

# CAP

## Categories, Algorithms, Programming

2020.10-01

19 October 2020

**Sebastian Gutsche**

**Sebastian Posur**

**Øystein Skartsæterhagen**

**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

# Contents

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>CAP Categories</b>                                     | <b>5</b>  |
| 1.1      | Categories . . . . .                                      | 5         |
| 1.2      | Constructor . . . . .                                     | 6         |
| 1.3      | Internal Attributes . . . . .                             | 6         |
| 1.4      | Logic switcher . . . . .                                  | 7         |
| 1.5      | Tool functions . . . . .                                  | 8         |
| 1.6      | Well-Definedness of Cells . . . . .                       | 9         |
| 1.7      | Unpacking data structures . . . . .                       | 9         |
| 1.8      | Caching . . . . .   | 10        |
| 1.9      | Sanity checks . . . . .                                   | 10        |
| 1.10     | Enable automatic calls of Add . . . . .                   | 11        |
| 1.11     | Performance tweaks . . . . .                              | 11        |
| 1.12     | LaTeX . . . . .   | 11        |
| <b>2</b> | <b>Objects</b>  | <b>13</b> |
| 2.1      | Attributes for the Type of Objects . . . . .              | 13        |
| 2.2      | Equality for Objects . . . . .                            | 13        |
| 2.3      | Categorical Properties of Objects . . . . .               | 14        |
| 2.4      | Random Objects . . . . .                                  | 15        |
| 2.5      | Tool functions for caches . . . . .                       | 16        |
| 2.6      | Adding Objects to a Category . . . . .                    | 17        |
| 2.7      | Well-Definedness of Objects . . . . .                     | 17        |
| 2.8      | Projectives . . . . .                                     | 18        |
| 2.9      | Injectives . . . . .                                      | 19        |
| 2.10     | Simplified Objects . . . . .                              | 21        |
| <b>3</b> | <b>Morphisms</b>  | <b>23</b> |
| 3.1      | Attributes for the Type of Morphisms . . . . .            | 23        |
| 3.2      | Categorical Properties of Morphisms . . . . .             | 23        |
| 3.3      | Random Morphisms . . . . .                                | 25        |
| 3.4      | Non-Categorical Properties of Morphisms . . . . .         | 29        |
| 3.5      | Adding Morphisms to a Category . . . . .                  | 30        |
| 3.6      | Equality and Congruence for Morphisms . . . . .           | 31        |
| 3.7      | Basic Operations for Morphisms in Ab-Categories . . . . . | 32        |
| 3.8      | Subobject and Factorobject Operations . . . . .           | 34        |
| 3.9      | Identity Morphism and Composition of Morphisms . . . . .  | 36        |

|          |  |            |
|----------|--|------------|
| 3.10     | Well-Definedness of Morphisms . . . . .              | 37         |
| 3.11     | Lift/Colift . . . . .                                | 37         |
| 3.12     | Inverses . . . . .                                   | 40         |
| 3.13     | Tool functions for caches . . . . .                  | 41         |
| 3.14     | IsHomSetInhabited . . . . .                          | 41         |
| 3.15     | Homomorphism structures . . . . .                    | 42         |
| 3.16     | Simplified Morphisms . . . . .                       | 46         |
| 3.17     | Reduction by split epi summands . . . . .            | 52         |
| <b>4</b> | <b>Category 2-Cells</b>                              | <b>54</b>  |
| 4.1      | Attributes for the Type of 2-Cells . . . . .         | 54         |
| 4.2      | Identity 2-Cell and Composition of 2-Cells . . . . . | 54         |
| 4.3      | Well-Definedness for 2-Cells . . . . .               | 56         |
| <b>5</b> | <b>Category of Categories</b>                        | <b>57</b>  |
| 5.1      | The Category Cat . . . . .                           | 57         |
| 5.2      | Categories . . . . .                                 | 57         |
| 5.3      | Constructors . . . . .                               | 58         |
| 5.4      | Functors . . . . .                                   | 58         |
| 5.5      | Natural transformations . . . . .                    | 61         |
| <b>6</b> | <b>Universal Objects</b>                             | <b>63</b>  |
| 6.1      | Kernel . . . . .                                     | 63         |
| 6.2      | Cokernel . . . . .                                   | 67         |
| 6.3      | Zero Object . . . . .                                | 71         |
| 6.4      | Terminal Object . . . . .                            | 74         |
| 6.5      | Initial Object . . . . .                             | 76         |
| 6.6      | Direct Sum . . . . .                                 | 78         |
| 6.7      | Coproduct . . . . .                                  | 86         |
| 6.8      | Direct Product . . . . .                             | 90         |
| 6.9      | Equalizer . . . . .                                  | 93         |
| 6.10     | Coequalizer . . . . .                                | 97         |
| 6.11     | Fiber Product . . . . .                              | 101        |
| 6.12     | Pushout . . . . .                                    | 109        |
| 6.13     | Image . . . . .                                      | 116        |
| 6.14     | Coimage . . . . .                                    | 120        |
| 6.15     | Homology objects . . . . .                           | 125        |
| <b>7</b> | <b>Add Functions</b>                                 | <b>128</b> |
| 7.1      | Functions Installed by Add . . . . .                 | 128        |
| 7.2      | Add Method . . . . .                                 | 129        |
| 7.3      | InstallAdd Function . . . . .                        | 130        |
| 7.4      | Enhancing the method name record . . . . .           | 132        |
| 7.5      | Install All Adds . . . . .                           | 133        |
| 7.6      | Prepare functions . . . . .                          | 133        |

|           |   |            |
|-----------|---|------------|
| <b>8</b>  | <b>Managing Derived Methods</b>   | <b>135</b> |
| 8.1       | Info Class . . . . .  | 135        |
| 8.2       | Derivation Objects . . . . .  | 135        |
| 8.3       | Derivation Graphs . . . . .   | 138        |
| 8.4       | Managing Derivations in a Category . . . . .  | 140        |
| 8.5       | Min Heaps for Strings . . . . .   | 142        |
| <b>9</b>  | <b>Technical Details</b>  | <b>144</b> |
| 9.1       | The Category Cat . . . . .  | 144        |
| 9.2       | Install Functions for IsWellDefined . . . . .   | 144        |
| <b>10</b> | <b>Limits and Colimits</b>  | <b>148</b> |
| 10.1      | Specification of Limits and Colimits . . . . .  | 148        |
| 10.2      | Enhancing Limit Specifications . . . . .  | 149        |
| 10.3      | Validating entries of a method name record which are part of a limit or colimit . . . . | 150        |
| <b>11</b> | <b>Examples and Tests</b>   | <b>151</b> |
| 11.1      | Functors . . . . .  | 151        |
| 11.2      | Homomorphism structure . . . . .  | 152        |
| 11.3      | Spectral Sequences . . . . .  | 155        |
| 11.4      | Homology object . . . . .   | 158        |
| 11.5      | Liftable . . . . .  | 160        |
| 11.6      | Monoidal Categories . . . . .   | 160        |
| 11.7      | MorphismFromSourceToPushout and MorphismFromFiberProductToSink . . . . .                | 162        |
| 11.8      | Opposite category . . . . .   | 163        |
| 11.9      | Generalized Morphisms Category . . . . .  | 163        |
| 11.10     | IsWellDefined . . . . .   | 170        |
| 11.11     | Kernel . . . . .  | 171        |
| 11.12     | FiberProduct . . . . .  | 173        |
|           | <b>Index</b>  | <b>174</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.

### 1.1 Categories

#### 1.1.1 IsCapCategory (for IsObject)

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

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

#### 1.1.2 IsCapCategoryCell (for IsAttributeStoringRep)

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

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

#### 1.1.3 IsCapCategoryObject (for IsCapCategoryCell)

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

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

### 1.1.4 IsCapCategoryMorphism (for IsCapCategoryCell)

▷ IsCapCategoryMorphism(*object*) (filter)

**Returns:** true or false

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

### 1.1.5 IsCapCategoryTwoCell (for IsCapCategoryCell)

▷ IsCapCategoryTwoCell(*object*) (filter)

**Returns:** true or false

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

### 1.1.6 AddCategoricalProperty

▷ AddCategoricalProperty(*list*) (function)

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

## 1.2 Constructor

### 1.2.1 CreateCapCategory

▷ CreateCapCategory() (operation)

**Returns:** a category

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

### 1.2.2 CreateCapCategory (for IsString)

▷ CreateCapCategory(*s*) (operation)

**Returns:** a category

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

## 1.3 Internal Attributes

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

### 1.3.1 CategoryFilter (for IsCapCategory)

▷ CategoryFilter(*C*) (attribute)

**Returns:** a filter

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

### 1.3.2 CellFilter (for IsCapCategory)

▷ `CellFilter( $C$ )` (attribute)

**Returns:** a filter

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

### 1.3.3 ObjectFilter (for IsCapCategory)

▷ `ObjectFilter( $C$ )` (attribute)

**Returns:** a filter

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

### 1.3.4 MorphismFilter (for IsCapCategory)

▷ `MorphismFilter( $C$ )` (attribute)

**Returns:** a filter

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

### 1.3.5 TwoCellFilter (for IsCapCategory)

▷ `TwoCellFilter( $C$ )` (attribute)

**Returns:** a filter

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

### 1.3.6 CommutativeRingOfLinearCategory (for IsCapCategory)

▷ `CommutativeRingOfLinearCategory( $C$ )` (attribute)

**Returns:** a ring

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

## 1.4 Logic switcher

### 1.4.1 CapCategorySwitchLogicPropagationForObjectsOn

▷ `CapCategorySwitchLogicPropagationForObjectsOn( $C$ )` (function)

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

### 1.4.2 CapCategorySwitchLogicPropagationForObjectsOff

▷ `CapCategorySwitchLogicPropagationForObjectsOff( $C$ )` (function)

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



### 1.4.3 CapCategorySwitchLogicPropagationForMorphismsOn

▷ `CapCategorySwitchLogicPropagationForMorphismsOn(C)` (function)

Activates the predicate logic propagation between equal morphisms for the category *C*.

### 1.4.4 CapCategorySwitchLogicPropagationForMorphismsOff

▷ `CapCategorySwitchLogicPropagationForMorphismsOff(C)` (function)

Deactivates the predicate logic propagation between equal morphisms for the category *C*.

### 1.4.5 CapCategorySwitchLogicPropagationOn

▷ `CapCategorySwitchLogicPropagationOn(C)` (function)

Activates the predicate logic propagation between equal cells for the category *C*.

### 1.4.6 CapCategorySwitchLogicPropagationOff

▷ `CapCategorySwitchLogicPropagationOff(C)` (function)

Deactivates the predicate logic propagation between equal cells for the category *C*.

### 1.4.7 CapCategorySwitchLogicOn

▷ `CapCategorySwitchLogicOn(C)` (function)

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

### 1.4.8 CapCategorySwitchLogicOff

▷ `CapCategorySwitchLogicOff(C)` (function)

Deactivates the predicate implication logic for the category *C*.

## 1.5 Tool functions

### 1.5.1 CanCompute (for IsCapCategory, IsString)

▷ `CanCompute(C, s)` (operation)

**Returns:** `true` or `false`

The argument is a category *C* and a string *s*, which should be the name of a basic operation, e.g., `PreCompose`. If applying this method is possible in *C*, the method returns `true`, `false` otherwise. If the string is not the name of a basic operation, an error is raised.

### 1.5.2 CheckConstructivenessOfCategory (for IsCapCategory, IsString)

▷ `CheckConstructivenessOfCategory(C, s)` (operation)

**Returns:** a list

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

## 1.6 Well-Definedness of Cells

### 1.6.1 IsWellDefined (for IsCapCategoryCell)

▷ `IsWellDefined(c)` (property)

**Returns:** a boolean

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

## 1.7 Unpacking data structures

### 1.7.1 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*).

### 1.7.2 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).

### 1.7.3 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.

## 1.8 Caching

### 1.8.1 SetCachingOfCategory

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

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

### 1.8.2 SetCachingOfCategoryWeak

▷ `SetCachingOfCategoryWeak(category)` (function)

▷ `SetCachingOfCategoryCrisp(category)` (function)

▷ `DeactivateCachingOfCategory(category)` (function)

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

### 1.8.3 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.

## 1.9 Sanity checks

### 1.9.1 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.

## 1.10 Enable automatic calls of Add

### 1.10.1 EnableAddForCategoricalOperations

- ▷ `EnableAddForCategoricalOperations(C)` (function)
- ▷ `DisableAddForCategoricalOperations(C)` (function)

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

## 1.11 Performance tweaks

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

- `DeactivateCachingOfCategory` or `DeactivateDefaultCaching`: see 1.8. This can either improve or degrade the performance depending on the concrete example.
- `CapCategorySwitchLogicOff` (on by default) or `CapCategorySwitchLogicPropagationOff` (off by default): see 1.4. This can either improve or degrade the performance depending on the concrete example.
- `DisableSanityChecks`: see 1.9.
- `DisableAddForCategoricalOperations`: see 1.10.
- `DeactivateToDoList`: see the package `ToolsForHomalg`.
- Use `ObjectifyObjectForCAPWithAttributes` (2.6) instead of `AddObject` and `ObjectifyMorphismWithSourceAndRangeForCAPWithAttributes` (3.5) instead of `AddMorphism`.
- Add all attribute testers (`Has...`) of your objects resp. morphisms to the filters passed to `AddObjectRepresentation` (2.6) resp. `AddMorphismRepresentation` (3.5).
- Pass the option `overhead := false` to `CreateCapCategory`. Note: this may have unintended effects. Use with care!

## 1.12 LaTeX

### 1.12.1 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.

### 1.12.2 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 Equality for Objects

#### 2.2.1 `IsEqualForObjects` (for `IsCapCategoryObject`, `IsCapCategoryObject`)

- ▷ `IsEqualForObjects(a, b)` (operation)  
**Returns:** a boolean  
The arguments are two objects  $a$  and  $b$ . The output is `true` if  $a = b$ , otherwise the output is `false`.

#### 2.2.2 `AddIsEqualForObjects` (for `IsCapCategory`, `IsFunction`)

- ▷ `AddIsEqualForObjects(C, F)` (operation)  
**Returns:** nothing  
The arguments are a category  $C$  and a function  $F$ . This operation adds the given function  $F$  to the category for the basic operation `IsEqualForObjects`.  $F : (a, b) \mapsto \text{IsEqualForObjects}(a, b)$ .

## 2.3 Categorical Properties of Objects

### 2.3.1 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.

### 2.3.2 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`.  $F : a \mapsto \text{IsProjective}(a)$ .

### 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 AddIsInjective (for IsCapCategory, IsFunction)

▷ `AddIsInjective(C, F)` (operation)  
**Returns:** nothing  
 The arguments are a category  $C$  and a function  $F$ . This operation adds the given function  $F$  to the category for the basic operation `IsInjective`.  $F : a \mapsto \text{IsInjective}(a)$ .

### 2.3.5 IsTerminal (for IsCapCategoryObject)

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

### 2.3.6 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`.  $F : a \mapsto \text{IsTerminal}(a)$ .

### 2.3.7 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`.

### 2.3.8 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`.  $F : a \mapsto \text{IsInitial}(a)$ .

### 2.3.9 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`.

### 2.3.10 IsZero (for IsCapCategoryObject)

▷ `IsZero( $a$ )` (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`.

### 2.3.11 AddIsZeroForObjects (for IsCapCategory, IsFunction)

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

## 2.4 Random Objects

CAP provides two principal methods to generate random objects:

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

### 2.4.1 RandomObjectByInteger (for IsCapCategory, IsInt)

▷ `RandomObjectByInteger( $\mathcal{C}$ ,  $n$ )` (operation)

**Returns:** an object or `fail`

The arguments are a category  $\mathcal{C}$  and an integer  $n$ . The output is a random object in  $\mathcal{C}$  or `fail`.



### 2.4.2 AddRandomObjectByInteger (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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

### 2.4.3 RandomObjectByList (for IsCapCategory, IsList)

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

**Returns:** an object or fail

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

### 2.4.4 AddRandomObjectByList (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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

### 2.4.5 RandomObject (for IsCapCategory, IsInt)

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

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

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

## 2.5 Tool functions for caches

### 2.5.1 IsEqualForCacheForObjects (for IsCapCategoryObject, IsCapCategoryObject)

▷ IsEqualForCacheForObjects( $\phi$ ,  $\psi$ ) (operation)

**Returns:** true or false

Compares two objects in the cache

### 2.5.2 AddIsEqualForCacheForObjects (for IsCapCategory, IsFunction)

▷ AddIsEqualForCacheForObjects( $c$ ,  $F$ ) (operation)

**Returns:** nothing

By default, CAP uses caches to store the values of Categorical operations. To get a value out of the cache, one needs to compare the input of a basic operation with its previous input. To compare objects in the category, 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, this function used instead. A

function  $F : a, b \mapsto \text{bool}$  is expected here. The output has to be true or false. Fail is not allowed in this context.

## 2.6 Adding Objects to a Category

### 2.6.1 Add (for IsCapCategory, IsCapCategoryObject)

▷ `Add(category, object)` (operation)

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

### 2.6.2 AddObject (for IsCapCategory, IsAttributeStoringRep)

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

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

### 2.6.3 AddObjectRepresentation (for IsCapCategory, IsObject)

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

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

### 2.6.4 ObjectifyObjectForCAPWithAttributes

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

**Returns:** an object

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

## 2.7 Well-Definedness of Objects

### 2.7.1 IsWellDefinedForObjects (for IsCapCategoryObject)

▷ `IsWellDefinedForObjects(a)` (operation)

**Returns:** a boolean

The argument is an object *a*. The output is true if *a* is well-defined, otherwise the output is false.

### 2.7.2 AddIsWellDefinedForObjects (for IsCapCategory, IsFunction)

▷ `AddIsWellDefinedForObjects( $C$ ,  $F$ )` (operation)

**Returns:** nothing

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

## 2.8 Projectives

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

### 2.8.1 SomeProjectiveObject (for IsCapCategoryObject)

▷ `SomeProjectiveObject( $A$ )` (attribute)

**Returns:** an object

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

### 2.8.2 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( $A$ )`.

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

### 2.8.4 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  $\epsilon : B \rightarrow A$ . The output is a morphism  $\lambda : P \rightarrow B$  such that  $\epsilon \circ \lambda = \pi$ .

### 2.8.5 AddSomeProjectiveObject (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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

### 2.8.6 AddEpimorphismFromSomeProjectiveObject (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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

### 2.8.7 AddEpimorphismFromSomeProjectiveObjectWithGivenSomeProjectiveObject (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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

### 2.8.8 AddProjectiveLift (for IsCapCategory, IsFunction)

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

**Returns:** nothing

The arguments are a category  $C$  and a function  $F$ . This operation adds the given function  $F$  to the category for the basic operation `ProjectiveLift`. The function  $F$  maps a pair  $(\pi, \varepsilon)$  to a projective lift  $\lambda$ .

## 2.9 Injectives

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

### 2.9.1 SomeInjectiveObject (for IsCapCategoryObject)

▷ `SomeInjectiveObject(A)` (attribute)

**Returns:** an object

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

### 2.9.2 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)`.

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

### 2.9.4 InjectiveColift (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

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

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

### 2.9.5 AddSomeInjectiveObject (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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

### 2.9.6 AddMonomorphismIntoSomeInjectiveObject (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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

### 2.9.7 AddMonomorphismIntoSomeInjectiveObjectWithGivenSomeInjectiveObject (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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

### 2.9.8 AddInjectiveColift (for IsCapCategory, IsFunction)

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

**Returns:** nothing

The arguments are a category  $C$  and a function  $F$ . This operation adds the given function  $F$  to the category for the basic operation `InjectiveColift`. The function  $F$  maps a pair  $(\iota, \beta)$  to an injective colift  $\lambda$  if it exists, and to fail otherwise.

## 2.10 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.

### 2.10.1 Simplify (for IsCapCategoryObject)

▷ `Simplify( $A$ )` (attribute)  
**Returns:** an object  
 The argument is an object  $A$ . The output is a simplified object  $A_\infty$ .

### 2.10.2 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  $A_i$ .

### 2.10.3 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`. The function  $F$  maps  $(A, i)$  to  $A_i$ .

### 2.10.4 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  $\iota_A^i : A \rightarrow A_i$ .

### 2.10.5 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`. The function  $F$  maps  $(A, i)$  to  $\iota_A^i$ .

### 2.10.6 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`.

### 2.10.7 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`. The function  $F$  maps  $(A, i)$  to  $(\iota_A^i)^{-1}$ .

