $\beta : b \rightarrow c$ . The output is the composition  $\beta \circ \alpha : a \rightarrow c$ .

### 3.10.3 PreCompose (for IsList)

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

### 3.10.4 PreComposeList (for IsCapCategoryObject, IsList, IsCapCategoryObject)

▷ `PreComposeList(s, L, r)` (operation)  
**Returns:** a morphism in  $\text{Hom}(s, r)$   
 The arguments are two objects  $s = a_1$ ,  $r = a_{n+1}$ , and a list of morphisms  $L = (\alpha_1 : a_1 \rightarrow a_2, \alpha_2 : a_2 \rightarrow a_3, \dots, \alpha_n : a_n \rightarrow a_{n+1})$  in  $C$ . The output is the composition  $\alpha_n \circ (\alpha_{n-1} \circ (\dots (\alpha_2 \circ \alpha_1)))$ . If  $L$  is empty, then  $s$  must be equal to  $r$  and the output is congruent to the identity morphism of  $s$ .

### 3.10.5 PostCompose (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

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

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

### 3.10.6 PostCompose (for IsList)

▷ `PostCompose(L)` (operation)

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

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

### 3.10.7 PostComposeList (for IsCapCategoryObject, IsList, IsCapCategoryObject)

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

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

The arguments are two objects  $s = a_1$ ,  $r = a_{n+1}$ , and a list of morphisms  $L = (\alpha_n : a_n \rightarrow a_{n+1}, \alpha_{n-1} : a_{n-1} \rightarrow a_n, \dots, \alpha_1 : a_1 \rightarrow a_2)$ . The output is the composition  $((\alpha_n \circ \alpha_{n-1}) \circ \dots \alpha_2) \circ \alpha_1$ . If  $L$  is empty, then  $s$  must be equal to  $r$  and the output is congruent to the identity morphism of  $s$ .

### 3.10.8 SumOfMorphisms (for IsCapCategoryObject, IsList, IsCapCategoryObject)

▷ `SumOfMorphisms(s, morphisms, r)` (operation)

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

The arguments are two objects  $s, r$  and a list *morphisms* of morphisms from  $s$  to  $r$ . The output is the sum of all elements in *morphisms*, or the zero-morphism from  $s$  to  $r$  if *morphisms* is empty.

### 3.10.9 LinearCombinationOfMorphisms (for IsCapCategoryObject, IsList, IsList, IsCapCategoryObject)

▷ `LinearCombinationOfMorphisms(s, coeffs, mors, r)` (operation)

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

The arguments are two objects  $s, r$  in some linear category over a ring  $R$ , a list *coeffs* of ring elements in  $R$  and a list *mors* of morphisms from  $s$  to  $r$ . The output is the linear combination of the morphisms in *mors* with respect to the coefficients list *coeffs*, or the zero morphism from  $s$  to  $r$  if *coeffs* and *mors* are the empty lists.

## 3.11 Well-Definedness of Morphisms

### 3.11.1 IsWellDefinedForMorphisms (for IsCapCategoryMorphism)

▷ `IsWellDefinedForMorphisms(alpha)` (operation)

**Returns:** a boolean

The argument is a morphism  $\alpha$ . The output is `true` if  $\alpha$  is well-defined, otherwise the output is `false`.

### 3.11.2 IsWellDefinedForMorphismsWithGivenSourceAndRange (for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryObject)

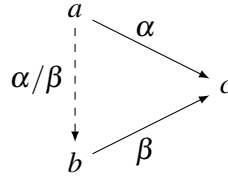
▷ `IsWellDefinedForMorphismsWithGivenSourceAndRange(source, alpha, range)` (operation)

**Returns:** a boolean

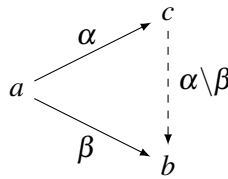
The arguments are two well-defined objects  $S$  and  $T$  and a morphism  $\alpha$ . The output is `true` if  $\alpha$  is a well-defined morphism from  $S$  to  $T$ , otherwise the output is `false`.

### 3.12 Lift/Colift

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



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



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.

#### 3.12.1 LiftAlongMonomorphism (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

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

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

#### 3.12.2 ColiftAlongEpimorphism (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

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

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

### 3.12.3 IsLiftableAlongMonomorphism (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

**Returns:** a boolean

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

### 3.12.4 IsColiftableAlongEpimorphism (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

**Returns:** a boolean

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

### 3.12.5 Lift (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `Lift(alpha, beta)` (operation)

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

The arguments are two morphisms  $\alpha : a \rightarrow c$ ,  $\beta : b \rightarrow c$  such that a lift  $\alpha/\beta : a \rightarrow b$  of  $\alpha$  along  $\beta$  exists. The output is such a lift  $\alpha/\beta : a \rightarrow b$ . Recall that a lift  $\alpha/\beta : a \rightarrow b$  of  $\alpha$  along  $\beta$  is a morphism such that  $\beta \circ (\alpha/\beta) \sim_{a,c} \alpha$ .

### 3.12.6 LiftOrFail (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

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

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

### 3.12.7 IsLiftable (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

**Returns:** a boolean

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

### 3.12.8 Colift (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `Colift(alpha, beta)` (operation)

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

The arguments are two morphisms  $\alpha : a \rightarrow c$ ,  $\beta : a \rightarrow b$  such that a colift  $\alpha \backslash \beta : c \rightarrow b$  of  $\beta$  along  $\alpha$  exists. The output is such a colift  $\alpha \backslash \beta : c \rightarrow b$ . Recall that a colift  $\alpha \backslash \beta : c \rightarrow b$  of  $\beta$  along  $\alpha$  is a morphism such that  $(\alpha \backslash \beta) \circ \alpha \sim_{a,b} \beta$ .



### 3.12.9 ColiftOrFail (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ ColiftOrFail(alpha, beta) (operation)

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

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

### 3.12.10 IsColiftable (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ IsColiftable(alpha, beta) (operation)

**Returns:** a boolean

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

## 3.13 Inverses

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

$$\begin{array}{ccc} \text{id}_a \circlearrowleft & a & \xrightarrow{\alpha} b \circlearrowright \text{id}_b \\ & & \alpha^{-1} \end{array}$$

### 3.13.1 InverseForMorphisms (for IsCapCategoryMorphism)

▷ InverseForMorphisms(alpha) (operation)

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

The argument is an isomorphism  $\alpha : a \rightarrow b$ . The output is its inverse  $\alpha^{-1} : b \rightarrow a$ .

### 3.13.2 PreInverseForMorphisms (for IsCapCategoryMorphism)

▷ PreInverseForMorphisms(alpha) (operation)

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

The argument is a split-epimorphism  $\alpha : a \rightarrow b$ . The output is a pre-inverse  $\iota : b \rightarrow a$  of  $\alpha$ , i.e.,  $\iota$  satisfies  $\alpha \circ \iota \sim_{b,b} \text{id}_b$ . The morphism  $\iota$  is also known as a section or a right-inverse of  $\alpha$ .

### 3.13.3 PostInverseForMorphisms (for IsCapCategoryMorphism)

▷ PostInverseForMorphisms(alpha) (operation)

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

The argument is a split-monomorphism  $\alpha : a \rightarrow b$ . The output is a post-inverse  $\pi : b \rightarrow a$  of  $\alpha$ , i.e.,  $\pi$  satisfies  $\pi \circ \alpha \sim_{a,a} \text{id}_a$ . The morphism  $\pi$  is also known as a contraction or a left-inverse of  $\alpha$ .

### 3.14 Tool functions for caches

#### 3.14.1 IsEqualForCacheForMorphisms (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

**Returns:** true or false

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, IsEqualForCacheForMorphisms is used. By default, IsEqualForCacheForMorphisms falls back to IsEqualForCache (see ToolsForHomalg), which in turn defaults to recursive comparison for lists and IsIdenticalObj in all other cases. If you add a function via AddIsEqualForCacheForMorphisms, that function is used instead. A function  $F : a, b \mapsto \text{bool}$  is expected there. The output has to be true or false. Fail is not allowed in this context.

### 3.15 IsHomSetInhabited

#### 3.15.1 IsHomSetInhabited (for IsCapCategoryObject, IsCapCategoryObject)

▷ IsHomSetInhabited(*A*, *B*) (operation)

**Returns:** a boolean

The arguments are two objects *A* and *B*. The output is true if there exists a morphism from *A* to *B*, otherwise the output is false.

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

#### 3.16.1 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  $H(a, b)$ .

#### 3.16.2 HomomorphismStructureOnMorphisms (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

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

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

### 3.16.3 HomomorphismStructureOnMorphismsWithGivenObjects (for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject)

▷ HomomorphismStructureOnMorphismsWithGivenObjects( $s, \alpha, \beta, r$ ) (operation)

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

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

### 3.16.4 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.

### 3.16.5 InterpretMorphismAsMorphismFromDistinguishedObjectToHomomorphismStructure (for IsCapCategoryMorphism)

▷ InterpretMorphismAsMorphismFromDistinguishedObjectToHomomorphismStructure( $\alpha$ ) (attribute)

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

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

### 3.16.6 InterpretMorphismAsMorphismFromDistinguishedObjectToHomomorphismStructureWithGivenObjects (for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryObject)

▷ InterpretMorphismAsMorphismFromDistinguishedObjectToHomomorphismStructureWithGivenObjects( $\alpha, r$ ) (operation)

**Returns:** a morphism in  $\text{Hom}_D(1, r)$

The arguments are the distinguished object 1, a morphism  $\alpha : a \rightarrow a'$ , and the object  $r = H(a, a')$ . The output is the corresponding morphism  $v(\alpha) : 1 \rightarrow r$  in  $D$  of the homomorphism structure.

### 3.16.7 InterpretMorphismFromDistinguishedObjectToHomomorphismStructureAsMorphism (for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryMorphism)

▷ InterpretMorphismFromDistinguishedObjectToHomomorphismStructureAsMorphism( $a, a', \iota$ ) (operation)

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

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

### 3.16.8 SolveLinearSystemInAbCategory (for IsList, IsList, IsList)

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

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

The arguments are three lists  $\alpha$ ,  $\beta$ , and  $\gamma$ . The first list  $\alpha$  (the left coefficients) is a list of list of morphisms  $\alpha_{ij} : A_i \rightarrow B_j$ , where  $i = 1 \dots m$  and  $j = 1 \dots n$  for integers  $m, n \geq 1$ . The second list  $\beta$  (the right coefficients) is a list of list of morphisms  $\beta_{ij} : C_j \rightarrow D_i$ , where  $i = 1 \dots m$  and  $j = 1 \dots n$ . The third list  $\gamma$  (the right side) is a list of morphisms  $\gamma_i : A_i \rightarrow D_i$ , where  $i = 1, \dots, m$ . Assumes that a solution to the linear system defined by  $\alpha$ ,  $\beta$ ,  $\gamma$  exists, i.e., there exist morphisms  $X_j : B_j \rightarrow C_j$  for  $j = 1 \dots n$  such that  $\sum_{j=1}^n \alpha_{ij} \cdot X_j \cdot \beta_{ij} = \gamma_i$  for all  $i = 1 \dots m$ . The output is list of such morphisms  $X_j : B_j \rightarrow C_j$  for  $j = 1 \dots n$ .

### 3.16.9 SolveLinearSystemInAbCategoryOrFail (for IsList, IsList, IsList)

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

**Returns:** a list of morphisms  $[X_1, \dots, X_n]$  or fail

Like `SolveLinearSystemInAbCategory`, but without the assumption that a solution exists. If no solution exists, fail is returned.

### 3.16.10 MereExistenceOfSolutionOfLinearSystemInAbCategory (for IsList, IsList, IsList)

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

**Returns:** a boolean

Like `SolveLinearSystemInAbCategory`, but the output is simply true if a solution exists, false otherwise.

### 3.16.11 HomStructure (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

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

This is a convenience method. The arguments are two morphisms  $\alpha : a \rightarrow a', \beta : b \rightarrow b'$  in  $C$ . The output is `HomomorphismStructureOnMorphisms` called on  $\alpha, \beta$ .

### 3.16.12 HomStructure (for IsCapCategoryMorphism, IsCapCategoryObject)

▷ `HomStructure(alpha, b)` (operation)

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

This is a convenience method. The arguments are a morphism  $\alpha : a \rightarrow a'$  and an object  $b$  in  $C$ . The output is `HomomorphismStructureOnMorphisms` called on  $\alpha, \text{id}_b$ .

### 3.16.13 HomStructure (for IsCapCategoryObject, IsCapCategoryMorphism)

▷ `HomStructure(a, beta)` (operation)

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

This is a convenience method. The arguments are an object  $a$  and a morphism  $\beta : b \rightarrow b'$  in  $C$ . The output is `HomomorphismStructureOnMorphisms` called on  $\text{id}_a, \beta$ .

### 3.16.14 HomStructure (for IsCapCategoryObject, IsCapCategoryObject)

▷ HomStructure(*a*, *b*) (operation)

**Returns:** an object

This is a convenience method. The arguments are two objects *a* and *b* in *C*. The output is HomomorphismStructureOnObjects called on *a, b*.

### 3.16.15 HomStructure (for IsCapCategoryMorphism)

▷ HomStructure(*arg*) (operation)

This is a convenience method for InterpretMorphismAsMorphismFromDistinguishedObjectToHomomorphismStructure.

### 3.16.16 HomStructure (for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryMorphism)

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

This is a convenience method for InterpretMorphismFromDistinguishedObjectToHomomorphismStructure.

### 3.16.17 HomStructure (for IsCapCategory)

▷ HomStructure(*arg*) (operation)

This is a convenience method for DistinguishedObjectOfHomomorphismStructure.

### 3.16.18 ExtendRangeOfHomomorphismStructureByFullEmbedding (for IsCapCategory, IsCapCategory, IsFunction, IsFunction, IsFunction, IsFunction)

▷ ExtendRangeOfHomomorphismStructureByFullEmbedding(*C*, *E*, *object\_function*, *morphism\_function*, *object\_function\_inverse*, *morphism\_function\_inverse*) (operation)

▷ HomomorphismStructureOnObjectsExtendedByFullEmbedding(*C*, *E*, *a*, *b*) (operation)

▷ HomomorphismStructureOnMorphismsExtendedByFullEmbedding(*C*, *E*, *alpha*, *beta*) (operation)

▷ HomomorphismStructureOnMorphismsWithGivenObjectsExtendedByFullEmbedding(*C*, *E*, *s*, *alpha*, *beta*, *r*) (operation)

▷ DistinguishedObjectOfHomomorphismStructureExtendedByFullEmbedding(*C*, *E*) (operation)

▷ InterpretMorphismAsMorphismFromDistinguishedObjectToHomomorphismStructureExtendedByFullEmbedding(*E*, *alpha*) (operation)

▷ InterpretMorphismAsMorphismFromDistinguishedObjectToHomomorphismStructureWithGivenObjectsExtendedByFullEmbedding(*E*, *distinguished\_object*, *alpha*, *r*) (operation)

▷ InterpretMorphismFromDistinguishedObjectToHomomorphismStructureAsMorphismExtendedByFullEmbedding(*E*, *a*, *a'*, *iota*) (operation)

**Returns:** nothing

If  $\iota: D \rightarrow E$  is a full embedding of categories, every  $D$ -homomorphism structure for a category  $C$  extends to a  $E$ -homomorphism structure for  $C$ . This operation accepts four functions and installs operations `DistinguishedObjectOfHomomorphismStructureExtendedByFullEmbedding`, `HomomorphismStructureOnObjectsExtendedByFullEmbedding` etc. which correspond to the  $E$ -homomorphism structure for  $C$ . Note: To distinguish embeddings in different categories, in addition to  $C$  also  $E$  is passed to the operations. When using this with different embeddings with the range category  $E$ , only the last embedding will be used. The arguments are:

- `object_function` gets the categories  $C$  and  $E$  and an object in  $D$ .
- `morphism_function` gets the categories  $C$  and  $E$ , an object in  $E$ , a morphism in  $D$  and another object in  $E$ . The objects are the results of `object_function` applied to the source and range of the morphism.
- `object_function_inverse` gets the categories  $C$  and  $E$  and an object in  $E$ .
- `morphism_function_inverse` gets the categories  $C$  and  $E$ , an object in  $D$ , a morphism in  $E$  and another object in  $D$ . The objects are the results of `object_function_inverse` applied to the source and range of the morphism.

`object_function` and `morphism_function` define the embedding. `object_function_inverse` and `morphism_function_inverse` define the inverse of the embedding on its image.

### 3.16.19 `ExtendRangeOfHomomorphismStructureByIdentityAsFullEmbedding` (for `IsCapCategory`)

▷ `ExtendRangeOfHomomorphismStructureByIdentityAsFullEmbedding(C)` (operation)

**Returns:** nothing

Chooses the identity on  $D$  as the full embedding in `ExtendRangeOfHomomorphismStructureByFullEmbedding` (3.16.18). This is useful to handle this case as a degenerate case of `ExtendRangeOfHomomorphismStructureByFullEmbedding` (3.16.18).

### 3.16.20 `MorphismsOfExternalHom` (for `IsCapCategoryObject`, `IsCapCategoryObject`)

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

**Returns:** a list of morphisms in  $\text{Hom}(a, b)$

The arguments are two objects  $a, b$ . The output is a list of all morphisms from  $a$  to  $b$ .

### 3.16.21 `BasisOfExternalHom` (for `IsCapCategoryObject`, `IsCapCategoryObject`)

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

**Returns:** a list of morphisms in  $\text{Hom}_C(a, b)$

The arguments are objects  $a, b$  in a  $k$ -linear category  $C$ . The output is a list  $L$  of morphisms which is a basis of  $\text{Hom}_C(a, b)$  in the sense that any given morphism  $\alpha: a \rightarrow b$  can uniquely be written as a linear combination of  $L$  with the coefficients in `CoefficientsOfMorphism( $\alpha$ )`.

### 3.16.22 CoefficientsOfMorphism (for IsCapCategoryMorphism)

▷ `CoefficientsOfMorphism(alpha)`

(attribute)

**Returns:** a list of elements in  $k$

This is a convenience method. The argument is a morphism  $\alpha : a \rightarrow b$  in a  $k$ -linear category  $C$ . The output is a list of coefficients of  $\alpha$  with respect to the list `BasisOfExternalHom(a,b)`.

## 3.17 Simplified Morphisms

Let  $\phi : A \rightarrow B$  be a morphism. There are several different natural ways to look at  $\phi$  as an object in an ambient category:

- $\text{Hom}(A, B)$ , the set of homomorphisms with the equivalence relation `IsCongruentForMorphisms` regarded as a category,
- $\sum_A \text{Hom}(A, B)$ , the category of morphisms where the range is fixed,
- $\sum_B \text{Hom}(A, B)$ , the category of morphisms where the source is fixed,
- $\sum_{A, B} \text{Hom}(A, B)$ , the category of morphisms where neither source nor range is fixed,

and furthermore, if  $\phi$  happens to be an endomorphism  $A \rightarrow A$ , we also have

- $\sum_A \text{Hom}(A, A)$ , the category of endomorphisms.

Let  $\mathbf{C}$  be one of the categories above in which  $\phi$  may reside as an object, and let  $i$  be a non-negative integer or  $\infty$ . CAP provides commands for passing from  $\phi$  to  $\phi_i$ , where  $\phi_i$  is isomorphic to  $\phi$  in  $\mathbf{C}$ , but "simpler". The idea is that the greater the  $i$ , the "simpler" the  $\phi_i$  (but this could mean the harder the computation), with  $\infty$  as a possible value. The case  $i = 0$  defaults to the identity operator for all simplifications. For the Add-operations, only the cases  $i \geq 1$  have to be given as functions.

If we regard  $\phi$  as an object in the category  $\text{Hom}(A, B)$ ,  $\phi_i$  is again in  $\text{Hom}(A, B)$  such that  $\phi \sim_{A, B} \phi_i$ . This case is handled by the following commands:

### 3.17.1 SimplifyMorphism (for IsCapCategoryMorphism, IsObject)

▷ `SimplifyMorphism(phi, i)`

(operation)

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

The arguments are a morphism  $\phi : A \rightarrow B$  and a non-negative integer  $i$  or infinity. The output is a simplified morphism  $\phi_i$ .

If we regard  $\phi$  as an object in the category  $\sum_A \text{Hom}(A, B)$ , then  $\phi_i$  is a morphism of type  $A_i \rightarrow B$  and there is an isomorphism  $\sigma_i : A \rightarrow A_i$  such that  $\phi_i \circ \sigma_i \sim_{A, B} \phi$ . This case is handled by the following commands:

### 3.17.2 SimplifySource (for IsCapCategoryMorphism, IsObject)

▷ `SimplifySource(phi, i)`

(operation)

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

The arguments are a morphism  $\phi : A \rightarrow B$  and a non-negative integer  $i$  or infinity. The output is a simplified morphism with simplified source  $\phi_i : A_i \rightarrow B$ .

### 3.17.3 SimplifySource\_IsoToInputObject (for IsCapCategoryMorphism, IsObject)

▷ `SimplifySource_IsoToInputObject(phi, i)` (operation)

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

The arguments are a morphism  $\phi : A \rightarrow B$  and a non-negative integer  $i$  or infinity. The output is the isomorphism  $(\sigma_i)^{-1} : A_i \rightarrow A$ .

### 3.17.4 SimplifySource\_IsoFromInputObject (for IsCapCategoryMorphism, IsObject)

▷ `SimplifySource_IsoFromInputObject(phi, i)` (operation)

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

The arguments are a morphism  $\phi : A \rightarrow B$  and a non-negative integer  $i$  or infinity. The output is the isomorphism  $\sigma_i : A \rightarrow A_i$ .

If we regard  $\phi$  as an object in the category  $\sum_B \text{Hom}(A, B)$ , then  $\phi_i$  is a morphism of type  $A \rightarrow B_i$  and there is an isomorphism  $\rho_i : B \rightarrow B_i$  such that  $\rho_i^{-1} \circ \phi_i \sim_{A,B} \phi$ . This case is handled by the following commands:

### 3.17.5 SimplifyRange (for IsCapCategoryMorphism, IsObject)

▷ `SimplifyRange(phi, i)` (operation)

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

The arguments are a morphism  $\phi : A \rightarrow B$  and a non-negative integer  $i$  or infinity. The output is a simplified morphism with simplified range  $\phi_i : A \rightarrow B_i$ .

### 3.17.6 SimplifyRange\_IsoToInputObject (for IsCapCategoryMorphism, IsObject)

▷ `SimplifyRange_IsoToInputObject(phi, i)` (operation)

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

The arguments are a morphism  $\phi : A \rightarrow B$  and a non-negative integer  $i$  or infinity. The output is the isomorphism  $(\rho_i)^{-1} : B_i \rightarrow B$ .

### 3.17.7 SimplifyRange\_IsoFromInputObject (for IsCapCategoryMorphism, IsObject)

▷ `SimplifyRange_IsoFromInputObject(phi, i)` (operation)

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

The arguments are a morphism  $\phi : A \rightarrow B$  and a non-negative integer  $i$  or infinity. The output is the isomorphism  $\rho_i : B \rightarrow B_i$ .

If we regard  $\phi$  as an object in the category  $\sum_{A,B} \text{Hom}(A, B)$ , then  $\phi_i$  is a morphism of type  $A_i \rightarrow B_i$  and there are isomorphisms  $\sigma_i : A \rightarrow A_i$  and  $\rho_i : B \rightarrow B_i$  such that  $\rho_i^{-1} \circ \phi_i \circ \sigma_i \sim_{A,B} \phi$ . This case is handled by the following commands:

### 3.17.8 SimplifySourceAndRange (for IsCapCategoryMorphism, IsObject)

▷ `SimplifySourceAndRange(phi, i)` (operation)

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



The arguments are a morphism  $\phi : A \rightarrow B$  and a non-negative integer  $i$  or infinity. The output is a simplified morphism with simplified source and range  $\phi_i : A_i \rightarrow B_i$ .

### 3.17.9 SimplifySourceAndRange\_IsoToInputRange (for IsCapCategoryMorphism, IsObject)

▷ `SimplifySourceAndRange_IsoToInputRange(phi, i)` (operation)

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

The arguments are a morphism  $\phi : A \rightarrow B$  and a non-negative integer  $i$  or infinity. The output is the isomorphism  $(\rho_i)^{-1} : B_i \rightarrow B$ .

### 3.17.10 SimplifySourceAndRange\_IsoFromInputRange (for IsCapCategoryMorphism, IsObject)

▷ `SimplifySourceAndRange_IsoFromInputRange(phi, i)` (operation)

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

The arguments are a morphism  $\phi : A \rightarrow B$  and a non-negative integer  $i$  or infinity. The output is the isomorphism  $\rho_i : B \rightarrow B_i$ .

### 3.17.11 SimplifySourceAndRange\_IsoToInputSource (for IsCapCategoryMorphism, IsObject)

▷ `SimplifySourceAndRange_IsoToInputSource(phi, i)` (operation)

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

The arguments are a morphism  $\phi : A \rightarrow B$  and a non-negative integer  $i$  or infinity. The output is the isomorphism  $(\sigma_i)^{-1} : A_i \rightarrow A$ .

### 3.17.12 SimplifySourceAndRange\_IsoFromInputSource (for IsCapCategoryMorphism, IsObject)

▷ `SimplifySourceAndRange_IsoFromInputSource(phi, i)` (operation)

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

The arguments are a morphism  $\phi : A \rightarrow B$  and a non-negative integer  $i$  or infinity. The output is the isomorphism  $\sigma_i : A \rightarrow A_i$ .

If  $\phi : A \rightarrow A$  is an endomorphism, we may regard it as an object in the category  $\sum_A \text{Hom}(A, A)$ . In this case  $\phi_i$  is a morphism of type  $A_i \rightarrow A_i$  and there is an isomorphism  $\sigma_i : A \rightarrow A_i$  such that  $\sigma_i^{-1} \circ \phi_i \circ \sigma_i \sim_{A,A} \phi$ . This case is handled by the following commands:

### 3.17.13 SimplifyEndo (for IsCapCategoryMorphism, IsObject)

▷ `SimplifyEndo(phi, i)` (operation)

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

The arguments are an endomorphism  $\phi : A \rightarrow A$  and a non-negative integer  $i$  or infinity. The output is a simplified endomorphism  $\phi_i : A_i \rightarrow A_i$ .

### 3.17.14 SimplifyEndo\_IsoToInputObject (for IsCapCategoryMorphism, IsObject)

▷ `SimplifyEndo_IsoToInputObject(phi, i)` (operation)

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

The arguments are an endomorphism  $\phi : A \rightarrow A$  and a non-negative integer  $i$  or infinity. The output is the isomorphism  $(\sigma_i)^{-1} : A_i \rightarrow A$ .

### 3.17.15 SimplifyEndo\_IsoFromInputObject (for IsCapCategoryMorphism, IsObject)

▷ `SimplifyEndo_IsoFromInputObject(phi, i)` (operation)

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

The arguments are an endomorphism  $\phi : A \rightarrow A$  and a non-negative integer  $i$  or infinity. The output is the isomorphism  $\sigma_i : A \rightarrow A_i$ .

### 3.17.16 Simplify (for IsCapCategoryMorphism)

▷ `Simplify(phi)` (attribute)

**Returns:** a morphism in  $\text{Hom}(A_\infty, B_\infty)$

This is a convenient method. The argument is a morphism  $\phi : A \rightarrow B$ . The output is a "simplified" version of  $\phi$  that may change the source and range of  $\phi$  (up to isomorphism). To be precise, the output is an  $\infty$ -th simplified morphism of  $(\iota_A^\infty)^{-1} \circ \phi \circ \iota_A^\infty$ .

## 3.18 Reduction by split epi summands

Let  $\alpha : A \rightarrow B$  be a morphism in an additive category. Suppose we are given direct sum decompositions of  $A \simeq A' \oplus A''$  and  $B \simeq B' \oplus B''$  such that

$$\begin{array}{ccc} A' \oplus A'' & \xrightarrow{\alpha' \oplus \alpha''} & B' \oplus B'' \\ \uparrow & & \uparrow \\ A & \xrightarrow{\alpha} & B \end{array}$$

If  $\alpha''$  is a split epimorphism, then we call  $\alpha' : A' \rightarrow B'$  *some reduction of  $\alpha$  by split epi summands*. The inclusions/projections of the decompositions into direct sums induce commutative diagrams

$$\begin{array}{ccc} A' & \xrightarrow{\alpha'} & B' \\ \uparrow & & \uparrow \beta \\ A & \xrightarrow{\alpha} & B \end{array}$$

and

$$\begin{array}{ccc}
 A' & \xrightarrow{\alpha'} & B' \\
 \downarrow & & \downarrow \beta' \\
 A & \xrightarrow{\alpha} & B
 \end{array}$$

### 3.18.1 SomeReductionBySplitEpiSummand (for IsCapCategoryMorphism)

▷ `SomeReductionBySplitEpiSummand(alpha)` (attribute)

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

The argument is a morphism  $\alpha : A \rightarrow B$ . The output is some reduction of  $\alpha$  by split epi summands  $\alpha' : A' \rightarrow B'$ .

### 3.18.2 SomeReductionBySplitEpiSummand\_MorphismToInputRange (for IsCapCategoryMorphism)

▷ `SomeReductionBySplitEpiSummand_MorphismToInputRange(alpha)` (attribute)

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

The argument is a morphism  $\alpha : A \rightarrow B$ . The output is the morphism  $\beta' : B' \rightarrow B$  linking  $\alpha$  with some reduction by split epi summands.

### 3.18.3 SomeReductionBySplitEpiSummand\_MorphismFromInputRange (for IsCapCategoryMorphism)

▷ `SomeReductionBySplitEpiSummand_MorphismFromInputRange(alpha)` (attribute)

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

The argument is a morphism  $\alpha : A \rightarrow B$ . The output is the morphism  $\beta : B \rightarrow B'$  linking  $\alpha$  with some reduction by split epi summands.

# 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  $c : \alpha \rightarrow \beta$ . The output is its source  $\alpha$ .

#### 4.1.2 Range (for IsCapCategoryTwoCell)

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

#### 4.1.3 Target (for IsCapCategoryTwoCell)

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

### 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  $\alpha$ . The output is its identity 2-cell  $\text{id}_\alpha : \alpha \rightarrow \alpha$ .

#### 4.2.2 HorizontalPreCompose (for IsCapCategoryTwoCell, IsCapCategoryTwoCell)

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

### 4.2.3 HorizontalPostCompose (for IsCapCategoryTwoCell, IsCapCategoryTwoCell)

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

**Returns:** a 2-cell

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

### 4.2.4 VerticalPreCompose (for IsCapCategoryTwoCell, IsCapCategoryTwoCell)

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

**Returns:** a 2-cell

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

### 4.2.5 VerticalPostCompose (for IsCapCategoryTwoCell, IsCapCategoryTwoCell)

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

**Returns:** a 2-cell

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

## 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`.

## Chapter 5

# Category of Categories

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

### 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  $\text{Cat}$ . 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  $\text{Cat}$ . 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  $\text{Cat}$ , 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  $\text{Cat}$ , 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  $A \times B \rightarrow C$  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  $[[A_1, b_1], \dots, [A_n, b_n]]$  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  $i = 1, \dots, n$ . Instead of a pair  $[A_i, b_i]$ , you can also give simply  $A_i$ , which will be interpreted as the pair  $[A_i, \text{false}]$ . 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  $D_1 \times \dots \times D_n$ , where  $D_i = A_i$  if  $b_i = \text{false}$ , and  $D_i = A_i^{\text{op}}$  otherwise.

### 5.4.3 SourceOfFunctor (for IsCapFunctor)

▷ `SourceOfFunctor(F)` (attribute)

The argument is a functor  $F$ . The output is its source as CAP category.

### 5.4.4 RangeOfFunctor (for IsCapFunctor)

▷ `RangeOfFunctor(F)` (attribute)

The argument is a functor  $F$ . The output is its range as CAP category.

### 5.4.5 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  $[[A_1, b_1], \dots, [A_n, b_n]]$ , 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.

### 5.4.6 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.

### 5.4.7 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  $[[A_1, b_1], \dots, [A_n, b_n]]$ , 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.

#### 5.4.8 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.

#### 5.4.9 ApplyFunctor

▷ `ApplyFunctor( $func$ ,  $A$  [,  $B$ , ...])` (function)

**Returns:** `IsCapCategoryCell`

Applies the functor  $func$  either to

- an object or morphism  $A$  in the source of  $func$  or
- to objects or morphisms belonging to the categories in the input signature of  $func$ .

#### 5.4.10 InputSignature (for IsCapFunctor)

▷ `InputSignature( $F$ )` (attribute)

**Returns:** `IsList`

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

#### 5.4.11 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  $[[A_1, b_1], \dots, [A_n, b_n]]$  denote the input signature of  $F$ . This method tries to install 3 operations: an operation  $\omega_1$  with the name  $s$ , an operation  $\omega_2$  with the name  $s\text{OnObjects}$ , and an operation  $\omega_3$  with the name  $s\text{OnMorphisms}$ . The operation  $\omega_1$  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.  $\omega_2$  and  $\omega_3$  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.

### 5.4.12 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.

### 5.4.13 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.

### 5.4.14 NaturalIsomorphismFromIdentityToCanonicalizeZeroObjects (for IsCapCategory)

▷ NaturalIsomorphismFromIdentityToCanonicalizeZeroObjects(*cat*) (attribute)

**Returns:** a natural transformation

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

### 5.4.15 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  $\phi$  to itself, unless it is congruent to the zero morphism; in this case it is mapped to ZeroMorphism(Source( $\phi$ ), Range( $\phi$ )). This functor is equivalent to the identity functor.

### 5.4.16 NaturalIsomorphismFromIdentityToCanonicalizeZeroMorphisms (for IsCapCategory)

▷ NaturalIsomorphismFromIdentityToCanonicalizeZeroMorphisms(*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  $F: A \rightarrow B$  and  $G: A \rightarrow B$ . 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  $N: F \rightarrow G$ , the arguments should be  $F(A), A, G(A)$ . The output should be a morphism,  $F(A) \rightarrow G(A)$ .

### 5.5.4 ApplyNaturalTransformation

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

**Returns:** a morphism

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

### 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

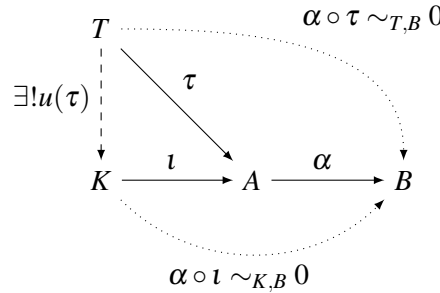
### 6.1 Kernel

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

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

The triple  $(K, \iota, u)$  is called a *kernel* of  $\alpha$  if the morphisms  $u(\tau)$  are uniquely determined up to congruence of morphisms. We denote the object  $K$  of such a triple by  $\text{KernelObject}(\alpha)$ . We say that the morphism  $u(\tau)$  is induced by the *universal property of the kernel*.

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



#### 6.1.1 KernelObject (for IsCapCategoryMorphism)

▷ `KernelObject(alpha)`

(attribute)

**Returns:** an object

The argument is a morphism  $\alpha$ . The output is the kernel  $K$  of  $\alpha$ .

### 6.1.2 KernelEmbedding (for IsCapCategoryMorphism)

▷ `KernelEmbedding(alpha)` (attribute)

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

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

### 6.1.3 KernelEmbeddingWithGivenKernelObject (for IsCapCategoryMorphism, IsCapCategoryObject)

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

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

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

### 6.1.4 MorphismFromKernelObjectToSink (for IsCapCategoryMorphism)

▷ `MorphismFromKernelObjectToSink(alpha)` (operation)

**Returns:** the zero morphism in  $\text{Hom}(\text{KernelObject}(\alpha), B)$

The argument is a morphism  $\alpha : A \rightarrow B$ . The output is the zero morphism  $0 : \text{KernelObject}(\alpha) \rightarrow B$ .

### 6.1.5 MorphismFromKernelObjectToSinkWithGivenKernelObject (for IsCapCategoryMorphism, IsCapCategoryObject)

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

**Returns:** the zero morphism in  $\text{Hom}(K, B)$

The arguments are a morphism  $\alpha : A \rightarrow B$  and an object  $K = \text{KernelObject}(\alpha)$ . The output is the zero morphism  $0 : K \rightarrow B$ .

### 6.1.6 KernelLift (for IsCapCategoryMorphism, IsCapCategoryObject, IsCapCategoryMorphism)

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

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

The arguments are a morphism  $\alpha : A \rightarrow B$ , a test object  $T$ , and a test morphism  $\tau : T \rightarrow A$  satisfying  $\alpha \circ \tau \sim_{T,B} 0$ . For convenience, the test object  $T$  can be omitted and is automatically derived from  $\tau$  in that case. The output is the morphism  $u(\tau) : T \rightarrow \text{KernelObject}(\alpha)$  given by the universal property of the kernel.

### 6.1.7 KernelLiftWithGivenKernelObject (for IsCapCategoryMorphism, IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryObject)

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

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

The arguments are a morphism  $\alpha : A \rightarrow B$ , a test object  $T$ , a test morphism  $\tau : T \rightarrow A$  satisfying  $\alpha \circ \tau \sim_{T,B} 0$ , and an object  $K = \text{KernelObject}(\alpha)$ . For convenience, the test object  $T$  can be omitted

and is automatically derived from `tau` in that case. The output is the morphism  $u(\tau) : T \rightarrow K$  given by the universal property of the kernel.

### 6.1.8 KernelObjectFunctorial (for IsList)

▷ `KernelObjectFunctorial(L)` (operation)

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

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

### 6.1.9 KernelObjectFunctorial (for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism)

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

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

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

### 6.1.10 KernelObjectFunctorialWithGivenKernelObjects (for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject)

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

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

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

### 6.1.11 KernelObjectFunctorialWithGivenKernelObjects (for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject)

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

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

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

## 6.2 Cokernel

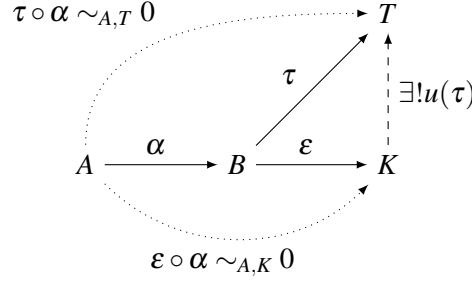
For a given morphism  $\alpha : A \rightarrow B$ , a cokernel of  $\alpha$  consists of three parts:

- an object  $K$ ,
- a morphism  $\varepsilon : B \rightarrow K$  such that  $\varepsilon \circ \alpha \sim_{A,K} 0$ ,

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

The triple  $(K, \varepsilon, u)$  is called a *cokernel* of  $\alpha$  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*.

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



### 6.2.1 CokernelObject (for IsCapCategoryMorphism)

▷ `CokernelObject(alpha)`

(attribute)

**Returns:** an object

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

### 6.2.2 CokernelProjection (for IsCapCategoryMorphism)

▷ `CokernelProjection(alpha)`

(attribute)

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

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

### 6.2.3 CokernelProjectionWithGivenCokernelObject (for IsCapCategoryMorphism, IsCapCategoryObject)

▷ `CokernelProjectionWithGivenCokernelObject(alpha, K)`

(operation)

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

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

### 6.2.4 MorphismFromSourceToCokernelObject (for IsCapCategoryMorphism)

▷ `MorphismFromSourceToCokernelObject(alpha)`

(operation)

**Returns:** the zero morphism in  $\text{Hom}(A, \text{CokernelObject}(\alpha))$ .

The argument is a morphism  $\alpha : A \rightarrow B$ . The output is the zero morphism  $0 : A \rightarrow \text{CokernelObject}(\alpha)$ .

### 6.2.5 MorphismFromSourceToCokernelObjectWithGivenCokernelObject (for IsCapCategoryMorphism, IsCapCategoryObject)

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

**Returns:** the zero morphism in  $\text{Hom}(A, K)$ .

The argument is a morphism  $\alpha : A \rightarrow B$  and an object  $K = \text{CokernelObject}(\alpha)$ . The output is the zero morphism  $0 : A \rightarrow K$ .

### 6.2.6 CokernelColift (for IsCapCategoryMorphism, IsCapCategoryObject, IsCapCategoryMorphism)

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

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

The arguments are a morphism  $\alpha : A \rightarrow B$ , a test object  $T$ , and a test morphism  $\tau : B \rightarrow T$  satisfying  $\tau \circ \alpha \sim_{A, T} 0$ . For convenience, the test object  $T$  can be omitted and is automatically derived from  $\tau$  in that case. The output is the morphism  $u(\tau) : \text{CokernelObject}(\alpha) \rightarrow T$  given by the universal property of the cokernel.

### 6.2.7 CokernelColiftWithGivenCokernelObject (for IsCapCategoryMorphism, IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryObject)

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

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

The arguments are a morphism  $\alpha : A \rightarrow B$ , a test object  $T$ , a test morphism  $\tau : B \rightarrow T$  satisfying  $\tau \circ \alpha \sim_{A, T} 0$ , and an object  $K = \text{CokernelObject}(\alpha)$ . For convenience, the test object  $T$  can be omitted and is automatically derived from  $\tau$  in that case. The output is the morphism  $u(\tau) : K \rightarrow T$  given by the universal property of the cokernel.

### 6.2.8 CokernelObjectFunctorial (for IsList)

▷ `CokernelObjectFunctorial(L)` (operation)

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

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

### 6.2.9 CokernelObjectFunctorial (for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism)

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

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

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



### 6.2.10 CokernelObjectFunctorialWithGivenCokernelObjects (for IsCap-CategoryObject, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject)

▷ CokernelObjectFunctorialWithGivenCokernelObjects( $s$ ,  $\alpha$ ,  $\nu$ ,  $\alpha'$ ,  $r$ ) (operation)

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

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

### 6.2.11 CokernelObjectFunctorialWithGivenCokernelObjects (for IsCap-CategoryObject, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject)

▷ CokernelObjectFunctorialWithGivenCokernelObjects( $s$ ,  $\alpha$ ,  $\mu$ ,  $\nu$ ,  $\alpha'$ ,  $r$ ) (operation)

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

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

## 6.3 Zero Object

A zero object consists of three parts:

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

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

### 6.3.1 ZeroObject (for IsCapCategory)

▷ 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 UniversalMorphismFromZeroObject (for IsCapCategoryObject)

▷ `UniversalMorphismFromZeroObject(A)` (attribute)

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

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

### 6.3.4 UniversalMorphismFromZeroObjectWithGivenZeroObject (for IsCapCategoryObject, IsCapCategoryObject)

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

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

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

### 6.3.5 UniversalMorphismIntoZeroObject (for IsCapCategoryObject)

▷ `UniversalMorphismIntoZeroObject(A)` (attribute)

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

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

### 6.3.6 UniversalMorphismIntoZeroObjectWithGivenZeroObject (for IsCapCategoryObject, IsCapCategoryObject)

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

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

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

### 6.3.7 MorphismFromZeroObject (for IsCapCategoryObject)

▷ `MorphismFromZeroObject(A)` (attribute)

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

This is a synonym for `UniversalMorphismFromZeroObject`.

### 6.3.8 MorphismIntoZeroObject (for IsCapCategoryObject)

▷ `MorphismIntoZeroObject(A)` (attribute)

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

This is a synonym for `UniversalMorphismIntoZeroObject`.

### 6.3.9 IsomorphismFromZeroObjectToInitialObject (for IsCapCategory)

▷ `IsomorphismFromZeroObjectToInitialObject(C)` (attribute)

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

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

### 6.3.10 IsomorphismFromInitialObjectToZeroObject (for IsCapCategory)

▷ `IsomorphismFromInitialObjectToZeroObject(C)` (attribute)

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

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

### 6.3.11 IsomorphismFromZeroObjectToTerminalObject (for IsCapCategory)

▷ `IsomorphismFromZeroObjectToTerminalObject(C)` (attribute)

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

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

### 6.3.12 IsomorphismFromTerminalObjectToZeroObject (for IsCapCategory)

▷ `IsomorphismFromTerminalObjectToZeroObject(C)` (attribute)

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

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

### 6.3.13 ZeroObjectFunctorial (for IsCapCategory)

▷ `ZeroObjectFunctorial(C)` (attribute)

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

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

### 6.3.14 ZeroObjectFunctorialWithGivenZeroObjects (for IsCapCategoryObject, IsCapCategoryObject)

▷ `ZeroObjectFunctorialWithGivenZeroObjects(C, zero_object1, zero_object2)` (operation)

**Returns:** a morphism in  $\text{Hom}(\text{zero\_object1}, \text{zero\_object2})$

The argument is a category  $C$  and a zero object  $\text{ZeroObject}(C)$  twice (for compatibility with other functors). The output is the unique morphism  $\text{zero\_object1} \rightarrow \text{zero\_object2}$ .

## 6.4 Terminal Object

A terminal object consists of two parts:

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

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

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

### 6.4.1 TerminalObject (for IsCapCategory)

▷ `TerminalObject(C)` (attribute)

**Returns:** an object

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

### 6.4.2 TerminalObject (for IsCapCategoryCell)

▷ `TerminalObject(c)` (attribute)

**Returns:** an object

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

### 6.4.3 UniversalMorphismIntoTerminalObject (for IsCapCategoryObject)

▷ `UniversalMorphismIntoTerminalObject(A)` (attribute)

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

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

### 6.4.4 UniversalMorphismIntoTerminalObjectWithGivenTerminalObject (for IsCapCategoryObject, IsCapCategoryObject)

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

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

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

### 6.4.5 TerminalObjectFunctorial (for IsCapCategory)

▷ `TerminalObjectFunctorial(C)` (attribute)

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

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

### 6.4.6 TerminalObjectFunctorialWithGivenTerminalObjects (for IsCapCategoryObject, IsCapCategoryObject)

▷ `TerminalObjectFunctorialWithGivenTerminalObjects(C, terminal_object1, terminal_object2)` (operation)

**Returns:** a morphism in  $\text{Hom}(terminal\_object1, terminal\_object2)$

The argument is a category  $C$  and a terminal object  $\text{TerminalObject}(C)$  twice (for compatibility with other functorials). The output is the unique morphism  $terminal\_object1 \rightarrow terminal\_object2$ .