## 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  $b$ .

### 3.2 Categorical Properties of Morphisms

#### 3.2.1 `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`.



### 3.2.2 AddIsMonomorphism (for IsCapCategory, IsFunction)

▷ AddIsMonomorphism( $C$ ,  $F$ ) (operation)

**Returns:** nothing

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.  $F : \alpha \mapsto \text{IsMonomorphism}(\alpha)$ .

### 3.2.3 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.

### 3.2.4 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.  $F : \alpha \mapsto \text{IsEpimorphism}(\alpha)$ .

### 3.2.5 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.

### 3.2.6 AddIsIsomorphism (for IsCapCategory, IsFunction)

▷ AddIsIsomorphism( $C$ ,  $F$ ) (operation)

**Returns:** nothing

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

### 3.2.7 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.

### 3.2.8 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.  $F : \alpha \mapsto \text{IsSplitMonomorphism}(\alpha)$ .

### 3.2.9 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.

### 3.2.10 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`.  $F : \alpha \mapsto \text{IsSplitEpimorphism}(\alpha)$ .

### 3.2.11 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.

### 3.2.12 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`.  $F : \alpha \mapsto \text{IsOne}(\alpha)$ .

### 3.2.13 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.

### 3.2.14 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`.  $F : \alpha \mapsto \text{IsIdempotent}(\alpha)$ .

## 3.3 Random Morphisms

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

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

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

### 3.3.1 RandomMorphismWithFixedSourceByInteger (for IsCapCategoryObject, IsInt)

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

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

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

### 3.3.2 AddRandomMorphismWithFixedSourceByInteger (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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

### 3.3.3 RandomMorphismWithFixedSourceByList (for IsCapCategoryObject, IsList)

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

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

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

### 3.3.4 AddRandomMorphismWithFixedSourceByList (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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

### 3.3.5 RandomMorphismWithFixedRangeByInteger (for IsCapCategoryObject, IsInt)

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

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

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

### 3.3.6 AddRandomMorphismWithFixedRangeByInteger (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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

### 3.3.7 RandomMorphismWithFixedRangeByList (for IsCapCategoryObject, IsList)

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

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

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

### 3.3.8 AddRandomMorphismWithFixedRangeByList (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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

### 3.3.9 RandomMorphismWithFixedSourceAndRangeByInteger (for IsCapCategoryObject, IsCapCategoryObject, IsInt)

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

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

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

### 3.3.10 AddRandomMorphismWithFixedSourceAndRangeByInteger (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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

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

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

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

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

### 3.3.12 AddRandomMorphismWithFixedSourceAndRangeByList (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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

### 3.3.13 RandomMorphismByInteger (for IsCapCategory, IsInt)

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

**Returns:** a morphism or fail

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

### 3.3.14 AddRandomMorphismByInteger (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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

### 3.3.15 RandomMorphismByList (for IsCapCategory, IsList)

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

**Returns:** a morphism or fail

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

### 3.3.16 AddRandomMorphismByList (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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

### 3.3.17 RandomMorphismWithFixedSource (for IsCapCategoryObject, IsInt)

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

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

▷ RandomMorphismWithFixedRange( $b, n$ ) (operation)

▷ RandomMorphismWithFixedRange( $b, L$ ) (operation)

▷ RandomMorphismWithFixedSourceAndRange( $a, b, n$ ) (operation)

- ▷ `RandomMorphismWithFixedSourceAndRange(a, b, L)` (operation)
- ▷ `RandomMorphism(C, n)` (operation)
- ▷ `RandomMorphism(C, L)` (operation)

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

### 3.4 Non-Categorical Properties of Morphisms

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

#### 3.4.1 `IsIdenticalToIdentityMorphism` (for `IsCapCategoryMorphism`)

- ▷ `IsIdenticalToIdentityMorphism(alpha)` (property)

**Returns:** a boolean

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

#### 3.4.2 `AddIsIdenticalToIdentityMorphism` (for `IsCapCategory`, `IsFunction`)

- ▷ `AddIsIdenticalToIdentityMorphism(C, F)` (operation)

**Returns:** nothing

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

#### 3.4.3 `IsIdenticalToZeroMorphism` (for `IsCapCategoryMorphism`)

- ▷ `IsIdenticalToZeroMorphism(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.

#### 3.4.4 `AddIsIdenticalToZeroMorphism` (for `IsCapCategory`, `IsFunction`)

- ▷ `AddIsIdenticalToZeroMorphism(C, F)` (operation)

**Returns:** nothing

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

#### 3.4.5 `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.

### 3.4.6 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.  $F : \alpha \mapsto \text{IsEndomorphism}(\alpha)$ .

### 3.4.7 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.

### 3.4.8 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.  $F : \alpha \mapsto \text{IsAutomorphism}(\alpha)$ .

## 3.5 Adding Morphisms to a Category

### 3.5.1 Add (for IsCapCategory, IsCapCategoryMorphism)

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

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

### 3.5.2 AddMorphism (for IsCapCategory, IsAttributeStoringRep)

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

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

### 3.5.3 AddMorphismRepresentation (for IsCapCategory, IsObject)

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

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

### 3.5.4 ObjectifyMorphismWithSourceAndRangeForCAPWithAttributes

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

**Returns:** a morphism

Objectifies the morphism *morphism* with the type created for morphisms in the category *category*. The type is created by passing a representation to `AddMorphismRepresentation`. Morphisms which are objectified using this method do not have to be passed to the `AddMorphism` function. The arguments *source* and *range* are assumed to be objectified. The optional arguments behave like the corresponding arguments in `ObjectifyWithAttributes`. Also returns the objectified morphism.

## 3.6 Equality and Congruence for Morphisms

### 3.6.1 `IsCongruentForMorphisms` (for `IsCapCategoryMorphism`, `IsCapCategoryMorphism`)

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

**Returns:** a boolean

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

### 3.6.2 `AddIsCongruentForMorphisms` (for `IsCapCategory`, `IsFunction`)

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

**Returns:** nothing

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

### 3.6.3 `IsEqualForMorphisms` (for `IsCapCategoryMorphism`, `IsCapCategoryMorphism`)

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

**Returns:** a boolean

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

### 3.6.4 `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`.  $F : (\alpha, \beta) \mapsto \text{IsEqualForMorphisms}(\alpha, \beta)$ .

### 3.6.5 `IsEqualForMorphismsOnMor` (for `IsCapCategoryMorphism`, `IsCapCategoryMorphism`)

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

**Returns:** a boolean

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



### 3.6.6 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.  $F : (\alpha, \beta) \mapsto \text{IsEqualForMorphismsOnMor}(\alpha, \beta)$ .

## 3.7 Basic Operations for Morphisms in Ab-Categories

### 3.7.1 IsZeroForMorphisms (for IsCapCategoryMorphism)

▷ IsZeroForMorphisms( $\alpha$ ) (property)

**Returns:** a boolean

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

### 3.7.2 AddIsZeroForMorphisms (for IsCapCategory, IsFunction)

▷ AddIsZeroForMorphisms( $C$ ,  $F$ ) (operation)

**Returns:** nothing

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

### 3.7.3 AdditionForMorphisms (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

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

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

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

### 3.7.5 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  $\alpha - \beta$ .

### 3.7.6 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.  $F : (\alpha, \beta) \mapsto \alpha - \beta$ .

### 3.7.7 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  $-\alpha$ .

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

### 3.7.9 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  $r \cdot \alpha$ .

### 3.7.10 \\* (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  $r \cdot \alpha$ .

### 3.7.11 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.  $F : (r, \alpha) \mapsto r \cdot \alpha$ .

### 3.7.12 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  $0 : a \rightarrow b$ .

### 3.7.13 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.  $F : (a, b) \mapsto (0 : a \rightarrow b)$ .

## 3.8 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.

### 3.8.1 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.

### 3.8.2 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.  $F : (\alpha, \beta) \mapsto \text{IsEqualAsSubobjects}(\alpha, \beta)$ .

### 3.8.3 IsEqualAsFactorobjects (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

**Returns:** a boolean

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

### 3.8.4 AddIsEqualAsFactorobjects (for IsCapCategory, IsFunction)

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

**Returns:** nothing

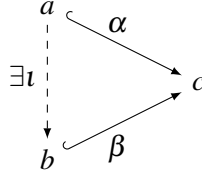
The arguments are a category  $C$  and a function  $F$ . This operation adds the given function  $F$  to the category for the basic operation `IsEqualAsFactorobjects`.  $F : (\alpha, \beta) \mapsto \text{IsEqualAsFactorobjects}(\alpha, \beta)$ .

### 3.8.5 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.

### 3.8.6 AddIsDominating (for IsCapCategory, IsFunction)

▷ `AddIsDominating(C, F)`

(operation)

**Returns:** nothing

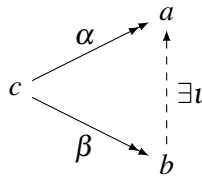
The arguments are a category  $C$  and a function  $F$ . This operation adds the given function  $F$  to the category for the basic operation `IsDominating`.  $F : (\alpha, \beta) \mapsto \text{IsDominating}(\alpha, \beta)$ .

### 3.8.7 IsCodominating (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `IsCodominating(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  $\iota : b \rightarrow a$  such that  $\iota \circ \beta \sim_{c,a} \alpha$ , otherwise the output is false.

### 3.8.8 AddIsCodominating (for IsCapCategory, IsFunction)

▷ `AddIsCodominating(C, F)`

(operation)

**Returns:** nothing

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

### 3.9 Identity Morphism and Composition of Morphisms

#### 3.9.1 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  $\text{id}_a$ .

#### 3.9.2 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.  $F : a \mapsto \text{id}_a$ .

#### 3.9.3 PreCompose (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

#### 3.9.4 PreCompose (for IsList)

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

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

#### 3.9.6 PostCompose (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

#### 3.9.7 PostCompose (for IsList)

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

### 3.9.8 AddPostCompose (for IsCapCategory, IsFunction)

▷ AddPostCompose( $\mathcal{C}$ ,  $F$ )

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

## 3.10 Well-Definedness of Morphisms

### 3.10.1 IsWellDefinedForMorphisms (for IsCapCategoryMorphism)

▷ IsWellDefinedForMorphisms( $\alpha$ )

(operation)

**Returns:** a boolean

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

### 3.10.2 AddIsWellDefinedForMorphisms (for IsCapCategory, IsFunction)

▷ AddIsWellDefinedForMorphisms( $\mathcal{C}$ ,  $F$ )

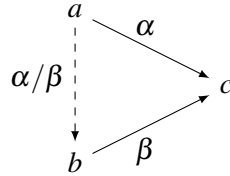
(operation)

**Returns:** nothing

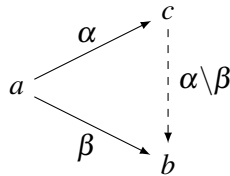
The arguments are a category  $\mathcal{C}$  and a function  $F$ . This operation adds the given function  $F$  to the category for the basic operation IsWellDefinedForMorphisms.  $F : \alpha \mapsto \text{IsWellDefinedForMorphisms}(\alpha)$ .

## 3.11 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.

### 3.11.1 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  $u$ .

### 3.11.2 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. The function  $F$  maps a pair  $(\iota, \tau)$  to a lift  $u$ .

### 3.11.3 ColiftAlongEpimorphism (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

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

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

### 3.11.4 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. The function  $F$  maps a pair  $(\varepsilon, \tau)$  to a lift  $u$ .

### 3.11.5 IsLiftableAlongMonomorphism (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

**Returns:** a boolean

The arguments are a monomorphism  $\iota : k \hookrightarrow a$  and a morphism  $\tau : t \rightarrow a$ . 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`.

### 3.11.6 AddIsLiftableAlongMonomorphism (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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

### 3.11.7 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`.

### 3.11.8 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.  $F : (\varepsilon, \tau) \mapsto \text{IsColiftableAlongEpimorphism}(\varepsilon, \tau)$ .

### 3.11.9 Lift (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ Lift(*alpha*, *beta*) (operation)

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

The arguments are two morphisms  $\alpha : a \rightarrow c$ ,  $\beta : b \rightarrow c$ . 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  $\beta \circ (\alpha/\beta) \sim_{a,c} \alpha$ .

### 3.11.10 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. The function *F* maps a pair  $(\alpha, \beta)$  to a lift  $\alpha/\beta$  if it exists, and to `fail` otherwise.

### 3.11.11 Colift (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ Colift(*alpha*, *beta*) (operation)

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

The arguments are two morphisms  $\alpha : a \rightarrow c$ ,  $\beta : a \rightarrow b$ . 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  $(\alpha \backslash \beta) \circ \alpha \sim_{a,b} \beta$ .

### 3.11.12 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. The function *F* maps a pair  $(\alpha, \beta)$  to a colift  $\alpha \backslash \beta$  if it exists, and to `fail` otherwise.



### 3.11.13 IsLiftable (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

**Returns:** a boolean

The arguments are two morphisms  $\alpha : a \rightarrow c$ ,  $\beta : b \rightarrow c$ . 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`.

### 3.11.14 AddIsLiftable (for IsCapCategory, IsFunction)

▷ `AddIsLiftable(C, F)` (operation)

**Returns:** nothing

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

### 3.11.15 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`.

### 3.11.16 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`.  $F : (\alpha, \beta) \mapsto \text{IsColiftable}(\alpha, \beta)$ .

## 3.12 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  $\alpha^{-1} \circ \alpha \sim_{a,a} \text{id}_a$ .

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

### 3.12.1 Inverse (for IsCapCategoryMorphism)

▷ `Inverse(alpha)` (operation)

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

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

### 3.12.2 AddInverse (for IsCapCategory, IsFunction)

▷ AddInverse( $C$ ,  $F$ ) (operation)

**Returns:** nothing

The arguments are a category  $C$  and a function  $F$ . This operation adds the given function  $F$  to the category for the basic operation Inverse.  $F : \alpha \mapsto \alpha^{-1}$ .

## 3.13 Tool functions for caches

### 3.13.1 IsEqualForCacheForMorphisms (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ IsEqualForCacheForMorphisms( $\phi$ ,  $\psi$ ) (operation)

**Returns:** true or false

Compares two objects in the cache

### 3.13.2 AddIsEqualForCacheForMorphisms (for IsCapCategory, IsFunction)

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

**Returns:** nothing

By default, CAP uses caches to store the values of Categorical operations. To get a value out of the cache, one needs to compare the input of a basic operation with its previous input. To compare morphisms in the category, 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, this function is used instead. A function  $F : a, b \mapsto \text{bool}$  is expected here. The output has to be true or false. Fail is not allowed in this context.

## 3.14 IsHomSetInhabited

### 3.14.1 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.

### 3.14.2 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.  $F : A, B \mapsto \text{IsHomSetInhabited}(A, B)$ .

### 3.15 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  $a, b \in C$ .

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

#### 3.15.2 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.  $F : (a, b) \mapsto H(a, b)$ .

#### 3.15.3 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  $H(\alpha, \beta)$ .

#### 3.15.4 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  $H(\alpha, \beta)$ .

#### 3.15.5 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`.  $F : (s, \alpha : a \rightarrow a', \beta : b \rightarrow b', r) \mapsto H(\alpha, \beta)$ .

### 3.15.6 DistinguishedObjectOfHomomorphismStructure (for IsCapCategory)

▷ `DistinguishedObjectOfHomomorphismStructure( $C$ )` (attribute)

**Returns:** an object in  $D$

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

### 3.15.7 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`.  $F : () \mapsto 1$ .

### 3.15.8 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.

### 3.15.9 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`.

$F : (\alpha : a \rightarrow a') \mapsto (v(\alpha) : 1 \rightarrow H(a, a'))$ .

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

▷ `InterpretMorphismFromDistinguishedObjectToHomomorphismStructureAsMorphism( $a, a', \text{iota}$ )` (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.

### 3.15.11 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.

$F : (a, a', \iota : 1 \rightarrow H(a, a')) \mapsto (v^{-1}(\iota) : a \rightarrow a')$ .

### 3.15.12 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  $i = 1, \dots, m$ . The output is either a list of morphisms  $X_j : B_j \rightarrow C_j$  for  $j = 1 \dots n$  solving the linear system defined by  $\alpha, \beta, \gamma$ , i.e.,  $\sum_{j=1}^n \alpha_{ij} \cdot X_j \cdot \beta_{ij} = \gamma_i$  for all  $i = 1 \dots m$ , or fail if no such solution exists.

### 3.15.13 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.

### 3.15.14 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  $\alpha, \beta$ .

### 3.15.15 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  $\alpha, \text{id}_b$ .

### 3.15.16 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  $\text{id}_a, \beta$ .

### 3.15.17 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  $a, b$ .

### 3.15.18 HomStructure (for IsCapCategoryMorphism)

▷ `HomStructure(arg)` (operation)

This is a convenience method for `InterpretMorphismAsMorphismFromDistinguishedObjectToHomomorphismStructure`.

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

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

This is a convenience method for `InterpretMorphismFromDistinguishedObjectToHomomorphismStructure`.

### 3.15.20 HomStructure (for IsCapCategory)

▷ `HomStructure(arg)` (operation)

This is a convenience method for `DistinguishedObjectOfHomomorphismStructure`.

### 3.15.21 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  $\text{CoefficientsOfMorphismWithGivenBasisOfExternalHom}(\alpha, L)$ .

### 3.15.22 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`.

### 3.15.23 CoefficientsOfMorphismWithGivenBasisOfExternalHom (for IsCapCategoryMorphism, IsList)

▷ `CoefficientsOfMorphismWithGivenBasisOfExternalHom(alpha, L)` (operation)

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

The arguments are a morphism  $\alpha : a \rightarrow b$  in a  $k$ -linear category  $C$  and a list  $L = \text{BasisOfExternalHom}(a, b)$ . The output is a list of coefficients of  $\alpha$  with respect to  $L$ .

### 3.15.24 AddCoefficientsOfMorphismWithGivenBasisOfExternalHom (for IsCapCategory, IsFunction)

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

**Returns:** nothing

The arguments are a category  $C$  and a function  $F$ . This operation adds the given function  $F$  to the category for the basic operation `CoefficientsOfMorphismWithGivenBasisOfExternalHom`.

### 3.15.25 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)`.

## 3.16 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:

### 3.16.1 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  $\phi_i$ .

### 3.16.2 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`. The function  $F$  maps  $(\phi, i \geq 1)$  to  $\phi_i$ .

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:

### 3.16.3 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  $\phi_i : A_i \rightarrow B$ .

### 3.16.4 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`. The function  $F$  maps  $(\phi, i \geq 1)$  to  $\phi_i$ .

### 3.16.5 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  $(\sigma_i)^{-1} : A_i \rightarrow A$ .

### 3.16.6 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`. The function  $F$  maps  $(\phi, i)$  to  $(\sigma_i)^{-1}$ .



### 3.16.7 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  $\sigma_i : A \rightarrow A_i$ .

### 3.16.8 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`. The function  $F$  maps  $(\phi, i)$  to  $(\sigma_i)$ .

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:

### 3.16.9 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  $\phi_i : A \rightarrow B_i$ .

### 3.16.10 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`. The function  $F$  maps  $(\phi, i \geq 1)$  to  $\phi_i$ .

### 3.16.11 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  $(\rho_i)^{-1} : B_i \rightarrow B$ .

### 3.16.12 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`. The function  $F$  maps  $(\phi, i)$  to  $(\rho_i)^{-1}$ .

### 3.16.13 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  $\rho_i : B \rightarrow B_i$ .

### 3.16.14 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`. The function  $F$  maps  $(\phi, i)$  to  $\rho_i$ .

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:

### 3.16.15 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  $\phi_i : A_i \rightarrow B_i$ .

### 3.16.16 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`. The function  $F$  maps  $(\phi, i \geq 1)$  to  $\phi_i$ .

### 3.16.17 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  $(\rho_i)^{-1} : B_i \rightarrow B$ .

### 3.16.18 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`. The function  $F$  maps  $(\phi, i)$  to  $(\rho_i)^{-1}$ .

### 3.16.19 `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  $\rho_i : B \rightarrow B_i$ .

### 3.16.20 `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`. The function  $F$  maps  $(\phi, i)$  to  $\rho_i$ .

### 3.16.21 `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  $(\sigma_i)^{-1} : A_i \rightarrow A$ .

### 3.16.22 `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`. The function  $F$  maps  $(\phi, i)$  to  $(\sigma_i)^{-1}$ .

### 3.16.23 `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  $\sigma_i : A \rightarrow A_i$ .

### 3.16.24 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. The function  $F$  maps  $(\phi, i)$  to  $(\sigma_i)$ .

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:

### 3.16.25 SimplifyEndo (for IsCapCategoryMorphism, IsObject)

▷ SimplifyEndo( $\phi_i, i$ ) (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  $\phi_i : A_i \rightarrow A_i$ .