## 6.5 Initial Object

An initial object consists of two parts:

- an object  $I$ ,

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

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

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

### 6.5.1 InitialObject (for IsCapCategory)

▷ `InitialObject(C)` (attribute)

**Returns:** an object

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

### 6.5.2 InitialObject (for IsCapCategoryCell)

▷ `InitialObject(c)` (attribute)

**Returns:** an object

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

### 6.5.3 UniversalMorphismFromInitialObject (for IsCapCategoryObject)

▷ `UniversalMorphismFromInitialObject(A)` (attribute)

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

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

### 6.5.4 UniversalMorphismFromInitialObjectWithGivenInitialObject (for IsCapCategoryObject, IsCapCategoryObject)

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

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

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

### 6.5.5 InitialObjectFunctorial (for IsCapCategory)

▷ `InitialObjectFunctorial(C)` (attribute)

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

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

### 6.5.6 InitialObjectFunctorialWithGivenInitialObjects (for IsCapCategoryObject, IsCapCategoryObject)

▷ `InitialObjectFunctorialWithGivenInitialObjects(C, initial_object1, initial_object2)` (operation)

**Returns:** a morphism in  $\text{Hom}(\text{initial\_object1}, \text{initial\_object2})$

The argument is a category  $C$  and an initial object `InitialObject(C)` twice (for compatibility with other functors). The output is the unique morphism  $\text{initial\_object1} \rightarrow \text{initial\_object2}$ .

## 6.6 Direct Sum

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

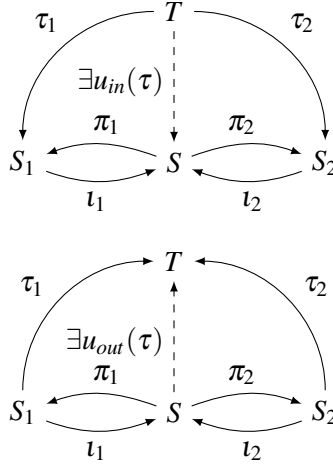
- an object  $S$ ,
- a list of morphisms  $\pi = (\pi_i : S \rightarrow S_i)_{i=1\dots n}$ ,
- a list of morphisms  $\iota = (\iota_i : S_i \rightarrow S)_{i=1\dots n}$ ,
- a dependent function  $u_{\text{in}}$  mapping every list  $\tau = (\tau_i : T \rightarrow S_i)_{i=1\dots n}$  to a morphism  $u_{\text{in}}(\tau)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

such that

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

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

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



### 6.6.1 DirectSum

▷ DirectSum(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 objects  $D = (S_1, \dots, S_n)$ .
- The arguments are objects  $S_1, \dots, S_n$ .

The output is the direct sum  $\bigoplus_{i=1}^n S_i$ .

### 6.6.2 DirectSumOp (for IsList)

- ▷ `DirectSumOp(D)` (operation)  
**Returns:** an object  
 The argument is a list of objects  $D = (S_1, \dots, S_n)$ . The output is the direct sum  $\bigoplus_{i=1}^n S_i$ .

### 6.6.3 ProjectionInFactorOfDirectSum (for IsList, IsInt)

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

### 6.6.4 ProjectionInFactorOfDirectSumWithGivenDirectSum (for IsList, IsInt, IsCapCategoryObject)

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

### 6.6.5 InjectionOfCofactorOfDirectSum (for IsList, IsInt)

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

### 6.6.6 InjectionOfCofactorOfDirectSumWithGivenDirectSum (for IsList, IsInt, IsCapCategoryObject)

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

### 6.6.7 UniversalMorphismIntoDirectSum (for IsList, IsCapCategoryObject, IsList)

- ▷ `UniversalMorphismIntoDirectSum(D, T, tau)` (operation)  
**Returns:** a morphism in  $\text{Hom}(T, \bigoplus_{i=1}^n S_i)$   
 The arguments are a list of objects  $D = (S_1, \dots, S_n)$ , a test object  $T$ , and a list of morphisms  $\tau = (\tau_i : T \rightarrow S_i)_{i=1 \dots n}$ . For convenience, the diagram  $D$  and/or the test object  $T$  can be omitted and are automatically derived from  $\tau$  in that case. The output is the morphism  $u_{\text{in}}(\tau) : T \rightarrow \bigoplus_{i=1}^n S_i$  given by the universal property of the direct sum.

### 6.6.8 UniversalMorphismIntoDirectSumWithGivenDirectSum (for IsList, IsCapCategoryObject, IsList, IsCapCategoryObject)

▷ UniversalMorphismIntoDirectSumWithGivenDirectSum( $D, T, \tau, S$ ) (operation)

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

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

### 6.6.9 UniversalMorphismFromDirectSum (for IsList, IsCapCategoryObject, IsList)

▷ UniversalMorphismFromDirectSum( $D, T, \tau$ ) (operation)

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

The arguments are a list of objects  $D = (S_1, \dots, S_n)$ , a test object  $T$ , and a list of morphisms  $\tau = (\tau_i : S_i \rightarrow T)_{i=1 \dots n}$ . For convenience, the diagram  $D$  and/or the test object  $T$  can be omitted and are automatically derived from  $\tau$  in that case. The output is the morphism  $u_{\text{out}}(\tau) : \bigoplus_{i=1}^n S_i \rightarrow T$  given by the universal property of the direct sum.

### 6.6.10 UniversalMorphismFromDirectSumWithGivenDirectSum (for IsList, IsCapCategoryObject, IsList, IsCapCategoryObject)

▷ UniversalMorphismFromDirectSumWithGivenDirectSum( $D, T, \tau, S$ ) (operation)

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

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

### 6.6.11 IsomorphismFromDirectSumToDirectProduct (for IsList)

▷ IsomorphismFromDirectSumToDirectProduct( $D$ ) (operation)

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

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

### 6.6.12 IsomorphismFromDirectProductToDirectSum (for IsList)

▷ IsomorphismFromDirectProductToDirectSum( $D$ ) (operation)

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

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

### 6.6.13 IsomorphismFromDirectSumToCoproduct (for IsList)

▷ IsomorphismFromDirectSumToCoproduct( $D$ ) (operation)

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



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

#### 6.6.14 IsomorphismFromCoproductToDirectSum (for IsList)

▷ `IsomorphismFromCoproductToDirectSum(D)` (operation)

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

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

#### 6.6.15 MorphismBetweenDirectSums (for IsList, IsList, IsList)

▷ `MorphismBetweenDirectSums(diagram_S, M, diagram_T)` (operation)

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

The arguments are given as follows:

- $diagram\_S$  is a list of objects  $(A_i)_{i=1\dots m}$ ,
- $diagram\_T$  is a list of objects  $(B_j)_{j=1\dots n}$ ,
- $M$  is a list of lists of morphisms  $((\phi_{i,j} : A_i \rightarrow B_j)_{j=1\dots n})_{i=1\dots m}$ .

The output is the morphism  $\bigoplus_{i=1}^m A_i \rightarrow \bigoplus_{j=1}^n B_j$  defined by the matrix  $M$ .

#### 6.6.16 MorphismBetweenDirectSums (for IsList)

▷ `MorphismBetweenDirectSums(M)` (operation)

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

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

#### 6.6.17 MorphismBetweenDirectSumsWithGivenDirectSums (for IsCapCategoryObject, IsList, IsList, IsList, IsCapCategoryObject)

▷ `MorphismBetweenDirectSumsWithGivenDirectSums(S, diagram_S, M, diagram_T, T)` (operation)

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

The arguments are given as follows:

- $diagram\_S$  is a list of objects  $(A_i)_{i=1\dots m}$ ,
- $diagram\_T$  is a list of objects  $(B_j)_{j=1\dots n}$ ,
- $S$  is the direct sum  $\bigoplus_{i=1}^m A_i$ ,
- $T$  is the direct sum  $\bigoplus_{j=1}^n B_j$ ,
- $M$  is a list of lists of morphisms  $((\phi_{i,j} : A_i \rightarrow B_j)_{j=1\dots n})_{i=1\dots m}$ .

The output is the morphism  $\bigoplus_{i=1}^m A_i \rightarrow \bigoplus_{j=1}^n B_j$  defined by the matrix  $M$ .

### 6.6.18 ComponentOfMorphismIntoDirectSum (for IsCapCategoryMorphism, IsList, IsInt)

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

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

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

### 6.6.19 ComponentOfMorphismFromDirectSum (for IsCapCategoryMorphism, IsList, IsInt)

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

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

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

### 6.6.20 DirectSumFunctorial (for IsList, IsList, IsList)

▷ `DirectSumFunctorial(source_diagram, L, range_diagram)` (operation)

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

The arguments are a list of objects  $(S_i)_{i=1\dots n}$ , a list of morphisms  $L = (\mu_1 : S_1 \rightarrow S'_1, \dots, \mu_n : S_n \rightarrow S'_n)$ , and a list of objects  $(S'_i)_{i=1\dots n}$ . For convenience, `source_diagram` and `range_diagram` can be omitted and are automatically derived from  $L$  in that case. The output is a morphism  $\bigoplus_{i=1}^n S_i \rightarrow \bigoplus_{i=1}^n S'_i$  given by the functoriality of the direct sum.

### 6.6.21 DirectSumFunctorialWithGivenDirectSums (for IsCapCategoryObject, IsList, IsList, IsList, IsCapCategoryObject)

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

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

The arguments are an object  $d_1 = \bigoplus_{i=1}^n S_i$ , a list of objects  $(S_i)_{i=1\dots n}$ , a list of morphisms  $L = (\mu_1 : S_1 \rightarrow S'_1, \dots, \mu_n : S_n \rightarrow S'_n)$ , a list of objects  $(S'_i)_{i=1\dots n}$ , and an object  $d_2 = \bigoplus_{i=1}^n S'_i$ . For convenience, `source_diagram` and `range_diagram` can be omitted and are automatically derived from  $L$  in that case. The output is a morphism  $d_1 \rightarrow d_2$  given by the functoriality of the direct sum.

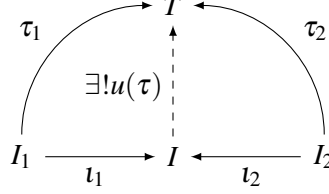
## 6.7 Coproduct

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

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

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

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



### 6.7.1 Coproduct (for IsList)

▷ Coproduct( $D$ ) (operation)

**Returns:** an object

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

### 6.7.2 Coproduct (for IsCapCategoryObject, IsCapCategoryObject)

▷ Coproduct( $I1, I2$ ) (operation)

**Returns:** an object

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

### 6.7.3 Coproduct (for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject)

▷ Coproduct( $I1, I2$ ) (operation)

**Returns:** an object

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

### 6.7.4 InjectionOfCofactorOfCoproduct (for IsList, IsInt)

▷ InjectionOfCofactorOfCoproduct( $D, k$ ) (operation)

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

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

### 6.7.5 InjectionOfCofactorOfCoproductWithGivenCoproduct (for IsList, IsInt, IsCapCategoryObject)

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

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

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

### 6.7.6 UniversalMorphismFromCoproduct (for IsList, IsCapCategoryObject, IsList)

▷ UniversalMorphismFromCoproduct( $D$ ,  $T$ ,  $\tau$ ) (operation)

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

The arguments are a list of objects  $D = (I_1, \dots, I_n)$ , a test object  $T$ , and a list of morphisms  $\tau = (\tau_i : I_i \rightarrow T)$ . For convenience, the diagram  $D$  and/or the test object  $T$  can be omitted and are automatically derived from  $\tau$  in that case. The output is the morphism  $u(\tau) : \bigsqcup_{i=1}^n I_i \rightarrow T$  given by the universal property of the coproduct.

### 6.7.7 UniversalMorphismFromCoproductWithGivenCoproduct (for IsList, IsCapCategoryObject, IsList, IsCapCategoryObject)

▷ UniversalMorphismFromCoproductWithGivenCoproduct( $D$ ,  $T$ ,  $\tau$ ,  $I$ ) (operation)

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

The arguments are a list of objects  $D = (I_1, \dots, I_n)$ , a test object  $T$ , a list of morphisms  $\tau = (\tau_i : I_i \rightarrow T)$ , and an object  $I = \bigsqcup_{i=1}^n I_i$ . For convenience, the test object  $T$  can be omitted and is automatically derived from  $\tau$  in that case. The output is the morphism  $u(\tau) : I \rightarrow T$  given by the universal property of the coproduct.

### 6.7.8 CoproductFunctorial (for IsList, IsList, IsList)

▷ CoproductFunctorial( $source\_diagram$ ,  $L$ ,  $range\_diagram$ ) (operation)

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

The arguments are a list of objects  $(I_i)_{i=1\dots n}$ , a list  $L = (\mu_1 : I_1 \rightarrow I'_1, \dots, \mu_n : I_n \rightarrow I'_n)$ , and a list of objects  $(I'_i)_{i=1\dots n}$ . For convenience,  $source\_diagram$  and  $range\_diagram$  can be omitted and are automatically derived from  $L$  in that case. The output is a morphism  $\bigsqcup_{i=1}^n I_i \rightarrow \bigsqcup_{i=1}^n I'_i$  given by the functoriality of the coproduct.

### 6.7.9 CoproductFunctorialWithGivenCoproducts (for IsCapCategoryObject, IsList, IsList, IsList, IsCapCategoryObject)

▷ CoproductFunctorialWithGivenCoproducts( $s$ ,  $source\_diagram$ ,  $L$ ,  $range\_diagram$ ,  $r$ ) (operation)

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

The arguments are an object  $s = \bigsqcup_{i=1}^n I_i$ , a list of objects  $(I_i)_{i=1\dots n}$ , a list  $L = (\mu_1 : I_1 \rightarrow I'_1, \dots, \mu_n : I_n \rightarrow I'_n)$ , a list of objects  $(I'_i)_{i=1\dots n}$ , and an object  $r = \bigsqcup_{i=1}^n I'_i$ . For convenience,  $source\_diagram$  and  $range\_diagram$  can be omitted and are automatically derived from  $L$  in that case. The output is a morphism  $\bigsqcup_{i=1}^n I_i \rightarrow \bigsqcup_{i=1}^n I'_i$  given by the functoriality of the coproduct.

### 6.7.10 ComponentOfMorphismFromCoproduct (for IsCapCategoryMorphism, IsList, IsInt)

▷ ComponentOfMorphismFromCoproduct( $\alpha$ ,  $D$ ,  $k$ ) (operation)

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

The arguments are a morphism  $\alpha : I \rightarrow A$ , a list  $D = (I_1, \dots, I_n)$  of objects with  $I = \bigsqcup_{j=1}^n I_j$ , and an integer  $k$ . The output is the component morphism  $I_k \rightarrow A$ .

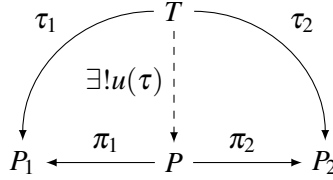
## 6.8 Direct Product

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

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

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

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



### 6.8.1 DirectProduct

▷ DirectProduct(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 objects  $D = (P_1, \dots, P_n)$ .
- The arguments are objects  $P_1, \dots, P_n$ .

The output is the direct product  $\prod_{i=1}^n P_i$ .

### 6.8.2 DirectProductOp (for IsList)

▷ DirectProductOp(D) (operation)

**Returns:** an object

The argument is a list of objects  $D = (P_1, \dots, P_n)$ . The output is the direct product  $\prod_{i=1}^n P_i$ .

### 6.8.3 ProjectionInFactorOfDirectProduct (for IsList, IsInt)

▷ ProjectionInFactorOfDirectProduct(D, k) (operation)

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

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

#### 6.8.4 ProjectionInFactorOfDirectProductWithGivenDirectProduct (for IsList, IsInt, IsCapCategoryObject)

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

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

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

#### 6.8.5 UniversalMorphismIntoDirectProduct (for IsList, IsCapCategoryObject, IsList)

▷ `UniversalMorphismIntoDirectProduct(D, T, tau)` (operation)

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

The arguments are a list of objects  $D = (P_1, \dots, P_n)$ , a test object  $T$ , and a list of morphisms  $\tau = (\tau_i : T \rightarrow P_i)_{i=1, \dots, n}$ . For convenience, the diagram  $D$  and/or the test object  $T$  can be omitted and are automatically derived from  $\tau$  in that case. The output is the morphism  $u(\tau) : T \rightarrow \prod_{i=1}^n P_i$  given by the universal property of the direct product.

#### 6.8.6 UniversalMorphismIntoDirectProductWithGivenDirectProduct (for IsList, IsCapCategoryObject, IsList, IsCapCategoryObject)

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

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

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

#### 6.8.7 DirectProductFunctorial (for IsList, IsList, IsList)

▷ `DirectProductFunctorial(source_diagram, L, range_diagram)` (operation)

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

The arguments are a list of objects  $(P_i)_{i=1, \dots, n}$ , a list of morphisms  $L = (\mu_i : P_i \rightarrow P'_i)_{i=1, \dots, n}$ , and a list of objects  $(P'_i)_{i=1, \dots, n}$ . For convenience, `source_diagram` and `range_diagram` can be omitted and are automatically derived from  $L$  in that case. The output is a morphism  $\prod_{i=1}^n P_i \rightarrow \prod_{i=1}^n P'_i$  given by the functoriality of the direct product.

#### 6.8.8 DirectProductFunctorialWithGivenDirectProducts (for IsCapCategoryObject, IsList, IsList, IsList, IsCapCategoryObject)

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

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

The arguments are an object  $s = \prod_{i=1}^n P_i$ , a list of objects  $(P_i)_{i=1, \dots, n}$ , a list of morphisms  $L = (\mu_i : P_i \rightarrow P'_i)_{i=1, \dots, n}$ , a list of objects  $(P'_i)_{i=1, \dots, n}$ , and an object  $r = \prod_{i=1}^n P'_i$ . For convenience, `source_diagram` and `range_diagram` can be omitted and are automatically derived from  $L$  in that case. The output is a morphism  $\prod_{i=1}^n P_i \rightarrow \prod_{i=1}^n P'_i$  given by the functoriality of the direct product.

### 6.8.9 ComponentOfMorphismIntoDirectProduct (for IsCapCategoryMorphism, Is-List, IsInt)

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

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

The arguments are a morphism  $\alpha : A \rightarrow P$ , a list  $D = (P_1, \dots, P_n)$  of objects with  $P = \prod_{j=1}^n P_j$ , and an integer  $k$ . The output is the component morphism  $A \rightarrow P_k$ .

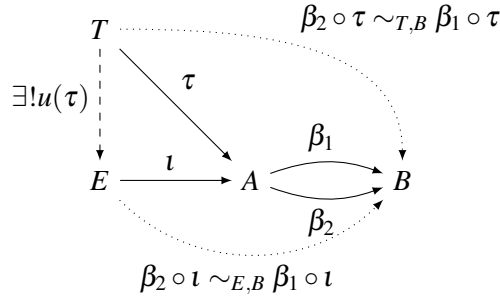
## 6.9 Equalizer

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

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

The triple  $(E, \iota, u)$  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  $\text{Equalizer}(D)$ . 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  $D' = (\beta'_i : A' \rightarrow B')_{i=1 \dots n}$  and a natural morphism between equalizer diagrams (i.e., a collection of morphisms  $\mu : A \rightarrow A'$  and  $\beta : B \rightarrow B'$  such that  $\beta'_i \circ \mu \sim_{A',B'} \beta \circ \beta_i$  for  $i = 1, \dots, n$ ) we obtain a morphism  $\text{Equalizer}(D) \rightarrow \text{Equalizer}(D')$ .



### 6.9.1 Equalizer

▷ `Equalizer(arg)` (function)

**Returns:** an object

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

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

The output is the equalizer  $\text{Equalizer}(D)$ .

### 6.9.2 EqualizerOp (for IsCapCategoryObject, IsList)

▷ `EqualizerOp(A, D)` (operation)

**Returns:** an object

The arguments are an object  $A$  and list of morphisms  $D = (\beta_i : A \rightarrow B)_{i=1\dots n}$ . For convenience, the object  $A$  can be omitted and is automatically derived from  $D$  in that case. The output is the equalizer  $\text{Equalizer}(D)$ .

### 6.9.3 EmbeddingOfEqualizer (for IsCapCategoryObject, IsList)

▷ `EmbeddingOfEqualizer(A, D)` (operation)

**Returns:** a morphism in  $\text{Hom}(\text{Equalizer}(D), A)$

The arguments are an object  $A$  and a list of morphisms  $D = (\beta_i : A \rightarrow B)_{i=1\dots n}$ . For convenience, the object  $A$  can be omitted and is automatically derived from  $D$  in that case. The output is the equalizer embedding  $\iota : \text{Equalizer}(D) \rightarrow A$ .

### 6.9.4 EmbeddingOfEqualizerWithGivenEqualizer (for IsCapCategoryObject, IsList, IsCapCategoryObject)

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

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

The arguments are an object  $A$ , a list of morphisms  $D = (\beta_i : A \rightarrow B)_{i=1\dots n}$ , and an object  $E = \text{Equalizer}(D)$ . For convenience, the object  $A$  can be omitted and is automatically derived from  $D$  in that case. The output is the equalizer embedding  $\iota : E \rightarrow A$ .

### 6.9.5 MorphismFromEqualizerToSink (for IsCapCategoryObject, IsList)

▷ `MorphismFromEqualizerToSink(A, D)` (operation)

**Returns:** a morphism in  $\text{Hom}(\text{Equalizer}(D), B)$

The arguments are an object  $A$  and a list of morphisms  $D = (\beta_i : A \rightarrow B)_{i=1\dots n}$ . For convenience, the object  $A$  can be omitted and is automatically derived from  $D$  in that case. The output is the composition  $\mu : \text{Equalizer}(D) \rightarrow B$  of the embedding  $\iota : \text{Equalizer}(D) \rightarrow A$  and  $\beta_1$ .

### 6.9.6 MorphismFromEqualizerToSinkWithGivenEqualizer (for IsCapCategoryObject, IsList, IsCapCategoryObject)

▷ `MorphismFromEqualizerToSinkWithGivenEqualizer(A, D, E)` (operation)

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

The arguments are an object  $A$ , a list of morphisms  $D = (\beta_i : A \rightarrow B)_{i=1\dots n}$  and an object  $E = \text{Equalizer}(D)$ . For convenience, the object  $A$  can be omitted and is automatically derived from  $D$  in that case. The output is the composition  $\mu : E \rightarrow B$  of the embedding  $\iota : E \rightarrow A$  and  $\beta_1$ .

### 6.9.7 UniversalMorphismIntoEqualizer (for IsCapCategoryObject, IsList, IsCapCategoryObject, IsCapCategoryMorphism)

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

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



The arguments are an object  $A$ , a list of morphisms  $D = (\beta_i : A \rightarrow B)_{i=1\dots n}$ , a test object  $T$ , and a morphism  $\tau : T \rightarrow A$  such that  $\beta_i \circ \tau \sim_{T,B} \beta_j \circ \tau$  for all pairs  $i, j$ . For convenience, the object  $A$  can be omitted and is automatically derived from  $D$  in that case. For convenience, the test object  $T$  can be omitted and is automatically derived from  $\tau$  in that case. The output is the morphism  $u(\tau) : T \rightarrow \text{Equalizer}(D)$  given by the universal property of the equalizer.

### 6.9.8 UniversalMorphismIntoEqualizerWithGivenEqualizer (for IsCapCategoryObject, IsList, IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryObject)

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

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

The arguments are an object  $A$ , a list of morphisms  $D = (\beta_i : A \rightarrow B)_{i=1\dots n}$ , a test object  $T$ , a morphism  $\tau : T \rightarrow A$  such that  $\beta_i \circ \tau \sim_{T,B} \beta_j \circ \tau$  for all pairs  $i, j$ , and an object  $E = \text{Equalizer}(D)$ . For convenience, the object  $A$  can be omitted and is automatically derived from  $D$  in that case. For convenience, the test object  $T$  can be omitted and is automatically derived from  $\tau$  in that case. The output is the morphism  $u(\tau) : T \rightarrow E$  given by the universal property of the equalizer.

### 6.9.9 EqualizerFunctorial (for IsList, IsCapCategoryMorphism, IsList)

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

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

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

### 6.9.10 EqualizerFunctorialWithGivenEqualizers (for IsCapCategoryObject, IsList, IsCapCategoryMorphism, IsList, IsCapCategoryObject)

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

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

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

### 6.9.11 JointPairwiseDifferencesOfMorphismsIntoDirectProduct (for IsCapCategoryObject, IsList)

▷ `JointPairwiseDifferencesOfMorphismsIntoDirectProduct(A, D)` (operation)

**Returns:** a morphism in  $\text{Hom}(A, \prod_{i=1}^{n-1} B)$

The arguments are an object  $A$  and a list of morphisms  $D = (\beta_i : A \rightarrow B)_{i=1\dots n}$ . The output is a morphism  $A \rightarrow \prod_{i=1}^{n-1} B$  such that its kernel equalizes the  $\beta_i$ .

### 6.9.12 IsomorphismFromEqualizerToKernelOfJointPairwiseDifferencesOfMorphismsIntoDirectProduct (for IsCapCategoryObject, IsList)

▷ `IsomorphismFromEqualizerToKernelOfJointPairwiseDifferencesOfMorphismsIntoDirectProduct(A, D)` (operation)

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

The arguments are an object  $A$  and a list of morphisms  $D = (\beta_i : A \rightarrow B)_{i=1\dots n}$ . The output is a morphism  $\text{Equalizer}(D) \rightarrow \Delta$ , where  $\Delta$  denotes the kernel object equalizing the morphisms  $\beta_i$ .

### 6.9.13 IsomorphismFromKernelOfJointPairwiseDifferencesOfMorphismsIntoDirectProductToEqualizer (for IsCapCategoryObject, IsList)

▷ `IsomorphismFromKernelOfJointPairwiseDifferencesOfMorphismsIntoDirectProductToEqualizer(A, D)` (operation)

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

The arguments are an object  $A$  and a list of morphisms  $D = (\beta_i : A \rightarrow B)_{i=1\dots n}$ . The output is a morphism  $\Delta \rightarrow \text{Equalizer}(D)$ , where  $\Delta$  denotes the kernel object equalizing the morphisms  $\beta_i$ .

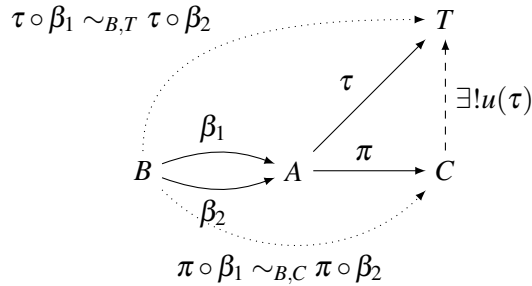
## 6.10 Coequalizer

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

- an object  $C$ ,
- a morphism  $\pi : A \rightarrow C$  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  $\tau : A \rightarrow T$  such that  $\tau \circ \beta_i \sim_{B,T} \tau \circ \beta_j$  to a morphism  $u(\tau) : C \rightarrow T$  such that  $u(\tau) \circ \pi \sim_{A,T} \tau$ .

The triple  $(C, \pi, u)$  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  $\text{Coequalizer}(D)$ . 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  $D' = (\beta'_i : B' \rightarrow A')_{i=1\dots n}$  and a natural morphism between coequalizer diagrams (i.e., a collection of morphisms  $\mu : A \rightarrow A'$  and  $\beta : B \rightarrow B'$  such that  $\beta'_i \circ \beta \sim_{B,A'} \mu \circ \beta_i$  for  $i = 1, \dots, n$ ) we obtain a morphism  $\text{Coequalizer}(D) \rightarrow \text{Coequalizer}(D')$ .



### 6.10.1 Coequalizer

▷ `Coequalizer(arg)` (function)

**Returns:** an object

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

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

The output is the coequalizer  $\text{Coequalizer}(D)$ .

### 6.10.2 CoequalizerOp (for IsCapCategoryObject, IsList)

▷ `CoequalizerOp(A, D)` (operation)

**Returns:** an object

The arguments are an object  $A$  and a list of morphisms  $D = (\beta_i : B \rightarrow A)_{i=1\dots n}$ . For convenience, the object  $A$  can be omitted and is automatically derived from  $D$  in that case. The output is the coequalizer  $\text{Coequalizer}(D)$ .

### 6.10.3 ProjectionOntoCoequalizer (for IsCapCategoryObject, IsList)

▷ `ProjectionOntoCoequalizer(A, D)` (operation)

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

The arguments are an object  $A$  and a list of morphisms  $D = (\beta_i : B \rightarrow A)_{i=1\dots n}$ . For convenience, the object  $A$  can be omitted and is automatically derived from  $D$  in that case. The output is the projection  $\pi : A \rightarrow \text{Coequalizer}(D)$ .

### 6.10.4 ProjectionOntoCoequalizerWithGivenCoequalizer (for IsCapCategoryObject, IsList, IsCapCategoryObject)

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

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

The arguments are an object  $A$ , a list of morphisms  $D = (\beta_i : B \rightarrow A)_{i=1\dots n}$ , and an object  $C = \text{Coequalizer}(D)$ . For convenience, the object  $A$  can be omitted and is automatically derived from  $D$  in that case. The output is the projection  $\pi : A \rightarrow C$ .

### 6.10.5 MorphismFromSourceToCoequalizer (for IsCapCategoryObject, IsList)

▷ `MorphismFromSourceToCoequalizer(A, D)` (operation)

**Returns:** a morphism in  $\text{Hom}(B, \text{Coequalizer}(D))$ .

The arguments are an object  $A$  and a list of morphisms  $D = (\beta_i : B \rightarrow A)_{i=1\dots n}$ . For convenience, the object  $A$  can be omitted and is automatically derived from  $D$  in that case. The output is the composition  $\mu : B \rightarrow \text{Coequalizer}(D)$  of  $\beta_1$  and the projection  $\pi : A \rightarrow \text{Coequalizer}(D)$ .

### 6.10.6 MorphismFromSourceToCoequalizerWithGivenCoequalizer (for IsCapCategoryObject, IsList, IsCapCategoryObject)

▷ MorphismFromSourceToCoequalizerWithGivenCoequalizer( $A, D, C$ ) (operation)

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

The arguments are an object  $A$ , a list of morphisms  $D = (\beta_i : B \rightarrow A)_{i=1\dots n}$  and an object  $C = \text{Coequalizer}(D)$ . For convenience, the object  $A$  can be omitted and is automatically derived from  $D$  in that case. The output is the composition  $\mu : B \rightarrow C$  of  $\beta_1$  and the projection  $\pi : A \rightarrow C$ .

### 6.10.7 UniversalMorphismFromCoequalizer (for IsCapCategoryObject, IsList, IsCapCategoryObject, IsCapCategoryMorphism)

▷ UniversalMorphismFromCoequalizer( $A, D, T, \tau$ ) (operation)

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

The arguments are an object  $A$ , a list of morphisms  $D = (\beta_i : B \rightarrow A)_{i=1\dots n}$ , a test object  $T$ , and a morphism  $\tau : A \rightarrow T$  such that  $\tau \circ \beta_i \sim_{B,T} \tau \circ \beta_j$  for all pairs  $i, j$ . For convenience, the object  $A$  can be omitted and is automatically derived from  $D$  in that case. For convenience, the test object  $T$  can be omitted and is automatically derived from  $\tau$  in that case. The output is the morphism  $u(\tau) : \text{Coequalizer}(D) \rightarrow T$  given by the universal property of the coequalizer.

### 6.10.8 UniversalMorphismFromCoequalizerWithGivenCoequalizer (for IsCapCategoryObject, IsList, IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryObject)

▷ UniversalMorphismFromCoequalizerWithGivenCoequalizer( $A, D, T, \tau, C$ ) (operation)

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

The arguments are an object  $A$ , a list of morphisms  $D = (\beta_i : B \rightarrow A)_{i=1\dots n}$ , a test object  $T$ , a morphism  $\tau : A \rightarrow T$  such that  $\tau \circ \beta_i \sim_{B,T} \tau \circ \beta_j$ , and an object  $C = \text{Coequalizer}(D)$ . For convenience, the object  $A$  can be omitted and is automatically derived from  $D$  in that case. For convenience, the test object  $T$  can be omitted and is automatically derived from  $\tau$  in that case. The output is the morphism  $u(\tau) : C \rightarrow T$  given by the universal property of the coequalizer.

### 6.10.9 CoequalizerFunctorial (for IsList, IsCapCategoryMorphism, IsList)

▷ CoequalizerFunctorial( $Ls, \mu, Lr$ ) (operation)

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

The arguments are a list of morphisms  $Ls = (\beta_i : B \rightarrow A)_{i=1\dots n}$ , a morphism  $\mu : A \rightarrow A'$ , and a list of morphisms  $Lr = (\beta'_i : B' \rightarrow A')_{i=1\dots n}$  such that there exists a morphism  $\beta : B \rightarrow B'$  such that  $\beta'_i \circ \beta \sim_{B,A'} \mu \circ \beta_i$  for  $i = 1, \dots, n$ . The output is the morphism  $\text{Coequalizer}((\beta_i)_{i=1}^n) \rightarrow \text{Coequalizer}((\beta'_i)_{i=1}^n)$  given by the functoriality of the coequalizer.

### 6.10.10 CoequalizerFunctorialWithGivenCoequalizers (for IsCapCategoryObject, IsList, IsCapCategoryMorphism, IsList, IsCapCategoryObject)

▷ CoequalizerFunctorialWithGivenCoequalizers( $s, Ls, \mu, Lr, r$ ) (operation)

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

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

### 6.10.11 JointPairwiseDifferencesOfMorphismsFromCoproduct (for IsCapCategory-Object, IsList)

▷ `JointPairwiseDifferencesOfMorphismsFromCoproduct(A, D)` (operation)

**Returns:** a morphism in  $\text{Hom}(\bigsqcup_{i=1}^{n-1} B, A)$

The arguments are an object  $A$  and a list of morphisms  $D = (\beta_i : B \rightarrow A)_{i=1\dots n}$ . The output is a morphism  $\bigsqcup_{i=1}^{n-1} B \rightarrow A$  such that its cokernel coequalizes the  $\beta_i$ .

### 6.10.12 IsomorphismFromCoequalizerToCokernelOfJointPairwiseDifferencesOfMorphismsFromCoproduct (for IsCapCategoryObject, IsList)

▷ `IsomorphismFromCoequalizerToCokernelOfJointPairwiseDifferencesOfMorphismsFromCoproduct(A, D)` (operation)

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

The arguments are an object  $A$  and a list of morphisms  $D = (\beta_i : B \rightarrow A)_{i=1\dots n}$ . The output is a morphism  $\text{Coequalizer}(D) \rightarrow \Delta$ , where  $\Delta$  denotes the cokernel object coequalizing the morphisms  $\beta_i$ .

### 6.10.13 IsomorphismFromCokernelOfJointPairwiseDifferencesOfMorphismsFromCoproductToCoequalizer (for IsCapCategoryObject, IsList)

▷ `IsomorphismFromCokernelOfJointPairwiseDifferencesOfMorphismsFromCoproductToCoequalizer(A, D)` (operation)

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

The arguments are an object  $A$  and a list of morphisms  $D = (\beta_i : B \rightarrow A)_{i=1\dots n}$ . The output is a morphism  $\Delta \rightarrow \text{Coequalizer}(D)$ , where  $\Delta$  denotes the cokernel object coequalizing the morphisms  $\beta_i$ .

## 6.11 Fiber Product

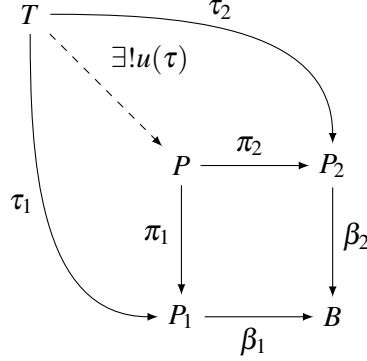
For an integer  $n \geq 1$  and a given list of morphisms  $D = (\beta_i : P_i \rightarrow B)_{i=1\dots n}$ , a fiber product of  $D$  consists of three parts:

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

The triple  $(P, \pi, u)$  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  $\text{FiberProduct}(D)$ . We say that the morphism  $u(\tau)$  is induced by the *universal property of the fiber product*.

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

and  $\beta : B \rightarrow B'$  such that  $\beta'_i \circ \mu_i \sim_{P_i, B'} \beta \circ \beta_i$  for  $i = 1, \dots, n$ ) we obtain a morphism  $\text{FiberProduct}(D) \rightarrow \text{FiberProduct}(D')$ .



### 6.11.1 IsomorphismFromFiberProductToEqualizerOfDirectProductDiagram (for Is-List)

▷ `IsomorphismFromFiberProductToEqualizerOfDirectProductDiagram(D)` (operation)

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

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

### 6.11.2 IsomorphismFromEqualizerOfDirectProductDiagramToFiberProduct (for Is-List)

▷ `IsomorphismFromEqualizerOfDirectProductDiagramToFiberProduct(D)` (operation)

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

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

### 6.11.3 FiberProductEmbeddingInDirectProduct (for IsList)

▷ `FiberProductEmbeddingInDirectProduct(D)` (operation)

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

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

### 6.11.4 FiberProductEmbeddingInDirectSum (for IsList)

▷ `FiberProductEmbeddingInDirectSum(D)` (operation)

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

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

### 6.11.5 FiberProduct

▷ `FiberProduct(arg)` (function)

**Returns:** an object

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

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

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

#### 6.11.6 FiberProductOp (for IsList)

▷  $\text{FiberProductOp}(D)$  (operation)

**Returns:** an object

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

#### 6.11.7 ProjectionInFactorOfFiberProduct (for IsList, IsInt)

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

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

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

#### 6.11.8 ProjectionInFactorOfFiberProductWithGivenFiberProduct (for IsList, IsInt, IsCapCategoryObject)

▷  $\text{ProjectionInFactorOfFiberProductWithGivenFiberProduct}(D, k, P)$  (operation)

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

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

#### 6.11.9 MorphismFromFiberProductToSink (for IsList)

▷  $\text{MorphismFromFiberProductToSink}(D)$  (operation)

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

The arguments are a list of morphisms  $D = (\beta_i : P_i \rightarrow B)_{i=1\dots n}$ . The output is the composition  $\mu : \text{FiberProduct}(D) \rightarrow B$  of the 1-st projection  $\pi_1 : \text{FiberProduct}(D) \rightarrow P_1$  and  $\beta_1$ .

#### 6.11.10 MorphismFromFiberProductToSinkWithGivenFiberProduct (for IsList, IsCapCategoryObject)

▷  $\text{MorphismFromFiberProductToSinkWithGivenFiberProduct}(D, P)$  (operation)

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

The arguments are a list of morphisms  $D = (\beta_i : P_i \rightarrow B)_{i=1\dots n}$  and an object  $P = \text{FiberProduct}(D)$ . The output is the composition  $\mu : P \rightarrow B$  of the 1-st projection  $\pi_1 : P \rightarrow P_1$  and  $\beta_1$ .

### 6.11.11 UniversalMorphismIntoFiberProduct (for IsList, IsCapCategoryObject, IsList)

▷ UniversalMorphismIntoFiberProduct( $D$ ,  $T$ ,  $\tau$ ) (operation)

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

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

### 6.11.12 UniversalMorphismIntoFiberProductWithGivenFiberProduct (for IsList, IsCapCategoryObject, IsList, IsCapCategoryObject)

▷ UniversalMorphismIntoFiberProductWithGivenFiberProduct( $D$ ,  $T$ ,  $\tau$ ,  $P$ ) (operation)

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

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

### 6.11.13 FiberProductFunctorial (for IsList, IsList, IsList)

▷ FiberProductFunctorial( $L_s$ ,  $L_m$ ,  $L_r$ ) (operation)

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

The arguments are three lists of morphisms  $L_s = (\beta_i : P_i \rightarrow B)_{i=1\dots n}$ ,  $L_m = (\mu_i : P_i \rightarrow P'_i)_{i=1\dots n}$ ,  $L_r = (\beta'_i : P'_i \rightarrow B')_{i=1\dots n}$  having the same length  $n$  such that there exists a morphism  $\beta : B \rightarrow B'$  such that  $\beta'_i \circ \mu_i \sim_{P_i, B'} \beta \circ \beta_i$  for  $i = 1, \dots, n$ . The output is the morphism  $\text{FiberProduct}((\beta_i)_{i=1\dots n}) \rightarrow \text{FiberProduct}((\beta'_i)_{i=1\dots n})$  given by the functoriality of the fiber product.

### 6.11.14 FiberProductFunctorialWithGivenFiberProducts (for IsCapCategoryObject, IsList, IsList, IsList, IsCapCategoryObject)

▷ FiberProductFunctorialWithGivenFiberProducts( $s$ ,  $L_s$ ,  $L_m$ ,  $L_r$ ,  $r$ ) (operation)

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

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

## 6.12 Pushout

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

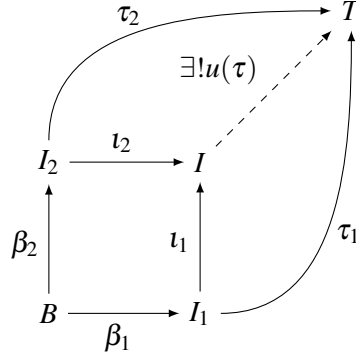
- an object  $I$ ,
- a list of morphisms  $\iota = (\iota_i : I_i \rightarrow I)_{i=1\dots n}$  such that  $\iota_i \circ \beta_i \sim_{B,I} \iota_j \circ \beta_j$  for all pairs  $i, j$ ,



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

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

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



### 6.12.1 IsomorphismFromPushoutToCoequalizerOfCoproductDiagram (for IsList)

▷ `IsomorphismFromPushoutToCoequalizerOfCoproductDiagram(D)` (operation)

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

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

### 6.12.2 IsomorphismFromCoequalizerOfCoproductDiagramToPushout (for IsList)

▷ `IsomorphismFromCoequalizerOfCoproductDiagramToPushout(D)` (operation)

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

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

### 6.12.3 PushoutProjectionFromCoproduct (for IsList)

▷ `PushoutProjectionFromCoproduct(D)` (operation)

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

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

### 6.12.4 PushoutProjectionFromDirectSum (for IsList)

▷ `PushoutProjectionFromDirectSum(D)` (operation)

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

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

### 6.12.5 Pushout (for IsList)

- ▷ `Pushout(D)` (operation)  
**Returns:** an object  
 The argument is a list of morphisms  $D = (\beta_i : B \rightarrow I_i)_{i=1\dots n}$ . The output is the pushout `Pushout(D)`.

### 6.12.6 Pushout (for IsCapCategoryMorphism, IsCapCategoryMorphism)

- ▷ `Pushout(D)` (operation)  
**Returns:** an object  
 This is a convenience method. The arguments are a morphism  $\alpha$  and a morphism  $\beta$ . The output is the pushout `Pushout( $\alpha, \beta$ )`.

### 6.12.7 InjectionOfCofactorOfPushout (for IsList, IsInt)

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

### 6.12.8 InjectionOfCofactorOfPushoutWithGivenPushout (for IsList, IsInt, IsCapCategoryObject)

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

### 6.12.9 MorphismFromSourceToPushout (for IsList)

- ▷ `MorphismFromSourceToPushout(D)` (operation)  
**Returns:** a morphism in  $\text{Hom}(B, \text{Pushout}(D))$ .  
 The arguments are a list of morphisms  $D = (\beta_i : B \rightarrow I_i)_{i=1\dots n}$ . The output is the composition  $\mu : B \rightarrow \text{Pushout}(D)$  of  $\beta_1$  and the 1-st injection  $\iota_1 : I_1 \rightarrow \text{Pushout}(D)$ .

### 6.12.10 MorphismFromSourceToPushoutWithGivenPushout (for IsList, IsCapCategoryObject)

- ▷ `MorphismFromSourceToPushoutWithGivenPushout(D, I)` (operation)  
**Returns:** a morphism in  $\text{Hom}(B, I)$ .  
 The arguments are a list of morphisms  $D = (\beta_i : B \rightarrow I_i)_{i=1\dots n}$  and an object  $I = \text{Pushout}(D)$ . The output is the composition  $\mu : B \rightarrow I$  of  $\beta_1$  and the 1-st injection  $\iota_1 : I_1 \rightarrow I$ .

### 6.12.11 UniversalMorphismFromPushout (for IsList, IsCapCategoryObject, IsList)

- ▷ `UniversalMorphismFromPushout(D, T, tau)` (operation)  
**Returns:** a morphism in  $\text{Hom}(\text{Pushout}(D), T)$

The arguments are a list of morphisms  $D = (\beta_i : B \rightarrow I_i)_{i=1\dots n}$ , a test object  $T$ , and a list of morphisms  $\tau = (\tau_i : I_i \rightarrow T)_{i=1\dots n}$  such that  $\tau_i \circ \beta_i \sim_{B,T} \tau_j \circ \beta_j$ . For convenience, the test object  $T$  can be omitted and is automatically derived from  $\tau$  in that case. The output is the morphism  $u(\tau) : \text{Pushout}(D) \rightarrow T$  given by the universal property of the pushout.

#### 6.12.12 UniversalMorphismFromPushoutWithGivenPushout (for IsList, IsCapCategoryObject, IsList, IsCapCategoryObject)

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

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

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

#### 6.12.13 PushoutFunctorial (for IsList, IsList, IsList)

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

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

The arguments are three lists of morphisms  $L_s = (\beta_i : B \rightarrow I_i)_{i=1\dots n}$ ,  $L_m = (\mu_i : I_i \rightarrow I'_i)_{i=1\dots n}$ ,  $L_r = (\beta'_i : B' \rightarrow I'_i)_{i=1\dots n}$  having the same length  $n$  such that there exists a morphism  $\beta : B \rightarrow B'$  such that  $\beta'_i \circ \beta \sim_{B,I'_i} \mu_i \circ \beta_i$  for  $i = 1, \dots, n$ . The output is the morphism  $\text{Pushout}((\beta_i)_{i=1}^n) \rightarrow \text{Pushout}((\beta'_i)_{i=1}^n)$  given by the functoriality of the pushout.