### 3.16.26 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. The function  $F$  maps  $(\phi, i \geq 1)$  to  $\phi_i$ .

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

▷ SimplifyEndo\_IsoToInputObject( $\phi_i, i$ ) (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  $(\sigma_i)^{-1} : A_i \rightarrow A$ .

### 3.16.28 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. The function  $F$  maps  $(\phi, i)$  to  $(\sigma_i)^{-1}$ .

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

▷ SimplifyEndo\_IsoFromInputObject( $\phi_i, i$ ) (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  $\sigma_i : A \rightarrow A_i$ .

### 3.16.30 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. The function  $F$  maps  $(\phi, i)$  to  $(\sigma_i)$ .

### 3.16.31 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  $(\iota_A^\infty)^{-1} \circ \phi \circ \iota_A^\infty$ .

## 3.17 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

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

### 3.17.1 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  $\alpha' : A' \rightarrow B'$ .

### 3.17.2 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`. The function  $F$  maps  $\alpha$  to  $\alpha'$ .

### 3.17.3 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.

### 3.17.4 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`. The function  $F$  maps  $\alpha$  to  $\beta'$ .

### 3.17.5 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.

### 3.17.6 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`. The function  $F$  maps  $\alpha$  to  $\beta$ .

# 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  $\beta$ .

### 4.2 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 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`.  $F : \alpha \mapsto \text{id}_\alpha$ .

#### 4.2.3 HorizontalPreCompose (for IsCapCategoryTwoCell, IsCapCategoryTwoCell)

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

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

#### 4.2.5 HorizontalPostCompose (for IsCapCategoryTwoCell, IsCapCategoryTwoCell)

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

**Returns:** a 2-cell

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

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

#### 4.2.7 VerticalPreCompose (for IsCapCategoryTwoCell, IsCapCategoryTwoCell)

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

**Returns:** a 2-cell

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

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

#### 4.2.9 VerticalPostCompose (for IsCapCategoryTwoCell, IsCapCategoryTwoCell)

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

**Returns:** a 2-cell

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

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



## 4.3 Well-Definedness for 2-Cells

### 4.3.1 IsWellDefinedForTwoCells (for IsCapCategoryTwoCell)

▷ `IsWellDefinedForTwoCells(c)` (operation)

**Returns:** a boolean

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

### 4.3.2 AddIsWellDefinedForTwoCells (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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

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

### 5.3.2 AsCapCategory (for IsCapCategoryAsCatObject)

▷ `AsCapCategory(C)` (attribute)

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

## 5.4 Functors

Functors are morphisms in `Cat`, thus they have source and target which are categories. A multivariate functor can be constructed via a product category as source, a presheaf is constructed via the opposite category as source. However, the user can explicitly decide the arity of a functor (which will only have technical implications). Thus, it is for example possible to consider a functor  $A \times B \rightarrow C$  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 `sOnObjects`, and an operation  $\omega_3$  with the name `sOnMorphisms`. 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

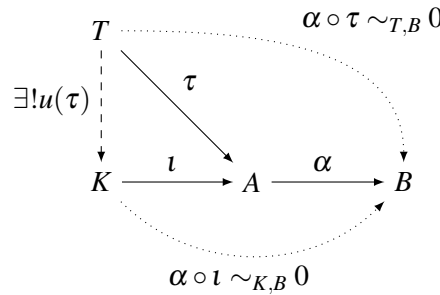
### 6.1 Kernel

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, IsCapCategoryMorphism)

- ▷ `KernelLift(alpha, tau)` (operation)  
**Returns:** a morphism in  $\text{Hom}(T, \text{KernelObject}(\alpha))$   
 The arguments are a morphism  $\alpha : A \rightarrow B$  and a test morphism  $\tau : T \rightarrow A$  satisfying  $\alpha \circ \tau \sim_{T,B} 0$ . The output is the morphism  $u(\tau) : T \rightarrow \text{KernelObject}(\alpha)$  given by the universal property of the kernel.

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

- ▷ `KernelLiftWithGivenKernelObject(alpha, tau, K)` (operation)  
**Returns:** a morphism in  $\text{Hom}(T, K)$   
 The arguments are a morphism  $\alpha : A \rightarrow B$ , a test morphism  $\tau : T \rightarrow A$  satisfying  $\alpha \circ \tau \sim_{T,B} 0$ , and an object  $K = \text{KernelObject}(\alpha)$ . The output is the morphism  $u(\tau) : T \rightarrow K$  given by the universal property of the kernel.

### 6.1.8 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.  $F : \alpha \mapsto \text{KernelObject}(\alpha)$ .

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

### 6.1.10 AddKernelEmbeddingWithGivenKernelObject (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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

### 6.1.11 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.  $F : \alpha \mapsto \mu$

### 6.1.12 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.  $F : (\alpha, K) \mapsto \mu$

### 6.1.13 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.  $F : (\alpha, \tau) \mapsto u(\tau)$ .

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

#### 6.1.15 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.

#### 6.1.16 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.

#### 6.1.17 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.

#### 6.1.18 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.

### 6.1.19 AddKernelObjectFunctorialWithGivenKernelObjects (for IsCapCategory, Is-Function)

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

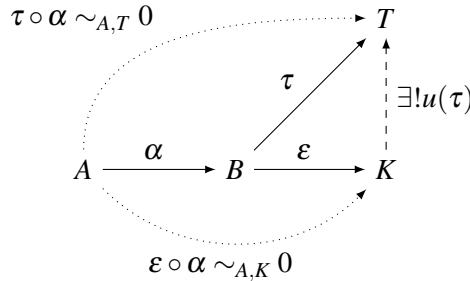
## 6.2 Cokernel

For a given morphism  $\alpha : A \rightarrow B$ , a cokernel of  $\alpha$  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*.

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, IsCapCategoryMorphism)

▷ CokernelColift(alpha, tau) (operation)

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

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

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

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

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

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

### 6.2.8 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  $\text{CokernelObject}$ .  $F : \alpha \mapsto K$ .

### 6.2.9 AddCokernelProjection (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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

### 6.2.10 AddCokernelProjectionWithGivenCokernelObject (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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

### 6.2.11 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.  $F : \alpha \mapsto \mu$

### 6.2.12 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.  $F : (\alpha, K) \mapsto \mu$

### 6.2.13 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 CokernelProjection.  $F : (\alpha, \tau) \mapsto u(\tau)$ .

### 6.2.14 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 CokernelProjection.  $F : (\alpha, \tau, K) \mapsto u(\tau)$ .

### 6.2.15 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.

### 6.2.16 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.

### 6.2.17 CokernelObjectFunctorialWithGivenCokernelObjects (for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject)

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

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

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

### 6.2.18 CokernelObjectFunctorialWithGivenCokernelObjects (for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject)

▷ `CokernelObjectFunctorialWithGivenCokernelObjects(s, alpha, mu, nu, alpha_prime, 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.

### 6.2.19 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`.  $F : (\text{CokernelObject}(\alpha), \alpha, \nu, \alpha', \text{CokernelObject}(\alpha')) \mapsto (\text{CokernelObject}(\alpha) \rightarrow \text{CokernelObject}(\alpha'))$ .

## 6.3 Zero Object

A zero object consists of three parts:

- an object  $Z$ ,
- a function  $u_{\text{in}}$  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 MorphismFromZeroObject (for IsCapCategoryObject)

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

### 6.3.4 MorphismIntoZeroObject (for IsCapCategoryObject)

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

### 6.3.5 UniversalMorphismFromZeroObject (for IsCapCategoryObject)

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



### 6.3.6 UniversalMorphismFromZeroObjectWithGivenZeroObject (for IsCapCategoryObject, IsCapCategoryObject)

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

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

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

### 6.3.7 UniversalMorphismIntoZeroObject (for IsCapCategoryObject)

▷ UniversalMorphismIntoZeroObject( $A$ ) (attribute)

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

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

### 6.3.8 UniversalMorphismIntoZeroObjectWithGivenZeroObject (for IsCapCategoryObject, IsCapCategoryObject)

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

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

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

### 6.3.9 IsomorphismFromZeroObjectToInitialObject (for IsCapCategory)

▷ IsomorphismFromZeroObjectToInitialObject( $C$ ) (attribute)

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

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

### 6.3.10 IsomorphismFromInitialObjectToZeroObject (for IsCapCategory)

▷ IsomorphismFromInitialObjectToZeroObject( $C$ ) (attribute)

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

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

### 6.3.11 IsomorphismFromZeroObjectToTerminalObject (for IsCapCategory)

▷ IsomorphismFromZeroObjectToTerminalObject( $C$ ) (attribute)

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

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

### 6.3.12 IsomorphismFromTerminalObjectToZeroObject (for IsCapCategory)

▷ IsomorphismFromTerminalObjectToZeroObject( $C$ ) (attribute)

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

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

### 6.3.13 AddZeroObject (for IsCapCategory, IsFunction)

▷ AddZeroObject( $C$ ,  $F$ ) (operation)

**Returns:** nothing

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

### 6.3.14 AddUniversalMorphismIntoZeroObject (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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

### 6.3.15 AddUniversalMorphismIntoZeroObjectWithGivenZeroObject (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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

### 6.3.16 AddUniversalMorphismFromZeroObject (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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

### 6.3.17 AddUniversalMorphismFromZeroObjectWithGivenZeroObject (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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

### 6.3.18 AddIsomorphismFromZeroObjectToInitialObject (for IsCapCategory, IsFunction)

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

**Returns:** nothing

The arguments are a category  $C$  and a function  $F$ . This operation adds the given function  $F$  to the category for the basic operation IsomorphismFromZeroObjectToInitialObject.  $F : () \mapsto (\text{ZeroObject} \rightarrow \text{InitialObject})$ .

### 6.3.19 AddIsomorphismFromInitialObjectToZeroObject (for IsCapCategory, IsFunction)

▷ AddIsomorphismFromInitialObjectToZeroObject( $C, F$ ) (operation)

**Returns:** nothing

The arguments are a category  $C$  and a function  $F$ . This operation adds the given function  $F$  to the category for the basic operation IsomorphismFromInitialObjectToZeroObject.  $F : () \mapsto (\text{InitialObject} \rightarrow \text{ZeroObject})$ .

### 6.3.20 AddIsomorphismFromZeroObjectToTerminalObject (for IsCapCategory, IsFunction)

▷ AddIsomorphismFromZeroObjectToTerminalObject( $C, F$ ) (operation)

**Returns:** nothing

The arguments are a category  $C$  and a function  $F$ . This operation adds the given function  $F$  to the category for the basic operation IsomorphismFromZeroObjectToTerminalObject.  $F : () \mapsto (\text{ZeroObject} \rightarrow \text{TerminalObject})$ .

### 6.3.21 AddIsomorphismFromTerminalObjectToZeroObject (for IsCapCategory, IsFunction)

▷ AddIsomorphismFromTerminalObjectToZeroObject( $C, F$ ) (operation)

**Returns:** nothing

The arguments are a category  $C$  and a function  $F$ . This operation adds the given function  $F$  to the category for the basic operation IsomorphismFromTerminalObjectToZeroObject.  $F : () \mapsto (\text{TerminalObject} \rightarrow \text{ZeroObject})$ .

### 6.3.22 ZeroObjectFunctorial (for IsCapCategory)

▷ ZeroObjectFunctorial( $C$ ) (attribute)

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

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

### 6.3.23 AddZeroObjectFunctorial (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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

## 6.4 Terminal Object

A terminal object consists of two parts:

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

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

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

#### 6.4.1 `TerminalObject` (for `IsCapCategory`)

▷ `TerminalObject(C)` (attribute)

**Returns:** an object

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

#### 6.4.2 `TerminalObject` (for `IsCapCategoryCell`)

▷ `TerminalObject(c)` (attribute)

**Returns:** an object

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

#### 6.4.3 `UniversalMorphismIntoTerminalObject` (for `IsCapCategoryObject`)

▷ `UniversalMorphismIntoTerminalObject(A)` (attribute)

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

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

#### 6.4.4 `UniversalMorphismIntoTerminalObjectWithGivenTerminalObject` (for `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 `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`.  $F : () \mapsto T$ .

#### 6.4.6 `AddUniversalMorphismIntoTerminalObject` (for `IsCapCategory`, `IsFunction`)

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

**Returns:** nothing

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

### 6.4.7 AddUniversalMorphismIntoTerminalObjectWithGivenTerminalObject (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismIntoTerminalObjectWithGivenTerminalObject( $C, F$ ) (operation)

**Returns:** nothing

The arguments are a category  $C$  and a function  $F$ . This operation adds the given function  $F$  to the category for the basic operation UniversalMorphismIntoTerminalObjectWithGivenTerminalObject.  $F : (A, T) \mapsto u(A)$ .

### 6.4.8 TerminalObjectFunctorial (for IsCapCategory)

▷ TerminalObjectFunctorial( $C$ ) (attribute)

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

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

### 6.4.9 AddTerminalObjectFunctorial (for IsCapCategory, IsFunction)

▷ AddTerminalObjectFunctorial( $C, F$ ) (operation)

**Returns:** nothing

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

## 6.5 Initial Object

An initial object consists of two parts:

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

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

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

### 6.5.1 InitialObject (for IsCapCategory)

▷ InitialObject( $C$ ) (attribute)

**Returns:** an object

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

### 6.5.2 InitialObject (for IsCapCategoryCell)

▷ InitialObject( $c$ ) (attribute)

**Returns:** an object

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

### 6.5.3 UniversalMorphismFromInitialObject (for IsCapCategoryObject)

▷ UniversalMorphismFromInitialObject( $A$ ) (attribute)

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

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

### 6.5.4 UniversalMorphismFromInitialObjectWithGivenInitialObject (for IsCapCategoryObject, IsCapCategoryObject)

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

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

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

### 6.5.5 AddInitialObject (for IsCapCategory, IsFunction)

▷ AddInitialObject( $C, F$ ) (operation)

**Returns:** nothing

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

### 6.5.6 AddUniversalMorphismFromInitialObject (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismFromInitialObject( $C, F$ ) (operation)

**Returns:** nothing

The arguments are a category  $C$  and a function  $F$ . This operation adds the given function  $F$  to the category for the basic operation UniversalMorphismFromInitialObject.  $F : A \mapsto u(A)$ .

### 6.5.7 AddUniversalMorphismFromInitialObjectWithGivenInitialObject (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismFromInitialObjectWithGivenInitialObject( $C, F$ ) (operation)

**Returns:** nothing

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

### 6.5.8 InitialObjectFunctorial (for IsCapCategory)

▷ InitialObjectFunctorial( $C$ ) (attribute)

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

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

### 6.5.9 AddInitialObjectFunctorial (for IsCapCategory, IsFunction)

▷ AddInitialObjectFunctorial( $C, F$ ) (operation)

**Returns:** nothing

The arguments are a category  $C$  and a function  $F$ . This operation adds the given function  $F$  to the category for the basic operation InitialObjectFunctorial.  $F : () \rightarrow (I \rightarrow I)$ .

## 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:

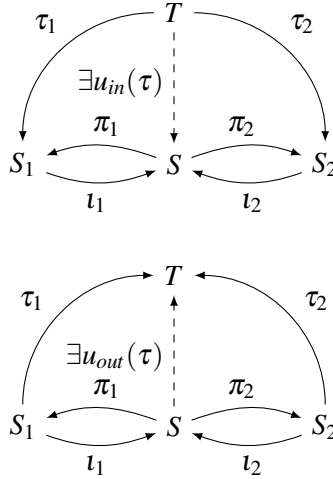
- an object  $S$ ,
- a list of morphisms  $\pi = (\pi_i : S \rightarrow S_i)_{i=1\dots n}$ ,
- a list of morphisms  $\iota = (\iota_i : S_i \rightarrow S)_{i=1\dots n}$ ,
- a dependent function  $u_{\text{in}}$  mapping every list  $\tau = (\tau_i : T \rightarrow S_i)_{i=1\dots n}$  to a morphism  $u_{\text{in}}(\tau)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

▷ DirectSumOp( $D$ ,  $method\_selection\_object$ )

(operation)

**Returns:** an object

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

### 6.6.2 ProjectionInFactorOfDirectSum (for IsList, IsInt)

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

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

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

### 6.6.3 ProjectionInFactorOfDirectSumOp (for IsList, IsInt, IsCapCategoryObject)

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

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

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

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

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

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

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

### 6.6.5 InjectionOfCofactorOfDirectSum (for IsList, IsInt)

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

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

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

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

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

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

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

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

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

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

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

### 6.6.8 UniversalMorphismIntoDirectSum

▷ `UniversalMorphismIntoDirectSum(arg)` (function)

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

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



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

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

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

▷ UniversalMorphismIntoDirectSumOp( $D$ ,  $\tau$ ,  $\text{method\_selection\_object}$ ) (operation)

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

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

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

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

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

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

### 6.6.11 UniversalMorphismFromDirectSum

▷ UniversalMorphismFromDirectSum( $\text{arg}$ ) (function)

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

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

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

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

### 6.6.12 UniversalMorphismFromDirectSumOp (for IsList, IsList, IsCapCategoryObject)

▷ UniversalMorphismFromDirectSumOp( $D$ ,  $\tau$ ,  $\text{method\_selection\_object}$ ) (operation)

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

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

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

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

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

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

### 6.6.14 IsomorphismFromDirectSumToDirectProduct (for IsList)

▷ IsomorphismFromDirectSumToDirectProduct( $D$ ) (operation)

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

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

### 6.6.15 IsomorphismFromDirectSumToDirectProductOp (for IsList, IsCapCategory-Object)

▷ IsomorphismFromDirectSumToDirectProductOp( $D$ ,  $\text{method\_selection\_object}$ ) (operation)

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

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

### 6.6.16 IsomorphismFromDirectProductToDirectSum (for IsList)

▷ IsomorphismFromDirectProductToDirectSum( $D$ ) (operation)

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

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

### 6.6.17 IsomorphismFromDirectProductToDirectSumOp (for IsList, IsCapCategory-Object)

▷ IsomorphismFromDirectProductToDirectSumOp( $D$ ,  $\text{method\_selection\_object}$ ) (operation)

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

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

### 6.6.18 IsomorphismFromDirectSumToCoproduct (for IsList)

▷ IsomorphismFromDirectSumToCoproduct( $D$ ) (operation)

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

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

### 6.6.19 IsomorphismFromDirectSumToCoproductOp (for IsList, IsCapCategoryObject)

▷ IsomorphismFromDirectSumToCoproductOp( $D$ ,  $method\_selection\_object$ ) (operation)

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

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

### 6.6.20 IsomorphismFromCoproductToDirectSum (for IsList)

▷ IsomorphismFromCoproductToDirectSum( $D$ ) (operation)

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

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

### 6.6.21 IsomorphismFromCoproductToDirectSumOp (for IsList, IsCapCategoryObject)

▷ IsomorphismFromCoproductToDirectSumOp( $D$ ,  $method\_selection\_object$ ) (operation)

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

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

### 6.6.22 MorphismBetweenDirectSums (for IsList)

▷ MorphismBetweenDirectSums( $M$ ) (operation)

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

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

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

### 6.6.23 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.  $F : (\bigoplus_{i=1}^m A_i, M, \bigoplus_{j=1}^n B_j) \mapsto (\bigoplus_{i=1}^m A_i \rightarrow \bigoplus_{j=1}^n B_j)$ .

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

▷ MorphismBetweenDirectSumsOp( $M$ ,  $m$ ,  $n$ ,  $method\_selection\_morphism$ ) (operation)

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

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

#### 6.6.25 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  $A \rightarrow S_k$ .

#### 6.6.26 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  $S_k \rightarrow A$ .

#### 6.6.27 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.  $F : (\alpha : A \rightarrow S, D, k) \mapsto (A \rightarrow S_k)$ .

#### 6.6.28 AddComponentOfMorphismFromDirectSum (for IsCapCategory, IsFunction)

▷ AddComponentOfMorphismFromDirectSum(C, F) (operation)

**Returns:** nothing

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

#### 6.6.29 AddProjectionInFactorOfDirectSum (for IsCapCategory, IsFunction)

▷ AddProjectionInFactorOfDirectSum(C, F) (operation)

**Returns:** nothing

The arguments are a category  $C$  and a function  $F$ . This operation adds the given function  $F$  to the category for the basic operation ProjectionInFactorOfDirectSum.  $F : (D, k) \mapsto \pi_k$ .

#### 6.6.30 AddProjectionInFactorOfDirectSumWithGivenDirectSum (for IsCapCategory, IsFunction)

▷ AddProjectionInFactorOfDirectSumWithGivenDirectSum(C, F) (operation)

**Returns:** nothing

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

### 6.6.31 AddInjectionOfCofactorOfDirectSum (for IsCapCategory, IsFunction)

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

**Returns:** nothing

The arguments are a category  $C$  and a function  $F$ . This operation adds the given function  $F$  to the category for the basic operation `InjectionOfCofactorOfDirectSum`.  $F : (D, k) \mapsto \iota_k$ .

### 6.6.32 AddInjectionOfCofactorOfDirectSumWithGivenDirectSum (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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

### 6.6.33 AddUniversalMorphismIntoDirectSum (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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

### 6.6.34 AddUniversalMorphismIntoDirectSumWithGivenDirectSum (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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

### 6.6.35 AddUniversalMorphismFromDirectSum (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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

### 6.6.36 AddUniversalMorphismFromDirectSumWithGivenDirectSum (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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

### 6.6.37 AddIsomorphismFromDirectSumToDirectProduct (for IsCapCategory, IsFunction)

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

**Returns:** nothing

The arguments are a category  $C$  and a function  $F$ . This operation adds the given function  $F$  to the category for the basic operation `IsomorphismFromDirectSumToDirectProduct`.  $F : D \mapsto (\bigoplus_{i=1}^n S_i \rightarrow \prod_{i=1}^n S_i)$ .

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

### 6.6.39 AddIsomorphismFromDirectSumToCoproduct (for IsCapCategory, IsFunction)

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

**Returns:** nothing

The arguments are a category  $C$  and a function  $F$ . This operation adds the given function  $F$  to the category for the basic operation `IsomorphismFromDirectSumToCoproduct`.  $F : D \mapsto (\bigoplus_{i=1}^n S_i \rightarrow \bigsqcup_{i=1}^n S_i)$ .

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

### 6.6.41 AddDirectSum (for IsCapCategory, IsFunction)

▷ `AddDirectSum(C, F)` (operation)

**Returns:** nothing

The arguments are a category  $C$  and a function  $F$ . This operation adds the given function  $F$  to the category for the basic operation `DirectSum`.  $F : D \mapsto \bigoplus_{i=1}^n S_i$ .

### 6.6.42 DirectSumFunctorial (for IsList)

▷ `DirectSumFunctorial(L)`

(operation)

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

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

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

▷ `DirectSumFunctorialWithGivenDirectSums(d_1, L, d_2)`

(operation)

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

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

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

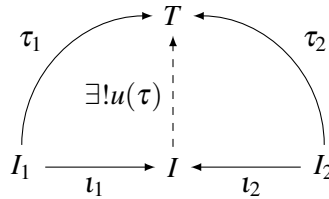
## 6.7 Coproduct

For 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)` (attribute)  
**Returns:** an object  
 The argument is a list of objects  $D = (I_1, \dots, I_n)$ . The output is the coproduct  $\bigsqcup_{i=1}^n I_i$ .

### 6.7.2 Coproduct (for IsCapCategoryObject, IsCapCategoryObject)

- ▷ `Coproduct(I1, I2)` (operation)  
**Returns:** an object  
 This is a convenience method. The arguments are two objects  $I_1, I_2$ . The output is the coproduct  $I_1 \sqcup I_2$ .

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

- ▷ `Coproduct(I1, I2)` (operation)  
**Returns:** an object  
 This is a convenience method. The arguments are three objects  $I_1, I_2, I_3$ . The output is the coproduct  $I_1 \sqcup I_2 \sqcup I_3$ .

### 6.7.4 CoproductOp (for IsList, IsCapCategoryObject)

- ▷ `CoproductOp(D, method_selection_object)` (operation)  
**Returns:** an object  
 The arguments are a list of objects  $D = (I_1, \dots, I_n)$  and a method selection object. The output is the coproduct  $\bigsqcup_{i=1}^n I_i$ .

### 6.7.5 InjectionOfCofactorOfCoproduct (for IsList, IsInt)

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

### 6.7.6 InjectionOfCofactorOfCoproductOp (for IsList, IsInt, IsCapCategoryObject)

- ▷ `InjectionOfCofactorOfCoproductOp(D, k, method_selection_object)` (operation)  
**Returns:** a morphism in  $\text{Hom}(I_k, \bigsqcup_{i=1}^n I_i)$   
 The arguments are a list of objects  $D = (I_1, \dots, I_n)$ , an integer  $k$ , and a method selection object. The output is the  $k$ -th injection  $\iota_k : I_k \rightarrow \bigsqcup_{i=1}^n I_i$ .

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

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



### 6.7.8 UniversalMorphismFromCoproduct

▷ `UniversalMorphismFromCoproduct(arg)` (function)

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

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

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

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

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

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

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

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

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

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

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

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

### 6.7.11 AddCoproduct (for IsCapCategory, IsFunction)

▷ `AddCoproduct(C, F)` (operation)

**Returns:** nothing

The arguments are a category  $C$  and a function  $F$ . This operation adds the given function  $F$  to the category for the basic operation `Coproduct`.  $F : ((I_1, \dots, I_n)) \mapsto I$ .

### 6.7.12 AddInjectionOfCofactorOfCoproduct (for IsCapCategory, IsFunction)

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

**Returns:** nothing

The arguments are a category  $C$  and a function  $F$ . This operation adds the given function  $F$  to the category for the basic operation `InjectionOfCofactorOfCoproduct`.  $F : ((I_1, \dots, I_n), i) \mapsto I_i$ .

### 6.7.13 AddInjectionOfCofactorOfCoproductWithGivenCoproduct (for IsCapCategory, IsFunction)

▷ AddInjectionOfCofactorOfCoproductWithGivenCoproduct( $C, F$ ) (operation)

**Returns:** nothing

The arguments are a category  $C$  and a function  $F$ . This operation adds the given function  $F$  to the category for the basic operation InjectionOfCofactorOfCoproductWithGivenCoproduct.  $F : ((I_1, \dots, I_n), i, I) \mapsto l_i$ .

### 6.7.14 AddUniversalMorphismFromCoproduct (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismFromCoproduct( $C, F$ ) (operation)

**Returns:** nothing

The arguments are a category  $C$  and a function  $F$ . This operation adds the given function  $F$  to the category for the basic operation UniversalMorphismFromCoproduct.  $F : ((I_1, \dots, I_n), \tau) \mapsto u(\tau)$ .

### 6.7.15 AddUniversalMorphismFromCoproductWithGivenCoproduct (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismFromCoproductWithGivenCoproduct( $C, F$ ) (operation)

**Returns:** nothing

The arguments are a category  $C$  and a function  $F$ . This operation adds the given function  $F$  to the category for the basic operation UniversalMorphismFromCoproductWithGivenCoproduct.  $F : ((I_1, \dots, I_n), \tau, I) \mapsto u(\tau)$ .

### 6.7.16 CoproductFunctorial (for IsList)

▷ CoproductFunctorial( $L$ ) (operation)

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

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

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

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

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

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

### 6.7.18 AddCoproductFunctorialWithGivenCoproducts (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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

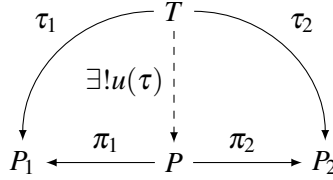
## 6.8 Direct Product

For 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 DirectProductOp (for IsList, IsCapCategoryObject)

▷ DirectProductOp( $D$ ) (operation)

**Returns:** an object

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

### 6.8.2 ProjectionInFactorOfDirectProduct (for IsList, IsInt)

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

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

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

### 6.8.3 ProjectionInFactorOfDirectProductOp (for IsList, IsInt, IsCapCategoryObject)

▷ ProjectionInFactorOfDirectProductOp( $D, k, \text{method\_selection\_object}$ ) (operation)

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

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

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

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

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

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

#### 6.8.5 UniversalMorphismIntoDirectProduct

▷ `UniversalMorphismIntoDirectProduct(arg)` (function)

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

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

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

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

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

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

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

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

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

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

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

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

#### 6.8.8 AddDirectProduct (for IsCapCategory, IsFunction)

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

**Returns:** nothing

The arguments are a category  $C$  and a function  $F$ . This operation adds the given function  $F$  to the category for the basic operation `DirectProduct`.  $F : ((P_1, \dots, P_n)) \mapsto P$

### 6.8.9 AddProjectionInFactorOfDirectProduct (for IsCapCategory, IsFunction)

▷ AddProjectionInFactorOfDirectProduct( $C, F$ ) (operation)

**Returns:** nothing

The arguments are a category  $C$  and a function  $F$ . This operation adds the given function  $F$  to the category for the basic operation ProjectionInFactorOfDirectProduct.  $F : ((P_1, \dots, P_n), k) \mapsto \pi_k$

### 6.8.10 AddProjectionInFactorOfDirectProductWithGivenDirectProduct (for IsCapCategory, IsFunction)

▷ AddProjectionInFactorOfDirectProductWithGivenDirectProduct( $C, F$ ) (operation)

**Returns:** nothing

The arguments are a category  $C$  and a function  $F$ . This operation adds the given function  $F$  to the category for the basic operation ProjectionInFactorOfDirectProductWithGivenDirectProduct.  $F : ((P_1, \dots, P_n), k, P) \mapsto \pi_k$

### 6.8.11 AddUniversalMorphismIntoDirectProduct (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismIntoDirectProduct( $C, F$ ) (operation)

**Returns:** nothing

The arguments are a category  $C$  and a function  $F$ . This operation adds the given function  $F$  to the category for the basic operation UniversalMorphismIntoDirectProduct.  $F : ((P_1, \dots, P_n), \tau) \mapsto u(\tau)$

### 6.8.12 AddUniversalMorphismIntoDirectProductWithGivenDirectProduct (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismIntoDirectProductWithGivenDirectProduct( $C, F$ ) (operation)

**Returns:** nothing

The arguments are a category  $C$  and a function  $F$ . This operation adds the given function  $F$  to the category for the basic operation UniversalMorphismIntoDirectProductWithGivenDirectProduct.  $F : ((P_1, \dots, P_n), \tau, P) \mapsto u(\tau)$

### 6.8.13 DirectProductFunctorial (for IsList)

▷ DirectProductFunctorial( $L$ ) (operation)

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

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

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

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

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

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

### 6.8.15 AddDirectProductFunctorialWithGivenDirectProducts (for IsCapCategory, IsFunction)

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

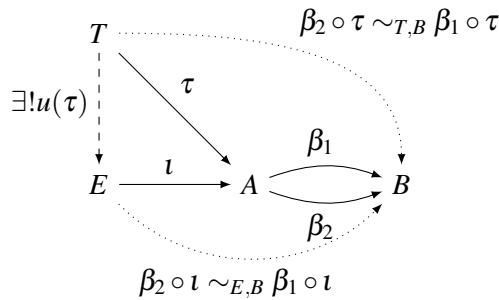
## 6.9 Equalizer

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 two different ways to use this method:

- 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 IsList, IsCapCategoryMorphism)

▷  $\text{EqualizerOp}(D, \text{method\_selection\_morphism})$  (operation)

**Returns:** an object

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

### 6.9.3 EmbeddingOfEqualizer (for IsList)

▷  $\text{EmbeddingOfEqualizer}(D)$  (operation)

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

The arguments are a list of morphisms  $D = (\beta_i : A \rightarrow B)_{i=1\dots n}$ . The output is the equalizer embedding  $\iota : \text{Equalizer}(D) \rightarrow A$ .

### 6.9.4 EmbeddingOfEqualizerOp (for IsList, IsCapCategoryMorphism)

▷  $\text{EmbeddingOfEqualizerOp}(D, \text{method\_selection\_morphism})$  (operation)

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

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

### 6.9.5 EmbeddingOfEqualizerWithGivenEqualizer (for IsList, IsCapCategoryObject)

▷  $\text{EmbeddingOfEqualizerWithGivenEqualizer}(D, E)$  (operation)

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

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

### 6.9.6 MorphismFromEqualizerToSink (for IsList)

▷  $\text{MorphismFromEqualizerToSink}(D)$  (operation)

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

The arguments are a list of morphisms  $D = (\beta_i : A \rightarrow B)_{i=1\dots n}$ . The output is the composition  $\mu : \text{Equalizer}(D) \rightarrow B$  of the embedding  $\iota : \text{Equalizer}(D) \rightarrow A$  and  $\beta_1$ .

### 6.9.7 MorphismFromEqualizerToSinkOp (for IsList, IsCapCategoryMorphism)

▷  $\text{MorphismFromEqualizerToSinkOp}(D, \text{method\_selection\_morphism})$  (operation)

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

The arguments are a list of morphisms  $D = (\beta_i : A \rightarrow B)_{i=1\dots n}$  and a morphism for method selection. The output is the composition  $\mu : \text{Equalizer}(D) \rightarrow B$  of the embedding  $\iota : \text{Equalizer}(D) \rightarrow A$  and  $\beta_1$ .

### 6.9.8 MorphismFromEqualizerToSinkWithGivenEqualizer (for IsList, IsCapCategoryObject)

▷ MorphismFromEqualizerToSinkWithGivenEqualizer( $D, E$ ) (operation)

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

The arguments are a list of morphisms  $D = (\beta_i : A \rightarrow B)_{i=1\dots n}$  and an object  $E = \text{Equalizer}(D)$ . The output is the composition  $\mu : E \rightarrow B$  of the embedding  $\iota : E \rightarrow A$  and  $\beta_1$ .

### 6.9.9 UniversalMorphismIntoEqualizer (for IsList, IsCapCategoryMorphism)

▷ UniversalMorphismIntoEqualizer( $D, \tau$ ) (operation)

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

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

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

▷ UniversalMorphismIntoEqualizerWithGivenEqualizer( $D, \tau, E$ ) (operation)

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

The arguments are a list of morphisms  $D = (\beta_i : A \rightarrow B)_{i=1\dots n}$ , 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  $E = \text{Equalizer}(D)$ . The output is the morphism  $u(\tau) : T \rightarrow E$  given by the universal property of the equalizer.

### 6.9.11 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.  $F : ((\beta_i : A \rightarrow B)_{i=1\dots n}) \mapsto E$

### 6.9.12 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.  $F : ((\beta_i : A \rightarrow B)_{i=1\dots n}, k) \mapsto \iota$

### 6.9.13 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.  $F : ((\beta_i : A \rightarrow B)_{i=1\dots n}, E) \mapsto \iota$



### 6.9.14 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.  $F : ((\beta_i : P \rightarrow B)_{i=1\dots n}) \mapsto \mu$

### 6.9.15 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.  $F : ((\beta_i : P \rightarrow B)_{i=1\dots n}, E) \mapsto \mu$

### 6.9.16 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.  $F : ((\beta_i : A \rightarrow B)_{i=1\dots n}, \tau) \mapsto u(\tau)$

### 6.9.17 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.  $F : ((\beta_i : A \rightarrow B)_{i=1\dots n}, \tau, E) \mapsto u(\tau)$

### 6.9.18 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.

### 6.9.19 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.

### 6.9.20 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`.  $F : (\text{Equalizer}((\beta_i)_{i=1\dots n}), (\beta_i : A \rightarrow B)_{i=1\dots n}, \mu : A \rightarrow A', (\beta'_i : A' \rightarrow B')_{i=1\dots n}, \text{Equalizer}((\beta'_i)_{i=1\dots n})) \mapsto (\text{Equalizer}((\beta_i)_{i=1\dots n}) \rightarrow \text{Equalizer}((\beta'_i)_{i=1\dots n}))$

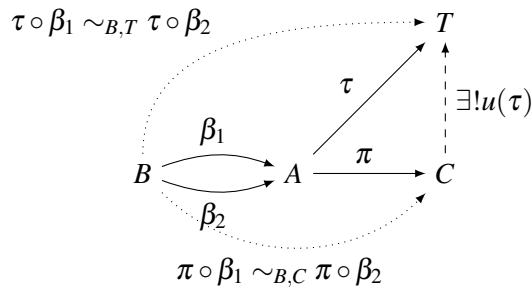
## 6.10 Coequalizer

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 two different ways to use this method:

- 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 IsList, IsCapCategoryMorphism)

▷  $\text{CoequalizerOp}(D, \text{method\_selection\_morphism})$  (operation)

**Returns:** an object

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

### 6.10.3 ProjectionOntoCoequalizer (for IsList)

▷  $\text{ProjectionOntoCoequalizer}(D)$  (operation)

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

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

### 6.10.4 ProjectionOntoCoequalizerOp (for IsList, IsCapCategoryMorphism)

▷  $\text{ProjectionOntoCoequalizerOp}(D, \text{method\_selection\_morphism})$  (operation)

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

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

### 6.10.5 ProjectionOntoCoequalizerWithGivenCoequalizer (for IsList, IsCapCategory-Object)

▷  $\text{ProjectionOntoCoequalizerWithGivenCoequalizer}(D, C)$  (operation)

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

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

### 6.10.6 MorphismFromSourceToCoequalizer (for IsList)

▷  $\text{MorphismFromSourceToCoequalizer}(D)$  (operation)

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

The arguments are a list of morphisms  $D = (\beta_i : B \rightarrow A)_{i=1\dots n}$ . The output is the composition  $\mu : B \rightarrow \text{Coequalizer}(D)$  of  $\beta_1$  and the projection  $\pi : A \rightarrow \text{Coequalizer}(D)$ .

### 6.10.7 MorphismFromSourceToCoequalizerOp (for IsList, IsCapCategoryMorphism)

▷  $\text{MorphismFromSourceToCoequalizerOp}(D, \text{method\_selection\_morphism})$  (operation)

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

The arguments are a list of morphisms  $D = (\beta_i : B \rightarrow A)_{i=1\dots n}$  and a morphism for method selection. The output is the composition  $\mu : B \rightarrow \text{Coequalizer}(D)$  of  $\beta_1$  and the projection  $\pi : A \rightarrow \text{Coequalizer}(D)$ .

#### 6.10.8 MorphismFromSourceToCoequalizerWithGivenCoequalizer (for IsList, IsCapCategoryObject)

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

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

The arguments are a list of morphisms  $D = (\beta_i : B \rightarrow A)_{i=1\dots n}$  and an object  $C = \text{Coequalizer}(D)$ . The output is the composition  $\mu : B \rightarrow C$  of  $\beta_1$  and the projection  $\pi : A \rightarrow C$ .

#### 6.10.9 UniversalMorphismFromCoequalizer (for IsList, IsCapCategoryMorphism)

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

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

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

#### 6.10.10 UniversalMorphismFromCoequalizerWithGivenCoequalizer (for IsList, IsCapCategoryMorphism, IsCapCategoryObject)

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

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

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

#### 6.10.11 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`.  $F : ((\beta_i : B \rightarrow A)_{i=1\dots n}) \mapsto C$

#### 6.10.12 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`.  $F : ((\beta_i : B \rightarrow A)_{i=1\dots n}, k) \mapsto \pi$

#### 6.10.13 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`.  $F : ((\beta_i : B \rightarrow A)_{i=1\dots n}, C) \mapsto \pi$

#### 6.10.14 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`.  $F : ((\beta_i : B \rightarrow A)_{i=1\dots n}) \mapsto \mu$

#### 6.10.15 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`.  $F : ((\beta_i : B \rightarrow A)_{i=1\dots n}, C) \mapsto \mu$

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

#### 6.10.17 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`.  $F : ((\beta_i : B \rightarrow A)_{i=1\dots n}, \tau, C) \mapsto u(\tau)$

#### 6.10.18 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  $L_s = (\beta_i : B \rightarrow A)_{i=1\dots n}$ , 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  $i = 1, \dots, n$ . 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.

### 6.10.19 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  $Ls = (\beta_i : B \rightarrow A)_{i=1\dots n}$ , 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  $i = 1, \dots, n$ , 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.