#### 6.12.14 PushoutFunctorialWithGivenPushouts (for IsCapCategoryObject, IsList, IsList, IsList, IsCapCategoryObject)

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

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

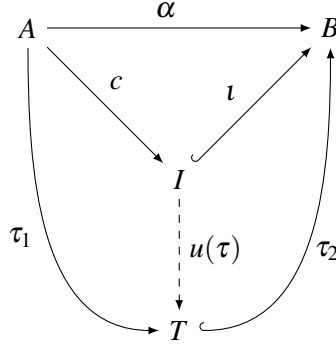
The arguments are an object  $s = \text{Pushout}((\beta_i)_{i=1}^n)$ , three lists of morphisms  $L_s = (\beta_i : B \rightarrow I_i)_{i=1\dots n}$ ,  $L_m = (\mu_i : I_i \rightarrow I'_i)_{i=1\dots n}$ ,  $L_r = (\beta'_i : B' \rightarrow I'_i)_{i=1\dots n}$  having the same length  $n$  such that there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

### 6.13 Image

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

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

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



### 6.13.1 IsomorphismFromImageObjectToKernelOfCokernel (for IsCapCategoryMorphism)

▷ `IsomorphismFromImageObjectToKernelOfCokernel(alpha)` (attribute)

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

The argument is a morphism  $\alpha$ . The output is the canonical morphism  $\text{im}(\alpha) \rightarrow \text{KernelObject}(\text{CokernelProjection}(\alpha))$ .

### 6.13.2 IsomorphismFromKernelOfCokernelToImageObject (for IsCapCategoryMorphism)

▷ `IsomorphismFromKernelOfCokernelToImageObject(alpha)` (attribute)

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

The argument is a morphism  $\alpha$ . The output is the canonical morphism  $\text{KernelObject}(\text{CokernelProjection}(\alpha)) \rightarrow \text{im}(\alpha)$ .

### 6.13.3 ImageObject (for IsCapCategoryMorphism)

▷ `ImageObject(alpha)` (attribute)

**Returns:** an object

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

### 6.13.4 ImageEmbedding (for IsCapCategoryMorphism)

▷ `ImageEmbedding(alpha)` (attribute)

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

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

### 6.13.5 ImageEmbeddingWithGivenImageObject (for IsCapCategoryMorphism, IsCapCategoryObject)

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

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

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

### 6.13.6 CostrictionToImage (for IsCapCategoryMorphism)

▷ `CostrictionToImage(alpha)` (attribute)  
**Returns:** a morphism in  $\text{Hom}(A, \text{im}(\alpha))$   
 The argument is a morphism  $\alpha : A \rightarrow B$ . The output is the costriction to image  $c : A \rightarrow \text{im}(\alpha)$ .

### 6.13.7 CostrictionToImageWithGivenImageObject (for IsCapCategoryMorphism, IsCapCategoryObject)

▷ `CostrictionToImageWithGivenImageObject(alpha, I)` (operation)  
**Returns:** a morphism in  $\text{Hom}(A, I)$   
 The argument is a morphism  $\alpha : A \rightarrow B$  and an object  $I = \text{im}(\alpha)$ . The output is the costriction to image  $c : A \rightarrow I$ .

### 6.13.8 UniversalMorphismFromImage (for IsCapCategoryMorphism, IsList)

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

### 6.13.9 UniversalMorphismFromImageWithGivenImageObject (for IsCapCategoryMorphism, IsList, IsCapCategoryObject)

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

### 6.13.10 ImageObjectFunctorial (for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `ImageObjectFunctorial(alpha, nu, alpha_prime)` (operation)  
**Returns:** a morphism in  $\text{Hom}(\text{ImageObject}(\alpha), \text{ImageObject}(\alpha'))$   
 The arguments are three morphisms  $\alpha : A \rightarrow B$ ,  $\nu : B \rightarrow B'$ ,  $\alpha' : A' \rightarrow B'$ . The output is the morphism  $\text{ImageObject}(\alpha) \rightarrow \text{ImageObject}(\alpha')$  given by the functoriality of the image.

### 6.13.11 ImageObjectFunctorialWithGivenImageObjects (for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject)

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

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

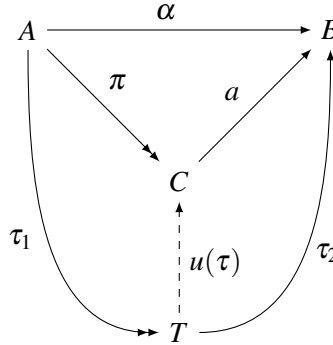
The arguments are an object  $s = \text{ImageObject}(\alpha)$ , three morphisms  $\alpha : A \rightarrow B$ ,  $v : B \rightarrow B'$ ,  $\alpha' : A' \rightarrow B'$ , and an object  $r = \text{ImageObject}(\alpha')$ . The output is the morphism  $\text{ImageObject}(\alpha) \rightarrow \text{ImageObject}(\alpha')$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

## 6.14 Coimage

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

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

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



### 6.14.1 MorphismFromCoimageToImage (for IsCapCategoryMorphism)

▷ `MorphismFromCoimageToImage(alpha)`

(attribute)

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

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

### 6.14.2 MorphismFromCoimageToImageWithGivenObjects (for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryObject)

▷ `MorphismFromCoimageToImageWithGivenObjects(C, alpha, I)`

(operation)

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

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

### 6.14.3 InverseOfMorphismFromCoimageToImage (for IsCapCategoryMorphism)

▷ `InverseOfMorphismFromCoimageToImage(alpha)` (attribute)

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

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

### 6.14.4 InverseOfMorphismFromCoimageToImageWithGivenObjects (for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryObject)

▷ `InverseOfMorphismFromCoimageToImageWithGivenObjects(I, alpha, C)` (operation)

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

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

### 6.14.5 IsomorphismFromCoimageToCokernelOfKernel (for IsCapCategoryMorphism)

▷ `IsomorphismFromCoimageToCokernelOfKernel(alpha)` (attribute)

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

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

### 6.14.6 IsomorphismFromCokernelOfKernelToCoimage (for IsCapCategoryMorphism)

▷ `IsomorphismFromCokernelOfKernelToCoimage(alpha)` (attribute)

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

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

### 6.14.7 CoimageObject (for IsCapCategoryMorphism)

▷ `CoimageObject(alpha)` (attribute)

**Returns:** an object

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

### 6.14.8 CoimageProjection (for IsCapCategoryMorphism)

▷ `CoimageProjection(alpha)` (attribute)

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

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

### 6.14.9 CoimageProjectionWithGivenCoimageObject (for IsCapCategoryMorphism, IsCapCategoryObject)

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

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

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

#### 6.14.10 AstrictionToCoimage (for IsCapCategoryMorphism)

▷ `AstrictionToCoimage(alpha)` (attribute)  
**Returns:** a morphism in  $\text{Hom}(\text{coim}(\alpha), B)$   
 The argument is a morphism  $\alpha : A \rightarrow B$ . The output is the astriction to coimage  $a : \text{coim}(\alpha) \rightarrow B$ .

#### 6.14.11 AstrictionToCoimageWithGivenCoimageObject (for IsCapCategoryMorphism, IsCapCategoryObject)

▷ `AstrictionToCoimageWithGivenCoimageObject(alpha, C)` (operation)  
**Returns:** a morphism in  $\text{Hom}(C, B)$   
 The argument are a morphism  $\alpha : A \rightarrow B$  and an object  $C = \text{coim}(\alpha)$ . The output is the astriction to coimage  $a : C \rightarrow B$ .

#### 6.14.12 UniversalMorphismIntoCoimage (for IsCapCategoryMorphism, IsList)

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

#### 6.14.13 UniversalMorphismIntoCoimageWithGivenCoimageObject (for IsCapCategoryMorphism, IsList, IsCapCategoryObject)

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

Whenever the `CoastrictionToImage` is an epi, or the `AstrictionToCoimage` is a mono, there is a canonical morphism from the image to the coimage. If this canonical morphism is an isomorphism, we call it the *canonical identification* (between image and coimage).

#### 6.14.14 CoimageObjectFunctorial (for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `CoimageObjectFunctorial(alpha, mu, alpha_prime)` (operation)  
**Returns:** a morphism in  $\text{Hom}(\text{CoimageObject}(\alpha), \text{CoimageObject}(\alpha'))$   
 The arguments are three morphisms  $\alpha : A \rightarrow B, \mu : A \rightarrow A', \alpha' : A' \rightarrow B'$ . The output is the morphism  $\text{CoimageObject}(\alpha) \rightarrow \text{CoimageObject}(\alpha')$  given by the functoriality of the coimage.



### 6.14.15 CoimageObjectFunctorialWithGivenCoimageObjects (for IsCapCategory-Object, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject)

▷ CoimageObjectFunctorialWithGivenCoimageObjects( $s$ ,  $\alpha$ ,  $\mu$ ,  $\alpha_{\text{prime}}$ ,  $r$ )  
(operation)

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

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

## 6.15 Homology objects

In an abelian category, we can define the operation that takes as an input a pair of morphisms  $\alpha : A \rightarrow B, \beta : B \rightarrow C$  and outputs the subquotient of  $B$  given by

$$\bullet H := \text{KernelObject}(\beta) / (\text{KernelObject}(\beta) \cap \text{ImageObject}(\alpha)).$$

This object is called a *homology object* of the pair  $\alpha, \beta$ . Note that we do not need the precomposition of  $\alpha$  and  $\beta$  to be zero in order to make sense of this notion. Moreover, given a second pair  $\gamma : D \rightarrow E, \delta : E \rightarrow F$  of morphisms, and a morphism  $\varepsilon : B \rightarrow E$  such that there exists  $\omega_1 : A \rightarrow D, \omega_2 : C \rightarrow F$  with  $\varepsilon \circ \alpha \sim_{A,E} \gamma \circ \omega_1$  and  $\omega_2 \circ \beta \sim_{B,F} \delta \circ \varepsilon$  there is a functorial way to obtain from these data a morphism between the two corresponding homology objects.

### 6.15.1 HomologyObject (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ HomologyObject( $\alpha$ ,  $\beta$ )  
(operation)

**Returns:** an object

The arguments are two morphisms  $\alpha : A \rightarrow B, \beta : B \rightarrow C$ . The output is the homology object  $H$  of this pair.

### 6.15.2 HomologyObjectFunctorial (for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ HomologyObjectFunctorial( $\alpha$ ,  $\beta$ ,  $\varepsilon$ ,  $\gamma$ ,  $\delta$ )  
(operation)

**Returns:** a morphism in  $\text{Hom}(H_1, H_2)$

The argument are five morphisms  $\alpha : A \rightarrow B, \beta : B \rightarrow C, \varepsilon : B \rightarrow E, \gamma : D \rightarrow E, \delta : E \rightarrow F$  such that there exists  $\omega_1 : A \rightarrow D, \omega_2 : C \rightarrow F$  with  $\varepsilon \circ \alpha \sim_{A,E} \gamma \circ \omega_1$  and  $\omega_2 \circ \beta \sim_{B,F} \delta \circ \varepsilon$ . The output is the functorial morphism induced by  $\varepsilon$  between the corresponding homology objects  $H_1$  and  $H_2$ , where  $H_1$  denotes the homology object of the pair  $\alpha, \beta$ , and  $H_2$  denotes the homology object of the pair  $\gamma, \delta$ .

### 6.15.3 HomologyObjectFunctorialWithGivenHomologyObjects (for IsCapCategory-Object, IsList, IsCapCategoryObject)

▷ HomologyObjectFunctorialWithGivenHomologyObjects( $H_1$ ,  $L$ ,  $H_2$ )  
(operation)

**Returns:** a morphism in  $\text{Hom}(H_1, H_2)$

The arguments are an object  $H_1$ , a list  $L$  consisting of five morphisms  $\alpha : A \rightarrow B$ ,  $\beta : B \rightarrow C$ ,  $\varepsilon : B \rightarrow E$ ,  $\gamma : D \rightarrow E$ ,  $\delta : E \rightarrow F$ , and an object  $H_2$ , such that  $H_1 = \text{HomologyObject}(\alpha, \beta)$  and  $H_2 = \text{HomologyObject}(\gamma, \delta)$ , and such that there exists  $\omega_1 : A \rightarrow D$ ,  $\omega_2 : C \rightarrow F$  with  $\varepsilon \circ \alpha \sim_{A,E} \gamma \circ \omega_1$  and  $\omega_2 \circ \beta \sim_{B,F} \delta \circ \varepsilon$ . The output is the functorial morphism induced by  $\varepsilon$  between the corresponding homology objects  $H_1$  and  $H_2$ , where  $H_1$  denotes the homology object of the pair  $\alpha, \beta$ , and  $H_2$  denotes the homology object of the pair  $\gamma, \delta$ .

#### 6.15.4 IsomorphismFromHomologyObjectToItsConstructionAsAnImageObject (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

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

The arguments are two morphisms  $\alpha : A \rightarrow B, \beta : B \rightarrow C$ . The output is the natural isomorphism from the homology object  $H$  of  $\alpha$  and  $\beta$  to the construction of the homology object as `ImageObject(PreCompose(KernelEmbedding( $\beta$ ), CokernelProjection( $\alpha$ )))`, denoted by  $I$ .

#### 6.15.5 IsomorphismFromItsConstructionAsAnImageObjectToHomologyObject (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

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

The arguments are two morphisms  $\alpha : A \rightarrow B, \beta : B \rightarrow C$ . The output is the natural isomorphism from the construction of the homology object as `ImageObject(PreCompose(KernelEmbedding( $\beta$ ), CokernelProjection( $\alpha$ )))`, denoted by  $I$ , to the homology object  $H$  of  $\alpha$  and  $\beta$ .

### 6.16 Projective covers and injective envelopes

#### 6.16.1 ProjectiveCoverObject (for IsCapCategoryObject)

▷ `ProjectiveCoverObject(A)` (attribute)

**Returns:** an object

The argument is an object  $A$ . The output is a projective cover of  $A$ .

#### 6.16.2 EpimorphismFromProjectiveCoverObject (for IsCapCategoryObject)

▷ `EpimorphismFromProjectiveCoverObject(A)` (attribute)

**Returns:** an epimorphism

The argument is an object  $A$ . The output is an epimorphism from a projective cover of  $A$ .

#### 6.16.3 EpimorphismFromProjectiveCoverObjectWithGivenProjectiveCoverObject (for IsCapCategoryObject, IsCapCategoryObject)

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

**Returns:** an epimorphism

The argument is an object  $A$ . The output is the epimorphism from the projective cover  $P$  of  $A$ .

#### 6.16.4 **InjectiveEnvelopeObject (for IsCapCategoryObject)**

▷ `InjectiveEnvelopeObject(A)` (attribute)

**Returns:** an object

The argument is an object  $A$ . The output is an injective envelope of  $A$ .

#### 6.16.5 **MonomorphismIntoInjectiveEnvelopeObject (for IsCapCategoryObject)**

▷ `MonomorphismIntoInjectiveEnvelopeObject(A)` (attribute)

**Returns:** a monomorphism

The argument is an object  $A$ . The output is a monomorphism into an injective envelope of  $A$ .

#### 6.16.6 **MonomorphismIntoInjectiveEnvelopeObjectWithGivenInjectiveEnvelopeObject (for IsCapCategoryObject, IsCapCategoryObject)**

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

**Returns:** a monomorphism

The argument is an object  $A$ . The output is a monomorphism into an injective envelope  $I$  of  $A$ .

## 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 have the following syntax:

```
Code
DeclareOperation( "AddSomeFunc",
                  [ IsCapCategory, IsList, IsInt ] );
```

The first argument is the category to which some function (e.g. `KernelObject`) is added, the second is a list containing pairs of functions and additional filters for the arguments, (e.g. if one argument is a morphism, an additional filter could be `IsMorphism`). The third is an optional weight which will then be the weight for `SomeFunc` (default value: 100). This is described later. If only one function is to be installed, the list can be replaced by the function. CAP installs the given function(s) as methods for `SomeFunc` (resp. `SomeFuncOp` if `SomeFunc` is not an operation).

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`.

- **Prefunction:** The prefunction should be used for error handling and plausibility checks of the input to `SomeFunc` (e.g. for `KernelLift` it should check whether range and source of the morphisms coincide). Generally, the prefunction is defined in the method record and only depends on the name `SomeFunc`. It is called with the same input as the function itself, and should return either `[ true ]`, which continues the computation, or `[ false, "message" ]`, which will cause an error with message "message" and some additional information.
- **Full prefunction:** The full prefunction 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 `KernelEmbedding`, it will set the `KernelObject` of the input morphism to result. The postfunction is also taken from the method record and does only depend on the name `SomeFunc`.
- **Addfunction:** If the result is a category cell, it is added to the category for which the function was installed. This is disabled by default and can be enabled via `EnableAddForCategoricalOperations` (1.12.1).

## 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:

- **Default weight:** If the weight parameter is `-1`, the default weight is assumed, which is `100`.
- **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.
- **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.

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:

```
Code
DeclareOperation( "CapInternalInstallAdd",
                  [ IsRecord ] );
```

The record can have the following components:

```
Code
[
    "filter_list",
    "input_arguments_names",
    "return_type",
    "output_source_getter_string",
    "output_source_getter_preconditions",
    "output_range_getter_string",
    "output_range_getter_preconditions",
    "with_given_object_position",
    "dual_operation",
    "dual_arguments_reversed",
    "dual_with_given_objects_reversed",
    "dual_preprocessor_func",
    "dual_preprocessor_func_string",
    "dual_postprocessor_func",
    "dual_postprocessor_func_string",
    "functorial",
    "compatible_with_congruence_of_morphisms",
    "redirect_function",
    "pre_function",
    "pre_function_full",
    "post_function",
]
```

- `pre_function` (optional): A function which is used as the prefunction of the installed methods, as described above. Can also be the name of another operation. In this case the pre function of the referenced operation is used.
- `pre_function_full` (optional): A function which is used as the full prefunction of the installed methods, as described above. Can also be the name of another operation. In this case the full pre function of the referenced operation is used.
- `redirect_function` (optional): A function which is used as the redirect function of the installed methods, as described above. Can also be the name of another operation. In this case the redirect function of the referenced operation is used.
- `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 entries are filters, or the strings listed below, which will be replaced by appropriate filters at the time the add method is called. The first entry of `filter_list` must be the string category. If the category can be inferred from the remaining arguments, a convenience method without the category as the first argument is installed automatically.

```

- category,
- object,
- morphism,
- twocell,
- object_in_range_category_of_homomorphism_structure,
- morphism_in_range_category_of_homomorphism_structure,
- list_of_objects,
- list_of_morphisms,
- list_of_twocells.

```

- `return_type`: The return type can either be a filter or one of the strings in the list below. For objects, morphisms and 2-cells the correct Add function (see above) is used for the result of the computation. Otherwise, no Add function is used after all.

Code

```

[
    "object",
    "object_or_fail",
    "morphism",
    "morphism_or_fail",
    "twocell",
    "object_in_range_category_of_homomorphism_structure",
    "morphism_in_range_category_of_homomorphism_structure",
    "bool",
    "list_of_objects",
    "list_of_morphisms",
    "list_of_morphisms_or_fail",
    "list_of_lists_of_morphisms",
    "object_datum",
    "morphism_datum",
    "nonneg_integer_or_infinity",
    "list_of_elements_of_commutative_ring_of_linear_structure",
]

```

- `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.
- `dual_operation` (optional): Name of the dual operation.
- `dual_arguments_reversed` (optional): Boolean, marks whether for the call of the dual operation all arguments have to be given in reversed order.

- `dual_with_given_objects_reversed` (optional): Boolean, marks whether for the call of the dual operation the source and range of a with given operation have to be given in reversed order.
- `dual_preprocessor_func[_string]` (optional): 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. The function can also be given as a string.
- `dual_postprocessor_func[_string]` (optional): 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. The function can also be given as a string.
- `input_arguments_names` (optional): A duplicate free list (of the same length as `filter_list`) of strings. For example, these strings will be used as the names of the arguments when automatically generating functions for this operation, e.g. in the opposite category.
- `output_source_getter_string` (optional): Only valid if the operation returns a morphism: a piece of GAP code which computes the source of the returned morphism. The input arguments are available via the names given in `input_arguments_names`.
- `output_source_getter_preconditions` (optional): Only valid if `output_source_getter_string` is set: The preconditions of `output_source_getter_string` in the same form as preconditions of derivations but with the CAP operations given as strings.
- `output_range_getter_string` (optional): Only valid if the operation returns a morphism: a piece of GAP code which computes the range of the returned morphism. The input arguments are available via the names given in `input_arguments_names`.
- `output_range_getter_preconditions` (optional): Only valid if `output_range_getter_string` is set: The preconditions of `output_range_getter_string` in the same form as preconditions of derivations but with the CAP operations given as strings.
- `with_given_object_position` (optional): One of the following strings: "Source", "Range", or "both". Set for the without given operation in a with given pair. Describes whether the source resp. range are given (as the last argument of the with given operation) or both (as the second and the last argument of the with given operation).
- `compatible_with_congruence_of_morphisms` (optional): Indicates if the operation is compatible with the congruence of morphisms, that is, if the output does not change with regard to `IsEqualForObjects` and `IsCongruentForMorphisms` if the input changes with regard to `IsEqualForObjects` and `IsCongruentForMorphisms`.

Using all those entries, the operation `CapInternalInstallAdd` installs add methods as described above. It first provides plausibility checks 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.

### 7.3.1 CapInternalInstallAdd

▷ `CapInternalInstallAdd(record)`

(function)

See 7.3.



## 7.4 Enhancing the method name record

The function `CAP_INTERNAL_ENHANCE_NAME_RECORD` can be applied to a method name record to make the following enhancements:

- Function name: Set the component `function_name` to the entry name.
- WithGiven special case: If the current entry belongs to a WithGiven operation or its without given pair, the `with_given_without_given_name_pair` is set. Additionally, the with given flag of the WithGiven operation is set to true.
- Redirect and post functions are created for all operations belonging to universal constructions (e.g. `KernelLift`) which are not a WithGiven operation.

## 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.

## 7.6 Available Add functions

### 7.6.1 AddAdditionForMorphisms (for IsCapCategory, IsFunction)

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

### 7.6.2 AddAdditiveGenerators (for IsCapCategory, IsFunction)

▷ AddAdditiveGenerators(*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 AdditiveGenerators.  $F : () \mapsto \text{AdditiveGenerators}()$ .

### 7.6.3 AddAdditiveInverseForMorphisms (for IsCapCategory, IsFunction)

▷ AddAdditiveInverseForMorphisms(*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 AdditiveInverseForMorphisms.  $F : (\alpha) \mapsto \text{AdditiveInverseForMorphisms}(\alpha)$ .

### 7.6.4 AddAstrictionToCoimage (for IsCapCategory, IsFunction)

▷ AddAstrictionToCoimage(*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 AstrictionToCoimage.  $F : (\alpha) \mapsto \text{AstrictionToCoimage}(\alpha)$ .

### 7.6.5 AddAstrictionToCoimageWithGivenCoimageObject (for IsCapCategory, IsFunction)

▷ AddAstrictionToCoimageWithGivenCoimageObject(*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 AstrictionToCoimageWithGivenCoimageObject.  $F : (\alpha, C) \mapsto \text{AstrictionToCoimageWithGivenCoimageObject}(\alpha, C)$ .

### 7.6.6 AddBasisOfExternalHom (for IsCapCategory, IsFunction)

▷ AddBasisOfExternalHom( $C, F$ ) (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 BasisOfExternalHom.  $F : (arg2, arg3) \mapsto \text{BasisOfExternalHom}(arg2, arg3)$ .

### 7.6.7 AddCostrictionToImage (for IsCapCategory, IsFunction)

▷ AddCostrictionToImage( $C, F$ ) (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 CostrictionToImage.  $F : (alpha) \mapsto \text{CostrictionToImage}(alpha)$ .

### 7.6.8 AddCostrictionToImageWithGivenImageObject (for IsCapCategory, IsFunction)

▷ AddCostrictionToImageWithGivenImageObject( $C, F$ ) (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 CostrictionToImageWithGivenImageObject.  $F : (alpha, I) \mapsto \text{CostrictionToImageWithGivenImageObject}(alpha, I)$ .

### 7.6.9 AddCoefficientsOfMorphism (for IsCapCategory, IsFunction)

▷ AddCoefficientsOfMorphism( $C, F$ ) (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 CoefficientsOfMorphism.  $F : (arg2) \mapsto \text{CoefficientsOfMorphism}(arg2)$ .

### 7.6.10 AddCoequalizer (for IsCapCategory, IsFunction)

▷ AddCoequalizer( $C, F$ ) (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 Coequalizer.  $F : (Y, morphisms) \mapsto \text{Coequalizer}(Y, morphisms)$ .

### 7.6.11 AddCoequalizerFunctorial (for IsCapCategory, IsFunction)

▷ AddCoequalizerFunctorial( $C, F$ ) (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 CoequalizerFunctorial.  $F : (morphisms, mu, morphismsp) \mapsto \text{CoequalizerFunctorial}(morphisms, mu, morphismsp)$ .

### 7.6.12 AddCoequalizerFunctorialWithGivenCoequalizers (for IsCapCategory, IsFunction)

▷ AddCoequalizerFunctorialWithGivenCoequalizers( $C, F$ ) (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 CoequalizerFunctorialWithGivenCoequalizers.  $F : (P, morphisms, mu, morphismsp, Pp) \mapsto \text{CoequalizerFunctorialWithGivenCoequalizers}(P, morphisms, mu, morphismsp, Pp)$ .

### 7.6.13 AddCoimageObject (for IsCapCategory, IsFunction)

▷ AddCoimageObject( $C, F$ ) (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 CoimageObject.  $F : (arg2) \mapsto \text{CoimageObject}(arg2)$ .

### 7.6.14 AddCoimageObjectFunctorial (for IsCapCategory, IsFunction)

▷ AddCoimageObjectFunctorial( $C, F$ ) (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 CoimageObjectFunctorial.  $F : (alpha, mu, alphap) \mapsto \text{CoimageObjectFunctorial}(alpha, mu, alphap)$ .

### 7.6.15 AddCoimageObjectFunctorialWithGivenCoimageObjects (for IsCapCategory, IsFunction)

▷ AddCoimageObjectFunctorialWithGivenCoimageObjects( $C, F$ ) (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 CoimageObjectFunctorialWithGivenCoimageObjects.  $F : (C, alpha, mu, alphap, Cp) \mapsto \text{CoimageObjectFunctorialWithGivenCoimageObjects}(C, alpha, mu, alphap, Cp)$ .

### 7.6.16 AddCoimageProjection (for IsCapCategory, IsFunction)

▷ AddCoimageProjection( $C, F$ ) (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 CoimageProjection.  $F : (alpha) \mapsto \text{CoimageProjection}(alpha)$ .

### 7.6.17 AddCoimageProjectionWithGivenCoimageObject (for IsCapCategory, IsFunction)

▷ AddCoimageProjectionWithGivenCoimageObject( $C, F$ ) (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 `CoimageProjectionWithGivenCoimageObject`.  $F : (\alpha, C) \mapsto \text{CoimageProjectionWithGivenCoimageObject}(\alpha, C)$ .

### 7.6.18 AddCokernelColift (for IsCapCategory, IsFunction)

▷ `AddCokernelColift(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 `CokernelColift`.  $F : (\alpha, T, \tau) \mapsto \text{CokernelColift}(\alpha, T, \tau)$ .

### 7.6.19 AddCokernelColiftWithGivenCokernelObject (for IsCapCategory, IsFunction)

▷ `AddCokernelColiftWithGivenCokernelObject(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 `CokernelColiftWithGivenCokernelObject`.  $F : (\alpha, T, \tau, P) \mapsto \text{CokernelColiftWithGivenCokernelObject}(\alpha, T, \tau, P)$ .

### 7.6.20 AddCokernelObject (for IsCapCategory, IsFunction)

▷ `AddCokernelObject(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 `CokernelObject`.  $F : (\alpha) \mapsto \text{CokernelObject}(\alpha)$ .

### 7.6.21 AddCokernelObjectFunctorial (for IsCapCategory, IsFunction)

▷ `AddCokernelObjectFunctorial(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 `CokernelObjectFunctorial`.  $F : (\alpha, \mu, \alpha\text{phap}) \mapsto \text{CokernelObjectFunctorial}(\alpha, \mu, \alpha\text{phap})$ .

### 7.6.22 AddCokernelObjectFunctorialWithGivenCokernelObjects (for IsCapCategory, IsFunction)

▷ `AddCokernelObjectFunctorialWithGivenCokernelObjects(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 `CokernelObjectFunctorialWithGivenCokernelObjects`.  $F : (P, \alpha, \mu, \alpha\text{phap}, Pp) \mapsto \text{CokernelObjectFunctorialWithGivenCokernelObjects}(P, \alpha, \mu, \alpha\text{phap}, Pp)$ .

### 7.6.23 AddCokernelProjection (for IsCapCategory, IsFunction)

▷ AddCokernelProjection( $C, F$ ) (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 CokernelProjection.  $F : (\alpha) \mapsto \text{CokernelProjection}(\alpha)$ .

### 7.6.24 AddCokernelProjectionWithGivenCokernelObject (for IsCapCategory, IsFunction)

▷ AddCokernelProjectionWithGivenCokernelObject( $C, F$ ) (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 CokernelProjectionWithGivenCokernelObject.  $F : (\alpha, P) \mapsto \text{CokernelProjectionWithGivenCokernelObject}(\alpha, P)$ .

### 7.6.25 AddColift (for IsCapCategory, IsFunction)

▷ AddColift( $C, F$ ) (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 Colift.  $F : (\alpha, \beta) \mapsto \text{Colift}(\alpha, \beta)$ .

### 7.6.26 AddColiftAlongEpimorphism (for IsCapCategory, IsFunction)

▷ AddColiftAlongEpimorphism( $C, F$ ) (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 ColiftAlongEpimorphism.  $F : (\epsilon, \tau) \mapsto \text{ColiftAlongEpimorphism}(\epsilon, \tau)$ .

### 7.6.27 AddColiftOrFail (for IsCapCategory, IsFunction)

▷ AddColiftOrFail( $C, F$ ) (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 ColiftOrFail.  $F : (\alpha, \beta) \mapsto \text{ColiftOrFail}(\alpha, \beta)$ .

### 7.6.28 AddComponentOfMorphismFromCoproduct (for IsCapCategory, IsFunction)

▷ AddComponentOfMorphismFromCoproduct( $C, F$ ) (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 ComponentOfMorphismFromCoproduct.  $F : (\alpha, I, i) \mapsto \text{ComponentOfMorphismFromCoproduct}(\alpha, I, i)$ .

### 7.6.29 AddComponentOfMorphismFromDirectSum (for IsCapCategory, IsFunction)

▷ AddComponentOfMorphismFromDirectSum( $C, F$ ) (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 ComponentOfMorphismFromDirectSum.  $F : (\alpha, S, i) \mapsto \text{ComponentOfMorphismFromDirectSum}(\alpha, S, i)$ .

### 7.6.30 AddComponentOfMorphismIntoDirectProduct (for IsCapCategory, IsFunction)

▷ AddComponentOfMorphismIntoDirectProduct( $C, F$ ) (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 ComponentOfMorphismIntoDirectProduct.  $F : (\alpha, P, i) \mapsto \text{ComponentOfMorphismIntoDirectProduct}(\alpha, P, i)$ .

### 7.6.31 AddComponentOfMorphismIntoDirectSum (for IsCapCategory, IsFunction)

▷ AddComponentOfMorphismIntoDirectSum( $C, F$ ) (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 ComponentOfMorphismIntoDirectSum.  $F : (\alpha, S, i) \mapsto \text{ComponentOfMorphismIntoDirectSum}(\alpha, S, i)$ .

### 7.6.32 AddCoproduct (for IsCapCategory, IsFunction)

▷ AddCoproduct( $C, F$ ) (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 Coproduct.  $F : (objects) \mapsto \text{Coproduct}(objects)$ .

### 7.6.33 AddCoproductFunctorial (for IsCapCategory, IsFunction)

▷ AddCoproductFunctorial( $C, F$ ) (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 CoproductFunctorial.  $F : (objects, L, objectsp) \mapsto \text{CoproductFunctorial}(objects, L, objectsp)$ .

### 7.6.34 AddCoproductFunctorialWithGivenCoproducts (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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

CoproductFunctorialWithGivenCoproducts.  $F : (P, objects, L, objectsp, Pp) \mapsto$   
 CoproductFunctorialWithGivenCoproducts( $P, objects, L, objectsp, Pp$ ).

### 7.6.35 AddDirectProduct (for IsCapCategory, IsFunction)

▷ AddDirectProduct( $C, F$ ) (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 DirectProduct.  $F : (objects) \mapsto \text{DirectProduct}(objects)$ .

### 7.6.36 AddDirectProductFunctorial (for IsCapCategory, IsFunction)

▷ AddDirectProductFunctorial( $C, F$ ) (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 DirectProductFunctorial.  $F : (objects, L, objectsp) \mapsto \text{DirectProductFunctorial}(objects, L, objectsp)$ .

### 7.6.37 AddDirectProductFunctorialWithGivenDirectProducts (for IsCapCategory, IsFunction)

▷ AddDirectProductFunctorialWithGivenDirectProducts( $C, F$ ) (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 DirectProductFunctorialWithGivenDirectProducts.  $F : (P, objects, L, objectsp, Pp) \mapsto \text{DirectProductFunctorialWithGivenDirectProducts}(P, objects, L, objectsp, Pp)$ .

### 7.6.38 AddDirectSum (for IsCapCategory, IsFunction)

▷ AddDirectSum( $C, F$ ) (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 DirectSum.  $F : (objects) \mapsto \text{DirectSum}(objects)$ .

### 7.6.39 AddDirectSumFunctorial (for IsCapCategory, IsFunction)

▷ AddDirectSumFunctorial( $C, F$ ) (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 DirectSumFunctorial.  $F : (objects, L, objectsp) \mapsto \text{DirectSumFunctorial}(objects, L, objectsp)$ .

### 7.6.40 AddDirectSumFunctorialWithGivenDirectSums (for IsCapCategory, IsFunction)

▷ AddDirectSumFunctorialWithGivenDirectSums( $C, F$ ) (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 `DirectSumFunctorialWithGivenDirectSums`.  $F : (P, objects, L, objectsp, Pp) \mapsto \text{DirectSumFunctorialWithGivenDirectSums}(P, objects, L, objectsp, Pp)$ .

#### 7.6.41 AddDistinguishedObjectOfHomomorphismStructure (for IsCapCategory, IsFunction)

▷ `AddDistinguishedObjectOfHomomorphismStructure(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 `DistinguishedObjectOfHomomorphismStructure`.  $F : () \mapsto \text{DistinguishedObjectOfHomomorphismStructure}()$ .

#### 7.6.42 AddEmbeddingOfEqualizer (for IsCapCategory, IsFunction)

▷ `AddEmbeddingOfEqualizer(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 `EmbeddingOfEqualizer`.  $F : (Y, morphisms) \mapsto \text{EmbeddingOfEqualizer}(Y, morphisms)$ .

#### 7.6.43 AddEmbeddingOfEqualizerWithGivenEqualizer (for IsCapCategory, IsFunction)

▷ `AddEmbeddingOfEqualizerWithGivenEqualizer(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 `EmbeddingOfEqualizerWithGivenEqualizer`.  $F : (Y, morphisms, P) \mapsto \text{EmbeddingOfEqualizerWithGivenEqualizer}(Y, morphisms, P)$ .

#### 7.6.44 AddEpimorphismFromProjectiveCoverObject (for IsCapCategory, IsFunction)

▷ `AddEpimorphismFromProjectiveCoverObject(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 `EpimorphismFromProjectiveCoverObject`.  $F : (A) \mapsto \text{EpimorphismFromProjectiveCoverObject}(A)$ .

#### 7.6.45 AddEpimorphismFromProjectiveCoverObjectWithGivenProjectiveCoverObject (for IsCapCategory, IsFunction)

▷ `AddEpimorphismFromProjectiveCoverObjectWithGivenProjectiveCoverObject(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

EpimorphismFromProjectiveCoverObjectWithGivenProjectiveCoverObject.  $F : (A, P) \mapsto \text{EpimorphismFromProjectiveCoverObjectWithGivenProjectiveCoverObject}(A, P)$ .

#### 7.6.46 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.  $F : (A) \mapsto \text{EpimorphismFromSomeProjectiveObject}(A)$ .

#### 7.6.47 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 EpimorphismFromSomeProjectiveObjectWithGivenSomeProjectiveObject.  $F : (A, P) \mapsto \text{EpimorphismFromSomeProjectiveObjectWithGivenSomeProjectiveObject}(A, P)$ .

#### 7.6.48 AddEqualizer (for IsCapCategory, IsFunction)

▷ AddEqualizer( $C, F$ ) (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 Equalizer.  $F : (Y, \text{morphisms}) \mapsto \text{Equalizer}(Y, \text{morphisms})$ .

#### 7.6.49 AddEqualizerFunctorial (for IsCapCategory, IsFunction)

▷ AddEqualizerFunctorial( $C, F$ ) (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 EqualizerFunctorial.  $F : (\text{morphisms}, \mu, \text{morphismsp}) \mapsto \text{EqualizerFunctorial}(\text{morphisms}, \mu, \text{morphismsp})$ .

#### 7.6.50 AddEqualizerFunctorialWithGivenEqualizers (for IsCapCategory, IsFunction)

▷ AddEqualizerFunctorialWithGivenEqualizers( $C, F$ ) (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 EqualizerFunctorialWithGivenEqualizers.  $F : (P, \text{morphisms}, \mu, \text{morphismsp}, Pp) \mapsto \text{EqualizerFunctorialWithGivenEqualizers}(P, \text{morphisms}, \mu, \text{morphismsp}, Pp)$ .

### 7.6.51 AddFiberProduct (for IsCapCategory, IsFunction)

▷ AddFiberProduct( $C, F$ ) (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 FiberProduct.  $F : (morphisms) \mapsto \text{FiberProduct}(morphisms)$ .

### 7.6.52 AddFiberProductFunctorial (for IsCapCategory, IsFunction)

▷ AddFiberProductFunctorial( $C, F$ ) (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 FiberProductFunctorial.  $F : (morphisms, L, morphismsp) \mapsto \text{FiberProductFunctorial}(morphisms, L, morphismsp)$ .

### 7.6.53 AddFiberProductFunctorialWithGivenFiberProducts (for IsCapCategory, IsFunction)

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

**Returns:** nothing

The arguments are a category  $C$  and a function  $F$ . This operation adds the given function  $F$  to the category for the basic operation FiberProductFunctorialWithGivenFiberProducts.  $F : (P, morphisms, L, morphismsp, Pp) \mapsto \text{FiberProductFunctorialWithGivenFiberProducts}(P, morphisms, L, morphismsp, Pp)$ .

### 7.6.54 AddHomologyObject (for IsCapCategory, IsFunction)

▷ AddHomologyObject( $C, F$ ) (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 HomologyObject.  $F : (alpha, beta) \mapsto \text{HomologyObject}(alpha, beta)$ .

### 7.6.55 AddHomologyObjectFunctorialWithGivenHomologyObjects (for IsCapCategory, IsFunction)

▷ AddHomologyObjectFunctorialWithGivenHomologyObjects( $C, F$ ) (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 HomologyObjectFunctorialWithGivenHomologyObjects.  $F : (H_1, L, H_2) \mapsto \text{HomologyObjectFunctorialWithGivenHomologyObjects}(H_1, L, H_2)$ .

### 7.6.56 AddHomomorphismStructureOnMorphisms (for IsCapCategory, IsFunction)

▷ AddHomomorphismStructureOnMorphisms( $C, F$ ) (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 `HomomorphismStructureOnMorphisms`.  $F : (\alpha, \beta) \mapsto \text{HomomorphismStructureOnMorphisms}(\alpha, \beta)$ .

### 7.6.57 AddHomomorphismStructureOnMorphismsWithGivenObjects (for IsCapCategory, IsFunction)

▷ `AddHomomorphismStructureOnMorphismsWithGivenObjects(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 `HomomorphismStructureOnMorphismsWithGivenObjects`.  $F : (\text{source}, \alpha, \beta, \text{range}) \mapsto \text{HomomorphismStructureOnMorphismsWithGivenObjects}(\text{source}, \alpha, \beta, \text{range})$ .

### 7.6.58 AddHomomorphismStructureOnObjects (for IsCapCategory, IsFunction)

▷ `AddHomomorphismStructureOnObjects(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 `HomomorphismStructureOnObjects`.  $F : (\text{arg2}, \text{arg3}) \mapsto \text{HomomorphismStructureOnObjects}(\text{arg2}, \text{arg3})$ .

### 7.6.59 AddHorizontalPostCompose (for IsCapCategory, IsFunction)

▷ `AddHorizontalPostCompose(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 `HorizontalPostCompose`.  $F : (\text{arg2}, \text{arg3}) \mapsto \text{HorizontalPostCompose}(\text{arg2}, \text{arg3})$ .

### 7.6.60 AddHorizontalPreCompose (for IsCapCategory, IsFunction)

▷ `AddHorizontalPreCompose(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 `HorizontalPreCompose`.  $F : (\text{arg2}, \text{arg3}) \mapsto \text{HorizontalPreCompose}(\text{arg2}, \text{arg3})$ .

### 7.6.61 AddIdentityMorphism (for IsCapCategory, IsFunction)

▷ `AddIdentityMorphism(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 `IdentityMorphism`.  $F : (a) \mapsto \text{IdentityMorphism}(a)$ .

### 7.6.62 AddIdentityTwoCell (for IsCapCategory, IsFunction)

▷ AddIdentityTwoCell( $C, F$ ) (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 IdentityTwoCell.  $F : (arg2) \mapsto \text{IdentityTwoCell}(arg2)$ .

### 7.6.63 AddImageEmbedding (for IsCapCategory, IsFunction)

▷ AddImageEmbedding( $C, F$ ) (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 ImageEmbedding.  $F : (alpha) \mapsto \text{ImageEmbedding}(alpha)$ .

### 7.6.64 AddImageEmbeddingWithGivenImageObject (for IsCapCategory, IsFunction)

▷ AddImageEmbeddingWithGivenImageObject( $C, F$ ) (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 ImageEmbeddingWithGivenImageObject.  $F : (alpha, I) \mapsto \text{ImageEmbeddingWithGivenImageObject}(alpha, I)$ .

### 7.6.65 AddImageObject (for IsCapCategory, IsFunction)

▷ AddImageObject( $C, F$ ) (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 ImageObject.  $F : (arg2) \mapsto \text{ImageObject}(arg2)$ .

### 7.6.66 AddImageObjectFunctorial (for IsCapCategory, IsFunction)

▷ AddImageObjectFunctorial( $C, F$ ) (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 ImageObjectFunctorial.  $F : (alpha, nu, alphap) \mapsto \text{ImageObjectFunctorial}(alpha, nu, alphap)$ .

### 7.6.67 AddImageObjectFunctorialWithGivenImageObjects (for IsCapCategory, IsFunction)

▷ AddImageObjectFunctorialWithGivenImageObjects( $C, F$ ) (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 ImageObjectFunctorialWithGivenImageObjects.  $F : (I, alpha, nu, alphap, Ip) \mapsto \text{ImageObjectFunctorialWithGivenImageObjects}(I, alpha, nu, alphap, Ip)$ .

### 7.6.68 AddIndecomposableInjectiveObjects (for IsCapCategory, IsFunction)

▷ AddIndecomposableInjectiveObjects( $C$ ,  $F$ ) (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 IndecomposableInjectiveObjects.  $F : () \mapsto \text{IndecomposableInjectiveObjects}()$ .

### 7.6.69 AddIndecomposableProjectiveObjects (for IsCapCategory, IsFunction)

▷ AddIndecomposableProjectiveObjects( $C$ ,  $F$ ) (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 IndecomposableProjectiveObjects.  $F : () \mapsto \text{IndecomposableProjectiveObjects}()$ .

### 7.6.70 AddInitialObject (for IsCapCategory, IsFunction)

▷ AddInitialObject( $C$ ,  $F$ ) (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 InitialObject.  $F : () \mapsto \text{InitialObject}()$ .

### 7.6.71 AddInitialObjectFunctorial (for IsCapCategory, IsFunction)

▷ AddInitialObjectFunctorial( $C$ ,  $F$ ) (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 InitialObjectFunctorial.  $F : () \mapsto \text{InitialObjectFunctorial}()$ .

### 7.6.72 AddInitialObjectFunctorialWithGivenInitialObjects (for IsCapCategory, IsFunction)

▷ AddInitialObjectFunctorialWithGivenInitialObjects( $C$ ,  $F$ ) (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 InitialObjectFunctorialWithGivenInitialObjects.  $F : (P, Pp) \mapsto \text{InitialObjectFunctorialWithGivenInitialObjects}(P, Pp)$ .

### 7.6.73 AddInjectionOfCofactorOfCoproduct (for IsCapCategory, IsFunction)

▷ AddInjectionOfCofactorOfCoproduct( $C$ ,  $F$ ) (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 InjectionOfCofactorOfCoproduct.  $F : (objects, k) \mapsto \text{InjectionOfCofactorOfCoproduct}(objects, k)$ .

#### 7.6.74 AddInjectionOfCofactorOfCoproductWithGivenCoproduct (for IsCapCategory, IsFunction)

▷ AddInjectionOfCofactorOfCoproductWithGivenCoproduct( $C, F$ ) (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 InjectionOfCofactorOfCoproductWithGivenCoproduct.  $F : (objects, k, P) \mapsto \text{InjectionOfCofactorOfCoproductWithGivenCoproduct}(objects, k, P)$ .

#### 7.6.75 AddInjectionOfCofactorOfDirectSum (for IsCapCategory, IsFunction)

▷ AddInjectionOfCofactorOfDirectSum( $C, F$ ) (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 InjectionOfCofactorOfDirectSum.  $F : (objects, k) \mapsto \text{InjectionOfCofactorOfDirectSum}(objects, k)$ .

#### 7.6.76 AddInjectionOfCofactorOfDirectSumWithGivenDirectSum (for IsCapCategory, IsFunction)

▷ AddInjectionOfCofactorOfDirectSumWithGivenDirectSum( $C, F$ ) (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 InjectionOfCofactorOfDirectSumWithGivenDirectSum.  $F : (objects, k, P) \mapsto \text{InjectionOfCofactorOfDirectSumWithGivenDirectSum}(objects, k, P)$ .

#### 7.6.77 AddInjectionOfCofactorOfPushout (for IsCapCategory, IsFunction)

▷ AddInjectionOfCofactorOfPushout( $C, F$ ) (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 InjectionOfCofactorOfPushout.  $F : (morphisms, k) \mapsto \text{InjectionOfCofactorOfPushout}(morphisms, k)$ .

#### 7.6.78 AddInjectionOfCofactorOfPushoutWithGivenPushout (for IsCapCategory, IsFunction)

▷ AddInjectionOfCofactorOfPushoutWithGivenPushout( $C, F$ ) (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 InjectionOfCofactorOfPushoutWithGivenPushout.  $F : (morphisms, k, P) \mapsto \text{InjectionOfCofactorOfPushoutWithGivenPushout}(morphisms, k, P)$ .

#### 7.6.79 AddInjectiveColift (for IsCapCategory, IsFunction)

▷ AddInjectiveColift( $C, F$ ) (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 `InjectiveColift`.  $F : (\alpha, \beta) \mapsto \text{InjectiveColift}(\alpha, \beta)$ .

### 7.6.80 AddInjectiveDimension (for IsCapCategory, IsFunction)

▷ `AddInjectiveDimension(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 `InjectiveDimension`.  $F : (\alpha) \mapsto \text{InjectiveDimension}(\alpha)$ .

### 7.6.81 AddInjectiveEnvelopeObject (for IsCapCategory, IsFunction)

▷ `AddInjectiveEnvelopeObject(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 `InjectiveEnvelopeObject`.  $F : (\alpha) \mapsto \text{InjectiveEnvelopeObject}(\alpha)$ .

### 7.6.82 AddInterpretMorphismAsMorphismFromDistinguishedObjectToHomomorphismStructure (for IsCapCategory, IsFunction)

▷ `AddInterpretMorphismAsMorphismFromDistinguishedObjectToHomomorphismStructure(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 `InterpretMorphismAsMorphismFromDistinguishedObjectToHomomorphismStructure`.  $F : (\alpha) \mapsto \text{InterpretMorphismAsMorphismFromDistinguishedObjectToHomomorphismStructure}(\alpha)$ .

### 7.6.83 AddInterpretMorphismAsMorphismFromDistinguishedObjectToHomomorphismStructureWithGivenObjects (for IsCapCategory, IsFunction)

▷ `AddInterpretMorphismAsMorphismFromDistinguishedObjectToHomomorphismStructureWithGivenObjects(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 `InterpretMorphismAsMorphismFromDistinguishedObjectToHomomorphismStructureWithGivenObjects`.  $F : (\text{source}, \alpha, \text{range}) \mapsto \text{InterpretMorphismAsMorphismFromDistinguishedObjectToHomomorphismStructure}(\text{source}, \alpha, \text{range})$ .

### 7.6.84 AddInterpretMorphismFromDistinguishedObjectToHomomorphismStructureAsMorphism (for IsCapCategory, IsFunction)

▷ `AddInterpretMorphismFromDistinguishedObjectToHomomorphismStructureAsMorphism(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 `InterpretMorphismFromDistinguishedObjectToHomomorphismStructureAsMorphism`.  $F : (source, range, alpha) \mapsto \text{InterpretMorphismFromDistinguishedObjectToHomomorphismStructureAsMorphism}$

### 7.6.85 AddInverseForMorphisms (for IsCapCategory, IsFunction)

▷ `AddInverseForMorphisms( $C, F$ )` (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 `InverseForMorphisms`.  $F : (alpha) \mapsto \text{InverseForMorphisms}(alpha)$ .

### 7.6.86 AddInverseOfMorphismFromCoimageToImage (for IsCapCategory, IsFunction)

▷ `AddInverseOfMorphismFromCoimageToImage( $C, F$ )` (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 `InverseOfMorphismFromCoimageToImage`.  $F : (alpha) \mapsto \text{InverseOfMorphismFromCoimageToImage}(alpha)$ .

### 7.6.87 AddInverseOfMorphismFromCoimageToImageWithGivenObjects (for IsCapCategory, IsFunction)

▷ `AddInverseOfMorphismFromCoimageToImageWithGivenObjects( $C, F$ )` (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 `InverseOfMorphismFromCoimageToImageWithGivenObjects`.  $F : (I, alpha, C) \mapsto \text{InverseOfMorphismFromCoimageToImageWithGivenObjects}(I, alpha, C)$ .

### 7.6.88 AddIsAutomorphism (for IsCapCategory, IsFunction)

▷ `AddIsAutomorphism( $C, F$ )` (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 `IsAutomorphism`.  $F : (arg2) \mapsto \text{IsAutomorphism}(arg2)$ .

### 7.6.89 AddIsBijectiveObject (for IsCapCategory, IsFunction)

▷ `AddIsBijectiveObject( $C, F$ )` (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 `IsBijectiveObject`.  $F : (arg2) \mapsto \text{IsBijectiveObject}(arg2)$ .

### 7.6.90 AddIsCodominating (for IsCapCategory, IsFunction)

▷ AddIsCodominating( $C$ ,  $F$ ) (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 IsCodominating.  $F : (arg2, arg3) \mapsto \text{IsCodominating}(arg2, arg3)$ .

### 7.6.91 AddIsColiftable (for IsCapCategory, IsFunction)

▷ AddIsColiftable( $C$ ,  $F$ ) (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 IsColiftable.  $F : (arg2, arg3) \mapsto \text{IsColiftable}(arg2, arg3)$ .

### 7.6.92 AddIsColiftableAlongEpimorphism (for IsCapCategory, IsFunction)

▷ AddIsColiftableAlongEpimorphism( $C$ ,  $F$ ) (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 IsColiftableAlongEpimorphism.  $F : (arg2, arg3) \mapsto \text{IsColiftableAlongEpimorphism}(arg2, arg3)$ .

### 7.6.93 AddIsCongruentForMorphisms (for IsCapCategory, IsFunction)

▷ AddIsCongruentForMorphisms( $C$ ,  $F$ ) (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 IsCongruentForMorphisms.  $F : (arg2, arg3) \mapsto \text{IsCongruentForMorphisms}(arg2, arg3)$ .

### 7.6.94 AddIsDominating (for IsCapCategory, IsFunction)

▷ AddIsDominating( $C$ ,  $F$ ) (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 IsDominating.  $F : (arg2, arg3) \mapsto \text{IsDominating}(arg2, arg3)$ .

### 7.6.95 AddIsEndomorphism (for IsCapCategory, IsFunction)

▷ AddIsEndomorphism( $C$ ,  $F$ ) (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 IsEndomorphism.  $F : (arg2) \mapsto \text{IsEndomorphism}(arg2)$ .

### 7.6.96 AddIsEpimorphism (for IsCapCategory, IsFunction)

▷ AddIsEpimorphism( $C$ ,  $F$ ) (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 IsEpimorphism.  $F : (arg2) \mapsto \text{IsEpimorphism}(arg2)$ .

### 7.6.97 AddIsEqualAsFactorobjects (for IsCapCategory, IsFunction)

▷ AddIsEqualAsFactorobjects( $C$ ,  $F$ ) (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 IsEqualAsFactorobjects.  $F : (arg2, arg3) \mapsto \text{IsEqualAsFactorobjects}(arg2, arg3)$ .

### 7.6.98 AddIsEqualAsSubobjects (for IsCapCategory, IsFunction)

▷ AddIsEqualAsSubobjects( $C$ ,  $F$ ) (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 IsEqualAsSubobjects.  $F : (arg2, arg3) \mapsto \text{IsEqualAsSubobjects}(arg2, arg3)$ .

### 7.6.99 AddIsEqualForCacheForMorphisms (for IsCapCategory, IsFunction)

▷ AddIsEqualForCacheForMorphisms( $C$ ,  $F$ ) (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 IsEqualForCacheForMorphisms.  $F : (arg2, arg3) \mapsto \text{IsEqualForCacheForMorphisms}(arg2, arg3)$ .

### 7.6.100 AddIsEqualForCacheForObjects (for IsCapCategory, IsFunction)

▷ AddIsEqualForCacheForObjects( $C$ ,  $F$ ) (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 IsEqualForCacheForObjects.  $F : (arg2, arg3) \mapsto \text{IsEqualForCacheForObjects}(arg2, arg3)$ .

### 7.6.101 AddIsEqualForMorphisms (for IsCapCategory, IsFunction)

▷ AddIsEqualForMorphisms( $C$ ,  $F$ ) (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 IsEqualForMorphisms.  $F : (arg2, arg3) \mapsto \text{IsEqualForMorphisms}(arg2, arg3)$ .

### 7.6.102 AddIsEqualForMorphismsOnMor (for IsCapCategory, IsFunction)

▷ AddIsEqualForMorphismsOnMor( $C, F$ ) (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 IsEqualForMorphismsOnMor.  $F : (arg2, arg3) \mapsto \text{IsEqualForMorphismsOnMor}(arg2, arg3)$ .

### 7.6.103 AddIsEqualForObjects (for IsCapCategory, IsFunction)

▷ AddIsEqualForObjects( $C, F$ ) (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 IsEqualForObjects.  $F : (arg2, arg3) \mapsto \text{IsEqualForObjects}(arg2, arg3)$ .

### 7.6.104 AddIsEqualToIdentityMorphism (for IsCapCategory, IsFunction)

▷ AddIsEqualToIdentityMorphism( $C, F$ ) (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 IsEqualToIdentityMorphism.  $F : (arg2) \mapsto \text{IsEqualToIdentityMorphism}(arg2)$ .

### 7.6.105 AddIsEqualToZeroMorphism (for IsCapCategory, IsFunction)

▷ AddIsEqualToZeroMorphism( $C, F$ ) (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 IsEqualToZeroMorphism.  $F : (arg2) \mapsto \text{IsEqualToZeroMorphism}(arg2)$ .

### 7.6.106 AddIsHomSetInhabited (for IsCapCategory, IsFunction)

▷ AddIsHomSetInhabited( $C, F$ ) (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 IsHomSetInhabited.  $F : (arg2, arg3) \mapsto \text{IsHomSetInhabited}(arg2, arg3)$ .

### 7.6.107 AddIsIdempotent (for IsCapCategory, IsFunction)

▷ AddIsIdempotent( $C, F$ ) (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 IsIdempotent.  $F : (arg2) \mapsto \text{IsIdempotent}(arg2)$ .

### 7.6.108 AddIsInitial (for IsCapCategory, IsFunction)

▷ AddIsInitial( $\mathcal{C}$ ,  $F$ ) (operation)

**Returns:** nothing

The arguments are a category  $\mathcal{C}$  and a function  $F$ . This operation adds the given function  $F$  to the category for the basic operation IsInitial.  $F : (arg2) \mapsto \text{IsInitial}(arg2)$ .

### 7.6.109 AddIsInjective (for IsCapCategory, IsFunction)

▷ AddIsInjective( $\mathcal{C}$ ,  $F$ ) (operation)

**Returns:** nothing

The arguments are a category  $\mathcal{C}$  and a function  $F$ . This operation adds the given function  $F$  to the category for the basic operation IsInjective.  $F : (arg2) \mapsto \text{IsInjective}(arg2)$ .

### 7.6.110 AddIsIsomorphicForObjects (for IsCapCategory, IsFunction)

▷ AddIsIsomorphicForObjects( $\mathcal{C}$ ,  $F$ ) (operation)

**Returns:** nothing

The arguments are a category  $\mathcal{C}$  and a function  $F$ . This operation adds the given function  $F$  to the category for the basic operation IsIsomorphicForObjects.  $F : (object_1, object_2) \mapsto \text{IsIsomorphicForObjects}(object_1, object_2)$ .

### 7.6.111 AddIsIsomorphism (for IsCapCategory, IsFunction)

▷ AddIsIsomorphism( $\mathcal{C}$ ,  $F$ ) (operation)

**Returns:** nothing

The arguments are a category  $\mathcal{C}$  and a function  $F$ . This operation adds the given function  $F$  to the category for the basic operation IsIsomorphism.  $F : (arg2) \mapsto \text{IsIsomorphism}(arg2)$ .

### 7.6.112 AddIsLiftable (for IsCapCategory, IsFunction)

▷ AddIsLiftable( $\mathcal{C}$ ,  $F$ ) (operation)

**Returns:** nothing

The arguments are a category  $\mathcal{C}$  and a function  $F$ . This operation adds the given function  $F$  to the category for the basic operation IsLiftable.  $F : (arg2, arg3) \mapsto \text{IsLiftable}(arg2, arg3)$ .

### 7.6.113 AddIsLiftableAlongMonomorphism (for IsCapCategory, IsFunction)

▷ AddIsLiftableAlongMonomorphism( $\mathcal{C}$ ,  $F$ ) (operation)

**Returns:** nothing

The arguments are a category  $\mathcal{C}$  and a function  $F$ . This operation adds the given function  $F$  to the category for the basic operation IsLiftableAlongMonomorphism.  $F : (arg2, arg3) \mapsto \text{IsLiftableAlongMonomorphism}(arg2, arg3)$ .

### 7.6.114 AddIsMonomorphism (for IsCapCategory, IsFunction)

▷ AddIsMonomorphism( $\mathcal{C}$ ,  $F$ ) (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 `IsMonomorphism`.  $F : (arg2) \mapsto \text{IsMonomorphism}(arg2)$ .

#### 7.6.115 AddIsOne (for IsCapCategory, IsFunction)

▷ `AddIsOne( $C$ ,  $F$ )` (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 `IsOne`.  $F : (arg2) \mapsto \text{IsOne}(arg2)$ .

#### 7.6.116 AddIsProjective (for IsCapCategory, IsFunction)

▷ `AddIsProjective( $C$ ,  $F$ )` (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 `IsProjective`.  $F : (arg2) \mapsto \text{IsProjective}(arg2)$ .

#### 7.6.117 AddIsSplitEpimorphism (for IsCapCategory, IsFunction)

▷ `AddIsSplitEpimorphism( $C$ ,  $F$ )` (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 `IsSplitEpimorphism`.  $F : (arg2) \mapsto \text{IsSplitEpimorphism}(arg2)$ .

#### 7.6.118 AddIsSplitMonomorphism (for IsCapCategory, IsFunction)

▷ `AddIsSplitMonomorphism( $C$ ,  $F$ )` (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 `IsSplitMonomorphism`.  $F : (arg2) \mapsto \text{IsSplitMonomorphism}(arg2)$ .

#### 7.6.119 AddIsTerminal (for IsCapCategory, IsFunction)

▷ `AddIsTerminal( $C$ ,  $F$ )` (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 `IsTerminal`.  $F : (arg2) \mapsto \text{IsTerminal}(arg2)$ .

#### 7.6.120 AddIsWellDefinedForMorphisms (for IsCapCategory, IsFunction)

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

### 7.6.121 AddIsWellDefinedForMorphismsWithGivenSourceAndRange (for IsCapCategory, IsFunction)

▷ AddIsWellDefinedForMorphismsWithGivenSourceAndRange( $C$ ,  $F$ ) (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 IsWellDefinedForMorphismsWithGivenSourceAndRange.  $F : (source, alpha, range) \mapsto \text{IsWellDefinedForMorphismsWithGivenSourceAndRange}(source, alpha, range)$ .

### 7.6.122 AddIsWellDefinedForObjects (for IsCapCategory, IsFunction)

▷ AddIsWellDefinedForObjects( $C$ ,  $F$ ) (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 IsWellDefinedForObjects.  $F : (arg2) \mapsto \text{IsWellDefinedForObjects}(arg2)$ .

### 7.6.123 AddIsWellDefinedForTwoCells (for IsCapCategory, IsFunction)

▷ AddIsWellDefinedForTwoCells( $C$ ,  $F$ ) (operation)

**Returns:** nothing

The arguments are a category  $C$  and a function  $F$ . This operation adds the given function  $F$  to the category for the basic operation IsWellDefinedForTwoCells.  $F : (arg2) \mapsto \text{IsWellDefinedForTwoCells}(arg2)$ .

### 7.6.124 AddIsZeroForMorphisms (for IsCapCategory, IsFunction)

▷ AddIsZeroForMorphisms( $C$ ,  $F$ ) (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 IsZeroForMorphisms.  $F : (arg2) \mapsto \text{IsZeroForMorphisms}(arg2)$ .

### 7.6.125 AddIsZeroForObjects (for IsCapCategory, IsFunction)

▷ AddIsZeroForObjects( $C$ ,  $F$ ) (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 IsZeroForObjects.  $F : (arg2) \mapsto \text{IsZeroForObjects}(arg2)$ .

### 7.6.126 AddIsomorphismFromCoequalizerOfCoproductDiagramToPushout (for IsCapCategory, IsFunction)

▷ AddIsomorphismFromCoequalizerOfCoproductDiagramToPushout( $C$ ,  $F$ ) (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 IsomorphismFromCoequalizerOfCoproductDiagramToPushout.  $F : (D) \mapsto \text{IsomorphismFromCoequalizerOfCoproductDiagramToPushout}(D)$ .

### 7.6.127 AddIsomorphismFromCoequalizerToCokernelOfJointPairwiseDifferencesOfMorphismsFromCoproduct (for IsCapCategory, IsFunction)

▷ AddIsomorphismFromCoequalizerToCokernelOfJointPairwiseDifferencesOfMorphismsFromCoproduct( $C, F$ ) (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 IsomorphismFromCoequalizerToCokernelOfJointPairwiseDifferencesOfMorphismsFromCoproduct.  $F : (A, D) \mapsto \text{IsomorphismFromCoequalizerToCokernelOfJointPairwiseDifferencesOfMorphismsFromCoproduct}(A, D)$ .

### 7.6.128 AddIsomorphismFromCoimageToCokernelOfKernel (for IsCapCategory, IsFunction)

▷ AddIsomorphismFromCoimageToCokernelOfKernel( $C, F$ ) (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 IsomorphismFromCoimageToCokernelOfKernel.  $F : (alpha) \mapsto \text{IsomorphismFromCoimageToCokernelOfKernel}(alpha)$ .

### 7.6.129 AddIsomorphismFromCokernelOfJointPairwiseDifferencesOfMorphismsFromCoproductToCoequalizer (for IsCapCategory, IsFunction)

▷ AddIsomorphismFromCokernelOfJointPairwiseDifferencesOfMorphismsFromCoproductToCoequalizer( $C, F$ ) (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 IsomorphismFromCokernelOfJointPairwiseDifferencesOfMorphismsFromCoproductToCoequalizer.  $F : (A, D) \mapsto \text{IsomorphismFromCokernelOfJointPairwiseDifferencesOfMorphismsFromCoproductToCoequalizer}(A, D)$ .

### 7.6.130 AddIsomorphismFromCokernelOfKernelToCoimage (for IsCapCategory, IsFunction)

▷ AddIsomorphismFromCokernelOfKernelToCoimage( $C, F$ ) (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 IsomorphismFromCokernelOfKernelToCoimage.  $F : (alpha) \mapsto \text{IsomorphismFromCokernelOfKernelToCoimage}(alpha)$ .

### 7.6.131 AddIsomorphismFromCoproductToDirectSum (for IsCapCategory, IsFunction)

▷ AddIsomorphismFromCoproductToDirectSum( $C, F$ ) (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 `IsomorphismFromCoproductToDirectSum`.  $F : (D) \mapsto \text{IsomorphismFromCoproductToDirectSum}(D)$ .

### 7.6.132 AddIsomorphismFromDirectProductToDirectSum (for IsCapCategory, IsFunction)

▷ `AddIsomorphismFromDirectProductToDirectSum(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 `IsomorphismFromDirectProductToDirectSum`.  $F : (D) \mapsto \text{IsomorphismFromDirectProductToDirectSum}(D)$ .

### 7.6.133 AddIsomorphismFromDirectSumToCoproduct (for IsCapCategory, IsFunction)

▷ `AddIsomorphismFromDirectSumToCoproduct(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 `IsomorphismFromDirectSumToCoproduct`.  $F : (D) \mapsto \text{IsomorphismFromDirectSumToCoproduct}(D)$ .

### 7.6.134 AddIsomorphismFromDirectSumToDirectProduct (for IsCapCategory, IsFunction)

▷ `AddIsomorphismFromDirectSumToDirectProduct(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 `IsomorphismFromDirectSumToDirectProduct`.  $F : (D) \mapsto \text{IsomorphismFromDirectSumToDirectProduct}(D)$ .

### 7.6.135 AddIsomorphismFromEqualizerOfDirectProductDiagramToFiberProduct (for IsCapCategory, IsFunction)

▷ `AddIsomorphismFromEqualizerOfDirectProductDiagramToFiberProduct(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 `IsomorphismFromEqualizerOfDirectProductDiagramToFiberProduct`.  $F : (D) \mapsto \text{IsomorphismFromEqualizerOfDirectProductDiagramToFiberProduct}(D)$ .

### 7.6.136 AddIsomorphismFromEqualizerToKernelOfJointPairwiseDifferencesOfMorphismsIntoDirectProduct (for IsCapCategory, IsFunction)

▷ `AddIsomorphismFromEqualizerToKernelOfJointPairwiseDifferencesOfMorphismsIntoDirectProduct(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 `IsomorphismFromEqualizerToKernelOfJointPairwiseDifferencesOfMorphismsIntoDirectProduct`.  $F : (A, D) \mapsto \text{IsomorphismFromEqualizerToKernelOfJointPairwiseDifferencesOfMorphismsIntoDirectPr}$

#### 7.6.137 AddIsomorphismFromFiberProductToEqualizerOfDirectProductDiagram (for IsCapCategory, IsFunction)

▷ `AddIsomorphismFromFiberProductToEqualizerOfDirectProductDiagram(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 `IsomorphismFromFiberProductToEqualizerOfDirectProductDiagram`.  $F : (D) \mapsto \text{IsomorphismFromFiberProductToEqualizerOfDirectProductDiagram}(D)$ .

#### 7.6.138 AddIsomorphismFromHomologyObjectToItsConstructionAsAnImageObject (for IsCapCategory, IsFunction)

▷ `AddIsomorphismFromHomologyObjectToItsConstructionAsAnImageObject(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 `IsomorphismFromHomologyObjectToItsConstructionAsAnImageObject`.  $F : (\alpha, \beta) \mapsto \text{IsomorphismFromHomologyObjectToItsConstructionAsAnImageObject}(\alpha, \beta)$ .

#### 7.6.139 AddIsomorphismFromImageObjectToKernelOfCokernel (for IsCapCategory, IsFunction)

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

#### 7.6.140 AddIsomorphismFromInitialObjectToZeroObject (for IsCapCategory, IsFunction)

▷ `AddIsomorphismFromInitialObjectToZeroObject(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 `IsomorphismFromInitialObjectToZeroObject`.  $F : () \mapsto \text{IsomorphismFromInitialObjectToZeroObject}()$ .

### 7.6.141 AddIsomorphismFromItsConstructionAsAnImageObjectToHomologyObject (for IsCapCategory, IsFunction)

▷ AddIsomorphismFromItsConstructionAsAnImageObjectToHomologyObject( $C, F$ ) (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 IsomorphismFromItsConstructionAsAnImageObjectToHomologyObject.  $F : (\alpha, \beta) \mapsto \text{IsomorphismFromItsConstructionAsAnImageObjectToHomologyObject}(\alpha, \beta)$ .

### 7.6.142 AddIsomorphismFromKernelOfCokernelToImageObject (for IsCapCategory, IsFunction)

▷ AddIsomorphismFromKernelOfCokernelToImageObject( $C, F$ ) (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 IsomorphismFromKernelOfCokernelToImageObject.  $F : (\alpha) \mapsto \text{IsomorphismFromKernelOfCokernelToImageObject}(\alpha)$ .

### 7.6.143 AddIsomorphismFromKernelOfJointPairwiseDifferencesOfMorphismsIntoDirectProductToEqualizer (for IsCapCategory, IsFunction)

▷ AddIsomorphismFromKernelOfJointPairwiseDifferencesOfMorphismsIntoDirectProductToEqualizer( $F$ ) (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 IsomorphismFromKernelOfJointPairwiseDifferencesOfMorphismsIntoDirectProductToEqualizer.  $F : (A, D) \mapsto \text{IsomorphismFromKernelOfJointPairwiseDifferencesOfMorphismsIntoDirectProductToEqualizer}(A, D)$ .

### 7.6.144 AddIsomorphismFromPushoutToCoequalizerOfCoproductDiagram (for IsCapCategory, IsFunction)

▷ AddIsomorphismFromPushoutToCoequalizerOfCoproductDiagram( $C, F$ ) (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 IsomorphismFromPushoutToCoequalizerOfCoproductDiagram.  $F : (D) \mapsto \text{IsomorphismFromPushoutToCoequalizerOfCoproductDiagram}(D)$ .

### 7.6.145 AddIsomorphismFromTerminalObjectToZeroObject (for IsCapCategory, IsFunction)

▷ AddIsomorphismFromTerminalObjectToZeroObject( $C, F$ ) (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 `IsomorphismFromTerminalObjectToZeroObject`.  $F : () \mapsto \text{IsomorphismFromTerminalObjectToZeroObject}()$ .

#### 7.6.146 **AddIsomorphismFromZeroObjectToInitialObject** (for **IsCapCategory**, **IsFunction**)

▷ `AddIsomorphismFromZeroObjectToInitialObject(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 `IsomorphismFromZeroObjectToInitialObject`.  $F : () \mapsto \text{IsomorphismFromZeroObjectToInitialObject}()$ .

#### 7.6.147 **AddIsomorphismFromZeroObjectToTerminalObject** (for **IsCapCategory**, **IsFunction**)

▷ `AddIsomorphismFromZeroObjectToTerminalObject(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 `IsomorphismFromZeroObjectToTerminalObject`.  $F : () \mapsto \text{IsomorphismFromZeroObjectToTerminalObject}()$ .

#### 7.6.148 **AddJointPairwiseDifferencesOfMorphismsFromCoproduct** (for **IsCapCategory**, **IsFunction**)

▷ `AddJointPairwiseDifferencesOfMorphismsFromCoproduct(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 `JointPairwiseDifferencesOfMorphismsFromCoproduct`.  $F : (A, D) \mapsto \text{JointPairwiseDifferencesOfMorphismsFromCoproduct}(A, D)$ .

#### 7.6.149 **AddJointPairwiseDifferencesOfMorphismsIntoDirectProduct** (for **IsCapCategory**, **IsFunction**)

▷ `AddJointPairwiseDifferencesOfMorphismsIntoDirectProduct(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 `JointPairwiseDifferencesOfMorphismsIntoDirectProduct`.  $F : (A, D) \mapsto \text{JointPairwiseDifferencesOfMorphismsIntoDirectProduct}(A, D)$ .

#### 7.6.150 **AddKernelEmbedding** (for **IsCapCategory**, **IsFunction**)

▷ `AddKernelEmbedding(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 `KernelEmbedding`.  $F : (\alpha) \mapsto \text{KernelEmbedding}(\alpha)$ .

### 7.6.151 AddKernelEmbeddingWithGivenKernelObject (for IsCapCategory, IsFunction)

▷ AddKernelEmbeddingWithGivenKernelObject( $C, F$ ) (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 KernelEmbeddingWithGivenKernelObject.  $F : (\alpha, P) \mapsto \text{KernelEmbeddingWithGivenKernelObject}(\alpha, P)$ .

### 7.6.152 AddKernelLift (for IsCapCategory, IsFunction)

▷ AddKernelLift( $C, F$ ) (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 KernelLift.  $F : (\alpha, T, \tau) \mapsto \text{KernelLift}(\alpha, T, \tau)$ .

### 7.6.153 AddKernelLiftWithGivenKernelObject (for IsCapCategory, IsFunction)

▷ AddKernelLiftWithGivenKernelObject( $C, F$ ) (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 KernelLiftWithGivenKernelObject.  $F : (\alpha, T, \tau, P) \mapsto \text{KernelLiftWithGivenKernelObject}(\alpha, T, \tau, P)$ .

### 7.6.154 AddKernelObject (for IsCapCategory, IsFunction)

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

### 7.6.155 AddKernelObjectFunctorial (for IsCapCategory, IsFunction)

▷ AddKernelObjectFunctorial( $C, F$ ) (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 KernelObjectFunctorial.  $F : (\alpha, \mu, \alpha\text{phap}) \mapsto \text{KernelObjectFunctorial}(\alpha, \mu, \alpha\text{phap})$ .

### 7.6.156 AddKernelObjectFunctorialWithGivenKernelObjects (for IsCapCategory, IsFunction)

▷ AddKernelObjectFunctorialWithGivenKernelObjects( $C, F$ ) (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 KernelObjectFunctorialWithGivenKernelObjects.  $F : (P, \alpha, \mu, \alpha\text{phap}, Pp) \mapsto \text{KernelObjectFunctorialWithGivenKernelObjects}(P, \alpha, \mu, \alpha\text{phap}, Pp)$ .

**7.6.157 AddLift (for IsCapCategory, IsFunction)**

▷ AddLift( $C, F$ ) (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 Lift.  $F : (\alpha, \beta) \mapsto \text{Lift}(\alpha, \beta)$ .

**7.6.158 AddLiftAlongMonomorphism (for IsCapCategory, IsFunction)**

▷ AddLiftAlongMonomorphism( $C, F$ ) (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 LiftAlongMonomorphism.  $F : (\iota, \tau) \mapsto \text{LiftAlongMonomorphism}(\iota, \tau)$ .

**7.6.159 AddLiftOrFail (for IsCapCategory, IsFunction)**

▷ AddLiftOrFail( $C, F$ ) (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 LiftOrFail.  $F : (\alpha, \beta) \mapsto \text{LiftOrFail}(\alpha, \beta)$ .

**7.6.160 AddLinearCombinationOfMorphisms (for IsCapCategory, IsFunction)**

▷ AddLinearCombinationOfMorphisms( $C, F$ ) (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 LinearCombinationOfMorphisms.  $F : (\text{source}, \text{list}_o f, \text{ing\_elements}, \text{list}_o f_m \text{morphisms}, \text{range}) \mapsto \text{LinearCombinationOfMorphisms}(\text{source}, \text{list}_o f, \text{ing\_elements}, \text{list}_o f_m \text{morphisms}, \text{range})$ .

**7.6.161 AddMereExistenceOfSolutionOfLinearSystemInAbCategory (for IsCapCategory, IsFunction)**

▷ AddMereExistenceOfSolutionOfLinearSystemInAbCategory( $C, F$ ) (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 MereExistenceOfSolutionOfLinearSystemInAbCategory.  $F : (\arg2, \arg3, \arg4) \mapsto \text{MereExistenceOfSolutionOfLinearSystemInAbCategory}(\arg2, \arg3, \arg4)$ .

**7.6.162 AddMonomorphismIntoInjectiveEnvelopeObject (for IsCapCategory, IsFunction)**

▷ AddMonomorphismIntoInjectiveEnvelopeObject( $C, F$ ) (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 `MonomorphismIntoInjectiveEnvelopeObject`.  $F : (A) \mapsto \text{MonomorphismIntoInjectiveEnvelopeObject}(A)$ .

### 7.6.163 AddMonomorphismIntoInjectiveEnvelopeObjectWithGivenInjectiveEnvelopeObject (for IsCapCategory, IsFunction)

▷ `AddMonomorphismIntoInjectiveEnvelopeObjectWithGivenInjectiveEnvelopeObject(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 `MonomorphismIntoInjectiveEnvelopeObjectWithGivenInjectiveEnvelopeObject`.  $F : (A, I) \mapsto \text{MonomorphismIntoInjectiveEnvelopeObjectWithGivenInjectiveEnvelopeObject}(A, I)$ .

### 7.6.164 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`.  $F : (A) \mapsto \text{MonomorphismIntoSomeInjectiveObject}(A)$ .

### 7.6.165 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 `MonomorphismIntoSomeInjectiveObjectWithGivenSomeInjectiveObject`.  $F : (A, I) \mapsto \text{MonomorphismIntoSomeInjectiveObjectWithGivenSomeInjectiveObject}(A, I)$ .

### 7.6.166 AddMorphismBetweenDirectSums (for IsCapCategory, IsFunction)

▷ `AddMorphismBetweenDirectSums(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 `MorphismBetweenDirectSums`.  $F : (\text{source\_diagram}, \text{mat}, \text{range\_diagram}) \mapsto \text{MorphismBetweenDirectSums}(\text{source\_diagram}, \text{mat}, \text{range\_diagram})$ .

### 7.6.167 AddMorphismBetweenDirectSumsWithGivenDirectSums (for IsCapCategory, IsFunction)

▷ AddMorphismBetweenDirectSumsWithGivenDirectSums( $C, F$ ) (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 MorphismBetweenDirectSumsWithGivenDirectSums.  $F : (S, source\_diagram, mat, range\_diagram, T) \mapsto \text{MorphismBetweenDirectSumsWithGivenDirectSums}(S, source\_diagram, mat, range\_diagram, T)$ .

### 7.6.168 AddMorphismConstructor (for IsCapCategory, IsFunction)

▷ AddMorphismConstructor( $C, F$ ) (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 MorphismConstructor.  $F : (arg2, arg3, arg4) \mapsto \text{MorphismConstructor}(arg2, arg3, arg4)$ .

### 7.6.169 AddMorphismDatum (for IsCapCategory, IsFunction)

▷ AddMorphismDatum( $C, F$ ) (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 MorphismDatum.  $F : (arg2) \mapsto \text{MorphismDatum}(arg2)$ .

### 7.6.170 AddMorphismFromCoimageToImage (for IsCapCategory, IsFunction)

▷ AddMorphismFromCoimageToImage( $C, F$ ) (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 MorphismFromCoimageToImage.  $F : (alpha) \mapsto \text{MorphismFromCoimageToImage}(alpha)$ .

### 7.6.171 AddMorphismFromCoimageToImageWithGivenObjects (for IsCapCategory, IsFunction)

▷ AddMorphismFromCoimageToImageWithGivenObjects( $C, F$ ) (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 MorphismFromCoimageToImageWithGivenObjects.  $F : (C, alpha, I) \mapsto \text{MorphismFromCoimageToImageWithGivenObjects}(C, alpha, I)$ .

### 7.6.172 AddMorphismFromEqualizerToSink (for IsCapCategory, IsFunction)

▷ AddMorphismFromEqualizerToSink( $C, F$ ) (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 `MorphismFromEqualizerToSink`.  $F : (Y, \text{morphisms}) \mapsto \text{MorphismFromEqualizerToSink}(Y, \text{morphisms})$ .

#### 7.6.173 **AddMorphismFromEqualizerToSinkWithGivenEqualizer** (for **IsCapCategory, IsFunction**)

▷ `AddMorphismFromEqualizerToSinkWithGivenEqualizer(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 `MorphismFromEqualizerToSinkWithGivenEqualizer`.  $F : (Y, \text{morphisms}, P) \mapsto \text{MorphismFromEqualizerToSinkWithGivenEqualizer}(Y, \text{morphisms}, P)$ .

#### 7.6.174 **AddMorphismFromFiberProductToSink** (for **IsCapCategory, IsFunction**)

▷ `AddMorphismFromFiberProductToSink(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 `MorphismFromFiberProductToSink`.  $F : (\text{morphisms}) \mapsto \text{MorphismFromFiberProductToSink}(\text{morphisms})$ .

#### 7.6.175 **AddMorphismFromFiberProductToSinkWithGivenFiberProduct** (for **IsCapCategory, IsFunction**)

▷ `AddMorphismFromFiberProductToSinkWithGivenFiberProduct(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 `MorphismFromFiberProductToSinkWithGivenFiberProduct`.  $F : (\text{morphisms}, P) \mapsto \text{MorphismFromFiberProductToSinkWithGivenFiberProduct}(\text{morphisms}, P)$ .

#### 7.6.176 **AddMorphismFromKernelObjectToSink** (for **IsCapCategory, IsFunction**)

▷ `AddMorphismFromKernelObjectToSink(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 `MorphismFromKernelObjectToSink`.  $F : (\alpha) \mapsto \text{MorphismFromKernelObjectToSink}(\alpha)$ .

#### 7.6.177 **AddMorphismFromKernelObjectToSinkWithGivenKernelObject** (for **IsCapCategory, IsFunction**)

▷ `AddMorphismFromKernelObjectToSinkWithGivenKernelObject(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 `MorphismFromKernelObjectToSinkWithGivenKernelObject`.  $F : (\alpha, P) \mapsto \text{MorphismFromKernelObjectToSinkWithGivenKernelObject}(\alpha, P)$ .

### 7.6.178 AddMorphismFromSourceToCoequalizer (for IsCapCategory, IsFunction)

▷ AddMorphismFromSourceToCoequalizer( $C, F$ ) (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 MorphismFromSourceToCoequalizer.  $F : (Y, morphisms) \mapsto \text{MorphismFromSourceToCoequalizer}(Y, morphisms)$ .

### 7.6.179 AddMorphismFromSourceToCoequalizerWithGivenCoequalizer (for IsCapCategory, IsFunction)

▷ AddMorphismFromSourceToCoequalizerWithGivenCoequalizer( $C, F$ ) (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 MorphismFromSourceToCoequalizerWithGivenCoequalizer.  $F : (Y, morphisms, P) \mapsto \text{MorphismFromSourceToCoequalizerWithGivenCoequalizer}(Y, morphisms, P)$ .

### 7.6.180 AddMorphismFromSourceToCokernelObject (for IsCapCategory, IsFunction)

▷ AddMorphismFromSourceToCokernelObject( $C, F$ ) (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 MorphismFromSourceToCokernelObject.  $F : (alpha) \mapsto \text{MorphismFromSourceToCokernelObject}(alpha)$ .

### 7.6.181 AddMorphismFromSourceToCokernelObjectWithGivenCokernelObject (for IsCapCategory, IsFunction)

▷ AddMorphismFromSourceToCokernelObjectWithGivenCokernelObject( $C, F$ ) (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 MorphismFromSourceToCokernelObjectWithGivenCokernelObject.  $F : (alpha, P) \mapsto \text{MorphismFromSourceToCokernelObjectWithGivenCokernelObject}(alpha, P)$ .

### 7.6.182 AddMorphismFromSourceToPushout (for IsCapCategory, IsFunction)

▷ AddMorphismFromSourceToPushout( $C, F$ ) (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 MorphismFromSourceToPushout.  $F : (morphisms) \mapsto \text{MorphismFromSourceToPushout}(morphisms)$ .

### 7.6.183 AddMorphismFromSourceToPushoutWithGivenPushout (for IsCapCategory, IsFunction)

▷ AddMorphismFromSourceToPushoutWithGivenPushout( $C, F$ ) (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 MorphismFromSourceToPushoutWithGivenPushout.  $F : (morphisms, P) \mapsto \text{MorphismFromSourceToPushoutWithGivenPushout}(morphisms, P)$ .

### 7.6.184 AddMorphismsOfExternalHom (for IsCapCategory, IsFunction)

▷ AddMorphismsOfExternalHom( $C, F$ ) (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 MorphismsOfExternalHom.  $F : (arg2, arg3) \mapsto \text{MorphismsOfExternalHom}(arg2, arg3)$ .

### 7.6.185 AddMultiplyWithElementOfCommutativeRingForMorphisms (for IsCapCategory, IsFunction)

▷ AddMultiplyWithElementOfCommutativeRingForMorphisms( $C, F$ ) (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 MultiplyWithElementOfCommutativeRingForMorphisms.  $F : (r, alpha) \mapsto \text{MultiplyWithElementOfCommutativeRingForMorphisms}(r, alpha)$ .

### 7.6.186 AddObjectConstructor (for IsCapCategory, IsFunction)

▷ AddObjectConstructor( $C, F$ ) (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 ObjectConstructor.  $F : (arg2) \mapsto \text{ObjectConstructor}(arg2)$ .

### 7.6.187 AddObjectDatum (for IsCapCategory, IsFunction)

▷ AddObjectDatum( $C, F$ ) (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 ObjectDatum.  $F : (arg2) \mapsto \text{ObjectDatum}(arg2)$ .

### 7.6.188 AddPostCompose (for IsCapCategory, IsFunction)

▷ AddPostCompose( $C, F$ ) (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 PostCompose.  $F : (beta, alpha) \mapsto \text{PostCompose}(beta, alpha)$ .

**7.6.189 AddPostComposeList (for IsCapCategory, IsFunction)**

▷ AddPostComposeList( $C, F$ ) (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 PostComposeList.  $F : (source, list_{of} morphisms, range) \mapsto \text{PostComposeList}(source, list_{of} morphisms, range)$ .

**7.6.190 AddPostInverseForMorphisms (for IsCapCategory, IsFunction)**

▷ AddPostInverseForMorphisms( $C, F$ ) (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 PostInverseForMorphisms.  $F : (alpha) \mapsto \text{PostInverseForMorphisms}(alpha)$ .

**7.6.191 AddPreCompose (for IsCapCategory, IsFunction)**

▷ AddPreCompose( $C, F$ ) (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 PreCompose.  $F : (alpha, beta) \mapsto \text{PreCompose}(alpha, beta)$ .

**7.6.192 AddPreComposeList (for IsCapCategory, IsFunction)**

▷ AddPreComposeList( $C, F$ ) (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 PreComposeList.  $F : (source, list_{of} morphisms, range) \mapsto \text{PreComposeList}(source, list_{of} morphisms, range)$ .

**7.6.193 AddPreInverseForMorphisms (for IsCapCategory, IsFunction)**

▷ AddPreInverseForMorphisms( $C, F$ ) (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 PreInverseForMorphisms.  $F : (alpha) \mapsto \text{PreInverseForMorphisms}(alpha)$ .

**7.6.194 AddProjectionInFactorOfDirectProduct (for IsCapCategory, IsFunction)**

▷ AddProjectionInFactorOfDirectProduct( $C, F$ ) (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 ProjectionInFactorOfDirectProduct.  $F : (objects, k) \mapsto \text{ProjectionInFactorOfDirectProduct}(objects, k)$ .

### 7.6.195 AddProjectionInFactorOfDirectProductWithGivenDirectProduct (for IsCapCategory, IsFunction)

▷ AddProjectionInFactorOfDirectProductWithGivenDirectProduct( $C, F$ ) (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 ProjectionInFactorOfDirectProductWithGivenDirectProduct.  $F : (objects, k, P) \mapsto \text{ProjectionInFactorOfDirectProductWithGivenDirectProduct}(objects, k, P)$ .

### 7.6.196 AddProjectionInFactorOfDirectSum (for IsCapCategory, IsFunction)

▷ AddProjectionInFactorOfDirectSum( $C, F$ ) (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 ProjectionInFactorOfDirectSum.  $F : (objects, k) \mapsto \text{ProjectionInFactorOfDirectSum}(objects, k)$ .

### 7.6.197 AddProjectionInFactorOfDirectSumWithGivenDirectSum (for IsCapCategory, IsFunction)

▷ AddProjectionInFactorOfDirectSumWithGivenDirectSum( $C, F$ ) (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 ProjectionInFactorOfDirectSumWithGivenDirectSum.  $F : (objects, k, P) \mapsto \text{ProjectionInFactorOfDirectSumWithGivenDirectSum}(objects, k, P)$ .

### 7.6.198 AddProjectionInFactorOfFiberProduct (for IsCapCategory, IsFunction)

▷ AddProjectionInFactorOfFiberProduct( $C, F$ ) (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 ProjectionInFactorOfFiberProduct.  $F : (morphisms, k) \mapsto \text{ProjectionInFactorOfFiberProduct}(morphisms, k)$ .

### 7.6.199 AddProjectionInFactorOfFiberProductWithGivenFiberProduct (for IsCapCategory, IsFunction)

▷ AddProjectionInFactorOfFiberProductWithGivenFiberProduct( $C, F$ ) (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 ProjectionInFactorOfFiberProductWithGivenFiberProduct.  $F : (morphisms, k, P) \mapsto \text{ProjectionInFactorOfFiberProductWithGivenFiberProduct}(morphisms, k, P)$ .

### 7.6.200 AddProjectionOntoCoequalizer (for IsCapCategory, IsFunction)

▷ AddProjectionOntoCoequalizer( $C, F$ ) (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 ProjectionOntoCoequalizer.  $F : (Y, morphisms) \mapsto \text{ProjectionOntoCoequalizer}(Y, morphisms)$ .

### 7.6.201 AddProjectionOntoCoequalizerWithGivenCoequalizer (for IsCapCategory, IsFunction)

▷ AddProjectionOntoCoequalizerWithGivenCoequalizer( $C, F$ ) (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 ProjectionOntoCoequalizerWithGivenCoequalizer.  $F : (Y, morphisms, P) \mapsto \text{ProjectionOntoCoequalizerWithGivenCoequalizer}(Y, morphisms, P)$ .

### 7.6.202 AddProjectiveCoverObject (for IsCapCategory, IsFunction)

▷ AddProjectiveCoverObject( $C, F$ ) (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 ProjectiveCoverObject.  $F : (arg2) \mapsto \text{ProjectiveCoverObject}(arg2)$ .

### 7.6.203 AddProjectiveDimension (for IsCapCategory, IsFunction)

▷ AddProjectiveDimension( $C, F$ ) (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 ProjectiveDimension.  $F : (arg2) \mapsto \text{ProjectiveDimension}(arg2)$ .

### 7.6.204 AddProjectiveLift (for IsCapCategory, IsFunction)

▷ AddProjectiveLift( $C, F$ ) (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 ProjectiveLift.  $F : (alpha, beta) \mapsto \text{ProjectiveLift}(alpha, beta)$ .

### 7.6.205 AddPushout (for IsCapCategory, IsFunction)

▷ AddPushout( $C, F$ ) (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 Pushout.  $F : (morphisms) \mapsto \text{Pushout}(morphisms)$ .

### 7.6.206 AddPushoutFunctorial (for IsCapCategory, IsFunction)

▷ AddPushoutFunctorial( $C, F$ ) (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 PushoutFunctorial.  $F : (morphisms, L, morphismsp) \mapsto \text{PushoutFunctorial}(morphisms, L, morphismsp)$ .