### 6.10.20 AddCoequalizerFunctorialWithGivenCoequalizers (for IsCapCategory, IsFunction)

▷ AddCoequalizerFunctorialWithGivenCoequalizers( $\mathcal{C}, F$ ) (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 CoequalizerFunctorialWithGivenCoequalizers.  $F : (\text{Coequalizer}((\beta_i)_{i=1}^n), (\beta_i : B \rightarrow A)_{i=1\dots n}, \mu : A \rightarrow A', (\beta'_i : B' \rightarrow A')_{i=1\dots n}, \text{Coequalizer}((\beta'_i)_{i=1}^n)) \mapsto (\text{Coequalizer}((\beta_i)_{i=1}^n) \rightarrow \text{Coequalizer}((\beta'_i)_{i=1}^n))$

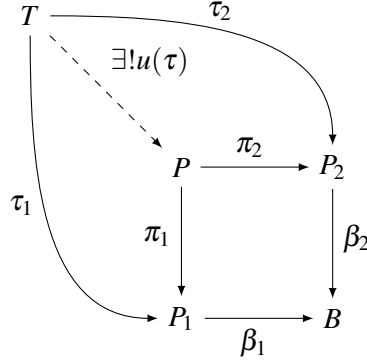
## 6.11 Fiber Product

For an integer  $n \geq 1$  and a given list of morphisms  $D = (\beta_i : P_i \rightarrow B)_{i=1\dots n}$ , 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 IsomorphismFromFiberProductToKernelOfDiagonalDifference (for IsList)

▷ `IsomorphismFromFiberProductToKernelOfDiagonalDifference(D)` (operation)

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

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

### 6.11.2 IsomorphismFromFiberProductToKernelOfDiagonalDifferenceOp (for IsList, IsCapCategoryMorphism)

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

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

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

### 6.11.3 AddIsomorphismFromFiberProductToKernelOfDiagonalDifference (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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

### 6.11.4 IsomorphismFromKernelOfDiagonalDifferenceToFiberProduct (for IsList)

▷ `IsomorphismFromKernelOfDiagonalDifferenceToFiberProduct(D)` (operation)

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

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

### 6.11.5 IsomorphismFromKernelOfDiagonalDifferenceToFiberProductOp (for IsList, IsCapCategoryMorphism)

▷ IsomorphismFromKernelOfDiagonalDifferenceToFiberProductOp( $D$ ) (operation)

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

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

### 6.11.6 AddIsomorphismFromKernelOfDiagonalDifferenceToFiberProduct (for IsCapCategory, IsFunction)

▷ AddIsomorphismFromKernelOfDiagonalDifferenceToFiberProduct( $C, F$ ) (operation)

**Returns:** nothing

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

### 6.11.7 IsomorphismFromFiberProductToEqualizerOfDirectProductDiagram (for IsList)

▷ IsomorphismFromFiberProductToEqualizerOfDirectProductDiagram( $D$ ) (operation)

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

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

### 6.11.8 IsomorphismFromFiberProductToEqualizerOfDirectProductDiagramOp (for IsList, IsCapCategoryMorphism)

▷ IsomorphismFromFiberProductToEqualizerOfDirectProductDiagramOp( $D, \text{method\_selection\_morphism}$ ) (operation)

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

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

### 6.11.9 AddIsomorphismFromFiberProductToEqualizerOfDirectProductDiagram (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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



### 6.11.10 IsomorphismFromEqualizerOfDirectProductDiagramToFiberProduct (for IsList)

▷ IsomorphismFromEqualizerOfDirectProductDiagramToFiberProduct( $D$ ) (operation)

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

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

### 6.11.11 IsomorphismFromEqualizerOfDirectProductDiagramToFiberProductOp (for IsList, IsCapCategoryMorphism)

▷ IsomorphismFromEqualizerOfDirectProductDiagramToFiberProductOp( $D$ ) (operation)

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

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

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

### 6.11.13 DirectSumDiagonalDifference (for IsList)

▷ DirectSumDiagonalDifference( $D$ ) (operation)

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

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

### 6.11.14 DirectSumDiagonalDifferenceOp (for IsList, IsCapCategoryMorphism)

▷ DirectSumDiagonalDifferenceOp( $D, \text{method\_selection\_morphism}$ ) (operation)

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

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

### 6.11.15 AddDirectSumDiagonalDifference (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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

### 6.11.16 FiberProductEmbeddingInDirectSum (for IsList)

▷ `FiberProductEmbeddingInDirectSum(D)` (operation)

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

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

### 6.11.17 FiberProductEmbeddingInDirectSumOp (for IsList, IsCapCategoryMorphism)

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

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

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

### 6.11.18 AddFiberProductEmbeddingInDirectSum (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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

### 6.11.19 FiberProduct

▷ `FiberProduct(arg)` (function)

**Returns:** an object

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

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

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

### 6.11.20 FiberProductOp (for IsList, IsCapCategoryMorphism)

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

**Returns:** an object

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

### 6.11.21 ProjectionInFactorOffFiberProduct (for IsList, IsInt)

▷ `ProjectionInFactorOffFiberProduct(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  $\pi_k : \text{FiberProduct}(D) \rightarrow P_k$ .

### 6.11.22 ProjectionInFactorOffFiberProductOp (for IsList, IsInt, IsCapCategoryMorphism)

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

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

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

### 6.11.23 ProjectionInFactorOffFiberProductWithGivenFiberProduct (for IsList, IsInt, IsCapCategoryObject)

▷ `ProjectionInFactorOffFiberProductWithGivenFiberProduct(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  $\pi_k : P \rightarrow P_k$ .

### 6.11.24 MorphismFromFiberProductToSink (for IsList)

▷ `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  $\beta_1$ .

### 6.11.25 MorphismFromFiberProductToSinkOp (for IsList, IsCapCategoryMorphism)

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

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

The arguments are a list of morphisms  $D = (\beta_i : P_i \rightarrow B)_{i=1\dots n}$  and a morphism for method selection. 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  $\beta_1$ .

### 6.11.26 MorphismFromFiberProductToSinkWithGivenFiberProduct (for IsList, IsCapCategoryObject)

▷ `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  $\beta_1$ .

### 6.11.27 UniversalMorphismIntoFiberProduct

▷ `UniversalMorphismIntoFiberProduct(arg)` (function)

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

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

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

### 6.11.28 UniversalMorphismIntoFiberProductOp (for IsList, IsList, IsCapCategory-Morphism)

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

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

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

### 6.11.29 UniversalMorphismIntoFiberProductWithGivenFiberProduct (for IsList, IsList, IsCapCategoryObject)

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

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

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

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

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

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

### 6.11.33 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.  $F : ((\beta_i : P_i \rightarrow B)_{i=1\dots n}) \mapsto \mu$

### 6.11.34 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.  $F : ((\beta_i : P_i \rightarrow B)_{i=1\dots n}, P) \mapsto \mu$

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

### 6.11.36 AddUniversalMorphismIntoFiberProductWithGivenFiberProduct (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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

### 6.11.37 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.

### 6.11.38 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.

### 6.11.39 AddFiberProductFunctorialWithGivenFiberProducts (for IsCapCategory, Is-Function)

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

**Returns:** nothing

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

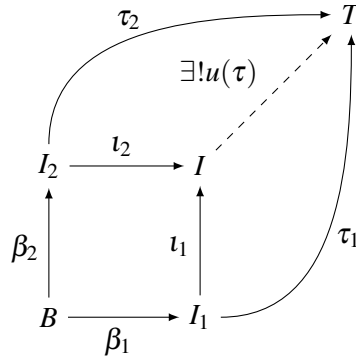
## 6.12 Pushout

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 IsomorphismFromPushoutToCokernelOfDiagonalDifference (for IsList)

▷ `IsomorphismFromPushoutToCokernelOfDiagonalDifference(D)` (operation)

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

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

### 6.12.2 IsomorphismFromPushoutToCokernelOfDiagonalDifferenceOp (for IsList, IsCapCategoryMorphism)

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

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

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

### 6.12.3 AddIsomorphismFromPushoutToCokernelOfDiagonalDifference (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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

### 6.12.4 IsomorphismFromCokernelOfDiagonalDifferenceToPushout (for IsList)

▷ `IsomorphismFromCokernelOfDiagonalDifferenceToPushout(D)` (operation)

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

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

### 6.12.5 IsomorphismFromCokernelOfDiagonalDifferenceToPushoutOp (for IsList, IsCapCategoryMorphism)

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

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

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

### 6.12.6 AddIsomorphismFromCokernelOfDiagonalDifferenceToPushout (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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

### 6.12.7 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  $\beta_i$ .

### 6.12.8 IsomorphismFromPushoutToCoequalizerOfCoproductDiagramOp (for IsList, IsCapCategoryMorphism)

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

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

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

### 6.12.9 AddIsomorphismFromPushoutToCoequalizerOfCoproductDiagram (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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

### 6.12.10 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  $\beta_i$ .

### 6.12.11 IsomorphismFromCoequalizerOfCoproductDiagramToPushoutOp (for IsList, IsCapCategoryMorphism)

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

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

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



### 6.12.12 AddIsomorphismFromCoequalizerOfCoproductDiagramToPushout (for Is-CapCategory, 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.  $F : ((\beta_i : B \rightarrow I_i)_{i=1\dots n}) \mapsto (\Delta \rightarrow \text{Pushout}(D))$

### 6.12.13 DirectSumCodiagonalDifference (for IsList)

▷ DirectSumCodiagonalDifference( $D$ ) (operation)

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

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

### 6.12.14 DirectSumCodiagonalDifferenceOp (for IsList, IsCapCategoryMorphism)

▷ DirectSumCodiagonalDifferenceOp( $D$ , *method\_selection\_morphism*) (operation)

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

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

### 6.12.15 AddDirectSumCodiagonalDifference (for IsCapCategory, IsFunction)

▷ AddDirectSumCodiagonalDifference( $C$ ,  $F$ ) (operation)

**Returns:** nothing

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

### 6.12.16 DirectSumProjectionInPushout (for IsList)

▷ DirectSumProjectionInPushout( $D$ ) (operation)

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

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

### 6.12.17 DirectSumProjectionInPushoutOp (for IsList, IsCapCategoryMorphism)

▷ DirectSumProjectionInPushoutOp( $D$ , *method\_selection\_morphism*) (operation)

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

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

### 6.12.18 AddDirectSumProjectionInPushout (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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

### 6.12.19 Pushout (for IsList)

▷ `Pushout(D)` (operation)

**Returns:** an object

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

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

### 6.12.21 PushoutOp (for IsList, IsCapCategoryMorphism)

▷ `PushoutOp(D)` (operation)

**Returns:** an object

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

### 6.12.22 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  $\iota_k : I_k \rightarrow \text{Pushout}(D)$ .

### 6.12.23 InjectionOfCofactorOfPushoutOp (for IsList, IsInt, IsCapCategoryMorphism)

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

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

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

### 6.12.24 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  $\iota_k : I_k \rightarrow I$ .

### 6.12.25 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  $\iota_1 : I_1 \rightarrow \text{Pushout}(D)$ .

### 6.12.26 MorphismFromSourceToPushoutOp (for IsList, IsCapCategoryMorphism)

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

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

The arguments are a list of morphisms  $D = (\beta_i : B \rightarrow I_i)_{i=1\dots n}$  and a morphism for method selection. The output is the composition  $\mu : B \rightarrow \text{Pushout}(D)$  of  $\beta_1$  and the 1-st injection  $\iota_1 : I_1 \rightarrow \text{Pushout}(D)$ .

### 6.12.27 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  $\iota_1 : I_1 \rightarrow I$ .

### 6.12.28 UniversalMorphismFromPushout

▷ `UniversalMorphismFromPushout(arg)` (function)

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

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

### 6.12.29 UniversalMorphismFromPushoutOp (for IsList, IsList, IsCapCategoryMorphism)

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

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

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

### 6.12.30 UniversalMorphismFromPushoutWithGivenPushout (for IsList, IsList, IsCapCategoryObject)

▷ UniversalMorphismFromPushoutWithGivenPushout( $D$ ,  $\tau$ ,  $I$ ) (operation)

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

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

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

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

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

### 6.12.34 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.  $F : ((\beta_i : B \rightarrow I_i)_{i=1\dots n}) \mapsto \mu$

### 6.12.35 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.  $F : ((\beta_i : B \rightarrow I_i)_{i=1\dots n}, I) \mapsto \mu$

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

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

### 6.12.38 PushoutFunctorial (for IsList, IsList, IsList)

▷ PushoutFunctorial( $L_s$ ,  $L_m$ ,  $L_r$ ) (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.

### 6.12.39 PushoutFunctorialWithGivenPushouts (for IsCapCategoryObject, IsList, IsList, IsList, IsCapCategoryObject)

▷ PushoutFunctorialWithGivenPushouts( $s$ ,  $L_s$ ,  $L_m$ ,  $L_r$ ,  $r$ ) (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.

### 6.12.40 AddPushoutFunctorialWithGivenPushouts (for IsCapCategory, IsFunction)

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

**Returns:** nothing

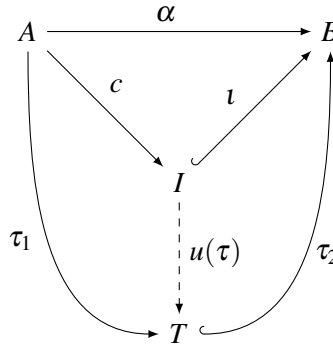
The arguments are a category  $C$  and a function  $F$ . This operation adds the given function  $F$  to the category for the basic operation PushoutFunctorial.  $F : (\text{Pushout}((\beta_i)_{i=1}^n), (\beta_i : B \rightarrow I_i)_{i=1\dots n}, (\mu_i : I_i \rightarrow I'_i)_{i=1\dots n}, (\beta'_i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

## 6.13 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 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`.  $F : \alpha \mapsto (\text{im}(\alpha) \rightarrow \text{KernelObject}(\text{CokernelProjection}(\alpha)))$

### 6.13.3 IsomorphismFromKernelOfCokernelToImageObject (for IsCapCategoryMorphism)

▷ `IsomorphismFromKernelOfCokernelToImageObject(alpha)` (attribute)

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

The argument is a morphism  $\alpha$ . The output is the canonical morphism  $\text{KernelObject}(\text{CokernelProjection}(\alpha)) \rightarrow \text{im}(\alpha)$ .

### 6.13.4 AddIsomorphismFromKernelOfCokernelToImageObject (for IsCapCategory, IsFunction)

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

### 6.13.5 ImageObject (for IsCapCategoryMorphism)

▷ `ImageObject( $\alpha$ )` (attribute)

**Returns:** an object

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

### 6.13.6 ImageEmbedding (for IsCapCategoryMorphism)

▷ `ImageEmbedding( $\alpha$ )` (attribute)

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

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

### 6.13.7 ImageEmbeddingWithGivenImageObject (for IsCapCategoryMorphism, IsCapCategoryObject)

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

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

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

### 6.13.8 CostrictionToImage (for IsCapCategoryMorphism)

▷ `CostrictionToImage( $\alpha$ )` (attribute)

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

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

### 6.13.9 CostrictionToImageWithGivenImageObject (for IsCapCategoryMorphism, IsCapCategoryObject)

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

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

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

### 6.13.10 UniversalMorphismFromImage (for IsCapCategoryMorphism, IsList)

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

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

The arguments are a morphism  $\alpha : A \rightarrow B$  and a pair of morphisms  $\tau = (\tau_1 : A \rightarrow T, \tau_2 : T \hookrightarrow B)$  where  $\tau_2$  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.

### 6.13.11 UniversalMorphismFromImageWithGivenImageObject (for IsCapCategory-Morphism, IsList, IsCapCategoryObject)

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

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

The arguments are a morphism  $\alpha : A \rightarrow B$ , a pair of morphisms  $\tau = (\tau_1 : A \rightarrow T, \tau_2 : T \hookrightarrow B)$  where  $\tau_2$  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.

### 6.13.12 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`.  $F : \alpha \mapsto I$ .

### 6.13.13 AddImageEmbedding (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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

### 6.13.14 AddImageEmbeddingWithGivenImageObject (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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

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

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

### 6.13.17 AddUniversalMorphismFromImage (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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

### 6.13.18 AddUniversalMorphismFromImageWithGivenImageObject (for IsCapCategory, IsFunction)

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

**Returns:** nothing

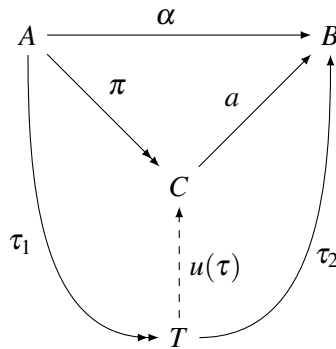
The arguments are a category  $C$  and a function  $F$ . This operation adds the given function  $F$  to the category for the basic operation `UniversalMorphismFromImageWithGivenImageObject`.  $F : (\alpha, \tau, I) \mapsto u(\tau)$ .

## 6.14 Coimage

For a given morphism  $\alpha : A \rightarrow B$ , 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(alpha)` (operation)

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

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

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

### 6.14.4 InverseMorphismFromCoimageToImage (for IsCapCategoryMorphism)

▷ `InverseMorphismFromCoimageToImage(alpha)` (attribute)

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

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

### 6.14.5 InverseMorphismFromCoimageToImageWithGivenObjects (for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryObject)

▷ `InverseMorphismFromCoimageToImageWithGivenObjects(C, alpha, I)` (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)  $I \rightarrow C$ .

### 6.14.6 AddInverseMorphismFromCoimageToImageWithGivenObjects (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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

### 6.14.7 IsomorphismFromCoimageToCokernelOfKernel (for IsCapCategoryMorphism)

▷ `IsomorphismFromCoimageToCokernelOfKernel(alpha)` (attribute)

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

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

### 6.14.8 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`.  $F : \alpha \mapsto (\text{coim}(\alpha) \rightarrow \text{CokernelObject}(\text{KernelEmbedding}(\alpha)))$ .

### 6.14.9 IsomorphismFromCokernelOfKernelToCoimage (for IsCapCategoryMorphism)

▷ `IsomorphismFromCokernelOfKernelToCoimage(alpha)` (attribute)

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

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

### 6.14.10 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`.  $F : \alpha \mapsto (\text{CokernelObject}(\text{KernelEmbedding}(\alpha)) \rightarrow \text{coim}(\alpha))$ .

### 6.14.11 Coimage (for IsCapCategoryMorphism)

▷ `Coimage(alpha)` (attribute)

**Returns:** an object

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

### 6.14.12 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  $\pi : A \twoheadrightarrow \text{coim}(\alpha)$ .

### 6.14.13 CoimageProjectionWithGivenCoimage (for IsCapCategoryMorphism, IsCapCategoryObject)

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

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

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

### 6.14.14 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  $a : \text{coim}(\alpha) \rightarrow B$ .

### 6.14.15 AstrictionToCoimageWithGivenCoimage (for IsCapCategoryMorphism, IsCapCategoryObject)

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

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

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

### 6.14.16 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.

### 6.14.17 UniversalMorphismIntoCoimageWithGivenCoimage (for IsCapCategoryMorphism, IsList, IsCapCategoryObject)

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

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

The arguments are a morphism  $\alpha : A \rightarrow B$ , a pair of morphisms  $\tau = (\tau_1 : A \twoheadrightarrow T, \tau_2 : T \rightarrow B)$  where  $\tau_1$  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.

### 6.14.18 AddCoimage (for IsCapCategory, IsFunction)

▷ `AddCoimage(C, F)` (operation)

**Returns:** nothing

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

### 6.14.19 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.  $F : \alpha \mapsto \pi$

### 6.14.20 AddCoimageProjectionWithGivenCoimage (for IsCapCategory, IsFunction)

▷ AddCoimageProjectionWithGivenCoimage( $C, F$ ) (operation)

**Returns:** nothing

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

### 6.14.21 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.  $F : \alpha \mapsto a$

### 6.14.22 AddAstrictionToCoimageWithGivenCoimage (for IsCapCategory, IsFunction)

▷ AddAstrictionToCoimageWithGivenCoimage( $C, F$ ) (operation)

**Returns:** nothing

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

### 6.14.23 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.  $F : (\alpha, \tau) \mapsto u(\tau)$

### 6.14.24 AddUniversalMorphismIntoCoimageWithGivenCoimage (for IsCapCategory, IsFunction)

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

**Returns:** nothing

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

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).

### 6.14.25 CanonicalIdentificationFromImageObjectToCoimage (for IsCapCategory-Morphism)

▷ CanonicalIdentificationFromImageObjectToCoimage(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 canonical identification  $c : \text{im}(\alpha) \rightarrow \text{coim}(\alpha)$ .

### 6.14.26 AddCanonicalIdentificationFromImageObjectToCoimage (for IsCapCategory, IsFunction)

▷ AddCanonicalIdentificationFromImageObjectToCoimage(C, F) (operation)

**Returns:** nothing

The arguments are a category  $C$  and a function  $F$ . This operation adds the given function  $F$  to the category for the basic operation CanonicalIdentificationFromImageObjectToCoimage.  $F : \alpha \mapsto c$

### 6.14.27 CanonicalIdentificationFromCoimageToImageObject (for IsCapCategory-Morphism)

▷ CanonicalIdentificationFromCoimageToImageObject(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 identification  $c : \text{coim}(\alpha) \rightarrow \text{im}(\alpha)$ .