### 7.6.207 AddPushoutFunctorialWithGivenPushouts (for IsCapCategory, IsFunction)

▷ AddPushoutFunctorialWithGivenPushouts( $C, F$ ) (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 PushoutFunctorialWithGivenPushouts.  $F : (P, morphisms, L, morphismsp, Pp) \mapsto \text{PushoutFunctorialWithGivenPushouts}(P, morphisms, L, morphismsp, Pp)$ .

### 7.6.208 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.  $F : (n) \mapsto \text{RandomMorphismByInteger}(n)$ .

### 7.6.209 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.  $F : (L) \mapsto \text{RandomMorphismByList}(L)$ .

### 7.6.210 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.  $F : (B, n) \mapsto \text{RandomMorphismWithFixedRangeByInteger}(B, n)$ .

### 7.6.211 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`.  $F : (B, L) \mapsto \text{RandomMorphismWithFixedRangeByList}(B, L)$ .

#### 7.6.212 **AddRandomMorphismWithFixedSourceAndRangeByInteger** (for **IsCapCategory, IsFunction**)

▷ `AddRandomMorphismWithFixedSourceAndRangeByInteger(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 `RandomMorphismWithFixedSourceAndRangeByInteger`.  $F : (A, B, n) \mapsto \text{RandomMorphismWithFixedSourceAndRangeByInteger}(A, B, n)$ .

#### 7.6.213 **AddRandomMorphismWithFixedSourceAndRangeByList** (for **IsCapCategory, IsFunction**)

▷ `AddRandomMorphismWithFixedSourceAndRangeByList(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 `RandomMorphismWithFixedSourceAndRangeByList`.  $F : (A, B, L) \mapsto \text{RandomMorphismWithFixedSourceAndRangeByList}(A, B, L)$ .

#### 7.6.214 **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`.  $F : (A, n) \mapsto \text{RandomMorphismWithFixedSourceByInteger}(A, n)$ .

#### 7.6.215 **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`.  $F : (A, L) \mapsto \text{RandomMorphismWithFixedSourceByList}(A, L)$ .

#### 7.6.216 **AddRandomObjectByInteger** (for **IsCapCategory, IsFunction**)

▷ `AddRandomObjectByInteger(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 `RandomObjectByInteger`.  $F : (n) \mapsto \text{RandomObjectByInteger}(n)$ .



### 7.6.217 AddRandomObjectByList (for IsCapCategory, IsFunction)

▷ AddRandomObjectByList( $C, F$ ) (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 RandomObjectByList.  $F : (L) \mapsto \text{RandomObjectByList}(L)$ .

### 7.6.218 AddSimplifyEndo (for IsCapCategory, IsFunction)

▷ AddSimplifyEndo( $C, F$ ) (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 SimplifyEndo.  $F : (mor, n) \mapsto \text{SimplifyEndo}(mor, n)$ .

### 7.6.219 AddSimplifyEndo\_IsoFromInputObject (for IsCapCategory, IsFunction)

▷ AddSimplifyEndo\_IsoFromInputObject( $C, F$ ) (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 SimplifyEndo\_IsoFromInputObject.  $F : (mor, n) \mapsto \text{SimplifyEndo\_IsoFromInputObject}(mor, n)$ .

### 7.6.220 AddSimplifyEndo\_IsoToInputObject (for IsCapCategory, IsFunction)

▷ AddSimplifyEndo\_IsoToInputObject( $C, F$ ) (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 SimplifyEndo\_IsoToInputObject.  $F : (mor, n) \mapsto \text{SimplifyEndo\_IsoToInputObject}(mor, n)$ .

### 7.6.221 AddSimplifyMorphism (for IsCapCategory, IsFunction)

▷ AddSimplifyMorphism( $C, F$ ) (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 SimplifyMorphism.  $F : (mor, n) \mapsto \text{SimplifyMorphism}(mor, n)$ .

### 7.6.222 AddSimplifyObject (for IsCapCategory, IsFunction)

▷ AddSimplifyObject( $C, F$ ) (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 SimplifyObject.  $F : (A, n) \mapsto \text{SimplifyObject}(A, n)$ .

### 7.6.223 AddSimplifyObject\_IsoFromInputObject (for IsCapCategory, IsFunction)

▷ AddSimplifyObject\_IsoFromInputObject( $C, F$ ) (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 `SimplifyObject_IsoFromInputObject`.  $F : (A, n) \mapsto \text{SimplifyObject\_IsoFromInputObject}(A, n)$ .

#### 7.6.224 AddSimplifyObject\_IsoToInputObject (for IsCapCategory, IsFunction)

▷ `AddSimplifyObject_IsoToInputObject(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 `SimplifyObject_IsoToInputObject`.  $F : (A, n) \mapsto \text{SimplifyObject\_IsoToInputObject}(A, n)$ .

#### 7.6.225 AddSimplifyRange (for IsCapCategory, IsFunction)

▷ `AddSimplifyRange(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 `SimplifyRange`.  $F : (mor, n) \mapsto \text{SimplifyRange}(mor, n)$ .

#### 7.6.226 AddSimplifyRange\_IsoFromInputObject (for IsCapCategory, IsFunction)

▷ `AddSimplifyRange_IsoFromInputObject(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 `SimplifyRange_IsoFromInputObject`.  $F : (mor, n) \mapsto \text{SimplifyRange\_IsoFromInputObject}(mor, n)$ .

#### 7.6.227 AddSimplifyRange\_IsoToInputObject (for IsCapCategory, IsFunction)

▷ `AddSimplifyRange_IsoToInputObject(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 `SimplifyRange_IsoToInputObject`.  $F : (mor, n) \mapsto \text{SimplifyRange\_IsoToInputObject}(mor, n)$ .

#### 7.6.228 AddSimplifySource (for IsCapCategory, IsFunction)

▷ `AddSimplifySource(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 `SimplifySource`.  $F : (mor, n) \mapsto \text{SimplifySource}(mor, n)$ .

#### 7.6.229 AddSimplifySourceAndRange (for IsCapCategory, IsFunction)

▷ `AddSimplifySourceAndRange(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 `SimplifySourceAndRange`.  $F : (mor, n) \mapsto \text{SimplifySourceAndRange}(mor, n)$ .

#### 7.6.230 **AddSimplifySourceAndRange\_IsoFromInputRange** (for **IsCapCategory**, **IsFunction**)

▷ `AddSimplifySourceAndRange_IsoFromInputRange(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 `SimplifySourceAndRange_IsoFromInputRange`.  $F : (mor, n) \mapsto \text{SimplifySourceAndRange\_IsoFromInputRange}(mor, n)$ .

#### 7.6.231 **AddSimplifySourceAndRange\_IsoFromInputSource** (for **IsCapCategory**, **IsFunction**)

▷ `AddSimplifySourceAndRange_IsoFromInputSource(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 `SimplifySourceAndRange_IsoFromInputSource`.  $F : (mor, n) \mapsto \text{SimplifySourceAndRange\_IsoFromInputSource}(mor, n)$ .

#### 7.6.232 **AddSimplifySourceAndRange\_IsoToInputRange** (for **IsCapCategory**, **IsFunction**)

▷ `AddSimplifySourceAndRange_IsoToInputRange(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 `SimplifySourceAndRange_IsoToInputRange`.  $F : (mor, n) \mapsto \text{SimplifySourceAndRange\_IsoToInputRange}(mor, n)$ .

#### 7.6.233 **AddSimplifySourceAndRange\_IsoToInputSource** (for **IsCapCategory**, **IsFunction**)

▷ `AddSimplifySourceAndRange_IsoToInputSource(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 `SimplifySourceAndRange_IsoToInputSource`.  $F : (mor, n) \mapsto \text{SimplifySourceAndRange\_IsoToInputSource}(mor, n)$ .

#### 7.6.234 **AddSimplifySource\_IsoFromInputObject** (for **IsCapCategory**, **IsFunction**)

▷ `AddSimplifySource_IsoFromInputObject(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 `SimplifySource_IsoFromInputObject`.  $F : (mor, n) \mapsto \text{SimplifySource\_IsoFromInputObject}(mor, n)$ .

### 7.6.235 AddSimplifySource\_IsoToInputObject (for IsCapCategory, IsFunction)

▷ AddSimplifySource\_IsoToInputObject( $C, F$ ) (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 `SimplifySource_IsoToInputObject`.  $F : (mor, n) \mapsto \text{SimplifySource\_IsoToInputObject}(mor, n)$ .

### 7.6.236 AddSolveLinearSystemInAbCategory (for IsCapCategory, IsFunction)

▷ AddSolveLinearSystemInAbCategory( $C, F$ ) (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 `SolveLinearSystemInAbCategory`.  $F : (arg2, arg3, arg4) \mapsto \text{SolveLinearSystemInAbCategory}(arg2, arg3, arg4)$ .

### 7.6.237 AddSolveLinearSystemInAbCategoryOrFail (for IsCapCategory, IsFunction)

▷ AddSolveLinearSystemInAbCategoryOrFail( $C, F$ ) (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 `SolveLinearSystemInAbCategoryOrFail`.  $F : (arg2, arg3, arg4) \mapsto \text{SolveLinearSystemInAbCategoryOrFail}(arg2, arg3, arg4)$ .

### 7.6.238 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`.  $F : (arg2) \mapsto \text{SomeInjectiveObject}(arg2)$ .

### 7.6.239 AddSomeIsomorphismBetweenObjects (for IsCapCategory, IsFunction)

▷ AddSomeIsomorphismBetweenObjects( $C, F$ ) (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 `SomeIsomorphismBetweenObjects`.  $F : (object_1, object_2) \mapsto \text{SomeIsomorphismBetweenObjects}(object_1, object_2)$ .

### 7.6.240 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`.  $F : (arg2) \mapsto \text{SomeProjectiveObject}(arg2)$ .

### 7.6.241 AddSomeReductionBySplitEpiSummand (for IsCapCategory, IsFunction)

▷ AddSomeReductionBySplitEpiSummand( $C, F$ ) (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 SomeReductionBySplitEpiSummand.  $F : (alpha) \mapsto \text{SomeReductionBySplitEpiSummand}(alpha)$ .

### 7.6.242 AddSomeReductionBySplitEpiSummand\_MorphismFromInputRange (for IsCapCategory, IsFunction)

▷ AddSomeReductionBySplitEpiSummand\_MorphismFromInputRange( $C, F$ ) (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 SomeReductionBySplitEpiSummand\_MorphismFromInputRange.  $F : (alpha) \mapsto \text{SomeReductionBySplitEpiSummand\_MorphismFromInputRange}(alpha)$ .

### 7.6.243 AddSomeReductionBySplitEpiSummand\_MorphismToInputRange (for IsCapCategory, IsFunction)

▷ AddSomeReductionBySplitEpiSummand\_MorphismToInputRange( $C, F$ ) (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 SomeReductionBySplitEpiSummand\_MorphismToInputRange.  $F : (alpha) \mapsto \text{SomeReductionBySplitEpiSummand\_MorphismToInputRange}(alpha)$ .

### 7.6.244 AddSubtractionForMorphisms (for IsCapCategory, IsFunction)

▷ AddSubtractionForMorphisms( $C, F$ ) (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 SubtractionForMorphisms.  $F : (alpha, beta) \mapsto \text{SubtractionForMorphisms}(alpha, beta)$ .

### 7.6.245 AddSumOfMorphisms (for IsCapCategory, IsFunction)

▷ AddSumOfMorphisms( $C, F$ ) (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 SumOfMorphisms.  $F : (source, list\_of\_morphisms, range) \mapsto \text{SumOfMorphisms}(source, list\_of\_morphisms, range)$ .

### 7.6.246 AddTerminalObject (for IsCapCategory, IsFunction)

▷ AddTerminalObject( $C, F$ ) (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 TerminalObject.  $F : () \mapsto \text{TerminalObject}()$ .

### 7.6.247 AddTerminalObjectFunctorial (for IsCapCategory, IsFunction)

▷ AddTerminalObjectFunctorial( $C, F$ ) (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 TerminalObjectFunctorial.  $F : () \mapsto \text{TerminalObjectFunctorial}()$ .

### 7.6.248 AddTerminalObjectFunctorialWithGivenTerminalObjects (for IsCapCategory, IsFunction)

▷ AddTerminalObjectFunctorialWithGivenTerminalObjects( $C, F$ ) (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 TerminalObjectFunctorialWithGivenTerminalObjects.  $F : (P, Pp) \mapsto \text{TerminalObjectFunctorialWithGivenTerminalObjects}(P, Pp)$ .

### 7.6.249 AddUniversalMorphismFromCoequalizer (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismFromCoequalizer( $C, F$ ) (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 UniversalMorphismFromCoequalizer.  $F : (Y, \text{morphisms}, T, \tau) \mapsto \text{UniversalMorphismFromCoequalizer}(Y, \text{morphisms}, T, \tau)$ .

### 7.6.250 AddUniversalMorphismFromCoequalizerWithGivenCoequalizer (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismFromCoequalizerWithGivenCoequalizer( $C, F$ ) (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 UniversalMorphismFromCoequalizerWithGivenCoequalizer.  $F : (Y, \text{morphisms}, T, \tau, P) \mapsto \text{UniversalMorphismFromCoequalizerWithGivenCoequalizer}(Y, \text{morphisms}, T, \tau, P)$ .

### 7.6.251 AddUniversalMorphismFromCoproduct (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismFromCoproduct( $C, F$ ) (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 UniversalMorphismFromCoproduct.  $F : (\text{objects}, T, \tau) \mapsto \text{UniversalMorphismFromCoproduct}(\text{objects}, T, \tau)$ .

### 7.6.252 AddUniversalMorphismFromCoproductWithGivenCoproduct (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismFromCoproductWithGivenCoproduct( $C, F$ ) (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 `UniversalMorphismFromCoproductWithGivenCoproduct`.  $F : (objects, T, tau, P) \mapsto \text{UniversalMorphismFromCoproductWithGivenCoproduct}(objects, T, tau, P)$ .

#### 7.6.253 AddUniversalMorphismFromDirectSum (for IsCapCategory, IsFunction)

▷ `AddUniversalMorphismFromDirectSum( $C, F$ )` (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 `UniversalMorphismFromDirectSum`.  $F : (objects, T, tau) \mapsto \text{UniversalMorphismFromDirectSum}(objects, T, tau)$ .

#### 7.6.254 AddUniversalMorphismFromDirectSumWithGivenDirectSum (for IsCapCategory, IsFunction)

▷ `AddUniversalMorphismFromDirectSumWithGivenDirectSum( $C, F$ )` (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 `UniversalMorphismFromDirectSumWithGivenDirectSum`.  $F : (objects, T, tau, P) \mapsto \text{UniversalMorphismFromDirectSumWithGivenDirectSum}(objects, T, tau, P)$ .

#### 7.6.255 AddUniversalMorphismFromImage (for IsCapCategory, IsFunction)

▷ `AddUniversalMorphismFromImage( $C, F$ )` (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 `UniversalMorphismFromImage`.  $F : (alpha, tau) \mapsto \text{UniversalMorphismFromImage}(alpha, tau)$ .

#### 7.6.256 AddUniversalMorphismFromImageWithGivenImageObject (for IsCapCategory, IsFunction)

▷ `AddUniversalMorphismFromImageWithGivenImageObject( $C, F$ )` (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 `UniversalMorphismFromImageWithGivenImageObject`.  $F : (alpha, tau, I) \mapsto \text{UniversalMorphismFromImageWithGivenImageObject}(alpha, tau, I)$ .

#### 7.6.257 AddUniversalMorphismFromInitialObject (for IsCapCategory, IsFunction)

▷ `AddUniversalMorphismFromInitialObject( $C, F$ )` (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 `UniversalMorphismFromInitialObject`.  $F : (T) \mapsto \text{UniversalMorphismFromInitialObject}(T)$ .

### 7.6.258 AddUniversalMorphismFromInitialObjectWithGivenInitialObject (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismFromInitialObjectWithGivenInitialObject( $C, F$ ) (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 UniversalMorphismFromInitialObjectWithGivenInitialObject.  $F : (T, P) \mapsto \text{UniversalMorphismFromInitialObjectWithGivenInitialObject}(T, P)$ .

### 7.6.259 AddUniversalMorphismFromPushout (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismFromPushout( $C, F$ ) (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 UniversalMorphismFromPushout.  $F : (\text{morphisms}, T, \text{tau}) \mapsto \text{UniversalMorphismFromPushout}(\text{morphisms}, T, \text{tau})$ .

### 7.6.260 AddUniversalMorphismFromPushoutWithGivenPushout (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismFromPushoutWithGivenPushout( $C, F$ ) (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 UniversalMorphismFromPushoutWithGivenPushout.  $F : (\text{morphisms}, T, \text{tau}, P) \mapsto \text{UniversalMorphismFromPushoutWithGivenPushout}(\text{morphisms}, T, \text{tau}, P)$ .

### 7.6.261 AddUniversalMorphismFromZeroObject (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismFromZeroObject( $C, F$ ) (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 UniversalMorphismFromZeroObject.  $F : (T) \mapsto \text{UniversalMorphismFromZeroObject}(T)$ .

### 7.6.262 AddUniversalMorphismFromZeroObjectWithGivenZeroObject (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismFromZeroObjectWithGivenZeroObject( $C, F$ ) (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 UniversalMorphismFromZeroObjectWithGivenZeroObject.  $F : (T, P) \mapsto \text{UniversalMorphismFromZeroObjectWithGivenZeroObject}(T, P)$ .



### 7.6.263 AddUniversalMorphismIntoCoimage (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismIntoCoimage( $C, F$ ) (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 UniversalMorphismIntoCoimage.  $F : (\alpha, \tau) \mapsto \text{UniversalMorphismIntoCoimage}(\alpha, \tau)$ .

### 7.6.264 AddUniversalMorphismIntoCoimageWithGivenCoimageObject (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismIntoCoimageWithGivenCoimageObject( $C, F$ ) (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 UniversalMorphismIntoCoimageWithGivenCoimageObject.  $F : (\alpha, \tau, C) \mapsto \text{UniversalMorphismIntoCoimageWithGivenCoimageObject}(\alpha, \tau, C)$ .

### 7.6.265 AddUniversalMorphismIntoDirectProduct (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismIntoDirectProduct( $C, F$ ) (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 UniversalMorphismIntoDirectProduct.  $F : (\text{objects}, T, \tau) \mapsto \text{UniversalMorphismIntoDirectProduct}(\text{objects}, T, \tau)$ .

### 7.6.266 AddUniversalMorphismIntoDirectProductWithGivenDirectProduct (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismIntoDirectProductWithGivenDirectProduct( $C, F$ ) (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 UniversalMorphismIntoDirectProductWithGivenDirectProduct.  $F : (\text{objects}, T, \tau, P) \mapsto \text{UniversalMorphismIntoDirectProductWithGivenDirectProduct}(\text{objects}, T, \tau, P)$ .

### 7.6.267 AddUniversalMorphismIntoDirectSum (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismIntoDirectSum( $C, F$ ) (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 UniversalMorphismIntoDirectSum.  $F : (\text{objects}, T, \tau) \mapsto \text{UniversalMorphismIntoDirectSum}(\text{objects}, T, \tau)$ .

### 7.6.268 AddUniversalMorphismIntoDirectSumWithGivenDirectSum (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismIntoDirectSumWithGivenDirectSum( $C, F$ ) (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 `UniversalMorphismIntoDirectSumWithGivenDirectSum`.  $F : (objects, T, tau, P) \mapsto \text{UniversalMorphismIntoDirectSumWithGivenDirectSum}(objects, T, tau, P)$ .

#### 7.6.269 AddUniversalMorphismIntoEqualizer (for IsCapCategory, IsFunction)

▷ `AddUniversalMorphismIntoEqualizer(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 `UniversalMorphismIntoEqualizer`.  $F : (Y, morphisms, T, tau) \mapsto \text{UniversalMorphismIntoEqualizer}(Y, morphisms, T, tau)$ .

#### 7.6.270 AddUniversalMorphismIntoEqualizerWithGivenEqualizer (for IsCapCategory, IsFunction)

▷ `AddUniversalMorphismIntoEqualizerWithGivenEqualizer(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 `UniversalMorphismIntoEqualizerWithGivenEqualizer`.  $F : (Y, morphisms, T, tau, P) \mapsto \text{UniversalMorphismIntoEqualizerWithGivenEqualizer}(Y, morphisms, T, tau, P)$ .

#### 7.6.271 AddUniversalMorphismIntoFiberProduct (for IsCapCategory, IsFunction)

▷ `AddUniversalMorphismIntoFiberProduct(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 `UniversalMorphismIntoFiberProduct`.  $F : (morphisms, T, tau) \mapsto \text{UniversalMorphismIntoFiberProduct}(morphisms, T, tau)$ .

#### 7.6.272 AddUniversalMorphismIntoFiberProductWithGivenFiberProduct (for IsCapCategory, IsFunction)

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

**Returns:** nothing

The arguments are a category  $C$  and a function  $F$ . This operation adds the given function  $F$  to the category for the basic operation `UniversalMorphismIntoFiberProductWithGivenFiberProduct`.  $F : (morphisms, T, tau, P) \mapsto \text{UniversalMorphismIntoFiberProductWithGivenFiberProduct}(morphisms, T, tau, P)$ .

#### 7.6.273 AddUniversalMorphismIntoTerminalObject (for IsCapCategory, IsFunction)

▷ `AddUniversalMorphismIntoTerminalObject(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 `UniversalMorphismIntoTerminalObject`.  $F : (T) \mapsto \text{UniversalMorphismIntoTerminalObject}(T)$ .

### 7.6.274 AddUniversalMorphismIntoTerminalObjectWithGivenTerminalObject (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismIntoTerminalObjectWithGivenTerminalObject( $C, F$ ) (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 UniversalMorphismIntoTerminalObjectWithGivenTerminalObject.  $F : (T, P) \mapsto \text{UniversalMorphismIntoTerminalObjectWithGivenTerminalObject}(T, P)$ .

### 7.6.275 AddUniversalMorphismIntoZeroObject (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismIntoZeroObject( $C, F$ ) (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 UniversalMorphismIntoZeroObject.  $F : (T) \mapsto \text{UniversalMorphismIntoZeroObject}(T)$ .

### 7.6.276 AddUniversalMorphismIntoZeroObjectWithGivenZeroObject (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismIntoZeroObjectWithGivenZeroObject( $C, F$ ) (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 UniversalMorphismIntoZeroObjectWithGivenZeroObject.  $F : (T, P) \mapsto \text{UniversalMorphismIntoZeroObjectWithGivenZeroObject}(T, P)$ .

### 7.6.277 AddVerticalPostCompose (for IsCapCategory, IsFunction)

▷ AddVerticalPostCompose( $C, F$ ) (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 VerticalPostCompose.  $F : (arg2, arg3) \mapsto \text{VerticalPostCompose}(arg2, arg3)$ .

### 7.6.278 AddVerticalPreCompose (for IsCapCategory, IsFunction)

▷ AddVerticalPreCompose( $C, F$ ) (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 VerticalPreCompose.  $F : (arg2, arg3) \mapsto \text{VerticalPreCompose}(arg2, arg3)$ .

### 7.6.279 AddZeroMorphism (for IsCapCategory, IsFunction)

▷ AddZeroMorphism( $C, F$ ) (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 `ZeroMorphism`.  $F : (a, b) \mapsto \text{ZeroMorphism}(a, b)$ .

#### 7.6.280 AddZeroObject (for IsCapCategory, IsFunction)

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

#### 7.6.281 AddZeroObjectFunctorial (for IsCapCategory, IsFunction)

▷ `AddZeroObjectFunctorial(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 `ZeroObjectFunctorial`.  $F : () \mapsto \text{ZeroObjectFunctorial}()$ .

#### 7.6.282 AddZeroObjectFunctorialWithGivenZeroObjects (for IsCapCategory, IsFunction)

▷ `AddZeroObjectFunctorialWithGivenZeroObjects(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 `ZeroObjectFunctorialWithGivenZeroObjects`.  $F : (P, Pp) \mapsto \text{ZeroObjectFunctorialWithGivenZeroObjects}(P, Pp)$ .

## 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 IsAttributeStoringRep)

▷ `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, IsString, IsDenseList, IsPosInt, IsFunction, IsFunction)

▷ `MakeDerivation(name, target_op, used_ops_with_multiples, weight, func, 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\_name* is the name of the operation which the derived method implements. The argument *used\_ops\_with\_multiples* contains

- the name of each operation used by the derived method,
- together with a positive integer specifying how many times that operation is used and
- either a category getter or `fail`.

This is given as a list of lists, where each sublist has as first entry the name of an operation, as second entry an integer and as third entry either a function or `fail`. This function should accept the category for which this derivation will be installed, and return a category for which the operation in the first entry must be installed for the derivation to be considered applicable. 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 *func* contains the actual implementation of the derived method. The argument *category\_filter* is a filter (or function) describing which categories the derivation is valid for. If it is valid for all categories, then this argument should have the value `IsCapCategory`. The output of *category\_filter* must not change during the installation of operations. In particular, it must not rely on `CanCompute` to check conditions.

### 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 DerivationFunction (for `IsDerivedMethod`)

▷ `DerivationFunction(d)` (attribute)

The implementation of the derivation.

### 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 UsedOperationsWithMultiplesAndCategoryGetters (for IsDerivedMethod)

▷ `UsedOperationsWithMultiplesAndCategoryGetters(d)` (attribute)

**Returns:** The names of the operations used by the derivation *d*, together with their multiplicities and category getters. The result is a list consisting of lists of the form [*op\_name*, *mult*, *getter*], where *op\_name* is a string, *mult* a positive integer and *getter* is a function or fail.

### 8.2.10 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.

### 8.2.11 FunctionCalledBeforeInstallation (for IsDerivedMethod)

▷ `FunctionCalledBeforeInstallation(d)` (attribute)

Input is a derived method. Output is a unary function that takes as an input a category and does not output anything. This function is always called before the installation of the derived method for a concrete instance of a category.

## 8.3 Derivation Graphs

### 8.3.1 IsDerivedMethodGraph (for IsAttributeStoringRep)

▷ `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

▷ `AddDerivation(graph, target_op, description, used_ops_with_multiples_and_category_getters, func, weight, category_filter, loop_multiplier, category_getters, function_called_before_installation)` (function)

Add a derivation to a derivation graph.

### 8.3.5 AddDerivationToCAP

▷ `AddDerivationToCAP(arg)` (function)

### 8.3.6 Operations (for IsDerivedMethodGraph)

▷ `Operations(G)` (attribute)

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

### 8.3.7 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.

### 8.3.8 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 IsAttributeStoringRep)

▷ `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).

### 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, IsDerived-Method)

▷ `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 IsAttributeStoringRep)

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

### 9.2 Tools

#### 9.2.1 FunctionWithNamedArguments

▷ `FunctionWithNamedArguments(specification, func)` (function)  
**Returns:** a function  
Simulates named arguments in GAP as follows:

- *specification* is a list of pairs with first entry the name of the argument and second entry a default value (which must be immutable).
- *func* must be a function with first argument `CAP_NAMED_ARGUMENTS`.
- The return value is a function with one argument fewer than *func*.

When calling the returned function, the arguments are passed on to *func*. To simulate named arguments, any `GAP` options appearing in *specification* are consumed and put into the record `CAP_NAMED_ARGUMENTS`.

### 9.2.2 CAP\_INTERNAL\_GET\_DATA\_TYPE\_FROM\_STRING

▷ CAP\_INTERNAL\_GET\_DATA\_TYPE\_FROM\_STRING(*filter\_string*[, *category*]) (function)

**Returns:** a record

The function takes one of the strings listed under *filter\_list* in 7.3 as input and returns the corresponding data type (see *CapJitInferredDataTypes* (**CompilerForCAP: CapJit-InferredDataTypes**) for details). If no category is given, data types with generic filters (*IsCapCategoryObject*, *IsCapCategoryMorphism* etc.) are returned. However, those cannot be used in the context of *CompilerForCAP* because the component category cannot be set in this case.

### 9.2.3 CAP\_INTERNAL\_GET\_DATA\_TYPES\_FROM\_STRINGS

▷ CAP\_INTERNAL\_GET\_DATA\_TYPES\_FROM\_STRINGS(*list\_of\_strings*[, *category*]) (function)

**Returns:** a list

Applies CAP\_INTERNAL\_GET\_DATA\_TYPE\_FROM\_STRING (9.2.2) to all elements of *list* and returns the result.

### 9.2.4 CAP\_INTERNAL\_REPLACED\_STRING\_WITH\_FILTER

▷ CAP\_INTERNAL\_REPLACED\_STRING\_WITH\_FILTER(*filter\_string*[, *category*]) (function)

**Returns:** a filter

The function takes one of the strings listed under *filter\_list* in 7.3 as input. The corresponding filter of the category *category* is returned. If no category is given, generic filters (*IsCapCategoryObject*, *IsCapCategoryMorphism* etc.) are used.

### 9.2.5 CAP\_INTERNAL\_REPLACED\_STRINGS\_WITH\_FILTERS

▷ CAP\_INTERNAL\_REPLACED\_STRINGS\_WITH\_FILTERS(*list*[, *category*]) (function)

**Returns:** Replaced list

Applies CAP\_INTERNAL\_REPLACED\_STRING\_WITH\_FILTER (9.2.4) to all elements of *list* and returns the result.

### 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 CAP\_INTERNAL\_ASSERT\_VALUE\_IS\_OF\_TYPE\_GETTER

▷ `CAP_INTERNAL_ASSERT_VALUE_IS_OF_TYPE_GETTER(data_type, human_readable_identifier_list)` (function)

Returns a unary function which throws an error if its argument is not of type *data\_type*. *human\_readable\_identifier\_list* is a list of arguments passed to Error to refer to the argument of the unary function in the error message.

### 9.2.11 CAP\_INTERNAL\_ASSERT\_IS\_CELL\_OF\_CATEGORY

▷ `CAP_INTERNAL_ASSERT_IS_CELL_OF_CATEGORY(cell, category, human_readable_identifier_list)` (function)

The function throws an error if *cell* is not a cell of *category*. If *category* is the boolean false, only general checks not specific to a concrete category are performed. *human\_readable\_identifier\_list* is a list of arguments passed to Error to refer to *cell* in the error message.

### 9.2.12 CAP\_INTERNAL\_ASSERT\_IS\_OBJECT\_OF\_CATEGORY

▷ `CAP_INTERNAL_ASSERT_IS_OBJECT_OF_CATEGORY(object, category, human_readable_identifier_list)` (function)

The function throws an error if *object* is not an object of *category*. If *category* is the boolean false, only general checks not specific to a concrete category are performed. *human\_readable\_identifier\_list* is a list of arguments passed to Error to refer to *object* in the error message.

### 9.2.13 CAP\_INTERNAL\_ASSERT\_IS\_MORPHISM\_OF\_CATEGORY

▷ CAP\_INTERNAL\_ASSERT\_IS\_MORPHISM\_OF\_CATEGORY(*morphism*, *category*, *human\_readable\_identifier\_list*) (function)

The function throws an error if *morphism* is not a morphism of *category*. If *category* is the boolean false, only general checks not specific to a concrete category are performed. *human\_readable\_identifier\_list* is a list of arguments passed to Error to refer to *morphism* in the error message.

### 9.2.14 CAP\_INTERNAL\_ASSERT\_IS\_TWO\_CELL\_OF\_CATEGORY

▷ CAP\_INTERNAL\_ASSERT\_IS\_TWO\_CELL\_OF\_CATEGORY(*two\_cell*, *category*, *human\_readable\_identifier\_list*) (function)

The function throws an error if *two\_cell* is not a 2-cell of *category*. If *category* is the boolean false, only general checks not specific to a concrete category are performed. *human\_readable\_identifier\_list* is a list of arguments passed to Error to refer to *two\_cell* in the error message.

### 9.2.15 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.

### 9.2.16 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.

### 9.2.17 InstallDeprecatedAlias

▷ InstallDeprecatedAlias(*alias\_name*, *function\_name*, *deprecation\_date*) (function)

Makes the function given by *function\_name* available under the alias *alias\_name* with a deprecation warning including the date *deprecation\_date*.

### 9.2.18 IsSpecializationOfFilter

▷ IsSpecializationOfFilter(*filter1*, *filter2*) (function)



Checks if *filter2* is more special than *filter1*, i.e. if *filter2* implies *filter1*. *filter1* and/or *filter2* can also be one of the strings listed under *filter\_list* in 7.3 and in this case are replaced by the corresponding filters (e.g. `IsCapCategory`, `IsCapCategoryObject`, `IsCapCategoryMorphism`, ...).

### 9.2.19 IsSpecializationOfFilterList

▷ `IsSpecializationOfFilterList(filter_list1, filter_list2)` (function)

Checks if *filter\_list2* is more special than *filter\_list1*, i.e. if both lists have the same length and any element of *filter\_list2* is more special than the corresponding element of *filter\_list1* in the sense of `IsSpecializationOfFilter` (9.2.18). *filter\_list1* and *filter\_list2* can also be the string "any", representing a most general filter list of any length.

### 9.2.20 InstallMethodForCompilerForCAP

▷ `InstallMethodForCompilerForCAP(same, as, for, InstallMethod)` (function)

Installs a method via `InstallMethod` and adds it to the list of methods known to the compiler. See `CapJitAddKnownMethod` (9.2.22) for requirements.

### 9.2.21 InstallOtherMethodForCompilerForCAP

▷ `InstallOtherMethodForCompilerForCAP(same, as, for, InstallOtherMethod)` (function)

Installs a method via `InstallOtherMethod` and adds it to the list of methods known to the compiler. See `CapJitAddKnownMethod` (9.2.22) for requirements.

### 9.2.22 CapJitAddKnownMethod

▷ `CapJitAddKnownMethod(operation, filters, method)` (function)

Adds a method to the list of methods known to the compiler. If the first filter implies `IsCapCategory`, method selection only takes the number of arguments and the first filter into account. This allows to resolve operations even in the case that the syntax tree cannot fully be typed. If the first filter does not imply `IsCapCategory`, method selection takes all filters into account. To strictly distinguish between the two cases, `IsCapCategory` must not imply the first filter (except if the first filter is equal to `IsCapCategory`). Method selection is strict in the sense that two different methods for the same operation must not be comparable. That is, they must have a different number of filters or the filters at at least one position must not be related via implication. In particular, adding two methods with a CAP category as first argument (or a convenience method for a CAP operation) with the same number of arguments and one category filter implying the other is not supported.

### 9.2.23 CapJitAddTypeSignature

▷ `CapJitAddTypeSignature(name, input_filters, output_data_type)` (function)

(experimental) Adds a type signature for the global function or operation given by *name* to the compiler. *input\_filters* must be a list of filters, or the string `"any"` representing a most general filter list of any length. *output\_data\_type* must be a filter, a data type, or a function. If it is a function with one argument, it must accept a list of input types and return the corresponding data type of the output. If it is a function with two arguments, it must accept the arguments of a function call of *name* (as syntax trees) and the function stack and return a record with components *args* (the possibly modified arguments) and *output\_type* (the data type of the output). See `CapJitInferredDataTypes` (**CompilerForCAP: CapJitInferredDataTypes**) for more details on data types.

### 9.2.24 CapJitAddTypeSignatureDeferred

▷ `CapJitAddTypeSignatureDeferred(package_name, name, input_filters, output_data_type)` (function)

(experimental) Same as `CapJitAddTypeSignature` (9.2.23), but the filters and the output data type must be given as strings which will be evaluated once *package\_name* is loaded. This should be used with care because errors will only be detected at runtime.

### 9.2.25 CapJitDataTypeOfListOf

▷ `CapJitDataTypeOfListOf(element_type)` (function)

(experimental) Returns the data type of a list whose elements are of type *element\_type*. *element\_type* must be a filter or a data type.

### 9.2.26 CapJitDataTypeOfNTupleOf

▷ `CapJitDataTypeOfNTupleOf(n, element_types...)` (function)

(experimental) Returns the data type of an *n*-tuple whose entries are of types corresponding to *element\_types*. *element\_types...* must be filters or data types.

### 9.2.27 CapJitDataTypeOfCategory

▷ `CapJitDataTypeOfCategory(category)` (function)  
 ▷ `CapJitDataTypeOfObjectOfCategory(category)` (function)  
 ▷ `CapJitDataTypeOfMorphismOfCategory(category)` (function)  
 ▷ `CapJitDataTypeOfTwoCellOfCategory(category)` (function)

(experimental) Returns the data type of the category (or objects, morphisms, or two cells in the category) *category*.

### 9.2.28 CapJitTypedExpression

▷ `CapJitTypedExpression(value, data_type_getter)` (function)

(experimental) Simply returns *value*, but allows to specify the data type of *value* for `CompilerForCAP`. *data\_type\_getter* must be a literal function or a global variable pointing to a function.

The function must accept no arguments or a single argument, and return a valid data type. If the function accepts a single argument, it must be inside a CAP operation or method known to `CompilerForCAP` (for example, see `InstallMethodForCompilerForCAP` (9.2.20)), and the current category (i.e. the first argument of the CAP operation or method known to `CompilerForCAP`) will be passed to the function. **IMPORTANT:** If `data_type_getter` is a literal function, it must not contain references to variables in its context. Otherwise the code might access random memory locations. See `CapJitInferredDataTypes` (**CompilerForCAP: CapJitInferredDataTypes**) for more details on data types.

### 9.2.29 CapFixpoint

▷ `CapFixpoint(predicate, func, initial_value)` (function)

Computes a fixpoint of `func` with regard to equality given by `predicate`, starting with `initial_value`. If no such fixpoint exists, the execution does not terminate.

### 9.2.30 Iterated (for IsList, IsFunction, IsObject)

▷ `Iterated(list, func, initial_value)` (operation)

Shorthand for `Iterated( Concatenation( [ initial_value ], list ), func )`.

### 9.2.31 Iterated (for IsList, IsFunction, IsObject, IsObject)

▷ `Iterated(list, func, initial_value, terminal_value)` (operation)

Shorthand for `Iterated( Concatenation( [ initial_value ], list, [ terminal_value ] ), func )`.

### 9.2.32 TransitivelyNeededOtherPackages

▷ `TransitivelyNeededOtherPackages(package_name)` (function)

Returns a list of package names which are transitively needed other packages of the package `package_name`.

### 9.2.33 PackageOfCAPOperation

▷ `PackageOfCAPOperation(operation_name)` (function)

Returns the name of the package to which the CAP operation given by `operation_name` belongs or fail if the package is not known.

### 9.2.34 SafePosition (for IsList, IsObject)

▷ `SafePosition(list, obj)` (operation)

**Returns:** an integer

Returns `Position( list, obj )` while asserting that this value is not fail.

### 9.2.35 SafeUniquePosition (for IsList, IsObject)

- ▷ `SafeUniquePosition(list, obj)` (operation)  
**Returns:** an integer  
 Returns `Position( list, obj )` while asserting that this value is not fail and the position is unique.

### 9.2.36 SafePositionProperty (for IsList, IsFunction)

- ▷ `SafePositionProperty(list, func)` (operation)  
**Returns:** an integer  
 Returns `PositionProperty( list, func )` while asserting that this value is not fail.

### 9.2.37 SafeUniquePositionProperty (for IsList, IsFunction)

- ▷ `SafeUniquePositionProperty(list, func)` (operation)  
**Returns:** an integer  
 Returns a position in `list` for which `func` returns true when applied to the corresponding entry while asserting that there exists exactly one such position.

### 9.2.38 SafeFirst (for IsList, IsFunction)

- ▷ `SafeFirst(list, func)` (operation)  
**Returns:** an element of the list  
 Returns `First( list, func )` while asserting that this value is not fail.

### 9.2.39 SafeUniqueEntry (for IsList, IsFunction)

- ▷ `SafeUniqueEntry(list, func)` (operation)  
**Returns:** an element of the list  
 Returns a value in `list` for which `func` returns true while asserting that there exists exactly one such entry.

### 9.2.40 NTuple

- ▷ `NTuple(n, args...)` (function)  
**Returns:** a list  
 Returns `args` while asserting that its length is `n`.

### 9.2.41 Pair

- ▷ `Pair(first, second)` (function)  
**Returns:** a list  
 Alias for `NTuple( 2, first, second )`.

### 9.2.42 Triple

▷ `Triple(first, second, third)` (function)

**Returns:** a list

Alias for `NTuple( 3, first, second, third )`.

### 9.2.43 TransposedMatWithGivenDimensions

▷ `TransposedMatWithGivenDimensions(nr_rows, nr_cols, listlist)` (function)

**Returns:** a list (of lists)

The arguments are two integers `nr_rows`, `nr_cols` and a list of lists `listlist` such that `nr_rows = Length(listlist)` and `nr_cols = Length(listlist[i])` for  $i = 1$  to `nr_rows`. The output is the transpose of `listlist` as a list consisting of `nr_cols` rows and `nr_rows` columns.

### 9.2.44 HandlePrecompiledTowers

▷ `HandlePrecompiledTowers(category, underlying_category, constructor_name)` (function)

Handles the information stored in `underlying_category!.compiler_hints.precompiled_towers` (if bound) which is a list of records with components:

- `remaining_constructors_in_tower`: a non-empty list of strings (names of category constructors)
- `precompiled_functions_adder`: a function accepting a CAP category as input

If `constructor_name` is the only entry of `remaining_constructors_in_tower`, `precompiled_functions_adder` is applied to `category` (except if the option `no_precompiled_code` is set to true) and should add precompiled code. Else, if `constructor_name` is the first entry of `remaining_constructors_in_tower`, the information is attached to `category!.compiler_hints.precompiled_towers` after removing `constructor_name` from `remaining_constructors_in_tower`. Note: Currently, there is no logic for finding the "optimal" code to install if `constructor_name` is the only entry of `remaining_constructors_in_tower` of multiple entries.

### 9.2.45 CAP\_JIT\_INCOMPLETE\_LOGIC

▷ `CAP_JIT_INCOMPLETE_LOGIC(value)` (function)

Simply returns `value`. Used to signify that the argument is not fully run through all logic functions/templates by `CompilerForCAP`.

### 9.2.46 CAP\_JIT\_EXPR\_CASE\_WRAPPER

▷ `CAP_JIT_EXPR_CASE_WRAPPER(func)` (function)

Simply returns `func`, which must be a literal function without arguments only containing an `if-elif-else` statement with each branch consisting of a single `return` statement. Used to write expressions of the form `function() if-elif-else end()` as

CAP\_JIT\_EXPR\_CASE\_WRAPPER(function() if-elif-else end)() because the former is not valid in Julia.

### 9.2.47 ListWithKeys

▷ ListWithKeys(*list*, *func*) (function)  
**Returns:** a list  
Same as List( *list*, *func* ) but *func* gets both the key *i* and *list*[*i*] as arguments.

### 9.2.48 SumWithKeys

▷ SumWithKeys(*list*, *func*) (function)  
**Returns:** a list  
Same as Sum( *list*, *func* ) but *func* gets both the key *i* and *list*[*i*] as arguments.

### 9.2.49 ProductWithKeys

▷ ProductWithKeys(*list*, *func*) (function)  
**Returns:** a list  
Same as Product( *list*, *func* ) but *func* gets both the key *i* and *list*[*i*] as arguments.

### 9.2.50 ForAllWithKeys

▷ ForAllWithKeys(*list*, *func*) (function)  
**Returns:** a list  
Same as ForAll( *list*, *func* ) but *func* gets both the key *i* and *list*[*i*] as arguments.

### 9.2.51 ForAnyWithKeys

▷ ForAnyWithKeys(*list*, *func*) (function)  
**Returns:** a list  
Same as ForAny( *list*, *func* ) but *func* gets both the key *i* and *list*[*i*] as arguments.

### 9.2.52 NumberWithKeys

▷ NumberWithKeys(*list*, *func*) (function)  
**Returns:** a list  
Same as Number( *list*, *func* ) but *func* gets both the key *i* and *list*[*i*] as arguments.

### 9.2.53 FilteredWithKeys

▷ FilteredWithKeys(*list*, *func*) (function)  
**Returns:** a list  
Same as Filtered( *list*, *func* ) but *func* gets both the key *i* and *list*[*i*] as arguments.

### 9.2.54 FirstWithKeys

▷ `FirstWithKeys(list, func)` (function)

**Returns:** a list

Same as `First( list, func )` but `func` gets both the key `i` and `list[i]` as arguments.

### 9.2.55 LastWithKeys

▷ `LastWithKeys(list, func)` (function)

**Returns:** a list

Same as `Last( list, func )` but `func` gets both the key `i` and `list[i]` as arguments.

## 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_specification`: 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 (non-)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 (non-)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 Functions

### 10.3.1 CAP\_INTERNAL\_GENERATE\_CONVENIENCE\_METHODS\_FOR\_LIMITS

▷ `CAP_INTERNAL_GENERATE_CONVENIENCE_METHODS_FOR_LIMITS(package_name, method_name_record, limits)` (function)

This function takes a package name, a method name record and a list of enhanced limits, and generates convenience methods for the limits as a string of GAP code. The result is compared to the content of the file `package_name/gap/LimitConvenienceOutput.gi`. If a difference is found, a warning is raised and the generated string is written to a temporary file for manual inspection.

### 10.3.2 CAP\_INTERNAL\_VALIDATE\_LIMITS\_IN\_NAME\_RECORD

▷ `CAP_INTERNAL_VALIDATE_LIMITS_IN_NAME_RECORD(method_name_record, limits)` (function)

This function 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

## The Category Constructor

### 11.1 Info class

#### 11.1.1 InfoCategoryConstructor

▷ InfoCategoryConstructor (info class)

Info class controlling the debugging output of CategoryConstructor (11.2.1).

### 11.2 Constructors

#### 11.2.1 CategoryConstructor (for IsRecord)

▷ CategoryConstructor(*options*) (operation)

**Returns:** a CAP category

Creates a CAP category subject to the options given via *options*, which is a record with the following keys:

- name (optional): name of the category
- category\_filter, category\_object\_filter, category\_morphism\_filter (mandatory): passed to CreateCapCategoryWithDataTypes (1.3.4)
- object\_datum\_type, morphism\_datum\_type (optional): passed to CreateCapCategoryWithDataTypes (1.3.4)
- commutative\_ring\_of\_linear\_category (optional): ring attached as CommutativeRingOfLinearCategory (1.4.9) to the category
- range\_category\_of\_homomorphism\_structure (optional): category attached as RangeCategoryOfHomomorphismStructure (1.4.10) to the category (or the string "self" to attach the category to itself)
- properties (optional): list of categorical properties the category will have, see CAP\_INTERNAL\_CATEGORICAL\_PROPERTIES\_LIST
- object\_constructor (optional): function added as an installation of ObjectConstructor (2.7.1) to the category

- `object_datum` (optional): function added as an installation of `ObjectDatum` (2.7.3) to the category
- `morphism_constructor` (optional): function added as an installation of `MorphismConstructor` (3.2.1) to the category
- `morphism_datum` (optional): function added as an installation of `MorphismDatum` (3.2.2) to the category
- `list_of_operations_to_install` (mandatory): a list of names of **CAP** operations which should be installed for the category
- `is_computable` (optional): whether the category can decide `IsCongruentForMorphisms`
- `supports_empty_limits` (optional): whether the category supports empty lists in inputs to operations of limits and colimits
- `underlying_category_getter_string` (optional): see below
- `underlying_object_getter_string` (optional): see below
- `underlying_morphism_getter_string` (optional): see below
- `top_object_getter_string` (optional): see below
- `top_morphism_getter_string` (optional): see below
- `generic_output_source_getter_string` (optional): see below
- `generic_output_range_getter_string` (optional): see below
- `create_func_bool`: see below
- `create_func_object`: see below
- `create_func_object_or_fail`: see below
- `create_func_morphism`: see below
- `create_func_morphism_or_fail`: see below
- `create_func_list_of_objects`: see below

The values of the keys `create_func_*` should be either the string "default" or functions which accept the category and the name of a **CAP** operation of the corresponding `return_type`. Values for return types occurring for operations in `list_of_operations_to_install` are mandatory. The functions must return strings, which (after some replacements described below) will be evaluated and added as an installation of the corresponding operation to the category. The value "default" chooses a suitable default string, see the implementation for details. The following placeholders may be used in the strings and are replaced automatically:

- `operation_name` will be replaced by the name of the operation

- `input_arguments...` will be replaced by the `input_arguments_names` specified in the method name record (see 7.3)
- `underlying_arguments...`: If the constructed category is created from another category, `underlying_category_getter_string`, `underlying_object_getter_string`, and `underlying_morphism_getter_string` may be strings of functions computing the underlying category (when applied to the constructed category) and the underlying object resp. morphism (when applied to the constructed category and an object resp. morphism in the constructed category). These functions are applied to `input_arguments` and `underlying_arguments` is replaced by the result.
- `number_of_arguments` will be replaced by the number of input/underlying arguments
- `top_source` and `top_range`: If the return type is `morphism` or `morphism_or_fail`, `source` and `range` are computed if possible and `top_source` and `top_range` are replaced by the results. For computing `source` and `range`, the `output_source_getter_string` and `output_range_getter_string` from the method name record are used if available (see 7.3). In some categories, `source` and `range` can always be obtained in a generic way (e.g. from the `morphism` datum). In this case, `generic_output_source_getter_string` and `generic_output_range_getter_string` can be set and are used if the required information is not available in the method name record.
- `top_object_getter` and `top_morphism_getter` are used in the "default" strings and are replaced by `top_object_getter_string` and `top_morphism_getter_string`, respectively.

## Chapter 12

# Reinterpretations of categories

### 12.1 Introduction

The support for building towers of category constructors is one of the main design features of CAP. Many categories that appear in the various applications can be modeled by towers of multiple category constructors. The category constructor `ReinterpretationOfCategory` (12.6.1) allows adding one last layer on top which allows expressing the desired (re)interpretation of such a modeling tower. In particular, this category constructor allows specifying the name of the category together with customized methods for the operations

- `ObjectConstructor`
- `MorphismConstructor`
- `ObjectDatum`
- `MorphismDatum`

in order to reflect the desired (re)interpretation with a user-interface that is independent of the modeling tower (see below for details). Note that the same tower might have multiple interpretations.

<code>R := ReinterpretationOfCategory( cat_n )</code>
<code>cat_n := CategoryConstructor_n( cat_{n-1} )</code>
...
<code>cat_1 := CategoryConstructor_1( non_categorical_input )</code>

**Table:** A tower of categories modeling the category `R`

The reinterpretation `R` is isomorphic to the top category `cat_n` in the tower. In practice, the word “tower” stands more generally for a finite poset with a greatest element.

### 12.2 Tutorial

We will show how one can reinterpret a category with the following guiding example: We reinterpret `Opposite( CategoryOfRows( R ) )` as `CategoryOfColumns( R )` using `ReinterpretationOfCategory` (12.6.1) with the options described in the following (see `CategoryOfColumnsAsOppositeOfCategoryOfRows.gi` in `FreydCategoriesForCAP` for a full implementation).

1. Set the options `category_filter`, `category_object_filter`, and `category_morphism_filter` to the filters corresponding to the data structure of the desired reinterpretation, e.g. `IsCategoryOfColumns`, `IsCategoryOfColumnsObject`, and `IsCategoryOfColumnsMorphism`.
2. Set `object_constructor`, `object_datum`, `morphism_constructor`, and `morphism_datum` to the functions one would write for `ObjectConstructor` and so on for a primitive implementation of the desired reinterpretation. In our example, `object_constructor` takes the reinterpretation `R` (which lies in `IsCategoryOfColumns` due to the filter set in the first step) and an integer, and returns a CAP object in the category with attribute `RankOfObject` set to the integer, just like a primitive implementation of `CategoryOfColumns` would do.
3. Set `modeling_tower_object_constructor`, `modeling_tower_object_datum`, `modeling_tower_morphism_constructor`, and `modeling_tower_morphism_datum`: `modeling_tower_object_constructor` gets the same input as `object_constructor` but must return the corresponding object in the tower `cat_n`. `modeling_tower_object_datum` has the same output as `object_datum` but gets the reinterpretation `R` and an object in the tower `cat_n` as an input. In our example, `modeling_tower_object_constructor` gets the reinterpretation `R` and an integer as in step 2 and wraps the integer as a `CategoryOfRowsObject` and the result as an object in the opposite category. `modeling_tower_object_datum` gets the reinterpretation `R` and an object in `Opposite( CategoryOfRows( R ) )` (that is, an integer boxed as a category of rows object boxed as an object in the opposite category) and returns the underlying integer. `modeling_tower_morphism_constructor` and `modeling_tower_morphism_datum` are given analogously.

By composing `modeling_tower_object_datum` with `object_constructor` and `modeling_tower_morphism_datum` with `morphism_constructor` (with suitable source and range), `ReinterpretationOfCategory` defines a functor "Reinterpretation" from `cat_n` to `R`. Similarly, it defines a functor "Model" from `R` to `cat_n` by composing `object_datum` with `modeling_tower_object_constructor` and `morphism_datum` with `modeling_tower_morphism_constructor` (with suitable source and range). "Reinterpretation" should be an isomorphism of categories with inverse "Model". More precisely, one has to take care of the following things:

- Since `R` should just be a reinterpretation of `cat_n` with a nicer data structure, we certainly want "Reinterpretation" to be an equivalence of categories with pseudo-inverse "Model".
- `ReinterpretationOfCategory` copies all properties from `cat_n` to `R`, including properties like `IsSkeletalCategory` which are not necessarily preserved by mere equivalences.
- To fulfill the specification of `WithGiven` operations, reinterpreting a `WithGiven` object `A` in `cat_n` as an object in `R` and taking its model again must give an object equal to `A`. So we require applying "Reinterpretation" and then "Model" to give the identity. Conversely, let `B_1` be an object in `R`. We take its model and reinterpret this again to form an object `B_2`. By construction of `R`, `B_1` and `B_2` are equal if and only if their models are equal. But since applying "Reinterpretation" and then "Model" gives the identity, taking the model of `B_2` simply gives an object equal to the model of `B_1`. Thus, also `B_1` and `B_2` are equal. Hence, "Reinterpretation" has to be an equivalence which is a bijection on objects, and hence an isomorphism (although "Model" is not necessarily its inverse). Note: Alternatively, one can make sure that

WithGiven operations in `cat_n` are only called via the corresponding non-WithGiven operation in `cat_n`. This can be achieved by reinterpreting all operations of `cat_n` (i.e. creating `R` with `only_primitive_operations := false`), disabling redirect functions (i.e. creating `R` with the option `overhead := false`) and not calling WithGiven operations of `R` manually.

## 12.3 Implementation details

Operations in `ReinterpretationOfCategory` are implemented as follows:

1. Apply `object_datum` and `morphism_datum` to the input to get the underlying data.
2. Apply `modeling_tower_object_constructor` and `modeling_tower_morphism_constructor` to the underlying data to get objects and morphisms in the tower `cat_n`.
3. Apply the operation of the tower `cat_n`.
4. Apply `modeling_tower_object_datum` or `modeling_tower_morphism_datum` to the result to get the underlying data.
5. Apply `object_constructor` or `morphism_constructor` to the underlying data to get an object or a morphism in the reinterpretation `R`.

The first two steps define the functor "Model" and the last two steps define the functor "Reinterpretation". "Reinterpretation" on objects and morphisms is called `ReinterpretationOfObject` and `ReinterpretationOfMorphism` in the code. "Model" on objects and morphisms is called `ModelingObject` and `ModelingMorphism` in the code.

## 12.4 Relation to `CompilerForCAP`

The operation of the tower `cat_n` (step 3 above) usually unboxes objects and morphisms, operates on the underlying data, and boxes the result. The unboxing usually cancels with step 2 above, and boxing the result usually cancels with step 4 above. If one now compiles the operations of the reinterpretation `R`, only the following steps remain:

1. Apply `object_datum` and `morphism_datum` to the input to get the underlying data.
2. Operate on the underlying data. (previously part of step 3)
3. Apply `object_constructor` or `morphism_constructor` to the underlying data to get an object or a morphism in the reinterpretation `R`. (previously step 5)

This is exactly what a primitive implementation would look like. Thus, in many cases compiling a reinterpretation immediately gives a primitive implementation with no remaining references to the tower `cat_n`.



## 12.5 Attributes

### 12.5.1 ModelingCategory (for IsCapCategory)

- ▷ `ModelingCategory(R)` (attribute)
- Returns:** a category
- The tower `cat_n` modeling the reinterpretation *R*.

## 12.6 Constructors

### 12.6.1 ReinterpretationOfCategory (for IsCapCategory, IsRecord)

- ▷ `ReinterpretationOfCategory(category, options)` (operation)
- Returns:** a category
- Reinterprets a category *category* (the "modeling category") to form a new category *R* (the "reinterpretation") subject to the options given via *options*, which is a record with the following keys:

- `name`: the name of the reinterpretation *R*,
- `category_filter`, `category_object_filter`, `category_morphism_filter`, `object_datum_type`, `morphism_datum_type`, `object_constructor`, `object_datum`, `morphism_constructor`, `morphism_datum`: same meaning as for `CategoryConstructor` (11.2.1), which is used to create the reinterpretation *R*,
- `modeling_tower_object_constructor`: a function which gets the reinterpretation *R* and an object datum (in the sense of `object_datum`) and returns the corresponding modeling object in the modeling category,
- `modeling_tower_object_datum`: a function which gets the reinterpretation *R* and an object in the modeling category and returns the corresponding object datum (in the sense of `object_datum`),
- `modeling_tower_morphism_constructor`: a function which gets the reinterpretation *R*, a source in the modeling category, a morphism datum (in the sense of `morphism_datum`), and a range in the modeling category and returns the corresponding modeling morphism in the modeling category,
- `modeling_tower_morphism_datum`: a function which gets the reinterpretation *R* and a morphism in the modeling category and returns the corresponding morphism datum (in the sense of `morphism_datum`),
- `only_primitive_operations` (optional, default `false`): whether to only reinterpret primitive operations or all operations.

### 12.6.2 ReinterpretationFunctor (for IsCapCategory)

- ▷ `ReinterpretationFunctor(R)` (attribute)
- Returns:** a functor
- Returns the functor from the modeling category `ModelingCategory(R)` to the reinterpretation *R* which maps each object/morphism to its reinterpretation.

### 12.6.3 ModelingObject (for IsCapCategory, IsCapCategoryObject)

▷ `ModelingObject(R, obj)` (operation)

**Returns:** a CAP category object

Returns the object in `ModelingCategory(R)` modeling the object *obj* in the reinterpretation *R*.

### 12.6.4 ReinterpretationOfObject (for IsCapCategory, IsCapCategoryObject)

▷ `ReinterpretationOfObject(R, obj)` (operation)

**Returns:** a CAP category object

Returns the reinterpretation in *R* of an object *obj* in `ModelingCategory(R)`.

### 12.6.5 ModelingMorphism (for IsCapCategory, IsCapCategoryMorphism)

▷ `ModelingMorphism(R, mor)` (operation)

**Returns:** a CAP category morphism

Returns the morphism in `ModelingCategory(R)` modeling the morphism *mor* in the reinterpretation *R*.

### 12.6.6 ReinterpretationOfMorphism (for IsCapCategory, IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryObject)

▷ `ReinterpretationOfMorphism(R, source, obj, range)` (operation)

**Returns:** a CAP category morphism

Returns the reinterpretation in *R* with given source and range of a morphism *mor* in `ModelingCategory(R)`.

## Chapter 13

# Create wrapper hulls of a category

### 13.1 GAP categories

#### 13.1.1 IsWrapperCapCategory (for IsCapCategory)

- ▷ `IsWrapperCapCategory(arg)` (filter)  
**Returns:** true or false  
The GAP category of a wrapper CAP category.

#### 13.1.2 IsWrapperCapCategoryObject (for IsCapCategoryObject)

- ▷ `IsWrapperCapCategoryObject(arg)` (filter)  
**Returns:** true or false  
The GAP category of objects in a wrapper CAP category.

#### 13.1.3 IsWrapperCapCategoryMorphism (for IsCapCategoryMorphism)

- ▷ `IsWrapperCapCategoryMorphism(arg)` (filter)  
**Returns:** true or false  
The GAP category of morphisms in a wrapper CAP category.

### 13.2 Attributes

#### 13.2.1 UnderlyingCell (for IsWrapperCapCategoryObject)

- ▷ `UnderlyingCell(object)` (attribute)  
**Returns:** a category object  
The cell underlying the wrapper category object *object*.

#### 13.2.2 UnderlyingCell (for IsWrapperCapCategoryMorphism)

- ▷ `UnderlyingCell(morphism)` (attribute)  
**Returns:** a category morphism  
The cell underlying the wrapper category morphism *morphism*.

## 13.3 Constructors

### 13.3.1 AsObjectInWrapperCategory (for IsWrapperCapCategory, IsCapCategory-Object)

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

**Returns:** an object

Wrap an object *object* (in the category underlying the wrapper category *category*) to form an object in *category*.

### 13.3.2 AsMorphismInWrapperCategory (for IsWrapperCapCategoryObject, IsCapCategoryMorphism, IsWrapperCapCategoryObject)

▷ `AsMorphismInWrapperCategory(source, morphism, range)` (operation)

**Returns:** a morphism

Wrap a morphism *morphism* (in the category underlying the wrapper category `CapCategory(source)`) to form a morphism in `CapCategory(source)` with given source and range.

### 13.3.3 AsMorphismInWrapperCategory (for IsWrapperCapCategory, IsCapCategoryMorphism)

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

**Returns:** a morphism

Wrap a morphism *morphism* (in the category underlying the wrapper category *category*) to form a morphism in *category*.

### 13.3.4 / (for IsCapCategoryCell, IsWrapperCapCategory)

▷ `/(cell, category)` (operation)

Convenience method for `AsObjectInWrapperCategory` (13.3.1) and `AsMorphismInWrapperCategory` (13.3.3).

### 13.3.5 WrapperCategory (for IsCapCategory, IsRecord)

▷ `WrapperCategory(category, options)` (operation)

**Returns:** a category

Wraps a category *category* to form a new category subject to the options given via *options*, which is a record with the following keys:

- `name` (optional): the name of the wrapper category
- `only_primitive_operations` (optional, default false): whether to only wrap primitive operations or all operations

Additionally, the following options of `CategoryConstructor` (11.2.1) are supported: `category_filter`, `category_object_filter`, `category_morphism_filter`. The filters must imply `IsWrapperCapCategory`, `IsWrapperCapCategoryObject`, and `IsWrapperCapCategoryMorphism`, respectively.

### 13.3.6 WrappingFunctor (for IsWrapperCapCategory)

▷ `WrappingFunctor( $W$ )`

(attribute)

**Returns:** a functor

Return the functor from the wrapped category `ModelingCategory( $W$ )` to the wrapper category  $W$  which simply wraps objects/morphisms.

## Chapter 14

# Dummy implementations

A dummy implementation of a concept seems to provide an interface for the concept, but calling any operation in this interface will simply signal an error. Hence, when using a dummy implementation, we can be sure that we only rely on the abstract interface but not on any implementation details, for the simple reason that there is no actual implementation. This is useful for testing or compilation against a generic implementation of a concept.

### 14.1 Dummy rings

#### 14.1.1 IsDummyRing

▷ IsDummyRing (filter)

The GAP filter of dummy rings.

#### 14.1.2 IsDummyRingElement

▷ IsDummyRingElement (filter)

The GAP filter of elements of a dummy ring.

#### 14.1.3 IsDummyCommutativeRing

▷ IsDummyCommutativeRing (filter)

The GAP filter of dummy commutative rings.

#### 14.1.4 IsDummyCommutativeRingElement

▷ IsDummyCommutativeRingElement (filter)

The GAP filter of elements of a dummy commutative ring.

### 14.1.5 IsDummyField

▷ IsDummyField (filter)

The GAP filter of dummy fields.

### 14.1.6 IsDummyFieldElement

▷ IsDummyFieldElement (filter)

The GAP filter of elements of a dummy commutative ring.

### 14.1.7 DummyRing

▷ DummyRing(*arg*) (function)

**Returns:** a dummy ring

### 14.1.8 DummyCommutativeRing

▷ DummyCommutativeRing(*arg*) (function)

**Returns:** a dummy commutative ring

### 14.1.9 DummyField

▷ DummyField(*arg*) (function)

**Returns:** a dummy field

## 14.2 Dummy categories

### 14.2.1 IsDummyCategory (for IsCapCategory)

▷ IsDummyCategory(*arg*) (filter)

**Returns:** true or false

The GAP category of a dummy CAP category.

### 14.2.2 IsDummyCategoryObject (for IsCapCategoryObject)

▷ IsDummyCategoryObject(*arg*) (filter)

**Returns:** true or false

The GAP category of objects in a dummy CAP category.

### 14.2.3 IsDummyCategoryMorphism (for IsCapCategoryMorphism)

▷ IsDummyCategoryMorphism(*arg*) (filter)

**Returns:** true or false

The GAP category of morphisms in a dummy CAP category.

#### 14.2.4 DummyCategory (for IsRecord)

▷ `DummyCategory(options)`

(operation)

**Returns:** a category

Creates a dummy category subject to the options given via *options*, which is a record passed on to `CategoryConstructor` (11.2.1). Note that the options `{category,object,morphism}_filter` will be set to `IsDummyCategory{,Object,Morphism}` and the options `{object,morphism}_{constructor,datum}` and `create_func_*` will be set to dummy implementations (throwing errors when actually called). The dummy category will pretend to support empty limits by default.



## Chapter 15

# Examples and Tests

### 15.1 Dummy implementations

#### 15.1.1 Dummy categories

Example

```
gap> LoadPackage( "CAP", false );
true
gap> list_of_operations_to_install := [
>   "ObjectConstructor",
>   "MorphismConstructor",
>   "ObjectDatum",
>   "MorphismDatum",
>   "IsCongruentForMorphisms",
>   "PreCompose",
>   "IdentityMorphism",
>   "DirectSum",
> ];
gap> dummy := DummyCategory( rec(
>   list_of_operations_to_install := list_of_operations_to_install,
>   properties := [ "IsAdditiveCategory" ],
> ) );
gap> ForAll( list_of_operations_to_install, o -> CanCompute( dummy, o ) );
true
gap> IsAdditiveCategory( dummy );
true
```

#### 15.1.2 Dummy rings

Example

```
gap> LoadPackage( "CAP", false );
true
gap> DummyRing( );
Dummy ring 1
gap> DummyRing( );
Dummy ring 2
gap> IsRing( DummyRing( ) );
true
gap> DummyCommutativeRing( );
```

```

Dummy commutative ring 1
gap> DummyCommutativeRing( );
Dummy commutative ring 2
gap> IsRing( DummyCommutativeRing( ) );
true
gap> IsCommutative( DummyCommutativeRing( ) );
true
gap> DummyField( );
Dummy field 1
gap> DummyField( );
Dummy field 2
gap> IsRing( DummyField( ) );
true
gap> IsField( DummyField( ) );
true

```

## 15.2 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 TensorProductOnMorphismsWithGivenTensors( source, alpha, beta, range ); end;;
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], range );;
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( Category of left presentations of Q ) ] ]
```

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 );;
```

## 15.3 HandlePrecompiledTowers

## Example

```
gap> LoadPackage( "CAP", false );
true
gap> dummy1 := CreateCapCategory( );;
gap> dummy2 := CreateCapCategory( );;
gap> dummy3 := CreateCapCategory( );;
gap> PrintAndReturn := function ( string )
>   Print( string, "\n" ); return string; end;;
gap> dummy1!.compiler_hints := rec( );;
gap> dummy1!.compiler_hints.precompiled_towers := [
>   rec(
>     remaining_constructors_in_tower := [ "Constructor1" ],
>     precompiled_functions_adder := cat ->
>       PrintAndReturn( "Adding precompiled operations for Constructor1" ),
>   ),
>   rec(
>     remaining_constructors_in_tower := [ "Constructor1", "Constructor2" ],
>     precompiled_functions_adder := cat ->
```

```

>      PrintAndReturn( "Adding precompiled operations for Constructor2" ),
>    ),
> ];;
gap> HandlePrecompiledTowers( dummy2, dummy1, "Constructor1" );
Adding precompiled operations for Constructor1
gap> HandlePrecompiledTowers( dummy3, dummy2, "Constructor2" );
Adding precompiled operations for Constructor2

```

## 15.4 Homomorphism structure

Example

```

gap> ReadPackage( "CAP", "examples/FieldAsCategory.g" );;
gap> Q := HomalgFieldOfRationals();;
gap> Qoid := FieldAsCategory( Q );;
gap> a := FieldAsCategoryMorphism( Qoid, 1/2 );;
gap> b := FieldAsCategoryMorphism( Qoid, -2/3 );;
gap> u := FieldAsCategoryUniqueObject( Qoid );;
gap> IsCongruentForMorphisms( a,
>   InterpretMorphismFromDistinguishedObjectToHomomorphismStructureAsMorphism(
>     u,u,
>     InterpretMorphismAsMorphismFromDistinguishedObjectToHomomorphismStructure(
>       a
>     )
>   )
> );
true
gap> a = HomStructure( u, u, HomStructure( a ) );
true
gap> IsEqualForObjects( HomStructure( Qoid ), DistinguishedObjectOfHomomorphismStructure( Qoid ) );
true
gap> c := FieldAsCategoryMorphism( Qoid, 3 );;
gap> d := FieldAsCategoryMorphism( Qoid, 0 );;
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> MereExistenceOfSolutionOfLinearSystemInAbCategory( left_coeffs, right_coeffs, right_side );
true
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][j] ] )
>     )
>   );
true
gap> IsLiftable( c, d );
false

```

```

gap> LiftOrFail( c, d );
fail
gap> IsLiftable( d, c );
true
gap> LiftOrFail( d, c );
0
gap> Lift( d, c );
0
gap> IsColiftable( c, d );
true
gap> ColiftOrFail( c, d );
0
gap> Colift( c, d );
0
gap> IsColiftable( d, c );
false
gap> ColiftOrFail( d, c );
fail

```

#### Example

```

gap> ReadPackage( "CAP", "examples/StringsAsCategory.g" );;
gap> C := StringsAsCategory();;
gap> obj1 := StringsAsCategoryObject( C, "qaeiou" );;
gap> obj2 := StringsAsCategoryObject( C, "qxayeziouT" );;
gap> mor := StringsAsCategoryMorphism( C, obj1, "xyzaTe", obj2 );;
gap> IsWellDefined( mor );
true
gap> ## Test SimplifyObject
> IsEqualForObjects( SimplifyObject( obj1, 0 ), obj1 );
true
gap> IsEqualForObjects( SimplifyObject( obj1, 1 ), obj1 );
false
gap> ForAny( [0,1,2,3,4], i -> IsEqualForObjects( SimplifyObject( obj1, i ), SimplifyObject( obj1, i ) );
false
gap> ForAll( [5,6,7,8], i -> IsEqualForObjects( SimplifyObject( obj1, i ), SimplifyObject( obj1, i ) );
true
gap> ## Test SimplifyMorphism
> IsEqualForMorphisms( SimplifyMorphism( mor, 0 ), mor );
true
gap> IsEqualForMorphisms( SimplifyMorphism( mor, 1 ), mor );
false
gap> ForAny( [0,1], i -> IsEqualForMorphisms( SimplifyMorphism( mor, i ), SimplifyMorphism( mor, i ) );
false
gap> ForAll( [2,3,4,5], i -> IsEqualForMorphisms( SimplifyMorphism( mor, i ), SimplifyMorphism( mor, i ) );
true
gap> ## Test SimplifySource
> IsEqualForMorphismsOnMor( SimplifySource( mor, 0 ), mor );
true
gap> IsEqualForMorphismsOnMor( SimplifySource( mor, 1 ), mor );
false
gap> ForAny( [0,1,2,3,4], i -> IsEqualForMorphismsOnMor( SimplifySource( mor, i ), SimplifySource( mor, i ) );
false
gap> ForAll( [5,6,7,8,9], i -> IsEqualForMorphismsOnMor( SimplifySource( mor, i ), SimplifySource( mor, i ) );

```

```

true
gap> IsCongruentForMorphisms(
>   PreCompose( SimplifySource_IsoFromInputObject( mor, infinity ), SimplifySource( mor, infinity )
> );
true
gap> IsCongruentForMorphisms(
>   PreCompose( SimplifySource_IsoToInputObject( mor, infinity ), mor ), SimplifySource( mor, infinity )
> );
true
gap> ## Test SimplifyRange
> IsEqualForMorphismsOnMor( SimplifyRange( mor, 0 ), mor );
true
gap> IsEqualForMorphismsOnMor( SimplifyRange( mor, 1 ), mor );
false
gap> ForAny( [0,1,2,3,4], i -> IsEqualForMorphismsOnMor( SimplifyRange( mor, i ), SimplifyRange( mor, 0 ) )
false
gap> ForAll( [5,6,7,8,9], i -> IsEqualForMorphismsOnMor( SimplifyRange( mor, i ), SimplifyRange( mor, 0 ) )
true
gap> IsCongruentForMorphisms(
>   PreCompose( SimplifyRange( mor, infinity ), SimplifyRange_IsoToInputObject( mor, infinity )
> );
true
gap> IsCongruentForMorphisms(
>   PreCompose( mor, SimplifyRange_IsoFromInputObject( mor, infinity ) ), SimplifyRange( mor, infinity )
> );
true
gap> ## Test SimplifySourceAndRange
> IsEqualForMorphismsOnMor( SimplifySourceAndRange( mor, 0 ), mor );
true
gap> IsEqualForMorphismsOnMor( SimplifySourceAndRange( mor, 1 ), mor );
false
gap> ForAny( [0,1,2,3,4], i -> IsEqualForMorphismsOnMor( SimplifySourceAndRange( mor, i ), SimplifySourceAndRange( mor, 0 ) )
false
gap> ForAll( [5,6,7,8,9], i -> IsEqualForMorphismsOnMor( SimplifySourceAndRange( mor, i ), SimplifySourceAndRange( mor, 0 ) )
true
gap> IsCongruentForMorphisms(
>   mor,
>   PreCompose( [ SimplifySourceAndRange_IsoFromInputSource( mor, infinity ),
>                 SimplifySourceAndRange( mor, infinity ),
>                 SimplifySourceAndRange_IsoToInputRange( mor, infinity ) ] )
> );
true
gap> IsCongruentForMorphisms(
>   SimplifySourceAndRange( mor, infinity ),
>   PreCompose( [ SimplifySourceAndRange_IsoToInputSource( mor, infinity ),
>                 mor,
>                 SimplifySourceAndRange_IsoFromInputRange( mor, infinity ) ] )
> );
true
gap> ## Test SimplifyEndo
> endo := StringsAsCategoryMorphism( C, obj1, "uoiea", obj1 );
gap> IsWellDefined( endo );

```

```

true
gap> IsEqualForMorphismsOnMor( SimplifyEndo( endo, 0 ), endo );
true
gap> IsEqualForMorphismsOnMor( SimplifyEndo( endo, 1 ), endo );
false
gap> ForAny( [0,1,2,3,4], i -> IsEqualForMorphismsOnMor( SimplifySourceAndRange( endo, i ), SimplifyEndo( endo, i ) );
false
gap> ForAll( [5,6,7,8,9], i -> IsEqualForMorphismsOnMor( SimplifySourceAndRange( endo, i ), SimplifyEndo( endo, i ) );
true
gap> iota := SimplifyEndo_IsoToInputObject( endo, infinity );
gap> iota_inv := SimplifyEndo_IsoFromInputObject( endo, infinity );
gap> IsCongruentForMorphisms( PreCompose( [ iota_inv, SimplifyEndo( endo, infinity ) ], iota ), endo );
true

```

## 15.5 Homology object

### Example

```

gap> field := HomalgFieldOfRationals( );
gap> A := VectorSpaceObject( 1, field );
gap> B := VectorSpaceObject( 2, field );
gap> C := VectorSpaceObject( 3, field );
gap> alpha := VectorSpaceMorphism( A, HomalgMatrix( [ [ 1, 0, 0 ] ], 1, 3, field ), C );
gap> beta := VectorSpaceMorphism( C, HomalgMatrix( [ [ 1, 0 ], [ 1, 1 ], [ 1, 2 ] ], 3, 2, field ), B );
gap> IsZero( PreCompose( alpha, beta ) );
false
gap> IsCongruentForMorphisms(
>   IdentityMorphism( HomologyObject( alpha, beta ) ),
>   HomologyObjectFunctorial( alpha, beta, IdentityMorphism( C ), alpha, beta )
> );
true
gap> kernel_beta := KernelEmbedding( beta );
gap> K := Source( kernel_beta );
gap> IsIsomorphism(
>   HomologyObjectFunctorial(
>     MorphismFromZeroObject( K ),
>     MorphismIntoZeroObject( K ),
>     kernel_beta,
>     MorphismFromZeroObject( Source( beta ) ),
>     beta
>   )
> );
true
gap> cokernel_alpha := CokernelProjection( alpha );
gap> Co := Range( cokernel_alpha );
gap> IsIsomorphism(
>   HomologyObjectFunctorial(
>     alpha,
>     MorphismIntoZeroObject( Range( alpha ) ),
>     cokernel_alpha,
>     MorphismFromZeroObject( Co ),
>     MorphismIntoZeroObject( Co )
>   )
> );

```

```

> );
true
gap> alpha_op := Opposite( alpha );;
gap> beta_op := Opposite( beta );;
gap> IsCongruentForMorphisms(
>   IdentityMorphism( HomologyObject( beta_op, alpha_op ) ),
>   HomologyObjectFunctorial( beta_op, alpha_op, IdentityMorphism( Opposite( C ) ), beta_op, al
> );
true
gap> kernel_beta := KernelEmbedding( beta_op );;
gap> K := Source( kernel_beta );;
gap> IsIsomorphism(
>   HomologyObjectFunctorial(
>     MorphismFromZeroObject( K ),
>     MorphismIntoZeroObject( K ),
>     kernel_beta,
>     MorphismFromZeroObject( Source( beta_op ) ),
>     beta_op
>   )
> );
true
gap> cokernel_alpha := CokernelProjection( alpha_op );;
gap> Co := Range( cokernel_alpha );;
gap> IsIsomorphism(
>   HomologyObjectFunctorial(
>     alpha_op,
>     MorphismIntoZeroObject( Range( alpha_op ) ),
>     cokernel_alpha,
>     MorphismFromZeroObject( Co ),
>     MorphismIntoZeroObject( Co )
>   )
> );
true

```

## 15.6 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
gap> PreCompose( PreInverseForMorphisms( gamma ), gamma ) = IdentityMorphism( V );
true
gap> PreCompose( alpha, PostInverseForMorphisms( alpha ) ) = IdentityMorphism( V );
true

```

## 15.7 Monoidal Categories

Example

```

gap> ZZZ := HomalgRingOfIntegers();
gap> M1 := AsLeftPresentation( HomalgMatrix( [ [ 2 ] ], 1, 1, ZZZ ) );
<An object in Category of left presentations of Z>
gap> N1 := AsLeftPresentation( HomalgMatrix( [ [ 3 ] ], 1, 1, ZZZ ) );
<An object in Category of left presentations of Z>
gap> T1 := TensorProductOnObjects( M1, N1 );
<An object in Category of left presentations of Z>
gap> Display( UnderlyingMatrix( T1 ) );
[ [ 3 ],
  [ 2 ] ]
gap> IsZeroForObjects( T1 );
true
gap> B1 := Braiding( DirectSum( M1, N1 ), DirectSum( M1, M1 ) );
<A morphism in Category of left presentations of Z>
gap> Display( UnderlyingMatrix( B1 ) );
[ [ 1, 0, 0, 0 ],
  [ 0, 0, 1, 0 ],
  [ 0, 1, 0, 0 ],
  [ 0, 0, 0, 1 ] ]
gap> IsWellDefined( B1 );
true
gap> U1 := TensorUnit( CapCategory( M1 ) );
<An object in Category of left presentations of Z>
gap> IntHom1 := InternalHomOnObjects( DirectSum( M1, U1 ), N1 );
<An object in Category of left presentations of Z>
gap> Display( UnderlyingMatrix( IntHom1 ) );
[ [ 1, 2 ],
  [ 0, 3 ] ]
gap> generator_l1 := StandardGeneratorMorphism( IntHom1, 1 );
<A morphism in Category of left presentations of Z>
gap> morphism_l1 := LambdaElimination( DirectSum( M1, U1 ), N1, generator_l1 );
<A morphism in Category of left presentations of Z>
gap> Display( UnderlyingMatrix( morphism_l1 ) );
[ [ -3 ],
  [ 2 ] ]
gap> generator_l2 := StandardGeneratorMorphism( IntHom1, 2 );
<A morphism in Category of left presentations of Z>
gap> morphism_l2 := LambdaElimination( DirectSum( M1, U1 ), N1, generator_l2 );
<A morphism in Category of left presentations of Z>
gap> Display( UnderlyingMatrix( morphism_l2 ) );
[ [ 0 ],

```

```

[ [-1] ]
gap> IsEqualForMorphisms( LambdaIntroduction( morphism_l1 ), generator_l1 );
false
gap> IsCongruentForMorphisms( LambdaIntroduction( morphism_l1 ), generator_l1 );
true
gap> IsEqualForMorphisms( LambdaIntroduction( morphism_l2 ), generator_l2 );
false
gap> IsCongruentForMorphisms( LambdaIntroduction( morphism_l2 ), generator_l2 );
true
gap> Mr := AsRightPresentation( HomalgMatrix( [[ 2 ] ], 1, 1, ZZZ ) );
<An object in Category of right presentations of Z>
gap> Nr := AsRightPresentation( HomalgMatrix( [[ 3 ] ], 1, 1, ZZZ ) );
<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, Nr ) );
<A morphism in Category of right presentations of Z>
gap> Display( UnderlyingMatrix( Br ) );
[[ 1, 0, 0, 0 ],
 [ 0, 0, 1, 0 ],
 [ 0, 1, 0, 0 ],
 [ 0, 0, 0, 1 ] ]
gap> IsWellDefined( Br );
true
gap> Ur := TensorUnit( CapCategory( Mr ) );
<An object in Category of right presentations of Z>
gap> IntHomr := InternalHomOnObjects( DirectSum( Mr, Ur ), Nr );
<An object in Category of right presentations of Z>
gap> Display( UnderlyingMatrix( IntHomr ) );
[[ 1, 0 ],
 [ 2, 3 ] ]
gap> generator_r1 := StandardGeneratorMorphism( IntHomr, 1 );
<A morphism in Category of right presentations of Z>
gap> morphism_r1 := LambdaElimination( DirectSum( Mr, Ur ), Nr, generator_r1 );
<A morphism in Category of right presentations of Z>
gap> Display( UnderlyingMatrix( morphism_r1 ) );
[[ -3, 2 ] ]
gap> generator_r2 := StandardGeneratorMorphism( IntHoml, 2 );
<A morphism in Category of left presentations of Z>
gap> morphism_r2 := LambdaElimination( DirectSum( Ml, Ul ), Nl, generator_r2 );
<A morphism in Category of left presentations of Z>
gap> Display( UnderlyingMatrix( morphism_r2 ) );
[[ 0 ],
 [ -1 ] ]
gap> IsEqualForMorphisms( LambdaIntroduction( morphism_r1 ), generator_r1 );
false
gap> IsCongruentForMorphisms( LambdaIntroduction( morphism_r1 ), generator_r1 );
true

```

```

gap> IsEqualForMorphisms( LambdaIntroduction( morphism_r2 ), generator_r2 );
false
gap> IsCongruentForMorphisms( LambdaIntroduction( morphism_r2 ), generator_r2 );
true

```

## 15.8 MorphismFromSourceToPushout and MorphismFromFiberProductToSink

Example

```

gap> field := HomalgFieldOfRationals( );
gap> A := VectorSpaceObject( 3, field );
gap> B := VectorSpaceObject( 2, field );
gap> alpha := VectorSpaceMorphism( B, HomalgMatrix( [ [ 1, -1, 1 ], [ 1, 1, 1 ] ], 2, 3, field ), A );
gap> beta := VectorSpaceMorphism( B, HomalgMatrix( [ [ 1, 2, 1 ], [ 2, 1, 1 ] ], 2, 3, field ), A );
gap> m := MorphismFromFiberProductToSink( [ alpha, beta ] );
gap> IsCongruentForMorphisms(
>   m,
>   PreCompose( ProjectionInFactorOfFiberProduct( [ alpha, beta ], 1 ), alpha )
> );
true
gap> IsCongruentForMorphisms(
>   m,
>   PreCompose( ProjectionInFactorOfFiberProduct( [ alpha, beta ], 2 ), beta )
> );
true
gap> IsCongruentForMorphisms(
>   MorphismFromKernelObjectToSink( alpha ),
>   PreCompose( KernelEmbedding( alpha ), alpha )
> );
true
gap> alpha_p := DualOnMorphisms( alpha );
gap> beta_p := DualOnMorphisms( beta );
gap> m_p := MorphismFromSourceToPushout( [ alpha_p, beta_p ] );
gap> IsCongruentForMorphisms(
>   m_p,
>   PreCompose( alpha_p, InjectionOfCofactorOfPushout( [ alpha_p, beta_p ], 1 ) )
> );
true
gap> IsCongruentForMorphisms(
>   m_p,
>   PreCompose( beta_p, InjectionOfCofactorOfPushout( [ alpha_p, beta_p ], 2 ) )
> );
true
gap> IsCongruentForMorphisms(
>   MorphismFromSourceToCokernelObject( alpha_p ),
>   PreCompose( alpha_p, CokernelProjection( alpha_p ) )
> );
true

```

## 15.9 Opposite category

Example

```
gap> QQ := HomalgFieldOfRationals();;
gap> vec := MatrixCategory( QQ );;
gap> op := Opposite( vec );;
gap> ListKnownCategoricalProperties( op );
[ "IsAbCategory", "IsAbelianCategory", "IsAbelianCategoryWithEnoughInjectives",
  "IsAbelianCategoryWithEnoughProjectives", "IsAdditiveCategory",
  "IsBraidedMonoidalCategory", "IsClosedMonoidalCategory",
  "IsCoclosedMonoidalCategory", "IsEnrichedOverCommutativeRegularSemigroup",
  "IsEquippedWithHomomorphismStructure", "IsLinearCategoryOverCommutativeRing",
  "IsLinearCategoryOverCommutativeRingWithFinitelyGeneratedFreeExternalHoms",
  "IsMonoidalCategory", "IsPreAbelianCategory",
  "IsRigidSymmetricClosedMonoidalCategory",
  "IsRigidSymmetricCoclosedMonoidalCategory", "IsSkeletalCategory",
  "IsStrictMonoidalCategory", "IsSymmetricClosedMonoidalCategory",
  "IsSymmetricCoclosedMonoidalCategory", "IsSymmetricMonoidalCategory" ]
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> IsWellDefined( MorphismBetweenDirectSums( op, [ ], [ ], [ V1 ] ) );
true
gap> IsWellDefined( MorphismBetweenDirectSums( op, [ V1 ], [ [ ] ], [ ] ) );
true
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, [ CostrictionToImage( mor ), ImageEmbedding( mor ) ] ) );
true
```

## 15.10 PreComposeList and PostComposeList

Example

```
gap> field := HomalgFieldOfRationals( );;
gap> A := VectorSpaceObject( 1, field );;
gap> B := VectorSpaceObject( 2, field );;
gap> C := VectorSpaceObject( 3, field );;
```

```

gap> alpha := VectorSpaceMorphism( A, HomalgMatrix( [ [ 1, 0, 0 ] ], 1, 3, field ), C );
gap> beta := VectorSpaceMorphism( C, HomalgMatrix( [ [ 1, 0 ], [ 1, 1 ], [ 1, 2 ] ], 3, 2, field
gap> IsCongruentForMorphisms( PreCompose( alpha, beta ), PostCompose( beta, alpha ) );
true
gap> IsOne( PreComposeList( A, [ ], A ) );
true
gap> IsCongruentForMorphisms( PreComposeList( A, [ alpha ], C ), alpha );
true
gap> IsCongruentForMorphisms( PreComposeList( A, [ alpha, beta ], B ), PreCompose( alpha, beta )
true
gap> IsOne( PostComposeList( A, [ ], A ) );
true
gap> IsCongruentForMorphisms( PostComposeList( A, [ alpha ], C ), alpha );
true
gap> IsCongruentForMorphisms( PostComposeList( A, [ beta, alpha ], B ), PostCompose( beta, alpha
true

```

## 15.11 Terminal category

Example

```

gap> LoadPackage( "MonoidalCategories", ">= 2024.06-01", false );
true
gap> T := TerminalCategoryWithMultipleObjects( );
TerminalCategoryWithMultipleObjects( )
gap> Display( T );
A CAP category with name TerminalCategoryWithMultipleObjects( ):

82 primitive operations were used to derive 391 operations for this category \
which algorithmically
* IsCategoryWithDecidableColifts
* IsCategoryWithDecidableLifts
* IsEquippedWithHomomorphismStructure
* IsLinearCategoryOverCommutativeRing
* IsLeftClosedMonoidalCategory
* IsLeftCoclosedMonoidalCategory
* IsAbelianCategoryWithEnoughInjectives
* IsAbelianCategoryWithEnoughProjectives
* IsRigidSymmetricClosedMonoidalCategory
* IsRigidSymmetricCoclosedMonoidalCategory
and not yet algorithmically
* IsLinearCategoryOverCommutativeRingWithFinitelyGeneratedFreeExternalHoms
and furthermore mathematically
* IsLocallyOfFiniteInjectiveDimension
* IsLocallyOfFiniteProjectiveDimension
* IsTerminalCategory
gap> i := InitialObject( T );
<A zero object in TerminalCategoryWithMultipleObjects( )>
gap> t := TerminalObject( T );
<A zero object in TerminalCategoryWithMultipleObjects( )>
gap> z := ZeroObject( T );
<A zero object in TerminalCategoryWithMultipleObjects( )>
gap> Display( i );

```

```

ZeroObject
gap> Display( t );
ZeroObject
gap> Display( z );
ZeroObject
gap> IsIdenticalObj( i, z );
true
gap> IsIdenticalObj( t, z );
true
gap> id_z := IdentityMorphism( z );
<A zero, identity morphism in TerminalCategoryWithMultipleObjects( )>
gap> fn_z := ZeroObjectFunctorial( T );
<A zero, isomorphism in TerminalCategoryWithMultipleObjects( )>
gap> IsEqualForMorphisms( id_z, fn_z );
false
gap> IsCongruentForMorphisms( id_z, fn_z );
true
gap> a := "a" / T;
<A zero object in TerminalCategoryWithMultipleObjects( )>
gap> Display( a );
a
gap> IsWellDefined( a );
true
gap> aa := ObjectConstructor( T, "a" );
<A zero object in TerminalCategoryWithMultipleObjects( )>
gap> Display( aa );
a
gap> IsEqualForObjects( a, aa );
true
gap> IsIsomorphicForObjects( a, aa );
true
gap> IsIsomorphism( SomeIsomorphismBetweenObjects( a, aa ) );
true
gap> b := "b" / T;
<A zero object in TerminalCategoryWithMultipleObjects( )>
gap> Display( b );
b
gap> IsEqualForObjects( a, b );
false
gap> IsIsomorphicForObjects( a, b );
true
gap> mor_ab := SomeIsomorphismBetweenObjects( a, b );
<A zero, isomorphism in TerminalCategoryWithMultipleObjects( )>
gap> IsIsomorphism( mor_ab );
true
gap> Display( mor_ab );
a
|
| SomeIsomorphismBetweenObjects
v
b
gap> Hom_ab := MorphismsOfExternalHom( a, b );;
```