### 6.14.28 AddCanonicalIdentificationFromCoimageToImageObject (for IsCapCategory, IsFunction)

▷ AddCanonicalIdentificationFromCoimageToImageObject(C, F) (operation)

**Returns:** nothing

The arguments are a category  $C$  and a function  $F$ . This operation adds the given function  $F$  to the category for the basic operation CanonicalIdentificationFromCoimageToImageObject.  $F : \alpha \mapsto c$

## 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)

▷ `HomologyObjectFunctorial(alpha, beta, epsilon, 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 IsCapCategoryObject, 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  $\text{ImageObject}(\text{PreCompose}(\text{KernelEmbedding}(\beta), \text{CokernelProjection}(\alpha)))$ , 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

$\text{ImageObject}(\text{PreCompose}(\text{KernelEmbedding}(\beta), \text{CokernelProjection}(\alpha)))$ , denoted by  $I$ , to the homology object  $H$  of  $\alpha$  and  $\beta$ .

### 6.15.6 AddHomologyObject (for IsCapCategory, IsFunction)

▷  $\text{AddHomologyObject}(C, F)$  (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  $\text{HomologyObject}$ .  $F : (\alpha, \beta) \mapsto \text{HomologyObject}(\alpha, \beta)$ .

### 6.15.7 AddHomologyObjectFunctorialWithGivenHomologyObjects (for IsCapCategory, IsFunction)

▷  $\text{AddHomologyObjectFunctorialWithGivenHomologyObjects}(C, F)$  (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  $\text{AddHomologyObjectFunctorialWithGivenHomologyObjects}$ .  $F : (H_1, \alpha, \beta, \varepsilon, \gamma, \delta, H_2) \mapsto (H_1 \rightarrow H_2)$



## Chapter 7

# Add Functions

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

### 7.1 Functions Installed by Add

Add functions (up to some exceptions) have the following syntax

```
DeclareOperation( "AddSomeFunc", [ IsCapCategory, IsList, IsInt ] );
```

The first argument is the category to which some function (e.g. `KernelObject`) is added, the second is a list containing pairs of functions and additional filters for the arguments, (e.g. if one argument is a morphism, an additional filter could be `IsMomomorphism`). The third is a weight which will then be the weight for `SomeFunc`. This is described later. If only one function is to be installed, the list can be replaced by the function. Via `InstallMethod`, CAP installs the given function(s) as methods for the install name of `SomeFunc`, as listed in the `MethodRecord`. If no install name is given, the name `SomeFunc` is used.

All installed methods follow the following steps, described below:

- Redirect function
- Prefunction
- Function
- Logic
- Postfunction
- Addfunction

Every other part, except from function, does only depend on the name `SomeFunc`. We now explain the steps in detail.

- Redirect function: The redirect is used to redirect the computation from the given functions to some other symbol. If there is for example a `with` given method for some universal property, and the universal object is already computed, the redirect function might detect such a thing, calls the `with` given operation with the universal object as additional argument and then returns the value. In general, the redirect can be an arbitrary function. It is called with the same arguments as the operation `SomeFunc` itself and can return an array containing `[ true, something ]`, which

will cause the installed method to simply return the object something, or [ false ]. If the output is false, the computation will continue with the step Prefunction. Additionally, for every category and every name like SomeFunc, there is a boolean, stored in the category's redirects component under the name of SomeFunc, which, when it is false, will prevent the redirect function from being executed.

- Prefunction: The prefunction should be used for error handling and soft checks of the sanity of the input to SomeFunc (e.g. for KernelLift it should check whether range and source of the morphisms coincide). Generally, the prefunction is defined in the method record and only depends on the name SomeFunc. It is called with the same input as the function itself, and should return either [ true ], which continues the computation, or [ false, "message" ], which will cause an error with message "message" and some additional information.
- 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.

## 7.2 Add Method

Except from installing a new method for the name SomeFunc, an Add method does slightly more. Every Add method has the same structure. The steps in the Add method are as follows:

- Weight check: If the current weight of the operation is lower than the given weight of the new functions, then the add function returns and installs nothing.
- Option check: There are two possible options for every add method: SetPrimitive and IsDerivation.
  - SetPrimitive should be a boolean, the default is true. If SetPrimitive is false, then the current call of this add will not set the installed function to be primitive. This is used for derivations.

- IsDerivation should be a boolean, default is false. If it is true, the add method assumes that the given function is a derivation and does not try to install a corresponding pair (See below).
- Standard weight: If the weight parameter is -1, the Standard weight is assumed, which is 100.
- Checking for pairs: If the function is not a with given operation, has a corresponding with given or is a with given, and is newly installed, i.e. the current installation weight which is given to the add function is less than the current weight, the add method is going to install a corresponding pair function, i.e. a function for the corresponding with or without given method, which redirects to the currently installed functions. It also deactivates the redirect for this function. Note that the pair install is only done for primitive functions, and if the current weight is higher than the given weight.
- Can compute: Set the corresponding can compute of the category to true
- Install methods: Decide on the methods used to install the function. Check whether InstallMethodWithCache, InstallMethodWithToDoForIsWellDefined, both, or simply InstallMethod is used. This is decided by the ToDo and the caching flags.
- Installation: Next, the method to install the functions is created. It creates the correct filter list, by merging the standard filters for the operation with the particular filters for the given functions, then installs the method as described above.
- SetPrimitive: If the set primitive flag is true, it is set as primitive in the weight list of the category.
- Pair install: If there is a function pair, as described above, it is installed.

After calling an add method, the corresponding Operation is available in the category. Also, some derivations, which are triggered by the setting of the primitive value, might be available.

### 7.3 InstallAdd Function

Almost all Add methods in the CAP kernel are installed by the CapInternalInstallAdd operation. The definition of this function is as follows:

```
DeclareOperation( "CapInternalInstallAdd", [ IsRecord ] );
```

The record can have the following components, used as described:

- function\_name: The name of the function. This does not have to coincide with the installation name. It is used for the derivation weight.
- installation\_name (optional): A string which is the name of the operation for which the functions given to the Add method are installed as methods.
- pre\_function (optional): A function which is used as the prefunction of the installed methods, as described above.
- redirect\_function (optional): A function which is used as the redirect function of the installed methods, as described above.

- `post_function` (optional): A function which is used as the postfunction of the installed methods, as described above.
- `filter_list`: A list containing the basic filters for the methods installed by the add methods. Possible are filters, or the following strings, which will be replaced by appropriate filters at the time the add method is called:
  - `category`,
  - `cell`,
  - `object`,
  - `morphism`,
  - `twocell`,
  - `other_category`,
  - `other_cell`,
  - `other_object`,
  - `other_morphism`,
  - `other_twocell`,
  - `list_of_objects`,
  - `list_of_morphisms`,
  - `list_of_twocells`.
- `well_defined_todo` (optional): A boolean, default value is true, which states whether there should be to do list entries which propagate well definedness from the input of the installed methods to their output. Please note that true only makes sense if at least one argument and the output of the installed method is a cell.
- `cache_name` (optional): The name of the cache which is used for the installed methods. If no cache name is given, the caching for the operation is deactivated completely.
- `argument_list` (optional): A list containing integers, which defines which arguments should be used for the additional functions, (e.g. `redirect`, `pre`, ...). This is important for the `Op` method constructions. If no argument list is given, all arguments are used. Please note that if you have a method selection argument for your function, you need to give the `argument_list` to explicitly state which argument is the method selection argument.
- `return_type` (optional): The return type can be one of the following:
  - `object` or `object_or_fail`,
  - `morphism` or `morphism_or_fail`,
  - `twocell`,
  - `bool`,
  - `other_object`,

– `other_morphism`.

If it is one of the first three options, the correct Add function (see above) is used for the result of the computation. Otherwise, no Add function is used after all.

- `is_with_given`: Boolean, marks whether the function which is to be installed is a with given function or not.
- `with_given_without_given_name_pair` (optional): If the currently installed operation has a corresponding with given operation or is the with given of another operation, the names of both should be in this list.
- `functorial` (optional): If an object has a corresponding functorial function, e.g., `KernelObject` and `KernelObjectFunctorial`, the name of the functorial is stored as a string.
- `number_of_diagram_arguments`: Specifies how many of the arguments (counting from the first argument) of the function specify the diagram of the universal object.
- `dual_arguments_reversed`: Boolean, marks whether for the call of the dual operation all arguments have to be given in reversed order.
- `dual_preprocessor_func`: let `f` be an operation with dual operation `g`. For the automatic installation of `g` from `f`, the arguments given to `g` are preprocessed by this given function.
- `dual_postprocessor_func`: let `f` be an operation with dual operation `g`. For the automatic installation of `g` from `f`, the computed value of `f` is postprocessed by the given function.
- `zero_arguments_for_add_method`: the add method of this operation should get a function without arguments

Using all those entries, the operation `CapInternalInstallAdd` installs add methods as described above. It first provides a sanity check for all the entries described, then installs the Add method in 4 ways, with list or functions as second argument, and with an optional third parameter for the weight.

## 7.4 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:

- Cache check: If there is no `cache_name`, set it to the `installation_name`.
- Function name: Set the component `function_name` to the entry name.
- Redirect and post functions: Since the redirect and post functions need the category to work correctly, the given functions in the method records are packed up to discard the first argument (which is the category) if necessary.
- `universal_object_arg_list`: An argument list for redirect and post functions is created, by looking at the filter list in the record. If the number of diagram arguments is zero, the first argument is used. If all diagram arguments are lists, the diagram arguments and additionally the last argument (method selection argument) are used. Otherwise, the diagram arguments are used.

- 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.
- `argument_list`: If `argument_list` is not bound, it is set to include all arguments.
- Redirect and post functions are created for all operations belonging to universal constructions (e.g. `KernelLift`) which are not a WithGiven operation.

## 7.5 Install All Adds

The function `CAP_INTERNAL_INSTALL_ALL_ADDS` does not take any arguments. It is an auxiliary function which first applies `CAP_INTERNAL_ENHANCE_NAME_RECORD` to `CAP_INTERNAL_METHOD_NAME_RECORD`. Afterwards it iterates over `CAP_INTERNAL_METHOD_NAME_RECORD` and calls the `CapInternalInstallAdd` with the corresponding method record entry except if the `no_install` component of the record is set to true.

## 7.6 Prepare functions

### 7.6.1 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.

### 7.6.2 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`.

### 7.6.3 ListCAPPrepareFunctions

▷ `ListCAPPrepareFunctions(arg)`

(function)

Lists all prepare functions.

## Chapter 8

# Managing Derived Methods

### 8.1 Info Class

#### 8.1.1 DerivationInfo

▷ `DerivationInfo` (info class)

Info class for derivations.

#### 8.1.2 ActivateDerivationInfo

▷ `ActivateDerivationInfo(arg)` (function)

#### 8.1.3 DeactivateDerivationInfo

▷ `DeactivateDerivationInfo(arg)` (function)

### 8.2 Derivation Objects

#### 8.2.1 IsDerivedMethod (for IsObject)

▷ `IsDerivedMethod(arg)` (filter)

**Returns:** true or false

A derivation object describes a derived method. It contains information about which operation the derived method implements, and which other operations it relies on.

#### 8.2.2 MakeDerivation (for IsString, IsFunction, IsDenseList, IsPosInt, IsDenseList, IsFunction)

▷ `MakeDerivation(name, target_op, used_ops_with_multiples, weight, implementations_with_extra_filters, category_filter)` (operation)



Creates a new derivation object. The argument *name* is an arbitrary name used to identify this derivation, and is useful only for debugging purposes. The argument *target\_op* is the operation which the derived method implements. The argument *used\_ops\_with\_multiples* contains each operation used by the derived method, together with a positive integer specifying how many times that operation is used. This is given as a list of lists, where each sublist has as first entry an operation and as second entry an integer. The argument *weight* is an additional number to add when calculating the resulting weight of the target operation using this derivation. Unless there is any particular reason to regard the derivation as exceedingly expensive, this number should be 1. The argument *implementations\_with\_extra\_filters* contains one or more functions with the actual implementation of the derived method, together with lists of extra argument filters for each function. The argument is a list with entries of the form `[fun, filters]`, where *fun* is a function and *filters* is a (not necessarily dense) list of argument filters. If only one function is given, then *filters* should be the empty list; in this case the argument's value would be `[[fun,[]]]`, where *fun* is the function. The argument *category\_filter* is a filter describing which categories the derivation is valid for. If it is valid for all categories, then this argument should have the value `IsCapCategory`. The Option `ConditionsListComplete` indicates if the manually given list of preconditions for this derivation is complete or should be extended by looking for categorical operations in the function body. The value `false` indicates it is not complete, every other value that it is complete. Default is `false`.

### 8.2.3 DerivationName (for IsDerivedMethod)

▷ `DerivationName(d)` (attribute)

The name of the derivation. This is a name identifying this particular derivation, and normally not the same as the name of the operation implemented by the derivation.

### 8.2.4 DerivationWeight (for IsDerivedMethod)

▷ `DerivationWeight(d)` (attribute)

Extra weight for the derivation.

### 8.2.5 DerivationFunctionsWithExtraFilters (for IsDerivedMethod)

▷ `DerivationFunctionsWithExtraFilters(d)` (attribute)

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

### 8.2.6 CategoryFilter (for IsDerivedMethod)

▷ `CategoryFilter(d)` (attribute)

Filter describing which categories the derivation is valid for.

### 8.2.7 IsApplicableToCategory (for IsDerivedMethod, IsCapCategory)

- ▷ `IsApplicableToCategory(d, C)` (operation)  
**Returns:** true if the category *C* is known to satisfy the category filter of the derivation *d*.  
 Checks if the derivation is known to be valid for a given category.

### 8.2.8 TargetOperation (for IsDerivedMethod)

- ▷ `TargetOperation(d)` (attribute)  
**Returns:** The name (as a string) of the operation implemented by the derivation *d*

### 8.2.9 UsedOperations (for IsDerivedMethod)

- ▷ `UsedOperations(d)` (attribute)  
**Returns:** The names (as strings) of the operations used by the derivation *d*

### 8.2.10 UsedOperationMultiples (for IsDerivedMethod)

- ▷ `UsedOperationMultiples(d)` (attribute)  
**Returns:** Multiplicities of each operation used by the derivation *d*, in order corresponding to the operation names returned by `UsedOperations(d)`.

### 8.2.11 UsedOperationsWithMultiples (for IsDerivedMethod)

- ▷ `UsedOperationsWithMultiples(d)` (attribute)  
**Returns:** The names of the operations used by the derivation *d*, together with their multiplicities. The result is a list consisting of lists of the form [*op\_name*, *mult*], where *op\_name* is a string and *mult* a positive integer.

### 8.2.12 InstallDerivationForCategory (for IsDerivedMethod, IsPosInt, IsCapCategory)

- ▷ `InstallDerivationForCategory(d, weight, C)` (operation)  
 Install the derived method *d* for the category *C*. The integer *weight* is the computed weight of the operation implemented by this derivation.

### 8.2.13 DerivationResultWeight (for IsDerivedMethod, IsDenseList)

- ▷ `DerivationResultWeight(d, op_weights)` (operation)  
 Computes the resulting weight of the target operation of this derivation given a list of weights for the operations it uses. The argument *op\_weights* should be a list of integers specifying weights for the operations given by `UsedOperations(d)`, in the same order.

### 8.2.14 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 IsObject)

▷ `IsDerivedMethodGraph(arg)` (filter)  
**Returns:** true or false

A derivation graph consists of a set of operations and a set of derivations specifying how some operations can be implemented in terms of other operations.

### 8.3.2 MakeDerivationGraph (for IsDenseList)

▷ `MakeDerivationGraph(operations)` (operation)

Make a derivation graph containing the given set of operations and no derivations. The argument *operations* should be a list of strings, the names of the operations. The set of operations is fixed once the graph is created. Derivations can be added to the graph by calling `AddDerivation`.

### 8.3.3 AddOperationsToDerivationGraph (for IsDerivedMethodGraph, IsDenseList)

▷ `AddOperationsToDerivationGraph(graph, operations)` (operation)

Adds a list of operation names *operations* to a given derivation graph *graph*. This is used in extensions of CAP which want to have their own basic operations, but do not want to pollute the CAP kernel any more. Please use it with caution. If a weight list/category was created before it will not be aware of the operations.

### 8.3.4 AddDerivation (for IsDerivedMethodGraph, IsDerivedMethod)

▷ `AddDerivation(G, d)` (operation)

Add a derivation to a derivation graph.

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

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

### 8.3.6 AddDerivation (for IsDerivedMethodGraph, IsFunction, IsDenseList)

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

### 8.3.7 AddDerivation (for IsDerivedMethodGraph, IsFunction, IsFunction)

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

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

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

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

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

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

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

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

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

### 8.3.12 AddDerivationToCAP

▷ AddDerivationToCAP(*arg*) (function)

### 8.3.13 AddDerivationPairToCAP

▷ AddDerivationPairToCAP(*arg*) (function)

### 8.3.14 AddWithGivenDerivationPairToCAP

▷ AddWithGivenDerivationPairToCAP(*arg*) (function)

### 8.3.15 Operations (for IsDerivedMethodGraph)

▷ Operations(*G*) (attribute)

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

### 8.3.16 DerivationsUsingOperation (for IsDerivedMethodGraph, IsString)

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

Finds all the derivations in the graph *G* that use the operation named *op\_name*, and returns them as a list.

### 8.3.17 DerivationsOfOperation (for IsDerivedMethodGraph, IsString)

▷ `DerivationsOfOperation(G, op_name)` (operation)

Finds all the derivations in the graph *G* targeting the operation named *op\_name* (that is, the derivations that provide implementations of this operation), and returns them as a list.

## 8.4 Managing Derivations in a Category

### 8.4.1 IsOperationWeightList (for IsObject)

▷ `IsOperationWeightList(arg)` (filter)

**Returns:** true or false

An operation weight list manages the use of derivations in a single category *C*. For every operation, it keeps a weight value which indicates how costly it is to perform that operation in the category *C*. Whenever a new operation is implemented in *C*, the operation weight list should be notified about this and given a weight to assign to this operation. It will then automatically install all possible derived methods for *C* in such a way that every operation has the smallest possible weight (the weight of a derived method is computed by using the weights of the operations it uses; see `DerivationResultWeight`).

### 8.4.2 MakeOperationWeightList (for IsCapCategory, IsDerivedMethodGraph)

▷ `MakeOperationWeightList(C, G)` (operation)

Create the operation weight list for a category. This should only be done once for every category, and the category should afterwards remember the returned object. The argument *C* is the CAP category this operation weight list is associated to, and the argument *G* is a derivation graph containing operation names and derivations.

### 8.4.3 DerivationGraph (for IsOperationWeightList)

▷ `DerivationGraph(owl)` (attribute)

Returns the derivation graph used by the operation weight list *owl*.

### 8.4.4 CategoryOfOperationWeightList (for IsOperationWeightList)

▷ `CategoryOfOperationWeightList(owl)` (attribute)

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

### 8.4.5 CurrentOperationWeight (for IsOperationWeightList, IsString)

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

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

### 8.4.6 OperationWeightUsingDerivation (for IsOperationWeightList, 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 IsObject)

▷ IsStringMinHeap(*arg*) (filter)

**Returns:** true or false

A string min heap is a min heap where every node contains a string label and an integer key.

#### 8.5.2 StringMinHeap

▷ StringMinHeap(*arg*) (function)

Create an empty string min heap.

#### 8.5.3 Add (for IsStringMinHeap, IsString, IsInt)

▷ Add(*H*, *string*, *key*) (operation)

Add a new node containing the label *string* and the key *key* to the heap *H*.

### 8.5.4 ExtractMin (for IsStringMinHeap)

▷ ExtractMin( $H$ ) (operation)

Remove a node with minimal key value from the heap  $H$ , and return it. The return value is a list [ *label*, *key* ], where *label* is the extracted node's label (a string) and *key* is the node's key (an integer).

### 8.5.5 DecreaseKey (for IsStringMinHeap, IsString, IsInt)

▷ DecreaseKey( $H$ , *string*, *key*) (operation)

Decrease the key value for the node with label *string* in the heap  $H$ . The new key value is given by *key* and must be smaller than the node's current value.

### 8.5.6 IsEmptyHeap (for IsStringMinHeap)

▷ IsEmptyHeap( $H$ ) (operation)

Returns true if the heap  $H$  is empty, false otherwise.

### 8.5.7 HeapSize (for IsStringMinHeap)

▷ HeapSize( $H$ ) (operation)

Returns the number of nodes in the heap  $H$ .

### 8.5.8 Contains (for IsStringMinHeap, IsString)

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

Returns true if the heap  $H$  contains a node with label *string*, and false otherwise.