```

gap> Length( Hom_ab );
1
gap> Hom_ab[1];
<A zero, isomorphism in TerminalCategoryWithMultipleObjects( )>
gap> Display( Hom_ab[1] );
a
|
| InterpretMorphismFromDistinguishedObjectToHomomorphismStructureAsMorphism
v
b
gap> Hom_ab[1] = mor_ab;
true
gap> HomStructure( mor_ab );
<A zero, identity morphism in TerminalCategoryWithSingleObject( )>
gap> t := TensorProduct( a, b );
<A zero object in TerminalCategoryWithMultipleObjects( )>
gap> Display( t );
TensorProductOnObjects
gap> a = t;
false
gap> TensorProduct( a, a ) = t;
true
gap> m := MorphismConstructor( a, "m", b );
<A zero, isomorphism in TerminalCategoryWithMultipleObjects( )>
gap> Display( m );
a
|
| m
v
b
gap> IsWellDefined( m );
true
gap> n := MorphismConstructor( a, "n", b );
<A zero, isomorphism in TerminalCategoryWithMultipleObjects( )>
gap> Display( n );
a
|
| n
v
b
gap> IsEqualForMorphisms( m, n );
false
gap> IsCongruentForMorphisms( m, n );
true
gap> m = n;
true
gap> hom_mn := HomStructure( m, n );
<A zero, identity morphism in TerminalCategoryWithSingleObject( )>
gap> id := IdentityMorphism( a );
<A zero, identity morphism in TerminalCategoryWithMultipleObjects( )>
gap> Display( id );
a

```

```

|
| IdentityMorphism
v
a
gap> m = id;
false
gap> id = MorphismConstructor( a, "xyz", a );
true
gap> zero := ZeroMorphism( a, a );
<A zero, isomorphism in TerminalCategoryWithMultipleObjects( )>
gap> Display( zero );
a
|
| ZeroMorphism
v
a
gap> id = zero;
true
gap> IsLiftable( m, n );
true
gap> lift := Lift( m, n );
<A zero, isomorphism in TerminalCategoryWithMultipleObjects( )>
gap> Display( lift );
a
|
| Lift
v
a
gap> IsColiftable( m, n );
true
gap> colift := Colift( m, n );
<A zero, isomorphism in TerminalCategoryWithMultipleObjects( )>
gap> Display( colift );
b
|
| Colift
v
b
gap> DirectProduct( T, [ ] );
<A zero object in TerminalCategoryWithMultipleObjects( )>
gap> Equalizer( T, z, [ ] );
<A zero object in TerminalCategoryWithMultipleObjects( )>
gap> Coproduct( T, [ ] );
<A zero object in TerminalCategoryWithMultipleObjects( )>
gap> Coequalizer( T, z, [ ] );
<A zero object in TerminalCategoryWithMultipleObjects( )>

```

Example

```

gap> LoadPackage( "MonoidalCategories", ">= 2024.06-01", false );
true
gap> T := TerminalCategoryWithSingleObject( );
TerminalCategoryWithSingleObject( )
gap> Display( T );

```



```

A CAP category with name TerminalCategoryWithSingleObject( ):

76 primitive operations were used to derive 391 operations for this category \
which algorithmically
* IsCategoryWithDecidableColifts
* IsCategoryWithDecidableLifts
* IsEquippedWithHomomorphismStructure
* IsLinearCategoryOverCommutativeRing
* IsLeftClosedMonoidalCategory
* IsLeftCoclosedMonoidalCategory
* IsAbelianCategoryWithEnoughInjectives
* IsAbelianCategoryWithEnoughProjectives
* IsRigidSymmetricClosedMonoidalCategory
* IsRigidSymmetricCoclosedMonoidalCategory
and not yet algorithmically
* IsLinearCategoryOverCommutativeRingWithFinitelyGeneratedFreeExternalHoms
and furthermore mathematically
* IsLocallyOfFiniteInjectiveDimension
* IsLocallyOfFiniteProjectiveDimension
* IsSkeletalCategory
* IsStrictMonoidalCategory
* IsTerminalCategory
gap> i := InitialObject( T );
<A zero object in TerminalCategoryWithSingleObject( )>
gap> t := TerminalObject( T );
<A zero object in TerminalCategoryWithSingleObject( )>
gap> z := ZeroObject( T );
<A zero object in TerminalCategoryWithSingleObject( )>
gap> Display( i );
A zero object in TerminalCategoryWithSingleObject( ).
gap> Display( t );
A zero object in TerminalCategoryWithSingleObject( ).
gap> Display( z );
A zero object in TerminalCategoryWithSingleObject( ).
gap> IsIdenticalObj( i, z );
true
gap> IsIdenticalObj( t, z );
true
gap> IsWellDefined( z );
true
gap> id_z := IdentityMorphism( z );
<A zero, identity morphism in TerminalCategoryWithSingleObject( )>
gap> fn_z := ZeroObjectFunctorial( T );
<A zero, identity morphism in TerminalCategoryWithSingleObject( )>
gap> IsWellDefined( fn_z );
true
gap> IsEqualForMorphisms( id_z, fn_z );
true
gap> IsCongruentForMorphisms( id_z, fn_z );
true
gap> IsLiftable( id_z, fn_z );
true

```

```

gap> Lift( id_z, fn_z );
<A zero, identity morphism in TerminalCategoryWithSingleObject( )>
gap> IsColiftable( id_z, fn_z );
true
gap> Colift( id_z, fn_z );
<A zero, identity morphism in TerminalCategoryWithSingleObject( )>
gap> DirectProduct( T, [ ] );
<A zero object in TerminalCategoryWithSingleObject( )>
gap> Equalizer( T, z, [ ] );
<A zero object in TerminalCategoryWithSingleObject( )>
gap> Coproduct( T, [ ] );
<A zero object in TerminalCategoryWithSingleObject( )>
gap> Coequalizer( T, z, [ ] );
<A zero object in TerminalCategoryWithSingleObject( )>

```

## 15.12 WrapperCategory

Example