### 8.5.9 Swap (for IsStringMinHeap, IsPosInt, IsPosInt)

▷ Swap( $H$ , *i*, *j*) (operation)

Swaps two elements in the list used to implement the heap, and updates the heap's internal mapping of labels to list indices. This is an internal function which should only be called from the functions that implement the heap functionality.

### 8.5.10 Heapify (for IsStringMinHeap, IsPosInt)

▷ Heapify( $H$ , *i*) (operation)

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



## Chapter 9

# Technical Details

### 9.1 The Category Cat

#### 9.1.1 ObjectCache (for IsCapFunctor)

▷ `ObjectCache(functor)` (attribute)  
**Returns:** `IsCachingObject`  
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 Install Functions for IsWellDefined

#### 9.2.1 InstallMethodWithToDoForIsWellDefined

▷ `InstallMethodWithToDoForIsWellDefined(arg)` (function)

The `IsWellDefined` filter is a basic function of CAP. For every categorial construction the outcome is well defined if and only if every input object or morphism of the construction is well defined. So for every implementation of a categorial construction a `ToDoListEntry` needs to be defined which propagates well definedness from the input cells to the output. For not writing this construction in every method, this function can be used to install a method which then installs the correct `ToDoListEntries` for the output. The input syntax works exactly like `InstallMethod`, with one extension: The method creates an auxilliary function which computes the output from the function given to `InstallMethodWithToDoForIsWellDefined`, then installs the `ToDoListEntries`, and then installs the auxilliary function instead of the original one. This is normally done with `InstallMethod`. However, one can change this via the option `InstallMethod`, which can be set to any other function which is then used instead of `InstallMethod`. This is used for the caching functions.

### 9.2.2 InstallSetWithToDoForIsWellDefined (for IsObject, IsString, IsList)

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

For the caching one needs the possibility to install setters for functions with multiple arguments. This is a setter function which automatically adds `ToDoListEntries` for `IsWellDefined` like described above for the manually setted result of a method.

### 9.2.3 DeclareAttributeWithToDoForIsWellDefined

▷ `DeclareAttributeWithToDoForIsWellDefined(arg)` (function)

Since attributes install their setters themselves, one needs to declare attributes in another way to ensure `ToDoListEntries` for `IsWellDefined` in the setter of an attribute. This function works like `DeclareAttribute`, but installs `ToDoListEntries` for the setter of the attribute. Please note that implementations still need to be done with `InstallMethodWithToDoForIsWellDefined`.

### 9.2.4 DeclareFamilyProperty

▷ `DeclareFamilyProperty(arg)` (function)

### 9.2.5 CAP\_INTERNAL\_REPLACE\_STRINGS\_WITH\_FILTERS

▷ `CAP_INTERNAL_REPLACE_STRINGS_WITH_FILTERS(list, category)` (function)

**Returns:** Replaced list

The function takes a list (of lists) of filters or strings, where the strings can be category, cell, object, morphism, or twocell. The strings are then recursively replaced by the corresponding filters of the category. The replaced list is returned.

### 9.2.6 CAP\_INTERNAL\_MERGE\_FILTER\_LISTS

▷ `CAP_INTERNAL_MERGE_FILTER_LISTS(list, additional, list)` (function)

**Returns:** merged lists

The first argument should be a dense list with filters, the second a sparse list containing filters not longer then the first one. The filters of the second list are then appended (via `and`) to the filters in the first list at the corresponding position, and the resulting list is returned.

### 9.2.7 CAP\_INTERNAL\_RETURN\_OPTION\_OR\_DEFAULT

▷ `CAP_INTERNAL_RETURN_OPTION_OR_DEFAULT(string, value)` (function)

**Returns:** option value

Returns the value of the option with name string, or, if this value is fail, the object value.

### 9.2.8 CAP\_INTERNAL\_FIND\_APPEARANCE\_OF\_SYMBOL\_IN\_FUNCTION

▷ `CAP_INTERNAL_FIND_APPEARANCE_OF_SYMBOL_IN_FUNCTION(function, symbol_list, loop_multiple, replacement_record)` (function)

**Returns:** a list of symbols with multiples

The function searches for the appearance of the strings in symbol list on the function function and returns a list of pairs, containing the name of the symbol and the number of appearance. If the symbol appears in a loop, the number of appearance is counted times the loop multiple. Moreover, if appearances of found strings should be replaced by collections of other strings, then these can be specified in the replacement record.

### 9.2.9 CAP\_INTERNAL\_MERGE\_PRECONDITIONS\_LIST

▷ `CAP_INTERNAL_MERGE_PRECONDITIONS_LIST(list1, list2)` (function)

**Returns:** merge list

The function takes two lists containing pairs of symbols (strings) and multiples. The lists are merged that pairs where the string only appears in one list is then added to the return list, if a pair with a string appears in both lists, the resulting lists only contains this pair once, with the higher multiple from both lists.

### 9.2.10 CAP\_INTERNAL\_ASSERT\_IS\_CELL\_OF\_CATEGORY

▷ `CAP_INTERNAL_ASSERT_IS_CELL_OF_CATEGORY(cell, category, human_readable_identifier_getter)` (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\_getter* is a 0-ary function returning a string which is used to refer to *cell* in the error message.

### 9.2.11 CAP\_INTERNAL\_ASSERT\_IS\_OBJECT\_OF\_CATEGORY

▷ `CAP_INTERNAL_ASSERT_IS_OBJECT_OF_CATEGORY(object, category, human_readable_identifier_getter)` (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\_getter* is a 0-ary function returning a string which is used to refer to *object* in the error message.

### 9.2.12 CAP\_INTERNAL\_ASSERT\_IS\_MORPHISM\_OF\_CATEGORY

▷ `CAP_INTERNAL_ASSERT_IS_MORPHISM_OF_CATEGORY(morphism, category, human_readable_identifier_getter)` (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\_getter* is a 0-ary function returning a string which is used to refer to *morphism* in the error message.

### 9.2.13 CAP\_INTERNAL\_ASSERT\_IS\_TWO\_CELL\_OF\_CATEGORY

▷ `CAP_INTERNAL_ASSERT_IS_TWO_CELL_OF_CATEGORY(two_cell, category, human_readable_identifier_getter)` (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\_getter* is a 0-ary function returning a string which is used to refer to *two\_cell* in the error message.

### 9.2.14 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.

### 9.2.15 BrowseCachingStatistic

▷ `BrowseCachingStatistic(category)` (function)

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

## Chapter 10

# Limits and Colimits

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

### 10.1 Specification of Limits and Colimits

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

- `object_specification`: see below
- `morphism_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 Validating entries of a method name record which are part of a limit or colimit**

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

## Chapter 11

# Examples and Tests

### 11.1 Functors

We create a binary functor  $F$  with one covariant and one contravariant component in two ways. Here is the first way to model a binary functor:

Example

```
gap> field := HomalgFieldOfRationals( );
gap> vec := LeftPresentations( field );
gap> F := CapFunctor( "CohomForVec", [ vec, [ vec, true ] ], vec );
gap> obj_func := function( A, B ) return TensorProductOnObjects( A, DualOnObjects( B ) ); end;;
gap> mor_func := function( source, alpha, beta, range ) return TensorProductOnMorphismsWithGivenT
gap> AddObjectFunction( F, obj_func );
gap> AddMorphismFunction( F, mor_func );
```

CAP regards  $F$  as a binary functor on a technical level, as we can see by looking at its input signature:

Example

```
gap> InputSignature( F );
[ [ Category of left presentations of Q, false ], [ Category of left presentations of Q, true ] ]
```

We can see that `ApplyFunctor` works both on two arguments and on one argument (in the product category).

Example

```
gap> V1 := TensorUnit( vec );
gap> V3 := DirectSum( V1, V1, V1 );
gap> pi1 := ProjectionInFactorOfDirectSum( [ V1, V1 ], 1 );
gap> pi2 := ProjectionInFactorOfDirectSum( [ V3, V1 ], 1 );
gap> value1 := ApplyFunctor( F, pi1, pi2 );
gap> input := Product( pi1, Opposite( pi2 ) );
gap> value2 := ApplyFunctor( F, input );
gap> IsCongruentForMorphisms( value1, value2 );
true
```

Here is the second way to model a binary functor:

Example

```
gap> F2 := CapFunctor( "CohomForVec2", Product( vec, Opposite( vec ) ), vec );
gap> AddObjectFunction( F2, a -> obj_func( a[1], Opposite( a[2] ) ) );
gap> AddMorphismFunction( F2, function( source, datum, range ) return mor_func( source, datum[1],
```



```
gap> value3 := ApplyFunctor( F2,input );;
gap> IsCongruentForMorphisms( value1, value3 );
true
```

CAP regards  $F2$  as a unary functor on a technical level, as we can see by looking at its input signature:

Example

```
gap> InputSignature( F2 );
[ [ Product of: Category of left presentations of Q, Opposite of Category of left presentations 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 );;
```

## 11.2 Homomorphism structure

Example

```
gap> ReadPackage( "CAP", "examples/FieldAsCategory.g" );;
gap> Q := HomalgFieldOfRationals();;
gap> Qoid := FieldAsCategory( Q );;
gap> a := FieldAsCategoryMorphism( 1/2, Qoid );;
gap> b := FieldAsCategoryMorphism( -2/3, Qoid );;
gap> u := FieldAsCategoryUniqueObject( Qoid );;
gap> IsCongruentForMorphisms( a,
>   InterpretMorphismFromDistinguishedObjectToHomomorphismStructureAsMorphism(
>     u,u,
>     InterpretMorphismAsMorphismFromDistinguishedObjectToHomomorphismStructure(
>       a
>     )
>   )
> );
true
gap> a = HomStructure( u, u, HomStructure( a ) );
true
gap> IsEqualForObjects( HomStructure( Qoid ), DistinguishedObjectOfHomomorphismStructure( Qoid ) );
true
gap> c := FieldAsCategoryMorphism( 3, Qoid );;
gap> d := FieldAsCategoryMorphism( 0, Qoid );;
```

```

gap> left_coeffs := [ [ a, b ], [ c, d ] ];;
gap> right_coeffs := [ [ PreCompose( a, b ), PreCompose( b, c ) ], [ c, PreCompose( a, a ) ] ];;
gap> right_side := [ a, b ];;
gap> 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]
>       right_side[i]
>     )
> );
true
gap> Lift( c, d );
fail
gap> Lift( d, c );
0
gap> Colift( c, d );
0
gap> Colift( d, c );
fail

```

#### Example

```

gap> ReadPackage( "CAP", "examples/StringsAsCategory.g" );;
gap> C := StringsAsCategory();;
gap> obj1 := StringsAsCategoryObject( "qaeiou", C );;
gap> obj2 := StringsAsCategoryObject( "qxayeziouT", C );;
gap> mor := StringsAsCategoryMorphism( 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, 0 ) );
false
gap> ForAll( [5,6,7,8,9], i -> IsEqualForMorphismsOnMor( SimplifySource( mor, i ), SimplifySource( mor, 0 ) );
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( 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

```

### 11.3 Spectral Sequences

#### Example

```

gap> ZZ := HomalgRingOfIntegersInSingular( );
Z
gap> C1 := FreeLeftPresentation( 1, ZZ );
<An object in Category of left presentations of Z>
gap> C2 := FreeLeftPresentation( 2, ZZ );
<An object in Category of left presentations of Z>
gap> h1 := PresentationMorphism( C2, HomalgMatrix( [ [ 0 ], [ 4 ] ], ZZ ), C1 );
<A morphism in Category of left presentations of Z>
gap> h2 := PresentationMorphism( C2, HomalgMatrix( [ [ 0 ], [ 2 ] ], ZZ ), C1 );
<A morphism in Category of left presentations of Z>
gap> v1 := PresentationMorphism( C2, HomalgMatrix( [ [ 2, 0 ], [ 1, 2 ] ], ZZ ), C2 );
<A morphism in Category of left presentations of Z>
gap> v2 := PresentationMorphism( C1, HomalgMatrix( [ [ 4 ] ], ZZ ), C1 );
<A morphism in Category of left presentations of Z>
gap> cocomplex_h1 := CocomplexFromMorphismList( [ h1 ] );
<An object in Cocomplex category of Category of left presentations of Z>
gap> cocomplex_h2 := CocomplexFromMorphismList( [ h2 ] );
<An object in Cocomplex category of Category of left presentations of Z>
gap> cocomplex_mor := CochainMap( cocomplex_h2, [ v1, v2 ], cocomplex_h1 );
<A morphism in Cocomplex category of Category of left presentations of Z>
gap> Zmod := CapCategory( C1 );
Category of left presentations of Z
gap> CH0 := CohomologyFunctor( Zmod, 0 );
0-th cohomology functor of Category of left presentations of Z
gap> cmor0 := ApplyFunctor( CH0, cocomplex_mor );
<A morphism in Category of left presentations of Z>
gap> Display( UnderlyingMatrix( cmor0 ) );

```

```

2
gap> CH1 := CohomologyFunctor( Zmod, 1 );
1-th cohomology functor of Category of left presentations of Z
gap> cmor1 := ApplyFunctor( CH1, cocomplex_mor );
<A morphism in Category of left presentations of Z>
gap> Display( UnderlyingMatrix( cmor1 ) );
4
gap> ToComplex := CocomplexToComplexFunctor( Zmod );
Cocomplex to complex functor of Category of left presentations of Z
gap> complex_mor := ApplyFunctor( ToComplex, cocomplex_mor );
<A morphism in Complex category of Category of left presentations of Z>
gap> H0 := HomologyFunctor( Zmod, 0 );
0-th homology functor of Category of left presentations of Z
gap> mor0 := ApplyFunctor( H0, complex_mor );
<A morphism in Category of left presentations of Z>
gap> Display( UnderlyingMatrix( mor0 ) );
2
gap> Hm1 := HomologyFunctor( Zmod, -1 );
-1-th homology functor of Category of left presentations of Z
gap> mor1 := ApplyFunctor( Hm1, complex_mor );
<A morphism in Category of left presentations of Z>
gap> Display( UnderlyingMatrix( mor1 ) );
4

```

#### Example

```

gap> QQ := HomalgFieldOfRationalsInSingular( );
gap> R := QQ * "x,y";
Q[x,y]
gap> SetRecursionTrapInterval( 10000 );
gap> category := LeftPresentations( R );
Category of left presentations of Q[x,y]
gap> S := FreeLeftPresentation( 1, R );
<An object in Category of left presentations of Q[x,y]>
gap> object_func := function( i ) return S; end;
function( i ) ... end
gap> morphism_func := function( i ) return IdentityMorphism( S ); end;
function( i ) ... end
gap> C0 := ZFunctorObjectExtendedByInitialAndIdentity( object_func, morphism_func, category, 0, 4 );
<An object in Functors from integers into Category of left presentations of Q[x,y]>
gap> S2 := FreeLeftPresentation( 2, R );
<An object in Category of left presentations of Q[x,y]>
gap> C1 := ZFunctorObjectFromMorphismList( [ InjectionOfCofactorOfDirectSum( [ S2, S ], 1 ) ], 2 );
<An object in Functors from integers into Category of left presentations of Q[x,y]>
gap> C1 := ZFunctorObjectExtendedByInitialAndIdentity( C1, 2, 3 );
<An object in Functors from integers into Category of left presentations of Q[x,y]>
gap> C2 := ZFunctorObjectFromMorphismList( [ InjectionOfCofactorOfDirectSum( [ S, S ], 1 ) ], 3 );
<An object in Functors from integers into Category of left presentations of Q[x,y]>
gap> C2 := ZFunctorObjectExtendedByInitialAndIdentity( C2, 3, 4 );
<An object in Functors from integers into Category of left presentations of Q[x,y]>
gap> delta_1_3 := PresentationMorphism( C1[3], HomalgMatrix( [ [ "x^2" ], [ "xy" ], [ "y^3" ] ], 3 );
<A morphism in Category of left presentations of Q[x,y]>
gap> delta_1_2 := PresentationMorphism( C1[2], HomalgMatrix( [ [ "x^2" ], [ "xy" ] ], 2, 1, R ),
<A morphism in Category of left presentations of Q[x,y]>

```

```

gap> delta1 := ZFunctorMorphism( C1, [ UniversalMorphismFromInitialObject( C0[1] ), UniversalMorphismFromInitialObject( C0[2] ) ], UniversalMorphismFromInitialObject( C0[3] ) );
<A morphism in Functors from integers into Category of left presentations of Q[x,y]>
gap> delta1 := ZFunctorMorphismExtendedByInitialAndIdentity( delta1, 0, 3 );
<A morphism in Functors from integers into Category of left presentations of Q[x,y]>
gap> delta1 := AsAscendingFilteredMorphism( delta1 );
<A morphism in Ascending filtered object category of Category of left presentations of Q[x,y]>
gap> delta_2_3 := PresentationMorphism( C2[3], HomalgMatrix( [ [ "y", "-x", "0" ] ], 1, 3, R ), C2[3] );
<A morphism in Category of left presentations of Q[x,y]>
gap> delta_2_4 := PresentationMorphism( C2[4], HomalgMatrix( [ [ "y", "-x", "0" ] ], [ "0", "y^2", "x^2" ] ], C2[4] );
<A morphism in Category of left presentations of Q[x,y]>
gap> delta2 := ZFunctorMorphism( C2, [ UniversalMorphismFromInitialObject( C1[2] ), delta_2_3, delta_2_4 ], UniversalMorphismFromInitialObject( C1[3] ) );
<A morphism in Functors from integers into Category of left presentations of Q[x,y]>
gap> delta2 := ZFunctorMorphismExtendedByInitialAndIdentity( delta2, 2, 4 );
<A morphism in Functors from integers into Category of left presentations of Q[x,y]>
gap> delta2 := AsAscendingFilteredMorphism( delta2 );
<A morphism in Ascending filtered object category of Category of left presentations of Q[x,y]>
gap> SetIsAdditiveCategory( CategoryOfAscendingFilteredObjects( category ), true );
gap> complex := ZFunctorObjectFromMorphismList( [ delta2, delta1 ], -2 );
<An object in Functors from integers into Ascending filtered object category of Category of left presentations of Q[x,y]>
gap> complex := AsComplex( complex );
<An object in Complex category of Ascending filtered object category of Category of left presentations of Q[x,y]>
gap> LessGenFunctor := FunctorLessGeneratorsLeft( R );
Less generators for Category of left presentations of Q[x,y]
gap> s := SpectralSequenceEntryOfAscendingFilteredComplex( complex, 0, 0, 0 );
<A morphism in Generalized morphism category of Category of left presentations of Q[x,y]>
gap> Display( UnderlyingMatrix( ApplyFunctor( LessGenFunctor, UnderlyingHonestObject( Source( s ) ) ), 0, 1 );
(an empty 0 x 1 matrix)
gap> s := SpectralSequenceEntryOfAscendingFilteredComplex( complex, 1, 0, 0 );
<A morphism in Generalized morphism category of Category of left presentations of Q[x,y]>
gap> Display( UnderlyingMatrix( ApplyFunctor( LessGenFunctor, UnderlyingHonestObject( Source( s ) ) ), 0, 1 );
(an empty 0 x 1 matrix)
gap> s := SpectralSequenceEntryOfAscendingFilteredComplex( complex, 2, 0, 0 );
<A morphism in Generalized morphism category of Category of left presentations of Q[x,y]>
gap> Display( UnderlyingMatrix( ApplyFunctor( LessGenFunctor, UnderlyingHonestObject( Source( s ) ) ), 0, 1 );
(an empty 0 x 1 matrix)
gap> s := SpectralSequenceEntryOfAscendingFilteredComplex( complex, 3, 0, 0 );
<A morphism in Generalized morphism category of Category of left presentations of Q[x,y]>
gap> Display( UnderlyingMatrix( ApplyFunctor( LessGenFunctor, UnderlyingHonestObject( Source( s ) ) ), 0, 1 );
x*y,
x^2
gap> s := SpectralSequenceEntryOfAscendingFilteredComplex( complex, 4, 0, 0 );
<A morphism in Generalized morphism category of Category of left presentations of Q[x,y]>
gap> Display( UnderlyingMatrix( ApplyFunctor( LessGenFunctor, UnderlyingHonestObject( Source( s ) ) ), 0, 1 );
x*y,
x^2,
y^3
gap> s := SpectralSequenceEntryOfAscendingFilteredComplex( complex, 5, 0, 0 );
<A morphism in Generalized morphism category of Category of left presentations of Q[x,y]>
gap> Display( UnderlyingMatrix( ApplyFunctor( LessGenFunctor, UnderlyingHonestObject( Source( s ) ) ), 0, 1 );
x*y,
x^2,
y^3