```

gap> LoadPackage( "LinearAlgebraForCAP", false );
true
gap> Q := HomalgFieldOfRationals( );
Q
gap> Qmat := MATRIX_CATEGORY( Q );
Category of matrices over Q
gap> Wrapper := WrapperCategory( Qmat, rec( ) );
WrapperCategory( Category of matrices over Q )
gap> mor := ZeroMorphism( ZeroObject( Wrapper ), ZeroObject( Wrapper ) );
gap> 2 * mor;;
gap> BasisOfExternalHom( Source( mor ), Range( mor ) );
gap> CoefficientsOfMorphism( mor );
gap> distinguished_object := DistinguishedObjectOfHomomorphismStructure( Wrapper );
gap> object := HomomorphismStructureOnObjects( Source( mor ), Source( mor ) );
gap> HomomorphismStructureOnMorphisms( mor, mor );
gap> HomomorphismStructureOnMorphismsWithGivenObjects( object, mor, mor, object );
gap> iota := InterpretMorphismAsMorphismFromDistinguishedObjectToHomomorphismStructure( mor );
gap> InterpretMorphismAsMorphismFromDistinguishedObjectToHomomorphismStructureWithGivenObjects( object, mor, mor, object );
gap> beta := InterpretMorphismFromDistinguishedObjectToHomomorphismStructureAsMorphism( Source( mor ), mor );
gap> IsCongruentForMorphisms( mor, beta );
true

```

## Chapter 16

# Terminal category

### 16.1 GAP Categories

#### 16.1.1 IsCapTerminalCategoryWithSingleObject (for IsCapCategory)

- ▷ IsCapTerminalCategoryWithSingleObject( $T$ ) (filter)  
**Returns:** true or false  
The GAP type of a terminal category with a single object.

#### 16.1.2 IsObjectInCapTerminalCategoryWithSingleObject (for IsCapCategoryObject)

- ▷ IsObjectInCapTerminalCategoryWithSingleObject( $T$ ) (filter)  
**Returns:** true or false  
The GAP type of an object in a terminal category with a single object.

#### 16.1.3 IsMorphismInCapTerminalCategoryWithSingleObject (for IsCapCategoryMorphism)

- ▷ IsMorphismInCapTerminalCategoryWithSingleObject( $T$ ) (filter)  
**Returns:** true or false  
The GAP type of a morphism in a terminal category with a single object.

#### 16.1.4 IsCapTerminalCategoryWithMultipleObjects (for IsCapCategory)

- ▷ IsCapTerminalCategoryWithMultipleObjects( $T$ ) (filter)  
**Returns:** true or false  
The GAP type of a terminal category with multiple objects.

#### 16.1.5 IsObjectInCapTerminalCategoryWithMultipleObjects (for IsCapCategoryObject)

- ▷ IsObjectInCapTerminalCategoryWithMultipleObjects( $T$ ) (filter)  
**Returns:** true or false  
The GAP type of an object in a terminal category with multiple objects.

### 16.1.6 IsMorphismInCapTerminalCategoryWithMultipleObjects (for IsCapCategoryMorphism)

▷ IsMorphismInCapTerminalCategoryWithMultipleObjects(*T*) (filter)

**Returns:** true or false

The GAP type of a morphism in a terminal category with multiple objects.

### 16.1.7 IsTerminalCategory (for IsCapCategory)

▷ IsTerminalCategory(*C*) (property)

**Returns:** true or false

The property of the category *C* being terminal.

## 16.2 Constructors

### 16.2.1 TerminalCategoryWithSingleObject

▷ TerminalCategoryWithSingleObject(*arg*) (function)

Construct a terminal category with a single object.

### 16.2.2 TerminalCategoryWithMultipleObjects

▷ TerminalCategoryWithMultipleObjects(*arg*) (function)

Construct a terminal category with multiple objects.

### 16.2.3 CAP\_INTERNAL\_CONSTRUCTOR\_FOR\_TERMINAL\_CATEGORY

▷ CAP\_INTERNAL\_CONSTRUCTOR\_FOR\_TERMINAL\_CATEGORY(*options*) (function)

**Returns:** a CAP category

This function takes a record of options suited for CategoryConstructor. It makes common adjustments for TerminalCategoryWithSingleObject and TerminalCategoryWithMultipleObjects to the list of operations to install and the categorical properties of the given record, before passing it on to CategoryConstructor.

## 16.3 Attributes

### 16.3.1 UniqueObject (for IsCapTerminalCategoryWithSingleObject)

▷ UniqueObject(*arg*) (attribute)

**Returns:** a CAP object

The unique object in a terminal category with a single object.

### 16.3.2 UniqueMorphism (for IsCapTerminalCategoryWithSingleObject)

▷ UniqueMorphism(*arg*) (attribute)

**Returns:** a CAP morphism

The unique morphism in a terminal category with a single object.

### 16.3.3 FunctorFromTerminalCategory (for IsCapCategoryObject)

▷ FunctorFromTerminalCategory(*object*) (attribute)

**Returns:** a CAP functor

A functor from AsCapCategory( TerminalObject( CapCat ) ) mapping the unique object to *object*.

# Index

- \*
  - for IsRingElement, IsCapCategoryMorphism, 34
- /
  - for IsCapCategoryCell, IsWrapperCapCategory, 187
  - for IsObject, IsCapCategory, 22
- ActivateDerivationInfo, 156
- Add
  - for IsCapCategory, IsCapCategoryMorphism, 32
  - for IsCapCategory, IsCapCategoryObject, 21
  - for IsStringMinHeap, IsString, IsInt, 162
- AddAdditionForMorphisms
  - for IsCapCategory, IsFunction, 105
- AddAdditiveGenerators
  - for IsCapCategory, IsFunction, 105
- AddAdditiveInverseForMorphisms
  - for IsCapCategory, IsFunction, 105
- AddAstrictionToCoimage
  - for IsCapCategory, IsFunction, 105
- AddAstrictionToCoimageWithGivenCoimageObject
  - for IsCapCategory, IsFunction, 105
- AddBasisOfExternalHom
  - for IsCapCategory, IsFunction, 106
- AddCategoricalProperty, 7
- AddCoastrictionToImage
  - for IsCapCategory, IsFunction, 106
- AddCoastrictionToImageWithGivenImageObject
  - for IsCapCategory, IsFunction, 106
- AddCoefficientsOfMorphism
  - for IsCapCategory, IsFunction, 106
- AddCoequalizer
  - for IsCapCategory, IsFunction, 106
- AddCoequalizerFunctorial
  - for IsCapCategory, IsFunction, 106
- AddCoequalizerFunctorialWithGivenCoequalizers
  - for IsCapCategory, IsFunction, 107
- AddCoimageObject
  - for IsCapCategory, IsFunction, 107
- AddCoimageObjectFunctorial
  - for IsCapCategory, IsFunction, 107
- AddCoimageObjectFunctorialWithGivenCoimageObjects
  - for IsCapCategory, IsFunction, 107
- AddCoimageProjection
  - for IsCapCategory, IsFunction, 107
- AddCoimageProjectionWithGivenCoimageObject
  - for IsCapCategory, IsFunction, 107
- AddCokernelColift
  - for IsCapCategory, IsFunction, 108
- AddCokernelColiftWithGivenCokernelObject
  - for IsCapCategory, IsFunction, 108
- AddCokernelObject
  - for IsCapCategory, IsFunction, 108
- AddCokernelObjectFunctorial
  - for IsCapCategory, IsFunction, 108
- AddCokernelObjectFunctorialWithGivenCokernelObjects
  - for IsCapCategory, IsFunction, 108
- AddCokernelProjection
  - for IsCapCategory, IsFunction, 109
- AddCokernelProjectionWithGivenCokernelObject
  - for IsCapCategory, IsFunction, 109
- AddColift
  - for IsCapCategory, IsFunction, 109
- AddColiftAlongEpimorphism
  - for IsCapCategory, IsFunction, 109
- AddColiftOrFail
  - for IsCapCategory, IsFunction, 109

- AddComponentOfMorphismFromCoproduct  
for IsCapCategory, IsFunction, [109](#)
- AddComponentOfMorphismFromDirectSum  
for IsCapCategory, IsFunction, [110](#)
- AddComponentOfMorphismIntoDirect-  
Product  
for IsCapCategory, IsFunction, [110](#)
- AddComponentOfMorphismIntoDirectSum  
for IsCapCategory, IsFunction, [110](#)
- AddCoproduct  
for IsCapCategory, IsFunction, [110](#)
- AddCoproductFunctorial  
for IsCapCategory, IsFunction, [110](#)
- AddCoproductFunctorialWithGiven-  
Coproducts  
for IsCapCategory, IsFunction, [110](#)
- AddDerivation, [159](#)
- AddDerivationToCAP, [159](#)
- AddDirectProduct  
for IsCapCategory, IsFunction, [111](#)
- AddDirectProductFunctorial  
for IsCapCategory, IsFunction, [111](#)
- AddDirectProductFunctorialWithGiven-  
DirectProducts  
for IsCapCategory, IsFunction, [111](#)
- AddDirectSum  
for IsCapCategory, IsFunction, [111](#)
- AddDirectSumFunctorial  
for IsCapCategory, IsFunction, [111](#)
- AddDirectSumFunctorialWithGivenDirect-  
Sums  
for IsCapCategory, IsFunction, [111](#)
- AddDistinguishedObjectOfHomomorphism-  
Structure  
for IsCapCategory, IsFunction, [112](#)
- AddEmbeddingOfEqualizer  
for IsCapCategory, IsFunction, [112](#)
- AddEmbeddingOfEqualizerWithGiven-  
Equalizer  
for IsCapCategory, IsFunction, [112](#)
- AddEpimorphismFromProjectiveCover-  
Object  
for IsCapCategory, IsFunction, [112](#)
- AddEpimorphismFromProjectiveCover-  
ObjectWithGivenProjectiveCover-  
Object  
for IsCapCategory, IsFunction, [112](#)
- AddEpimorphismFromSomeProjectiveObject  
for IsCapCategory, IsFunction, [113](#)
- AddEpimorphismFromSomeProjective-  
ObjectWithGivenSomeProjective-  
Object  
for IsCapCategory, IsFunction, [113](#)
- AddEqualizer  
for IsCapCategory, IsFunction, [113](#)
- AddEqualizerFunctorial  
for IsCapCategory, IsFunction, [113](#)
- AddEqualizerFunctorialWithGiven-  
Equalizers  
for IsCapCategory, IsFunction, [113](#)
- AddFiberProduct  
for IsCapCategory, IsFunction, [114](#)
- AddFiberProductFunctorial  
for IsCapCategory, IsFunction, [114](#)
- AddFiberProductFunctorialWithGiven-  
FiberProducts  
for IsCapCategory, IsFunction, [114](#)
- AddHomologyObject  
for IsCapCategory, IsFunction, [114](#)
- AddHomologyObjectFunctorialWithGiven-  
HomologyObjects  
for IsCapCategory, IsFunction, [114](#)
- AddHomomorphismStructureOnMorphisms  
for IsCapCategory, IsFunction, [114](#)
- AddHomomorphismStructureOnMorphisms-  
WithGivenObjects  
for IsCapCategory, IsFunction, [115](#)
- AddHomomorphismStructureOnObjects  
for IsCapCategory, IsFunction, [115](#)
- AddHorizontalPostCompose  
for IsCapCategory, IsFunction, [115](#)
- AddHorizontalPreCompose  
for IsCapCategory, IsFunction, [115](#)
- AddIdentityMorphism  
for IsCapCategory, IsFunction, [115](#)
- AddIdentityTwoCell  
for IsCapCategory, IsFunction, [116](#)
- AddImageEmbedding  
for IsCapCategory, IsFunction, [116](#)
- AddImageEmbeddingWithGivenImageObject  
for IsCapCategory, IsFunction, [116](#)
- AddImageObject

- for IsCapCategory, IsFunction, 116
- AddImageObjectFunctorial
  - for IsCapCategory, IsFunction, 116
- AddImageObjectFunctorialWithGiven-ImageObjects
  - for IsCapCategory, IsFunction, 116
- AddIndecomposableInjectiveObjects
  - for IsCapCategory, IsFunction, 117
- AddIndecomposableProjectiveObjects
  - for IsCapCategory, IsFunction, 117
- AddInitialObject
  - for IsCapCategory, IsFunction, 117
- AddInitialObjectFunctorial
  - for IsCapCategory, IsFunction, 117
- AddInitialObjectFunctorialWithGiven-InitialObjects
  - for IsCapCategory, IsFunction, 117
- AddInjectionOfCofactorOfCoproduct
  - for IsCapCategory, IsFunction, 117
- AddInjectionOfCofactorOfCoproductWith-GivenCoproduct
  - for IsCapCategory, IsFunction, 118
- AddInjectionOfCofactorOfDirectSum
  - for IsCapCategory, IsFunction, 118
- AddInjectionOfCofactorOfDirectSumWith-GivenDirectSum
  - for IsCapCategory, IsFunction, 118
- AddInjectionOfCofactorOfPushout
  - for IsCapCategory, IsFunction, 118
- AddInjectionOfCofactorOfPushoutWith-GivenPushout
  - for IsCapCategory, IsFunction, 118
- AddInjectiveColift
  - for IsCapCategory, IsFunction, 118
- AddInjectiveDimension
  - for IsCapCategory, IsFunction, 119
- AddInjectiveEnvelopeObject
  - for IsCapCategory, IsFunction, 119
- AddInterpretMorphismAsMorphism-FromDistinguishedObjectTo-HomomorphismStructure
  - for IsCapCategory, IsFunction, 119
- AddInterpretMorphismAsMorphism-FromDistinguishedObjectTo-HomomorphismStructureWithGiven-Objects
  - for IsCapCategory, IsFunction, 119
- AddInterpretMorphismFromDistinguished-ObjectToHomomorphismStructure-AsMorphism
  - for IsCapCategory, IsFunction, 119
- AddInverseForMorphisms
  - for IsCapCategory, IsFunction, 120
- AddInverseOfMorphismFromCoimageToImage
  - for IsCapCategory, IsFunction, 120
- AddInverseOfMorphismFromCoimageTo-ImageWithGivenObjects
  - for IsCapCategory, IsFunction, 120
- AddIsAutomorphism
  - for IsCapCategory, IsFunction, 120
- AddIsBijectiveObject
  - for IsCapCategory, IsFunction, 120
- AddIsCodominating
  - for IsCapCategory, IsFunction, 121
- AddIsColiftable
  - for IsCapCategory, IsFunction, 121
- AddIsColiftableAlongEpimorphism
  - for IsCapCategory, IsFunction, 121
- AddIsCongruentForMorphisms
  - for IsCapCategory, IsFunction, 121
- AddIsDominating
  - for IsCapCategory, IsFunction, 121
- AddIsEndomorphism
  - for IsCapCategory, IsFunction, 121
- AddIsEpimorphism
  - for IsCapCategory, IsFunction, 122
- AddIsEqualAsFactorobjects
  - for IsCapCategory, IsFunction, 122
- AddIsEqualAsSubobjects
  - for IsCapCategory, IsFunction, 122
- AddIsEqualForCacheForMorphisms
  - for IsCapCategory, IsFunction, 122
- AddIsEqualForCacheForObjects
  - for IsCapCategory, IsFunction, 122
- AddIsEqualForMorphisms
  - for IsCapCategory, IsFunction, 122
- AddIsEqualForMorphismsOnMor
  - for IsCapCategory, IsFunction, 123
- AddIsEqualForObjects
  - for IsCapCategory, IsFunction, 123
- AddIsEqualToIdentityMorphism
  - for IsCapCategory, IsFunction, 123



- AddIsEqualToZeroMorphism
  - for IsCapCategory, IsFunction, [123](#)
- AddIsHomSetInhabited
  - for IsCapCategory, IsFunction, [123](#)
- AddIsIdempotent
  - for IsCapCategory, IsFunction, [123](#)
- AddIsInitial
  - for IsCapCategory, IsFunction, [124](#)
- AddIsInjective
  - for IsCapCategory, IsFunction, [124](#)
- AddIsIsomorphicForObjects
  - for IsCapCategory, IsFunction, [124](#)
- AddIsIsomorphism
  - for IsCapCategory, IsFunction, [124](#)
- AddIsLiftable
  - for IsCapCategory, IsFunction, [124](#)
- AddIsLiftableAlongMonomorphism
  - for IsCapCategory, IsFunction, [124](#)
- AddIsMonomorphism
  - for IsCapCategory, IsFunction, [124](#)
- AddIsomorphismFromCoequalizerOf-
  - CoproductDiagramToPushout
  - for IsCapCategory, IsFunction, [126](#)
- AddIsomorphismFromCoequalizerTo-
  - CokernelOfJointPairwise-
    - DifferencesOfMorphismsFrom-
      - Coproduct
  - for IsCapCategory, IsFunction, [127](#)
- AddIsomorphismFromCoimageToCokernelOf-
  - Kernel
  - for IsCapCategory, IsFunction, [127](#)
- AddIsomorphismFromCokernelOfJoint-
  - PairwiseDifferencesOfMorphisms-
    - FromCoproductToCoequalizer
  - for IsCapCategory, IsFunction, [127](#)
- AddIsomorphismFromCokernelOfKernelTo-
  - Coimage
  - for IsCapCategory, IsFunction, [127](#)
- AddIsomorphismFromCoproductToDirectSum
  - for IsCapCategory, IsFunction, [127](#)
- AddIsomorphismFromDirectProductTo-
  - DirectSum
  - for IsCapCategory, IsFunction, [128](#)
- AddIsomorphismFromDirectSumToCoproduct
  - for IsCapCategory, IsFunction, [128](#)
- AddIsomorphismFromDirectSumToDirect-
  - Product
  - for IsCapCategory, IsFunction, [128](#)
- AddIsomorphismFromEqualizerOfDirect-
  - ProductDiagramToFiberProduct
  - for IsCapCategory, IsFunction, [128](#)
- AddIsomorphismFromEqualizerToKernel-
  - OfJointPairwiseDifferencesOf-
    - MorphismsIntoDirectProduct
  - for IsCapCategory, IsFunction, [128](#)
- AddIsomorphismFromFiberProductTo-
  - EqualizerOfDirectProductDiagram
  - for IsCapCategory, IsFunction, [129](#)
- AddIsomorphismFromHomologyObjectToIts-
  - ConstructionAsAnImageObject
  - for IsCapCategory, IsFunction, [129](#)
- AddIsomorphismFromImageObjectToKernel-
  - OfCokernel
  - for IsCapCategory, IsFunction, [129](#)
- AddIsomorphismFromInitialObjectToZero-
  - Object
  - for IsCapCategory, IsFunction, [129](#)
- AddIsomorphismFromItsConstructionAsAn-
  - ImageObjectToHomologyObject
  - for IsCapCategory, IsFunction, [130](#)
- AddIsomorphismFromKernelOfCokernelTo-
  - ImageObject
  - for IsCapCategory, IsFunction, [130](#)
- AddIsomorphismFromKernelOfJoint-
  - PairwiseDifferencesOfMorphisms-
    - IntoDirectProductToEqualizer
  - for IsCapCategory, IsFunction, [130](#)
- AddIsomorphismFromPushoutTo-
  - CoequalizerOfCoproductDiagram
  - for IsCapCategory, IsFunction, [130](#)
- AddIsomorphismFromTerminalObjectTo-
  - ZeroObject
  - for IsCapCategory, IsFunction, [130](#)
- AddIsomorphismFromZeroObjectToInitial-
  - Object
  - for IsCapCategory, IsFunction, [131](#)
- AddIsomorphismFromZeroObjectTo-
  - TerminalObject
  - for IsCapCategory, IsFunction, [131](#)
- AddIsOne
  - for IsCapCategory, IsFunction, [125](#)
- AddIsProjective

- for IsCapCategory, IsFunction, [125](#)
- AddIsSplitEpimorphism
  - for IsCapCategory, IsFunction, [125](#)
- AddIsSplitMonomorphism
  - for IsCapCategory, IsFunction, [125](#)
- AddIsTerminal
  - for IsCapCategory, IsFunction, [125](#)
- AddIsWellDefinedForMorphisms
  - for IsCapCategory, IsFunction, [125](#)
- AddIsWellDefinedForMorphismsWithGiven-SourceAndRange
  - for IsCapCategory, IsFunction, [126](#)
- AddIsWellDefinedForObjects
  - for IsCapCategory, IsFunction, [126](#)
- AddIsWellDefinedForTwoCells
  - for IsCapCategory, IsFunction, [126](#)
- AddIsZeroForMorphisms
  - for IsCapCategory, IsFunction, [126](#)
- AddIsZeroForObjects
  - for IsCapCategory, IsFunction, [126](#)
- AdditionForMorphisms
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, [34](#)
- AdditiveGenerators
  - for IsCapCategory, [12](#)
- AdditiveInverseForMorphisms
  - for IsCapCategoryMorphism, [34](#)
- AddJointPairwiseDifferencesOf-MorphismsFromCoproduct
  - for IsCapCategory, IsFunction, [131](#)
- AddJointPairwiseDifferencesOf-MorphismsIntoDirectProduct
  - for IsCapCategory, IsFunction, [131](#)
- AddKernelEmbedding
  - for IsCapCategory, IsFunction, [131](#)
- AddKernelEmbeddingWithGivenKernel-Object
  - for IsCapCategory, IsFunction, [132](#)
- AddKernelLift
  - for IsCapCategory, IsFunction, [132](#)
- AddKernelLiftWithGivenKernelObject
  - for IsCapCategory, IsFunction, [132](#)
- AddKernelObject
  - for IsCapCategory, IsFunction, [132](#)
- AddKernelObjectFunctorial
  - for IsCapCategory, IsFunction, [132](#)
- AddKernelObjectFunctorialWithGiven-KernelObjects
  - for IsCapCategory, IsFunction, [132](#)
- AddLift
  - for IsCapCategory, IsFunction, [133](#)
- AddLiftAlongMonomorphism
  - for IsCapCategory, IsFunction, [133](#)
- AddLiftOrFail
  - for IsCapCategory, IsFunction, [133](#)
- AddLinearCombinationOfMorphisms
  - for IsCapCategory, IsFunction, [133](#)
- AddMereExistenceOfSolutionOfLinear-SystemInAbCategory
  - for IsCapCategory, IsFunction, [133](#)
- AddMonomorphismIntoInjectiveEnvelope-Object
  - for IsCapCategory, IsFunction, [133](#)
- AddMonomorphismIntoInjectiveEnvelope-ObjectWithGivenInjective-EnvelopeObject
  - for IsCapCategory, IsFunction, [134](#)
- AddMonomorphismIntoSomeInjectiveObject
  - for IsCapCategory, IsFunction, [134](#)
- AddMonomorphismIntoSomeInjective-ObjectWithGivenSomeInjective-Object
  - for IsCapCategory, IsFunction, [134](#)
- AddMorphism
  - for IsCapCategory, IsAttributeStoringRep, [32](#)
- AddMorphismBetweenDirectSums
  - for IsCapCategory, IsFunction, [134](#)
- AddMorphismBetweenDirectSumsWithGiven-DirectSums
  - for IsCapCategory, IsFunction, [135](#)
- AddMorphismConstructor
  - for IsCapCategory, IsFunction, [135](#)
- AddMorphismDatum
  - for IsCapCategory, IsFunction, [135](#)
- AddMorphismFromCoimageToImage
  - for IsCapCategory, IsFunction, [135](#)
- AddMorphismFromCoimageToImageWith-GivenObjects
  - for IsCapCategory, IsFunction, [135](#)
- AddMorphismFromEqualizerToSink
  - for IsCapCategory, IsFunction, [135](#)

- AddMorphismFromEqualizerToSinkWith-  
    GivenEqualizer  
    for IsCapCategory, IsFunction, [136](#)
- AddMorphismFromFiberProductToSink  
    for IsCapCategory, IsFunction, [136](#)
- AddMorphismFromFiberProductToSinkWith-  
    GivenFiberProduct  
    for IsCapCategory, IsFunction, [136](#)
- AddMorphismFromKernelObjectToSink  
    for IsCapCategory, IsFunction, [136](#)
- AddMorphismFromKernelObjectToSinkWith-  
    GivenKernelObject  
    for IsCapCategory, IsFunction, [136](#)
- AddMorphismFromSourceToCoequalizer  
    for IsCapCategory, IsFunction, [137](#)
- AddMorphismFromSourceToCoequalizer-  
    WithGivenCoequalizer  
    for IsCapCategory, IsFunction, [137](#)
- AddMorphismFromSourceToCokernelObject  
    for IsCapCategory, IsFunction, [137](#)
- AddMorphismFromSourceToCokernelObject-  
    WithGivenCokernelObject  
    for IsCapCategory, IsFunction, [137](#)
- AddMorphismFromSourceToPushout  
    for IsCapCategory, IsFunction, [137](#)
- AddMorphismFromSourceToPushoutWith-  
    GivenPushout  
    for IsCapCategory, IsFunction, [138](#)
- AddMorphismFunction  
    for IsCapFunctor, IsFunction, [55](#)
- AddMorphismRepresentation  
    for IsCapCategory, IsObject, [32](#)
- AddMorphismsOfExternalHom  
    for IsCapCategory, IsFunction, [138](#)
- AddMultiplyWithElementOfCommutative-  
    RingForMorphisms  
    for IsCapCategory, IsFunction, [138](#)
- AddNaturalTransformationFunction  
    for IsCapNaturalTransformation, IsFunction,  
    [58](#)
- AddObject  
    for IsCapCategory, IsAttributeStoringRep,  
    [21](#)
- AddObjectConstructor  
    for IsCapCategory, IsFunction, [138](#)
- AddObjectDatum  
    for IsCapCategory, IsFunction, [138](#)
- AddObjectFunction  
    for IsCapFunctor, IsFunction, [55](#)
- AddObjectRepresentation  
    for IsCapCategory, IsObject, [21](#)
- AddOperationsToDerivationGraph  
    for IsDerivedMethodGraph, IsDenseList,  
    [159](#)
- AddPostCompose  
    for IsCapCategory, IsFunction, [138](#)
- AddPostComposeList  
    for IsCapCategory, IsFunction, [139](#)
- AddPostInverseForMorphisms  
    for IsCapCategory, IsFunction, [139](#)
- AddPreCompose  
    for IsCapCategory, IsFunction, [139](#)
- AddPreComposeList  
    for IsCapCategory, IsFunction, [139](#)
- AddPreInverseForMorphisms  
    for IsCapCategory, IsFunction, [139](#)
- AddPrimitiveOperation  
    for IsOperationWeightList, IsString, IsInt,  
    [161](#)
- AddProjectionInFactorOfDirectProduct  
    for IsCapCategory, IsFunction, [139](#)
- AddProjectionInFactorOfDirectProduct-  
    WithGivenDirectProduct  
    for IsCapCategory, IsFunction, [140](#)
- AddProjectionInFactorOfDirectSum  
    for IsCapCategory, IsFunction, [140](#)
- AddProjectionInFactorOfDirectSumWith-  
    GivenDirectSum  
    for IsCapCategory, IsFunction, [140](#)
- AddProjectionInFactorOfFiberProduct  
    for IsCapCategory, IsFunction, [140](#)
- AddProjectionInFactorOfFiberProduct-  
    WithGivenFiberProduct  
    for IsCapCategory, IsFunction, [140](#)
- AddProjectionOntoCoequalizer  
    for IsCapCategory, IsFunction, [141](#)
- AddProjectionOntoCoequalizerWithGiven-  
    Coequalizer  
    for IsCapCategory, IsFunction, [141](#)
- AddProjectiveCoverObject  
    for IsCapCategory, IsFunction, [141](#)
- AddProjectiveDimension

- for IsCapCategory, IsFunction, [141](#)
- AddProjectiveLift
  - for IsCapCategory, IsFunction, [141](#)
- AddPushout
  - for IsCapCategory, IsFunction, [141](#)
- AddPushoutFunctorial
  - for IsCapCategory, IsFunction, [142](#)
- AddPushoutFunctorialWithGivenPushouts
  - for IsCapCategory, IsFunction, [142](#)
- AddRandomMorphismByInteger
  - for IsCapCategory, IsFunction, [142](#)
- AddRandomMorphismByList
  - for IsCapCategory, IsFunction, [142](#)
- AddRandomMorphismWithFixedRangeByInteger
  - for IsCapCategory, IsFunction, [142](#)
- AddRandomMorphismWithFixedRangeByList
  - for IsCapCategory, IsFunction, [142](#)
- AddRandomMorphismWithFixedSourceAndRangeByInteger
  - for IsCapCategory, IsFunction, [143](#)
- AddRandomMorphismWithFixedSourceAndRangeByList
  - for IsCapCategory, IsFunction, [143](#)
- AddRandomMorphismWithFixedSourceByInteger
  - for IsCapCategory, IsFunction, [143](#)
- AddRandomMorphismWithFixedSourceByList
  - for IsCapCategory, IsFunction, [143](#)
- AddRandomObjectByInteger
  - for IsCapCategory, IsFunction, [143](#)
- AddRandomObjectByList
  - for IsCapCategory, IsFunction, [144](#)
- AddSimplifyEndo
  - for IsCapCategory, IsFunction, [144](#)
- AddSimplifyEndo\_IsoFromInputObject
  - for IsCapCategory, IsFunction, [144](#)
- AddSimplifyEndo\_IsoToInputObject
  - for IsCapCategory, IsFunction, [144](#)
- AddSimplifyMorphism
  - for IsCapCategory, IsFunction, [144](#)
- AddSimplifyObject
  - for IsCapCategory, IsFunction, [144](#)
- AddSimplifyObject\_IsoFromInputObject
  - for IsCapCategory, IsFunction, [144](#)
- AddSimplifyObject\_IsoToInputObject
  - for IsCapCategory, IsFunction, [145](#)
- AddSimplifyRange
  - for IsCapCategory, IsFunction, [145](#)
- AddSimplifyRange\_IsoFromInputObject
  - for IsCapCategory, IsFunction, [145](#)
- AddSimplifyRange\_IsoToInputObject
  - for IsCapCategory, IsFunction, [145](#)
- AddSimplifySource
  - for IsCapCategory, IsFunction, [145](#)
- AddSimplifySourceAndRange
  - for IsCapCategory, IsFunction, [145](#)
- AddSimplifySourceAndRange\_IsoFromInputRange
  - for IsCapCategory, IsFunction, [146](#)
- AddSimplifySourceAndRange\_IsoFromInputSource
  - for IsCapCategory, IsFunction, [146](#)
- AddSimplifySourceAndRange\_IsoToInputRange
  - for IsCapCategory, IsFunction, [146](#)
- AddSimplifySourceAndRange\_IsoToInputSource
  - for IsCapCategory, IsFunction, [146](#)
- AddSimplifySource\_IsoFromInputObject
  - for IsCapCategory, IsFunction, [146](#)
- AddSimplifySource\_IsoToInputObject
  - for IsCapCategory, IsFunction, [147](#)
- AddSolveLinearSystemInAbCategory
  - for IsCapCategory, IsFunction, [147](#)
- AddSolveLinearSystemInAbCategoryOrFail
  - for IsCapCategory, IsFunction, [147](#)
- AddSomeInjectiveObject
  - for IsCapCategory, IsFunction, [147](#)
- AddSomeIsomorphismBetweenObjects
  - for IsCapCategory, IsFunction, [147](#)
- AddSomeProjectiveObject
  - for IsCapCategory, IsFunction, [147](#)
- AddSomeReductionBySplitEpiSummand
  - for IsCapCategory, IsFunction, [148](#)
- AddSomeReductionBySplitEpiSummand\_MorphismFromInputRange
  - for IsCapCategory, IsFunction, [148](#)
- AddSomeReductionBySplitEpiSummand\_MorphismToInputRange
  - for IsCapCategory, IsFunction, [148](#)
- AddSubtractionForMorphisms

for IsCapCategory, IsFunction, 148  
 AddSumOfMorphisms  
   for IsCapCategory, IsFunction, 148  
 AddTerminalObject  
   for IsCapCategory, IsFunction, 148  
 AddTerminalObjectFunctorial  
   for IsCapCategory, IsFunction, 149  
 AddTerminalObjectFunctorialWithGiven-  
   TerminalObjects  
   for IsCapCategory, IsFunction, 149  
 AddUniversalMorphismFromCoequalizer  
   for IsCapCategory, IsFunction, 149  
 AddUniversalMorphismFromCoequalizer-  
   WithGivenCoequalizer  
   for IsCapCategory, IsFunction, 149  
 AddUniversalMorphismFromCoproduct  
   for IsCapCategory, IsFunction, 149  
 AddUniversalMorphismFromCoproductWith-  
   GivenCoproduct  
   for IsCapCategory, IsFunction, 149  
 AddUniversalMorphismFromDirectSum  
   for IsCapCategory, IsFunction, 150  
 AddUniversalMorphismFromDirectSumWith-  
   GivenDirectSum  
   for IsCapCategory, IsFunction, 150  
 AddUniversalMorphismFromImage  
   for IsCapCategory, IsFunction, 150  
 AddUniversalMorphismFromImageWith-  
   GivenImageObject  
   for IsCapCategory, IsFunction, 150  
 AddUniversalMorphismFromInitialObject  
   for IsCapCategory, IsFunction, 150  
 AddUniversalMorphismFromInitialObject-  
   WithGivenInitialObject  
   for IsCapCategory, IsFunction, 151  
 AddUniversalMorphismFromPushout  
   for IsCapCategory, IsFunction, 151  
 AddUniversalMorphismFromPushoutWith-  
   GivenPushout  
   for IsCapCategory, IsFunction, 151  
 AddUniversalMorphismFromZeroObject  
   for IsCapCategory, IsFunction, 151  
 AddUniversalMorphismFromZeroObject-  
   WithGivenZeroObject  
   for IsCapCategory, IsFunction, 151  
 AddUniversalMorphismIntoCoimage

  for IsCapCategory, IsFunction, 152  
 AddUniversalMorphismIntoCoimageWith-  
   GivenCoimageObject  
   for IsCapCategory, IsFunction, 152  
 AddUniversalMorphismIntoDirectProduct  
   for IsCapCategory, IsFunction, 152  
 AddUniversalMorphismIntoDirectProduct-  
   WithGivenDirectProduct  
   for IsCapCategory, IsFunction, 152  
 AddUniversalMorphismIntoDirectSum  
   for IsCapCategory, IsFunction, 152  
 AddUniversalMorphismIntoDirectSumWith-  
   GivenDirectSum  
   for IsCapCategory, IsFunction, 152  
 AddUniversalMorphismIntoEqualizer  
   for IsCapCategory, IsFunction, 153  
 AddUniversalMorphismIntoEqualizerWith-  
   GivenEqualizer  
   for IsCapCategory, IsFunction, 153  
 AddUniversalMorphismIntoFiberProduct  
   for IsCapCategory, IsFunction, 153  
 AddUniversalMorphismIntoFiberProduct-  
   WithGivenFiberProduct  
   for IsCapCategory, IsFunction, 153  
 AddUniversalMorphismIntoTerminalObject  
   for IsCapCategory, IsFunction, 153  
 AddUniversalMorphismIntoTerminal-  
   ObjectWithGivenTerminalObject  
   for IsCapCategory, IsFunction, 154  
 AddUniversalMorphismIntoZeroObject  
   for IsCapCategory, IsFunction, 154  
 AddUniversalMorphismIntoZeroObject-  
   WithGivenZeroObject  
   for IsCapCategory, IsFunction, 154  
 AddVerticalPostCompose  
   for IsCapCategory, IsFunction, 154  
 AddVerticalPreCompose  
   for IsCapCategory, IsFunction, 154  
 AddZeroMorphism  
   for IsCapCategory, IsFunction, 154  
 AddZeroObject  
   for IsCapCategory, IsFunction, 155  
 AddZeroObjectFunctorial  
   for IsCapCategory, IsFunction, 155  
 AddZeroObjectFunctorialWithGivenZero-  
   Objects

- for IsCapCategory, IsFunction, 155
- ApplyFunctor, 56
- ApplyNaturalTransformation, 58
- AsCapCategory
  - for IsCapCategoryAsCatObject, 54
- AsCapCategoryMorphism, 33
- AsCapCategoryObject, 22
- AsCatObject
  - for IsCapCategory, 54
- AsHomalgMatrix
  - for IsCapCategoryMorphism, 33
  - for IsCapCategoryObject, 22
- AsInteger
  - for IsCapCategoryMorphism, 33
  - for IsCapCategoryObject, 22
- AsMorphismInWrapperCategory
  - for IsWrapperCapCategory, IsCapCategoryMorphism, 187
  - for IsWrapperCapCategoryObject, IsCapCategoryMorphism, IsWrapperCapCategoryObject, 187
- AsObjectInWrapperCategory
  - for IsWrapperCapCategory, IsCapCategoryObject, 187
- AsPrimitiveValue
  - for IsCapCategoryMorphism, 33
  - for IsCapCategoryObject, 22
- AstrictionToCoimage
  - for IsCapCategoryMorphism, 95
- AstrictionToCoimageWithGivenCoimageObject
  - for IsCapCategoryMorphism, IsCapCategoryObject, 95
- BasisOfExternalHom
  - for IsCapCategoryObject, IsCapCategoryObject, 45
- BrowseCachingStatistic, 167
- BrowseTimingStatistics, 16
- CachingStatistic, 167
- CanCompute
  - for IsCapCategory, IsFunction, 13
  - for IsCapCategory, IsString, 13
- CAPAddPrepareFunction, 104
- CapCat, 53
- CapCategory
  - for IsCapCategoryMorphism, 26
  - for IsCapCategoryObject, 18
- CapCategorySwitchLogicOff, 13
- CapCategorySwitchLogicOn, 13
- CapCategorySwitchLogicPropagationForMorphismsOff, 13
- CapCategorySwitchLogicPropagationForMorphismsOn, 13
- CapCategorySwitchLogicPropagationForObjectsOff, 13
- CapCategorySwitchLogicPropagationForObjectsOn, 12
- CapCategorySwitchLogicPropagationOff, 13
- CapCategorySwitchLogicPropagationOn, 13
- CapFixpoint, 170
- CapFunctor
  - for IsString, IsCapCategory, IsCapCategory, 54
  - for IsString, IsCapCategory, IsCapCategoryAsCatObject, 54
  - for IsString, IsCapCategoryAsCatObject, IsCapCategory, 54
  - for IsString, IsCapCategoryAsCatObject, IsCapCategoryAsCatObject, 54
  - for IsString, IsList, IsCapCategory, 54
  - for IsString, IsList, IsCapCategoryAsCatObject, 54
- CapInternalInstallAdd, 103
- CapJitAddKnownMethod, 168
- CapJitAddTypeSignature, 168
- CapJitAddTypeSignatureDeferred, 169
- CapJitDataTypeOfCategory, 169
- CapJitDataTypeOfListOf, 169
- CapJitDataTypeOfMorphismOfCategory, 169
- CapJitDataTypeOfNTupleOf, 169
- CapJitDataTypeOfObjectOfCategory, 169
- CapJitDataTypeOfTwoCellOfCategory, 169
- CapJitTypedExpression, 169
- CAPOperationPrepareFunction, 104
- CAP\_INTERNAL\_ASSERT\_IS\_CELL\_OF\_CATEGORY, 166
- CAP\_INTERNAL\_ASSERT\_IS\_MORPHISM\_OF\_CATEGORY, 167
- CAP\_INTERNAL\_ASSERT\_IS\_OBJECT\_OF\_CATEGORY, 166

- CAP\_INTERNAL\_ASSERT\_IS\_TWO\_CELL\_OF\_CATEGORY, [167](#)
- CAP\_INTERNAL\_ASSERT\_VALUE\_IS\_OF\_TYPE\_GETTER, [166](#)
- CAP\_INTERNAL\_CONSTRUCTOR\_FOR\_TERMINAL\_CATEGORY, [211](#)
- CAP\_INTERNAL\_FIND\_APPEARANCE\_OF\_SYMBOL\_IN\_FUNCTION, [166](#)
- CAP\_INTERNAL\_GENERATE\_CONVENIENCE\_METHODS\_FOR\_LIMITS, [177](#)
- CAP\_INTERNAL\_GET\_DATA\_TYPES\_FROM\_STRINGS, [165](#)
- CAP\_INTERNAL\_GET\_DATA\_TYPE\_FROM\_STRING, [165](#)
- CAP\_INTERNAL\_MERGE\_FILTER\_LISTS, [165](#)
- CAP\_INTERNAL\_MERGE\_PRECONDITIONS\_LIST, [166](#)
- CAP\_INTERNAL\_REPLACED\_STRINGS\_WITH\_FILTERS, [165](#)
- CAP\_INTERNAL\_REPLACED\_STRING\_WITH\_FILTER, [165](#)
- CAP\_INTERNAL\_RETURN\_OPTION\_OR\_DEFAULT, [165](#)
- CAP\_INTERNAL\_VALIDATE\_LIMITS\_IN\_NAME\_RECORD, [177](#)
- CAP\_JIT\_EXPR\_CASE\_WRAPPER, [172](#)
- CAP\_JIT\_INCOMPLETE\_LOGIC, [172](#)
- CategoryConstructor
  - for IsRecord, [178](#)
- CategoryFilter
  - for IsCapCategory, [11](#)
  - for IsDerivedMethod, [157](#)
- CategoryOfOperationWeightList
  - for IsOperationWeightList, [160](#)
- CostrictionToImage
  - for IsCapCategoryMorphism, [92](#)
- CostrictionToImageWithGivenImageObject
  - for IsCapCategoryMorphism, IsCapCategoryObject, [92](#)
- CoefficientsOfMorphism
  - for IsCapCategoryMorphism, [46](#)
- Coequalizer, [82](#)
- CoequalizerFunctorial
  - for IsList, IsCapCategoryMorphism, IsList, [83](#)
- CoequalizerFunctorialWithGivenCoequalizers
  - for IsCapCategoryObject, IsList, IsCapCategoryMorphism, IsList, IsCapCategoryObject, [83](#)
- CoequalizerOp
  - for IsCapCategoryObject, IsList, [82](#)
- CoimageObject
  - for IsCapCategoryMorphism, [94](#)
- CoimageObjectFunctorial
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, [95](#)
- CoimageObjectFunctorialWithGivenCoimageObjects
  - for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject, [96](#)
- CoimageProjection
  - for IsCapCategoryMorphism, [94](#)
- CoimageProjectionWithGivenCoimageObject
  - for IsCapCategoryMorphism, IsCapCategoryObject, [94](#)
- CokernelColift
  - for IsCapCategoryMorphism, IsCapCategoryObject, IsCapCategoryMorphism, [63](#)
- CokernelColiftWithGivenCokernelObject
  - for IsCapCategoryMorphism, IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryObject, [63](#)
- CokernelObject
  - for IsCapCategoryMorphism, [62](#)
- CokernelObjectFunctorial
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, [63](#)
  - for IsList, [63](#)
- CokernelObjectFunctorialWithGivenCokernelObjects
  - for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject, [64](#)



- for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject, 64
- CokernelProjection
  - for IsCapCategoryMorphism, 62
- CokernelProjectionWithGivenCokernelObject
  - for IsCapCategoryMorphism, IsCapCategoryObject, 62
- Colift
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, 39
- ColiftAlongEpimorphism
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, 38
- ColiftOrFail
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, 40
- CommutativeRingOfLinearCategory
  - for IsCapCategory, 12
- ComponentOfMorphismFromCoproduct
  - for IsCapCategoryMorphism, IsList, IsInt, 75
- ComponentOfMorphismFromDirectSum
  - for IsCapCategoryMorphism, IsList, IsInt, 73
- ComponentOfMorphismIntoDirectProduct
  - for IsCapCategoryMorphism, IsList, IsInt, 78
- ComponentOfMorphismIntoDirectSum
  - for IsCapCategoryMorphism, IsList, IsInt, 73
- Contains
  - for IsStringMinHeap, IsString, 163
- Coproduct
  - for IsCapCategoryObject, IsCapCategoryObject, 74
  - for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, 74
  - for IsList, 74
- CoproductFunctorial
  - for IsList, IsList, IsList, 75
- CoproductFunctorialWithGivenCoproducts
  - for IsCapCategoryObject, IsList, IsList, IsList, IsCapCategoryObject, 75
- CreateCapCategory, 10
  - for IsString, 10
  - for IsString, IsFunction, IsFunction, IsFunction, IsFunction, 10
- CreateCapCategoryMorphismWithAttributes, 32
- CreateCapCategoryObjectWithAttributes, 21
- CreateCapCategoryWithDataTypes, 10
- CurrentOperationWeight
  - for IsOperationWeightList, IsString, 160
- DeactivateCachingOfCategory, 15
- DeactivateDefaultCaching, 15
- DeactivateDerivationInfo, 156
- DecreaseKey
  - for IsStringMinHeap, IsString, IsInt, 162
- DerivationFunction
  - for IsDerivedMethod, 157
- DerivationGraph
  - for IsOperationWeightList, 160
- DerivationInfo, 156
- DerivationName
  - for IsDerivedMethod, 157
- DerivationOfOperation
  - for IsOperationWeightList, IsString, 161
- DerivationsOfOperation
  - for IsDerivedMethodGraph, IsString, 159
- DerivationsUsingOperation
  - for IsDerivedMethodGraph, IsString, 159
- DerivationWeight
  - for IsDerivedMethod, 157
- DirectProduct, 76
- DirectProductFunctorial
  - for IsList, IsList, IsList, 77
- DirectProductFunctorialWithGivenDirectProducts
  - for IsCapCategoryObject, IsList, IsList, IsList, IsCapCategoryObject, 77
- DirectProductOp
  - for IsList, 76
- DirectSum, 69
- DirectSumFunctorial
  - for IsList, IsList, IsList, 73
- DirectSumFunctorialWithGivenDirectSums
  - for IsCapCategoryObject, IsList, IsList, IsList, IsCapCategoryObject, 73
- DirectSumOp
  - for IsList, 70
- DisableAddForCategoricalOperations, 16
- DisableInputSanityChecks, 15



- DisableOutputSanityChecks, [15](#)
- DisableSanityChecks, [15](#)
- DisableTimingStatistics, [16](#)
- DisplayTimingStatistics, [16](#)
- DistinguishedObjectOfHomomorphism-  
Structure
  - for IsCapCategory, [42](#)
- DistinguishedObjectOfHomomorphism-  
StructureExtendedByFull-  
Embedding
  - for IsCapCategory, IsCapCategory, [44](#)
- Down
  - for IsObject, [14](#)
- DownOnlyMorphismData
  - for IsCapCategoryMorphism, [14](#)
- DownToBottom
  - for IsObject, [14](#)
- DummyCategory
  - for IsRecord, [191](#)
- DummyCommutativeRing, [190](#)
- DummyField, [190](#)
- DummyRing, [190](#)
- EmbeddingOfEqualizer
  - for IsCapCategoryObject, IsList, [79](#)
- EmbeddingOfEqualizerWithGivenEqualizer
  - for IsCapCategoryObject, IsList, IsCapCategoryObject, [79](#)
- EnableAddForCategoricalOperations, [16](#)
- EnableFullInputSanityChecks, [15](#)
- EnableFullOutputSanityChecks, [15](#)
- EnableFullSanityChecks, [15](#)
- EnablePartialInputSanityChecks, [15](#)
- EnablePartialOutputSanityChecks, [15](#)
- EnablePartialSanityChecks, [15](#)
- EnableTimingStatistics, [16](#)
- EpimorphismFromProjectiveCoverObject
  - for IsCapCategoryObject, [97](#)
- EpimorphismFromProjectiveCoverObject-  
WithGivenProjectiveCoverObject
  - for IsCapCategoryObject, IsCapCategoryObject, [97](#)
- EpimorphismFromSomeProjectiveObject
  - for IsCapCategoryObject, [23](#)
- EpimorphismFromSomeProjectiveObject-  
WithGivenSomeProjectiveObject
  - for IsCapCategoryObject, IsCapCategoryObject, [23](#)
- Equalizer, [78](#)
- EqualizerFunctorial
  - for IsList, IsCapCategoryMorphism, IsList, [80](#)
- EqualizerFunctorialWithGivenEqualizers
  - for IsCapCategoryObject, IsList, IsCapCategoryMorphism, IsList, IsCapCategoryObject, [80](#)
- EqualizerOp
  - for IsCapCategoryObject, IsList, [79](#)
- ExtendRangeOfHomomorphismStructureBy-  
FullEmbedding
  - for IsCapCategory, IsCapCategory, IsFunction, IsFunction, IsFunction, [44](#)
- ExtendRangeOfHomomorphismStructureBy-  
IdentityAsFullEmbedding
  - for IsCapCategory, [45](#)
- ExtractMin
  - for IsStringMinHeap, [162](#)
- FiberProduct, [85](#)
- FiberProductEmbeddingInDirectProduct
  - for IsList, [85](#)
- FiberProductEmbeddingInDirectSum
  - for IsList, [85](#)
- FiberProductFunctorial
  - for IsList, IsList, IsList, [87](#)
- FiberProductFunctorialWithGivenFiber-  
Products
  - for IsCapCategoryObject, IsList, IsList, IsList, IsCapCategoryObject, [87](#)
- FiberProductOp
  - for IsList, [86](#)
- FilteredWithKeys, [173](#)
- FirstWithKeys, [174](#)
- ForAllWithKeys, [173](#)
- ForAnyWithKeys, [173](#)
- FunctionCalledBeforeInstallation
  - for IsDerivedMethod, [158](#)
- FunctionWithNamedArguments, [164](#)
- FunctorCanonicalizeZeroMorphisms
  - for IsCapCategory, [57](#)
- FunctorCanonicalizeZeroObjects
  - for IsCapCategory, [57](#)

- FunctorFromTerminalCategory
  - for IsCapCategoryObject, [212](#)
- FunctorMorphismOperation
  - for IsCapFunctor, [56](#)
- FunctorObjectOperation
  - for IsCapFunctor, [55](#)
- HandlePrecompiledTowers, [172](#)
- Heapify
  - for IsStringMinHeap, IsPosInt, [163](#)
- HeapSize
  - for IsStringMinHeap, [163](#)
- HomologyObject
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, [96](#)
- HomologyObjectFunctorial
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, [96](#)
- HomologyObjectFunctorialWithGivenHomologyObjects
  - for IsCapCategoryObject, IsList, IsCapCategoryObject, [96](#)
- HomomorphismStructureOnMorphisms
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, [41](#)
- HomomorphismStructureOnMorphismsExtendedByFullEmbedding
  - for IsCapCategory, IsCapCategory, IsCapCategoryMorphism, IsCapCategoryMorphism, [44](#)
- HomomorphismStructureOnMorphismsWithGivenObjects
  - for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject, [42](#)
- HomomorphismStructureOnMorphismsWithGivenObjectsExtendedByFullEmbedding
  - for IsCapCategory, IsCapCategory, IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject, [44](#)
- HomomorphismStructureOnObjects
  - for IsCapCategoryObject, IsCapCategoryObject, [41](#)
- HomomorphismStructureOnObjectsExtendedByFullEmbedding
  - for IsCapCategory, IsCapCategory, IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, [44](#)
- HomStructure
  - for IsCapCategory, [44](#)
  - for IsCapCategoryMorphism, [44](#)
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, [43](#)
  - for IsCapCategoryMorphism, IsCapCategoryObject, [43](#)
  - for IsCapCategoryObject, IsCapCategoryMorphism, [43](#)
  - for IsCapCategoryObject, IsCapCategoryObject, [44](#)
  - for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryMorphism, [44](#)
- HorizontalPostCompose
  - for IsCapCategoryTwoCell, IsCapCategoryTwoCell, [52](#)
- HorizontalPreCompose
  - for IsCapCategoryTwoCell, IsCapCategoryTwoCell, [51](#)
- HorizontalPreComposeFunctorWithNaturalTransformation
  - for IsCapFunctor, IsCapNaturalTransformation, [58](#)
- HorizontalPreComposeNaturalTransformationWithFunctor
  - for IsCapNaturalTransformation, IsCapFunctor, [58](#)
- IdentityFunctor
  - for IsCapCategory, [57](#)
- IdentityMorphism
  - for IsCapCategoryObject, [36](#)
- IdentityTwoCell
  - for IsCapCategoryMorphism, [51](#)
- ImageEmbedding
  - for IsCapCategoryMorphism, [91](#)
- ImageEmbeddingWithGivenImageObject
  - for IsCapCategoryMorphism, IsCapCategoryObject, [91](#)
- ImageObject
  - for IsCapCategoryMorphism, [91](#)
- ImageObjectFunctorial

- for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, [92](#)
- ImageObjectFunctorialWithGivenImageObjects
  - for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject, [92](#)
- IndecomposableInjectiveObjects
  - for IsCapCategory, [12](#)
- IndecomposableProjectiveObjects
  - for IsCapCategory, [12](#)
- InfoCategoryConstructor, [178](#)
- InitialObject
  - for IsCapCategory, [68](#)
  - for IsCapCategoryCell, [68](#)
- InitialObjectFunctorial
  - for IsCapCategory, [68](#)
- InitialObjectFunctorialWithGivenInitialObjects
  - for IsCapCategoryObject, IsCapCategoryObject, [68](#)
- InjectionOfCofactorOfCoproduct
  - for IsList, IsInt, [74](#)
- InjectionOfCofactorOfCoproductWithGivenCoproduct
  - for IsList, IsInt, IsCapCategoryObject, [74](#)
- InjectionOfCofactorOfDirectSum
  - for IsList, IsInt, [70](#)
- InjectionOfCofactorOfDirectSumWithGivenDirectSum
  - for IsList, IsInt, IsCapCategoryObject, [70](#)
- InjectionOfCofactorOfPushout
  - for IsList, IsInt, [89](#)
- InjectionOfCofactorOfPushoutWithGivenPushout
  - for IsList, IsInt, IsCapCategoryObject, [89](#)
- InjectiveColift
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, [24](#)
- InjectiveDimension
  - for IsCapCategoryObject, [25](#)
- InjectiveEnvelopeObject
  - for IsCapCategoryObject, [98](#)
- InputSignature
  - for IsCapFunctor, [56](#)
- InstallDeprecatedAlias, [167](#)
- InstallDerivationForCategory
  - for IsDerivedMethod, IsPosInt, IsCapCategory, [158](#)
- InstallDerivationsUsingOperation
  - for IsOperationWeightList, IsString, [161](#)
- InstallFunctor
  - for IsCapFunctor, IsString, [56](#)
- InstallMethodForCompilerForCAP, [168](#)
- InstallNaturalTransformation
  - for IsCapNaturalTransformation, IsString, [58](#)
- InstallOtherMethodForCompilerForCAP, [168](#)
- InterpretMorphismAsMorphismFromDistinguishedObjectToHomomorphismStructure
  - for IsCapCategoryMorphism, [42](#)
- InterpretMorphismAsMorphismFromDistinguishedObjectToHomomorphismStructureExtendedByFullEmbedding
  - for IsCapCategory, IsCapCategory, IsCapCategoryMorphism, [44](#)
- InterpretMorphismAsMorphismFromDistinguishedObjectToHomomorphismStructureWithGivenObjects
  - for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryObject, [42](#)
- InterpretMorphismAsMorphismFromDistinguishedObjectToHomomorphismStructureWithGivenObjectsExtendedByFullEmbedding
  - for IsCapCategory, IsCapCategory, IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryObject, [44](#)
- InterpretMorphismFromDistinguishedObjectToHomomorphismStructureAsMorphism
  - for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryMorphism, [42](#)
- InterpretMorphismFromDistinguishedObjectToHomomorphismStructureAsMorphismExtendedByFullEmbedding

- for IsCapCategory, IsCapCategory, IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryMorphism, 44
- InverseForMorphisms
  - for IsCapCategoryMorphism, 40
- InverseOfMorphismFromCoimageToImage
  - for IsCapCategoryMorphism, 94
- InverseOfMorphismFromCoimageToImage-
  - WithGivenObjects
    - for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryObject, 94
- IsAbCategory
  - for IsCapCategory, 8
- IsAbelianCategory
  - for IsCapCategory, 9
- IsAbelianCategoryWithEnoughInjectives
  - for IsCapCategory, 9
- IsAbelianCategoryWithEnoughProjectives
  - for IsCapCategory, 9
- IsAdditiveCategory
  - for IsCapCategory, 9
- IsApplicableToCategory
  - for IsDerivedMethod, IsCapCategory, 158
- IsAutomorphism
  - for IsCapCategoryMorphism, 32
- IsBijectiveObject
  - for IsCapCategoryObject, 19
- IsCapCategory
  - for IsAttributeStoringRep, 7
- IsCapCategoryAsCatObject
  - for IsCapCategoryObject, 53
- IsCapCategoryCell
  - for IsAttributeStoringRep, 7
- IsCapCategoryMorphism
  - for IsCapCategoryCell, 7
- IsCapCategoryObject
  - for IsCapCategoryCell, 7
- IsCapCategoryTwoCell
  - for IsCapCategoryCell, 7
- IsCapFunctor
  - for IsCapCategoryMorphism, 53
- IsCapNaturalTransformation
  - for IsCapCategoryTwoCell, 53
- IsCapTerminalCategoryWithMultiple-
  - Objects
    - for IsCapCategory, 210
- IsCapTerminalCategoryWithSingleObject
  - for IsCapCategory, 210
- IsCategoryWithDecidableColifts
  - for IsCapCategory, 8
- IsCategoryWithDecidableLifts
  - for IsCapCategory, 8
- IsCodominating
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, 36
- IsColiftable
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, 40
- IsColiftableAlongEpimorphism
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, 39
- IsCongruentForMorphisms
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, 33
- IsDerivedMethod
  - for IsAttributeStoringRep, 156
- IsDerivedMethodGraph
  - for IsAttributeStoringRep, 158
- IsDominating
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, 35
- IsDummyCategory
  - for IsCapCategory, 190
- IsDummyCategoryMorphism
  - for IsCapCategoryMorphism, 190
- IsDummyCategoryObject
  - for IsCapCategoryObject, 190
- IsDummyCommutativeRing, 189
- IsDummyCommutativeRingElement, 189
- IsDummyField, 190
- IsDummyFieldElement, 190
- IsDummyRing, 189
- IsDummyRingElement, 189
- IsEmptyHeap
  - for IsStringMinHeap, 163
- IsEndomorphism
  - for IsCapCategoryMorphism, 31
- IsEnrichedOverCommutativeRegular-
  - Semigroup
    - for IsCapCategory, 8
- IsEpimorphism
  - for IsCapCategoryMorphism, 27

- IsEqualAsFactorobjects
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, [35](#)
- IsEqualAsSubobjects
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, [35](#)
- IsEqualForCacheForMorphisms
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, [41](#)
- IsEqualForCacheForObjects
  - for IsCapCategoryObject, IsCapCategoryObject, [20](#)
- IsEqualForMorphisms
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, [33](#)
- IsEqualForMorphismsOnMor
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, [33](#)
- IsEqualForObjects
  - for IsCapCategoryObject, IsCapCategoryObject, [18](#)
- IsEqualToIdentityMorphism
  - for IsCapCategoryMorphism, [31](#)
- IsEqualToZeroMorphism
  - for IsCapCategoryMorphism, [31](#)
- IsEquippedWithHomomorphismStructure
  - for IsCapCategory, [8](#)
- IsHomSetInhabited
  - for IsCapCategoryObject, IsCapCategoryObject, [41](#)
- IsIdempotent
  - for IsCapCategoryMorphism, [28](#)
- IsInitial
  - for IsCapCategoryObject, [19](#)
- IsInjective
  - for IsCapCategoryObject, [19](#)
- IsIsomorphicForObjects
  - for IsCapCategoryObject, IsCapCategoryObject, [18](#)
- IsIsomorphism
  - for IsCapCategoryMorphism, [27](#)
- IsLiftable
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, [39](#)
- IsLiftableAlongMonomorphism
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, [39](#)
- IsLinearCategoryOverCommutativeRing
  - for IsCapCategory, [8](#)
- IsLinearCategoryOverCommutativeRing-
  - WithFinitelyGeneratedFree-ExternalHoms
  - for IsCapCategory, [8](#)
- IsLocallyOfFiniteInjectiveDimension
  - for IsCapCategory, [9](#)
- IsLocallyOfFiniteProjectiveDimension
  - for IsCapCategory, [9](#)
- IsMonomorphism
  - for IsCapCategoryMorphism, [27](#)
- IsMorphismInCapTerminalCategoryWith-
  - MultipleObjects
  - for IsCapCategoryMorphism, [211](#)
- IsMorphismInCapTerminalCategoryWith-
  - SingleObject
  - for IsCapCategoryMorphism, [210](#)
- IsObjectInCapTerminalCategoryWith-
  - MultipleObjects
  - for IsCapCategoryObject, [210](#)
- IsObjectInCapTerminalCategoryWith-
  - SingleObject
  - for IsCapCategoryObject, [210](#)
- IsomorphismFromCoequalizerOfCoproduct-
  - DiagramToPushout
  - for IsList, [88](#)
- IsomorphismFromCoequalizerToCokernel-
  - OfJointPairwiseDifferencesOf-MorphismsFromCoproduct
  - for IsCapCategoryObject, IsList, [84](#)
- IsomorphismFromCoimageToCokernelOf-
  - Kernel
  - for IsCapCategoryMorphism, [94](#)
- IsomorphismFromCokernelOfJoint-
  - PairwiseDifferencesOfMorphisms-FromCoproductToCoequalizer
  - for IsCapCategoryObject, IsList, [84](#)
- IsomorphismFromCokernelOfKernelTo-
  - Coimage
  - for IsCapCategoryMorphism, [94](#)
- IsomorphismFromCoproductToDirectSum
  - for IsList, [72](#)
- IsomorphismFromDirectProductTo-
  - DirectSum

- for IsList, [71](#)
- IsomorphismFromDirectSumToCoproduct
  - for IsList, [71](#)
- IsomorphismFromDirectSumToDirect-Product
  - for IsList, [71](#)
- IsomorphismFromEqualizerOfDirect-ProductDiagramToFiberProduct
  - for IsList, [85](#)
- IsomorphismFromEqualizerToKernelOf-JointPairwiseDifferencesOf-MorphismsIntoDirectProduct
  - for IsCapCategoryObject, IsList, [81](#)
- IsomorphismFromFiberProductTo-EqualizerOfDirectProductDiagram
  - for IsList, [85](#)
- IsomorphismFromHomologyObjectToIts-ConstructionAsAnImageObject
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, [97](#)
- IsomorphismFromImageObjectToKernelOf-Cokernel
  - for IsCapCategoryMorphism, [91](#)
- IsomorphismFromInitialObjectToZero-Object
  - for IsCapCategory, [66](#)
- IsomorphismFromItsConstructionAsAn-ImageObjectToHomologyObject
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, [97](#)
- IsomorphismFromKernelOfCokernelTo-ImageObject
  - for IsCapCategoryMorphism, [91](#)
- IsomorphismFromKernelOfJointPairwise-DifferencesOfMorphismsInto-DirectProductToEqualizer
  - for IsCapCategoryObject, IsList, [81](#)
- IsomorphismFromPushoutToCoequalizerOf-CoproductDiagram
  - for IsList, [88](#)
- IsomorphismFromTerminalObjectToZero-Object
  - for IsCapCategory, [66](#)
- IsomorphismFromZeroObjectToInitial-Object
  - for IsCapCategory, [65](#)
- IsomorphismFromZeroObjectToTerminal-Object
  - for IsCapCategory, [66](#)
- IsOne
  - for IsCapCategoryMorphism, [28](#)
- IsOperationWeightList
  - for IsAttributeStoringRep, [160](#)
- IsPreAbelianCategory
  - for IsCapCategory, [9](#)
- IsProjective
  - for IsCapCategoryObject, [19](#)
- IsSkeletalCategory
  - for IsCapCategory, [8](#)
- IsSpecializationOfFilter, [167](#)
- IsSpecializationOfFilterList, [168](#)
- IsSplitEpimorphism
  - for IsCapCategoryMorphism, [28](#)
- IsSplitMonomorphism
  - for IsCapCategoryMorphism, [28](#)
- IsStringMinHeap
  - for IsAttributeStoringRep, [162](#)
- IsTerminal
  - for IsCapCategoryObject, [19](#)
- IsTerminalCategory
  - for IsCapCategory, [211](#)
- IsWellDefined
  - for IsCapCategoryCell, [14](#)
- IsWellDefinedForMorphisms
  - for IsCapCategoryMorphism, [37](#)
- IsWellDefinedForMorphismsWithGiven-SourceAndRange
  - for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryObject, [38](#)
- IsWellDefinedForObjects
  - for IsCapCategoryObject, [23](#)
- IsWellDefinedForTwoCells
  - for IsCapCategoryTwoCell, [52](#)
- IsWrapperCapCategory
  - for IsCapCategory, [186](#)
- IsWrapperCapCategoryMorphism
  - for IsCapCategoryMorphism, [186](#)
- IsWrapperCapCategoryObject
  - for IsCapCategoryObject, [186](#)
- IsZero
  - for IsCapCategoryObject, [20](#)
- IsZeroForMorphisms

- for IsCapCategoryMorphism, [34](#)
- IsZeroForObjects
  - for IsCapCategoryObject, [19](#)
- Iterated
  - for IsList, IsFunction, IsObject, [170](#)
  - for IsList, IsFunction, IsObject, IsObject, [170](#)
- JointPairwiseDifferencesOfMorphisms-FromCoproduct
  - for IsCapCategoryObject, IsList, [84](#)
- JointPairwiseDifferencesOfMorphisms-IntoDirectProduct
  - for IsCapCategoryObject, IsList, [80](#)
- KernelEmbedding
  - for IsCapCategoryMorphism, [60](#)
- KernelEmbeddingWithGivenKernelObject
  - for IsCapCategoryMorphism, IsCapCategoryObject, [60](#)
- KernelLift
  - for IsCapCategoryMorphism, IsCapCategoryObject, IsCapCategoryMorphism, [60](#)
- KernelLiftWithGivenKernelObject
  - for IsCapCategoryMorphism, IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryObject, [60](#)
- KernelObject
  - for IsCapCategoryMorphism, [59](#)
- KernelObjectFunctorial
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, [61](#)
  - for IsList, [61](#)
- KernelObjectFunctorialWithGivenKernelObjects
  - for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject, [61](#)
  - for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject, [61](#)
- LastWithKeys, [174](#)
- LaTeXOutput
  - for IsCapCategory, [17](#)
  - for IsCapCategoryCell, [17](#)
- Lift
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, [39](#)
- LiftAlongMonomorphism
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, [38](#)
- LiftOrFail
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, [39](#)
- LinearCombinationOfMorphisms
  - for IsCapCategoryObject, IsList, IsList, IsCapCategoryObject, [37](#)
- ListCAPPrepareFunctions, [105](#)
- ListWithKeys, [173](#)
- MakeDerivation
  - for IsString, IsString, IsDenseList, IsPosInt, IsFunction, IsFunction, [156](#)
- MakeDerivationGraph
  - for IsDenseList, [158](#)
- MakeOperationWeightList
  - for IsCapCategory, IsDerivedMethodGraph, [160](#)
- MereExistenceOfSolutionOfLinearSystem-InAbCategory
  - for IsList, IsList, IsList, [43](#)
- MissingOperationsForConstructiveness-OfCategory
  - for IsCapCategory, IsString, [14](#)
- ModelingCategory
  - for IsCapCategory, [184](#)
- ModelingMorphism
  - for IsCapCategory, IsCapCategoryMorphism, [185](#)
- ModelingObject
  - for IsCapCategory, IsCapCategoryObject, [185](#)
- MonomorphismIntoInjectiveEnvelopeObject
  - for IsCapCategoryObject, [98](#)
- MonomorphismIntoInjectiveEnvelopeObjectWithGivenInjectiveEnvelopeObject

- for IsCapCategoryObject, IsCapCategory-Object, 98
- MonomorphismIntoSomeInjectiveObject
  - for IsCapCategoryObject, 24
- MonomorphismIntoSomeInjectiveObject-WithGivenSomeInjectiveObject
  - for IsCapCategoryObject, IsCapCategory-Object, 24
- MorphismBetweenDirectSums
  - for IsList, 72
  - for IsList, IsList, IsList, 72
- MorphismBetweenDirectSumsWithGiven-DirectSums
  - for IsCapCategoryObject, IsList, IsList, Is-List, IsCapCategoryObject, 72
- MorphismCache
  - for IsCapFunctor, 164
- MorphismConstructor
  - for IsCapCategoryObject, IsObject, IsCap-CategoryObject, 27
- MorphismDatum
  - for IsCapCategoryMorphism, 27
- MorphismDatumType
  - for IsCapCategory, 11
- MorphismFilter
  - for IsCapCategory, 11
- MorphismFromCoimageToImage
  - for IsCapCategoryMorphism, 93
- MorphismFromCoimageToImageWithGiven-Objects
  - for IsCapCategoryObject, IsCapCategory-Morphism, IsCapCategoryObject, 93
- MorphismFromEqualizerToSink
  - for IsCapCategoryObject, IsList, 79
- MorphismFromEqualizerToSinkWithGiven-Equalizer
  - for IsCapCategoryObject, IsList, IsCapCate-goryObject, 79
- MorphismFromFiberProductToSink
  - for IsList, 86
- MorphismFromFiberProductToSinkWith-GivenFiberProduct
  - for IsList, IsCapCategoryObject, 86
- MorphismFromKernelObjectToSink
  - for IsCapCategoryMorphism, 60
- MorphismFromKernelObjectToSinkWith-
  - GivenKernelObject
    - for IsCapCategoryMorphism, IsCapCatego-ryObject, 60
- MorphismFromSourceToCoequalizer
  - for IsCapCategoryObject, IsList, 82
- MorphismFromSourceToCoequalizerWith-GivenCoequalizer
  - for IsCapCategoryObject, IsList, IsCapCate-goryObject, 83
- MorphismFromSourceToCokernelObject
  - for IsCapCategoryMorphism, 62
- MorphismFromSourceToCokernelObject-WithGivenCokernelObject
  - for IsCapCategoryMorphism, IsCapCatego-ryObject, 63
- MorphismFromSourceToPushout
  - for IsList, 89
- MorphismFromSourceToPushoutWithGiven-Pushout
  - for IsList, IsCapCategoryObject, 89
- MorphismFromZeroObject
  - for IsCapCategoryObject, 65
- MorphismIntoZeroObject
  - for IsCapCategoryObject, 65
- MorphismsOfExternalHom
  - for IsCapCategoryObject, IsCapCategory-Object, 45
- MultiplyWithElementOfCommutativeRing-ForMorphisms
  - for IsRingElement, IsCapCategoryMor-phism, 34
- Name
  - for IsCapCategory, 10
  - for IsCapNaturalTransformation, 57
- NaturalIsomorphismFromIdentityTo-CanonicalizeZeroMorphisms
  - for IsCapCategory, 57
- NaturalIsomorphismFromIdentityTo-CanonicalizeZeroObjects
  - for IsCapCategory, 57
- NaturalTransformation
  - for IsCapFunctor, IsCapFunctor, 58
- NTuple, 171
- NumberWithKeys, 173
- ObjectCache



- for IsCapFunctor, 164
- ObjectConstructor
  - for IsCapCategory, IsObject, 22
- ObjectDatum
  - for IsCapCategoryObject, 22
- ObjectDatumType
  - for IsCapCategory, 11
- ObjectFilter
  - for IsCapCategory, 11
- ObjectifyMorphismWithSourceAndRange-
  - ForCAPWithAttributes, 32
- ObjectifyObjectForCAPWithAttributes, 21
- Operations
  - for IsDerivedMethodGraph, 159
- OperationWeightUsingDerivation
  - for IsOperationWeightList, IsDerived-
    - Method, 160
- PackageOfCAPOperation, 170
- Pair, 171
- PostCompose
  - for IsCapCategoryMorphism, IsCapCatego-
    - ryMorphism, 37
  - for IsList, 37
- PostComposeList
  - for IsCapCategoryObject, IsList, IsCapCate-
    - goryObject, 37
- PostInverseForMorphisms
  - for IsCapCategoryMorphism, 40
- PreCompose
  - for IsCapCategoryMorphism, IsCapCatego-
    - ryMorphism, 36
  - for IsList, 36
- PreComposeList
  - for IsCapCategoryObject, IsList, IsCapCate-
    - goryObject, 36
- PreInverseForMorphisms
  - for IsCapCategoryMorphism, 40
- PrintDerivationTree
  - for IsOperationWeightList, IsString, 161
- PrintTree
  - for IsObject, IsFunction, IsFunction, 162
- PrintTreeRec
  - for IsObject, IsFunction, IsFunction, IsInt,
    - 162
- ProductWithKeys, 173
- ProjectionInFactorOfDirectProduct
  - for IsList, IsInt, 76
- ProjectionInFactorOfDirectProductWith-
  - GivenDirectProduct
    - for IsList, IsInt, IsCapCategoryObject, 77
- ProjectionInFactorOfDirectSum
  - for IsList, IsInt, 70
- ProjectionInFactorOfDirectSumWith-
  - GivenDirectSum
    - for IsList, IsInt, IsCapCategoryObject, 70
- ProjectionInFactorOfFiberProduct
  - for IsList, IsInt, 86
- ProjectionInFactorOfFiberProductWith-
  - GivenFiberProduct
    - for IsList, IsInt, IsCapCategoryObject, 86
- ProjectionOntoCoequalizer
  - for IsCapCategoryObject, IsList, 82
- ProjectionOntoCoequalizerWithGiven-
  - Coequalizer
    - for IsCapCategoryObject, IsList, IsCapCate-
      - goryObject, 82
- ProjectiveCoverObject
  - for IsCapCategoryObject, 97
- ProjectiveDimension
  - for IsCapCategoryObject, 25
- ProjectiveLift
  - for IsCapCategoryMorphism, IsCapCatego-
    - ryMorphism, 23
- Pushout
  - for IsCapCategoryMorphism, IsCapCatego-
    - ryMorphism, 89
  - for IsList, 89
- PushoutFunctorial
  - for IsList, IsList, IsList, 90
- PushoutFunctorialWithGivenPushouts
  - for IsCapCategoryObject, IsList, IsList, Is-
    - List, IsCapCategoryObject, 90
- PushoutProjectionFromCoproduct
  - for IsList, 88
- PushoutProjectionFromDirectSum
  - for IsList, 88
- RandomMorphism
  - for IsCapCategory, IsInt, 31
  - for IsCapCategory, IsList, 31
  - for IsCapCategoryObject, IsCapCategory-
    - Object, IsInt, 31

- for IsCapCategoryObject, IsCapCategory-Object, IsList, [31](#)
- RandomMorphismByInteger
  - for IsCapCategory, IsInt, [30](#)
- RandomMorphismByList
  - for IsCapCategory, IsList, [30](#)
- RandomMorphismWithFixedRange
  - for IsCapCategoryObject, IsInt, [31](#)
  - for IsCapCategoryObject, IsList, [31](#)
- RandomMorphismWithFixedRangeByInteger
  - for IsCapCategoryObject, IsInt, [29](#)
- RandomMorphismWithFixedRangeByList
  - for IsCapCategoryObject, IsList, [29](#)
- RandomMorphismWithFixedSource
  - for IsCapCategoryObject, IsInt, [31](#)
  - for IsCapCategoryObject, IsList, [31](#)
- RandomMorphismWithFixedSourceAndRange
  - for IsCapCategoryObject, IsCapCategory-Object, IsInt, [31](#)
  - for IsCapCategoryObject, IsCapCategory-Object, IsList, [31](#)
- RandomMorphismWithFixedSourceAndRange-ByInteger
  - for IsCapCategoryObject, IsCapCategory-Object, IsInt, [29](#)
- RandomMorphismWithFixedSourceAndRange-ByList
  - for IsCapCategoryObject, IsCapCategory-Object, IsList, [30](#)
- RandomMorphismWithFixedSourceByInteger
  - for IsCapCategoryObject, IsInt, [28](#)
- RandomMorphismWithFixedSourceByList
  - for IsCapCategoryObject, IsList, [29](#)
- RandomObject
  - for IsCapCategory, IsInt, [20](#)
  - for IsCapCategory, IsList, [20](#)
- RandomObjectByInteger
  - for IsCapCategory, IsInt, [20](#)
- RandomObjectByList
  - for IsCapCategory, IsList, [20](#)
- Range
  - for IsCapCategoryMorphism, [26](#)
  - for IsCapCategoryTwoCell, [51](#)
- RangeCategoryOfHomomorphismStructure
  - for IsCapCategory, [12](#)
- RangeOfFunctor
  - for IsCapFunctor, [55](#)
- Reevaluate
  - for IsOperationWeightList, [161](#)
- ReinterpretationFunctor
  - for IsCapCategory, [184](#)
- ReinterpretationOfCategory
  - for IsCapCategory, IsRecord, [184](#)
- ReinterpretationOfMorphism
  - for IsCapCategory, IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryObject, [185](#)
- ReinterpretationOfObject
  - for IsCapCategory, IsCapCategoryObject, [185](#)
- ResetTimingStatistics, [16](#)
- SafeFirst
  - for IsList, IsFunction, [171](#)
- SafePosition
  - for IsList, IsObject, [170](#)
- SafePositionProperty
  - for IsList, IsFunction, [171](#)
- SafeUniqueEntry
  - for IsList, IsFunction, [171](#)
- SafeUniquePosition
  - for IsList, IsObject, [171](#)
- SafeUniquePositionProperty
  - for IsList, IsFunction, [171](#)
- Saturate
  - for IsOperationWeightList, [161](#)
- SetCachingOfCategory, [15](#)
- SetCachingOfCategoryCrisp, [15](#)
- SetCachingOfCategoryWeak, [15](#)
- SetDefaultCaching, [15](#)
- SetDefaultCachingCrisp, [15](#)
- SetDefaultCachingWeak, [15](#)
- Simplify
  - for IsCapCategoryMorphism, [49](#)
  - for IsCapCategoryObject, [24](#)
- SimplifyEndo
  - for IsCapCategoryMorphism, IsObject, [48](#)
- SimplifyEndo\_IsoFromInputObject
  - for IsCapCategoryMorphism, IsObject, [49](#)
- SimplifyEndo\_IsoToInputObject
  - for IsCapCategoryMorphism, IsObject, [49](#)
- SimplifyMorphism
  - for IsCapCategoryMorphism, IsObject, [46](#)

- SimplifyObject
  - for IsCapCategoryObject, IsObject, [25](#)
- SimplifyObject\_IsoFromInputObject
  - for IsCapCategoryObject, IsObject, [25](#)
- SimplifyObject\_IsoToInputObject
  - for IsCapCategoryObject, IsObject, [25](#)
- SimplifyRange
  - for IsCapCategoryMorphism, IsObject, [47](#)
- SimplifyRange\_IsoFromInputObject
  - for IsCapCategoryMorphism, IsObject, [47](#)
- SimplifyRange\_IsoToInputObject
  - for IsCapCategoryMorphism, IsObject, [47](#)
- SimplifySource
  - for IsCapCategoryMorphism, IsObject, [46](#)
- SimplifySourceAndRange
  - for IsCapCategoryMorphism, IsObject, [47](#)
- SimplifySourceAndRange\_IsoFromInput-Range
  - for IsCapCategoryMorphism, IsObject, [48](#)
- SimplifySourceAndRange\_IsoFromInput-Source
  - for IsCapCategoryMorphism, IsObject, [48](#)
- SimplifySourceAndRange\_IsoToInputRange
  - for IsCapCategoryMorphism, IsObject, [48](#)
- SimplifySourceAndRange\_IsoToInput-Source
  - for IsCapCategoryMorphism, IsObject, [48](#)
- SimplifySource\_IsoFromInputObject
  - for IsCapCategoryMorphism, IsObject, [47](#)
- SimplifySource\_IsoToInputObject
  - for IsCapCategoryMorphism, IsObject, [47](#)
- SolveLinearSystemInAbCategory
  - for IsList, IsList, IsList, [43](#)
- SolveLinearSystemInAbCategoryOrFail
  - for IsList, IsList, IsList, [43](#)
- SomeInjectiveObject
  - for IsCapCategoryObject, [24](#)
- SomeIsomorphismBetweenObjects
  - for IsCapCategoryObject, IsCapCategory-Object, [18](#)
- SomeProjectiveObject
  - for IsCapCategoryObject, [23](#)
- SomeReductionBySplitEpiSummand
  - for IsCapCategoryMorphism, [50](#)
- SomeReductionBySplitEpiSummand\_-MorphismFromInputRange
  - for IsCapCategoryMorphism, [50](#)
- SomeReductionBySplitEpiSummand\_-MorphismToInputRange
  - for IsCapCategoryMorphism, [50](#)
- Source
  - for IsCapCategoryMorphism, [26](#)
  - for IsCapCategoryTwoCell, [51](#)
- SourceOfFunctor
  - for IsCapFunctor, [55](#)
- StringMinHeap, [162](#)
- SubtractionForMorphisms
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, [34](#)
- SumOfMorphisms
  - for IsCapCategoryObject, IsList, IsCapCategoryObject, [37](#)
- SumWithKeys, [173](#)
- Swap
  - for IsStringMinHeap, IsPosInt, IsPosInt, [163](#)
- Target
  - for IsCapCategoryMorphism, [26](#)
  - for IsCapCategoryTwoCell, [51](#)
- TargetOperation
  - for IsDerivedMethod, [158](#)
- TerminalCategoryWithMultipleObjects, [211](#)
- TerminalCategoryWithSingleObject, [211](#)
- TerminalObject
  - for IsCapCategory, [67](#)
  - for IsCapCategoryCell, [67](#)
- TerminalObjectFunctorial
  - for IsCapCategory, [67](#)
- TerminalObjectFunctorialWithGiven-TerminalObjects
  - for IsCapCategoryObject, IsCapCategory-Object, [67](#)
- TransitivelyNeededOtherPackages, [170](#)
- TransposedMatWithGivenDimensions, [172](#)
- Triple, [172](#)
- TwoCellDatumType
  - for IsCapCategory, [11](#)
- TwoCellFilter
  - for IsCapCategory, [11](#)
- UnderlyingCell
  - for IsWrapperCapCategoryMorphism, [186](#)

- for IsWrapperCapCategoryObject, [186](#)
- UniqueMorphism
  - for IsCapTerminalCategoryWithSingleObject, [212](#)
- UniqueObject
  - for IsCapTerminalCategoryWithSingleObject, [211](#)
- UniversalMorphismFromCoequalizer
  - for IsCapCategoryObject, IsList, IsCapCategoryObject, IsCapCategoryMorphism, [83](#)
- UniversalMorphismFromCoequalizerWithGivenCoequalizer
  - for IsCapCategoryObject, IsList, IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryObject, [83](#)
- UniversalMorphismFromCoproduct
  - for IsList, IsCapCategoryObject, IsList, [75](#)
- UniversalMorphismFromCoproductWithGivenCoproduct
  - for IsList, IsCapCategoryObject, IsList, IsCapCategoryObject, [75](#)
- UniversalMorphismFromDirectSum
  - for IsList, IsCapCategoryObject, IsList, [71](#)
- UniversalMorphismFromDirectSumWithGivenDirectSum
  - for IsList, IsCapCategoryObject, IsList, IsCapCategoryObject, [71](#)
- UniversalMorphismFromImage
  - for IsCapCategoryMorphism, IsList, [92](#)
- UniversalMorphismFromImageWithGivenImageObject
  - for IsCapCategoryMorphism, IsList, IsCapCategoryObject, [92](#)
- UniversalMorphismFromInitialObject
  - for IsCapCategoryObject, [68](#)
- UniversalMorphismFromInitialObjectWithGivenInitialObject
  - for IsCapCategoryObject, IsCapCategoryObject, [68](#)
- UniversalMorphismFromPushout
  - for IsList, IsCapCategoryObject, IsList, [89](#)
- UniversalMorphismFromPushoutWithGivenPushout
  - for IsList, IsCapCategoryObject, IsList, IsCapCategoryObject, [90](#)
- UniversalMorphismFromZeroObject
  - for IsCapCategoryObject, [65](#)
- UniversalMorphismFromZeroObjectWithGivenZeroObject
  - for IsCapCategoryObject, IsCapCategoryObject, [65](#)
- UniversalMorphismIntoCoimage
  - for IsCapCategoryMorphism, IsList, [95](#)
- UniversalMorphismIntoCoimageWithGivenCoimageObject
  - for IsCapCategoryMorphism, IsList, IsCapCategoryObject, [95](#)
- UniversalMorphismIntoDirectProduct
  - for IsList, IsCapCategoryObject, IsList, [77](#)
- UniversalMorphismIntoDirectProductWithGivenDirectProduct
  - for IsList, IsCapCategoryObject, IsList, IsCapCategoryObject, [77](#)
- UniversalMorphismIntoDirectSum
  - for IsList, IsCapCategoryObject, IsList, [70](#)
- UniversalMorphismIntoDirectSumWithGivenDirectSum
  - for IsList, IsCapCategoryObject, IsList, IsCapCategoryObject, [71](#)
- UniversalMorphismIntoEqualizer
  - for IsCapCategoryObject, IsList, IsCapCategoryObject, IsCapCategoryMorphism, [79](#)
- UniversalMorphismIntoEqualizerWithGivenEqualizer
  - for IsCapCategoryObject, IsList, IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryObject, [80](#)
- UniversalMorphismIntoFiberProduct
  - for IsList, IsCapCategoryObject, IsList, [87](#)
- UniversalMorphismIntoFiberProductWithGivenFiberProduct
  - for IsList, IsCapCategoryObject, IsList, IsCapCategoryObject, [87](#)
- UniversalMorphismIntoTerminalObject
  - for IsCapCategoryObject, [67](#)
- UniversalMorphismIntoTerminalObjectWithGivenTerminalObject
  - for IsCapCategoryObject, IsCapCategoryObject, [67](#)
- UniversalMorphismIntoZeroObject

- for IsCapCategoryObject, [65](#)
- UniversalMorphismIntoZeroObjectWith-  
    GivenZeroObject
  - for IsCapCategoryObject, IsCapCategory-  
    Object, [65](#)
- UsedOperationsWithMultiplesAnd-  
    CategoryGetters
  - for IsDerivedMethod, [158](#)
- VerticalPostCompose
  - for IsCapCategoryTwoCell, IsCapCatego-  
    ryTwoCell, [52](#)
- VerticalPreCompose
  - for IsCapCategoryTwoCell, IsCapCatego-  
    ryTwoCell, [52](#)
- WrapperCategory
  - for IsCapCategory, IsRecord, [187](#)
- WrappingFunctor
  - for IsWrapperCapCategory, [188](#)
- ZeroMorphism
  - for IsCapCategoryObject, IsCapCategory-  
    Object, [35](#)
- ZeroObject
  - for IsCapCategory, [64](#)
  - for IsCapCategoryCell, [64](#)
- ZeroObjectFunctorial
  - for IsCapCategory, [66](#)
- ZeroObjectFunctorialWithGivenZero-  
    Objects
  - for IsCapCategoryObject, IsCapCategory-  
    Object, [66](#)