```

```

gap> s := SpectralSequenceDifferentialOfAscendingFilteredComplex( complex, 3, 3, -2 );
<A morphism in Category of left presentations of Q[x,y]>
gap> Display( UnderlyingMatrix( ApplyFunctor( LessGenFunctor, s ) ) );
y^3
gap> AscToDescFunctor := AscendingToDescendingFilteredObjectFunctor( category );
Ascending to descending filtered object functor of Category of left presentations of Q[x,y]
gap> cocomplex := ZFunctorObjectFromMorphismList( [ ApplyFunctor( AscToDescFunctor, delta2 ), App
<An object in Functors from integers into Descending filtered object category of Category of left
gap> SetIsAdditiveCategory( CategoryOfDescendingFilteredObjects( category ), true );
gap> cocomplex := AsCocomplex( cocomplex );
<An object in Cocomplex category of Descending filtered object category of Category of left prese
gap> s := SpectralSequenceEntryOfDescendingFilteredCocomplex( cocomplex, 0, -2, 1 );
<A morphism in Generalized morphism category of Category of left presentations of Q[x,y]>
gap> Display( UnderlyingMatrix( ApplyFunctor( LessGenFunctor, UnderlyingHonestObject( Source( s )
(an empty 0 x 2 matrix)
gap> s := SpectralSequenceEntryOfDescendingFilteredCocomplex( cocomplex, 1, -2, 1 );
<A morphism in Generalized morphism category of Category of left presentations of Q[x,y]>
gap> Display( UnderlyingMatrix( ApplyFunctor( LessGenFunctor, UnderlyingHonestObject( Source( s )
(an empty 0 x 2 matrix)
gap> s := SpectralSequenceEntryOfDescendingFilteredCocomplex( cocomplex, 2, -2, 1 );
<A morphism in Generalized morphism category of Category of left presentations of Q[x,y]>
gap> Display( UnderlyingMatrix( ApplyFunctor( LessGenFunctor, UnderlyingHonestObject( Source( s )
-y,x
gap> s := SpectralSequenceEntryOfDescendingFilteredCocomplex( cocomplex, 3, -2, 1 );
<A morphism in Generalized morphism category of Category of left presentations of Q[x,y]>
gap> Display( UnderlyingMatrix( ApplyFunctor( LessGenFunctor, UnderlyingHonestObject( Source( s )
(an empty 0 x 0 matrix)
gap> s := SpectralSequenceDifferentialOfDescendingFilteredCocomplex( cocomplex, 2, -2, 1 );
<A morphism in Category of left presentations of Q[x,y]>
gap> Display( UnderlyingMatrix( ApplyFunctor( LessGenFunctor, s ) ) );
x^2,
x*y

```

## 11.4 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
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

```



## 11.5 Lifiable

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> IsLifiable( alpha, beta );
true
gap> IsLifiable( beta, alpha );
false
gap> IsLifiableAlongMonomorphism( beta, alpha );
true
gap> gamma := VectorSpaceMorphism( W, HomalgMatrix( [ [ 1 ], [ 1 ] ], 2, 1, field ), V );
gap> IsColifiable( beta, gamma );
true
gap> IsColifiable( gamma, beta );
false
gap> IsColifiableAlongEpimorphism( beta, gamma );
true
```

## 11.6 Monoidal Categories

Example

```
gap> ZZ := HomalgRingOfIntegers();
gap> M1 := AsLeftPresentation( HomalgMatrix( [ [ 2 ] ], 1, 1, ZZ ) );
<An object in Category of left presentations of Z>
gap> N1 := AsLeftPresentation( HomalgMatrix( [ [ 3 ] ], 1, 1, ZZ ) );
<An object in Category of left presentations of Z>
gap> T1 := TensorProductOnObjects( M1, N1 );
<An object in Category of left presentations of Z>
gap> Display( UnderlyingMatrix( T1 ) );
[ [ 3 ],
  [ 2 ] ]
gap> IsZeroForObjects( T1 );
true
gap> B1 := Braiding( DirectSum( M1, N1 ), DirectSum( M1, M1 ) );
<A morphism in Category of left presentations of Z>
gap> Display( UnderlyingMatrix( B1 ) );
[ [ 1, 0, 0, 0 ],
  [ 0, 0, 1, 0 ],
  [ 0, 1, 0, 0 ],
  [ 0, 0, 0, 1 ] ]
gap> IsWellDefined( B1 );
true
gap> U1 := TensorUnit( CapCategory( M1 ) );
<An object in Category of left presentations of Z>
gap> IntHom1 := InternalHomOnObjects( DirectSum( M1, U1 ), N1 );
<An object in Category of left presentations of Z>
gap> Display( UnderlyingMatrix( IntHom1 ) );
[ [ -2, -1 ],
  [ 1, -1 ] ]
```

```

gap> generator_l1 := StandardGeneratorMorphism( 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 ) );
[ [ 0 ],
  [ 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 ],
  [ 2 ] ]
gap> IsEqualForMorphisms( LambdaIntroduction( morphism_l1 ), generator_l1 );
false
gap> IsCongruentForMorphisms( LambdaIntroduction( morphism_l1 ), generator_l1 );
true
gap> IsEqualForMorphisms( LambdaIntroduction( morphism_l2 ), generator_l2 );
false
gap> IsCongruentForMorphisms( LambdaIntroduction( morphism_l2 ), generator_l2 );
true
gap> Mr := AsRightPresentation( HomalgMatrix( [ [ 2 ] ], 1, 1, ZZ ) );
<An object in Category of right presentations of Z>
gap> Nr := AsRightPresentation( HomalgMatrix( [ [ 3 ] ], 1, 1, ZZ ) );
<An object in Category of right presentations of Z>
gap> Tr := TensorProductOnObjects( Mr, Nr );
<An object in Category of right presentations of Z>
gap> Display( UnderlyingMatrix( Tr ) );
[ [ 3, 2 ] ]
gap> IsZeroForObjects( Tr );
true
gap> Br := Braiding( DirectSum( Mr, Nr ), DirectSum( Mr, Mr ) );
<A morphism in Category of right presentations of Z>
gap> Display( UnderlyingMatrix( Br ) );
[ [ 1, 0, 0, 0 ],
  [ 0, 0, 1, 0 ],
  [ 0, 1, 0, 0 ],
  [ 0, 0, 0, 1 ] ]
gap> IsWellDefined( Br );
true
gap> Ur := TensorUnit( CapCategory( Mr ) );
<An object in Category of right presentations of Z>
gap> IntHomr := InternalHomOnObjects( DirectSum( Mr, Ur ), Nr );
<An object in Category of right presentations of Z>
gap> Display( UnderlyingMatrix( IntHomr ) );
[ [ -2, 1 ],
  [ -1, -1 ] ]
gap> generator_r1 := StandardGeneratorMorphism( IntHomr, 1 );
<A morphism in Category of right presentations of Z>
gap> morphism_r1 := LambdaElimination( DirectSum( Mr, Ur ), Nr, generator_r1 );
<A morphism in Category of right presentations of Z>

```

```

gap> Display( UnderlyingMatrix( morphism_r1 ) );
[ [ 0, 2 ] ]
gap> generator_r2 := StandardGeneratorMorphism( IntHom1, 2 );
<A morphism in Category of left presentations of Z>
gap> morphism_r2 := LambdaElimination( DirectSum( M1, U1 ), N1, generator_r2 );
<A morphism in Category of left presentations of Z>
gap> Display( UnderlyingMatrix( morphism_r2 ) );
[ [ 0 ],
  [ 2 ] ]
gap> IsEqualForMorphisms( LambdaIntroduction( morphism_r1 ), generator_r1 );
false
gap> IsCongruentForMorphisms( LambdaIntroduction( morphism_r1 ), generator_r1 );
true
gap> IsEqualForMorphisms( LambdaIntroduction( morphism_r2 ), generator_r2 );
false
gap> IsCongruentForMorphisms( LambdaIntroduction( morphism_r2 ), generator_r2 );
true

```

## 11.7 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 ) );
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

```

## 11.8 Opposite category

Example

```

gap> QQ := HomalgFieldOfRationals();;
gap> vec := MatrixCategory( QQ );;
gap> V1 := Opposite( TensorUnit( vec ) );;
gap> V2 := DirectSum( V1, V1 );;
gap> V3 := DirectSum( V1, V2 );;
gap> V4 := DirectSum( V1, V3 );;
gap> V5 := DirectSum( V1, V4 );;
gap> alpha13 := InjectionOfCofactorOfDirectSum( [ V1, V2 ], 1 );;
gap> alpha14 := InjectionOfCofactorOfDirectSum( [ V1, V2, V1 ], 3 );;
gap> alpha15 := InjectionOfCofactorOfDirectSum( [ V2, V1, V2 ], 2 );;
gap> alpha23 := InjectionOfCofactorOfDirectSum( [ V2, V1 ], 1 );;
gap> alpha24 := InjectionOfCofactorOfDirectSum( [ V1, V2, V1 ], 2 );;
gap> alpha25 := InjectionOfCofactorOfDirectSum( [ V2, V2, V1 ], 1 );;
gap> mat := [
>      [ alpha13, alpha14, alpha15 ],
>      [ alpha23, alpha24, alpha25 ]
> ];;
gap> mor := MorphismBetweenDirectSums( mat );;
gap> IsWellDefined( mor );
true
gap> IsWellDefined( Opposite( mor ) );
true
gap> IsOne( UniversalMorphismFromImage( mor, [ CostrictionToImage( mor ), ImageEmbedding( mor ) ] ) );
true

```

## 11.9 Generalized Morphisms Category

Example

```

gap> vecspaces := CreateCapCategory( "VectorSpacesForGeneralizedMorphismsTest" );
VectorSpacesForGeneralizedMorphismsTest
gap> ReadPackage( "CAP", "examples/VectorSpacesAllMethods.g" );
true
gap> LoadPackage( "GeneralizedMorphismsForCAP" );
true
gap> B := QVectorSpace( 2 );
<A rational vector space of dimension 2>
gap> C := QVectorSpace( 3 );
<A rational vector space of dimension 3>

```

```

gap> B_1 := QVectorSpace( 1 );
<A rational vector space of dimension 1>
gap> C_1 := QVectorSpace( 2 );
<A rational vector space of dimension 2>
gap> c1_source_aid := VectorSpaceMorphism( B_1, [ [ 1, 0 ] ], B );
A rational vector space homomorphism with matrix:
[ [ 1, 0 ] ]

gap> SetIsSubobject( c1_source_aid, true );
gap> c1_range_aid := VectorSpaceMorphism( C, [ [ 1, 0 ], [ 0, 1 ], [ 0, 0 ] ], C_1 );
A rational vector space homomorphism with matrix:
[ [ 1, 0 ],
  [ 0, 1 ],
  [ 0, 0 ] ]

gap> SetIsFactorobject( c1_range_aid, true );
gap> c1_associated := VectorSpaceMorphism( B_1, [ [ 1, 1 ] ], C_1 );
A rational vector space homomorphism with matrix:
[ [ 1, 1 ] ]

gap> c1 := GeneralizedMorphism( c1_source_aid, c1_associated, c1_range_aid );
<A morphism in Generalized morphism category of VectorSpacesForGeneralizedMorphismsTest>
gap> B_2 := QVectorSpace( 1 );
<A rational vector space of dimension 1>
gap> C_2 := QVectorSpace( 2 );
<A rational vector space of dimension 2>
gap> c2_source_aid := VectorSpaceMorphism( B_2, [ [ 2, 0 ] ], B );
A rational vector space homomorphism with matrix:
[ [ 2, 0 ] ]

gap> SetIsSubobject( c2_source_aid, true );
gap> c2_range_aid := VectorSpaceMorphism( C, [ [ 3, 0 ], [ 0, 3 ], [ 0, 0 ] ], C_2 );
A rational vector space homomorphism with matrix:
[ [ 3, 0 ],
  [ 0, 3 ],
  [ 0, 0 ] ]

gap> SetIsFactorobject( c2_range_aid, true );
gap> c2_associated := VectorSpaceMorphism( B_2, [ [ 6, 6 ] ], C_2 );
A rational vector space homomorphism with matrix:
[ [ 6, 6 ] ]

gap> c2 := GeneralizedMorphism( c2_source_aid, c2_associated, c2_range_aid );
<A morphism in Generalized morphism category of VectorSpacesForGeneralizedMorphismsTest>
gap> IsCongruentForMorphisms( c1, c2 );
true
gap> IsCongruentForMorphisms( c1, c1 );
true
gap> c3_associated := VectorSpaceMorphism( B_1, [ [ 2, 2 ] ], C_1 );
A rational vector space homomorphism with matrix:
[ [ 2, 2 ] ]

```

```

gap> c3 := GeneralizedMorphism( c1_source_aid, c3_associated, c1_range_aid );
<A morphism in Generalized morphism category of VectorSpacesForGeneralizedMorphismsTest>
gap> IsCongruentForMorphisms( c1, c3 );
false
gap> IsCongruentForMorphisms( c2, c3 );
false
gap> c1 + c2;
<A morphism in Generalized morphism category of VectorSpacesForGeneralizedMorphismsTest>
gap> Arrow( c1 + c2 );
A rational vector space homomorphism with matrix:
[ [ 12, 12 ] ]

```

First composition test:

Example

```

gap> vecspaces := CreateCapCategory( "VectorSpacesForGeneralizedMorphismsTest" );
VectorSpacesForGeneralizedMorphismsTest
gap> ReadPackage( "CAP", "examples/VectorSpacesAllMethods.g" );
true
gap> A := QVectorSpace( 1 );
<A rational vector space of dimension 1>
gap> B := QVectorSpace( 2 );
<A rational vector space of dimension 2>
gap> C := QVectorSpace( 3 );
<A rational vector space of dimension 3>
gap> phi_tilde_associated := VectorSpaceMorphism( A, [ [ 1, 2, 0 ] ], C );
A rational vector space homomorphism with matrix:
[ [ 1, 2, 0 ] ]

gap> phi_tilde_source_aid := VectorSpaceMorphism( A, [ [ 1, 2 ] ], B );
A rational vector space homomorphism with matrix:
[ [ 1, 2 ] ]

gap> phi_tilde := GeneralizedMorphismWithSourceAid( phi_tilde_source_aid, phi_tilde_associated );
<A morphism in Generalized morphism category of VectorSpacesForGeneralizedMorphismsTest>
gap> psi_tilde_associated := IdentityMorphism( B );
A rational vector space homomorphism with matrix:
[ [ 1, 0 ],
  [ 0, 1 ] ]

gap> psi_tilde_source_aid := VectorSpaceMorphism( B, [ [ 1, 0, 0 ], [ 0, 1, 0 ] ], C );
A rational vector space homomorphism with matrix:
[ [ 1, 0, 0 ],
  [ 0, 1, 0 ] ]

gap> psi_tilde := GeneralizedMorphismWithSourceAid( psi_tilde_source_aid, psi_tilde_associated );
<A morphism in Generalized morphism category of VectorSpacesForGeneralizedMorphismsTest>
gap> composition := PreCompose( phi_tilde, psi_tilde );
<A morphism in Generalized morphism category of VectorSpacesForGeneralizedMorphismsTest>
gap> Arrow( composition );
A rational vector space homomorphism with matrix:
[ [ 1/2, 1 ] ]

```

```
gap> SourceAid( composition );
A rational vector space homomorphism with matrix:
[ [ 1/2, 1 ] ]

gap> RangeAid( composition );
A rational vector space homomorphism with matrix:
[ [ 1, 0 ],
  [ 0, 1 ] ]
```

## Second composition test

### Example

```
gap> vecspaces := CreateCapCategory( "VectorSpacesForGeneralizedMorphismsTest" );
VectorSpacesForGeneralizedMorphismsTest
gap> ReadPackage( "CAP", "examples/VectorSpacesAllMethods.g" );
true
gap> A := QVectorSpace( 1 );
<A rational vector space of dimension 1>
gap> B := QVectorSpace( 2 );
<A rational vector space of dimension 2>
gap> C := QVectorSpace( 3 );
<A rational vector space of dimension 3>
gap> phi2_tilde_associated := VectorSpaceMorphism( A, [ [ 1, 5 ] ], B );
A rational vector space homomorphism with matrix:
[ [ 1, 5 ] ]

gap> phi2_tilde_range_aid := VectorSpaceMorphism( C, [ [ 1, 0 ], [ 0, 1 ], [ 1, 1 ] ], B );
A rational vector space homomorphism with matrix:
[ [ 1, 0 ],
  [ 0, 1 ],
  [ 1, 1 ] ]

gap> phi2_tilde := GeneralizedMorphismWithRangeAid( phi2_tilde_associated, phi2_tilde_range_aid );
<A morphism in Generalized morphism category of VectorSpacesForGeneralizedMorphismsTest>
gap> psi2_tilde_associated := VectorSpaceMorphism( C, [ [ 1 ], [ 3 ], [ 4 ] ], A );
A rational vector space homomorphism with matrix:
[ [ 1 ],
  [ 3 ],
  [ 4 ] ]

gap> psi2_tilde_range_aid := VectorSpaceMorphism( B, [ [ 1 ], [ 1 ] ], A );
A rational vector space homomorphism with matrix:
[ [ 1 ],
  [ 1 ] ]

gap> psi2_tilde := GeneralizedMorphismWithRangeAid( psi2_tilde_associated, psi2_tilde_range_aid );
<A morphism in Generalized morphism category of VectorSpacesForGeneralizedMorphismsTest>
gap> composition2 := PreCompose( phi2_tilde, psi2_tilde );
<A morphism in Generalized morphism category of VectorSpacesForGeneralizedMorphismsTest>
gap> Arrow( composition2 );
A rational vector space homomorphism with matrix:
[ [ 16 ] ]
```

```
gap> RangeAid( composition2 );
A rational vector space homomorphism with matrix:
[ [ 1 ],
  [ 1 ] ]

gap> SourceAid( composition2 );
A rational vector space homomorphism with matrix:
[ [ 1 ] ]
```

### Third composition test

Example

```
gap> vecspaces := CreateCapCategory( "VectorSpacesForGeneralizedMorphismsTest" );
VectorSpacesForGeneralizedMorphismsTest
gap> ReadPackage( "CAP", "examples/VectorSpacesAllMethods.g" );
true
gap> A := QVectorSpace( 3 );
<A rational vector space of dimension 3>
gap> Asub := QVectorSpace( 2 );
<A rational vector space of dimension 2>
gap> B := QVectorSpace( 3 );
<A rational vector space of dimension 3>
gap> Bfac := QVectorSpace( 1 );
<A rational vector space of dimension 1>
gap> Bsub := QVectorSpace( 2 );
<A rational vector space of dimension 2>
gap> C := QVectorSpace( 3 );
<A rational vector space of dimension 3>
gap> Cfac := QVectorSpace( 1 );
<A rational vector space of dimension 1>
gap> Asub_into_A := VectorSpaceMorphism( Asub, [ [ 1, 0, 0 ], [ 0, 1, 0 ] ], A );
A rational vector space homomorphism with matrix:
[ [ 1, 0, 0 ],
  [ 0, 1, 0 ] ]

gap> Asub_to_Bfac := VectorSpaceMorphism( Asub, [ [ 1 ], [ 1 ] ], Bfac );
A rational vector space homomorphism with matrix:
[ [ 1 ],
  [ 1 ] ]

gap> B_onto_Bfac := VectorSpaceMorphism( B, [ [ 1 ], [ 1 ], [ 1 ] ], Bfac );
A rational vector space homomorphism with matrix:
[ [ 1 ],
  [ 1 ],
  [ 1 ] ]

gap> Bsub_into_B := VectorSpaceMorphism( Bsub, [ [ 2, 2, 0 ], [ 0, 2, 2 ] ], B );
A rational vector space homomorphism with matrix:
[ [ 2, 2, 0 ],
  [ 0, 2, 2 ] ]

gap> Bsub_to_Cfac := VectorSpaceMorphism( Bsub, [ [ 3 ], [ 0 ] ], Cfac );
A rational vector space homomorphism with matrix:
[ [ 3 ],
```



```

[ 0 ] ]

gap> C_onto_Cfac := VectorSpaceMorphism( C, [ [ 1 ], [ 2 ], [ 3 ] ], Cfac );
A rational vector space homomorphism with matrix:
[ [ 1 ],
  [ 2 ],
  [ 3 ] ]

gap> generalized_morphism1 := GeneralizedMorphism( Asub_into_A, Asub_to_Bfac, B_onto_Bfac );
<A morphism in Generalized morphism category of VectorSpacesForGeneralizedMorphismsTest>
gap> generalized_morphism2 := GeneralizedMorphism( Bsub_into_B, Bsub_to_Cfac, C_onto_Cfac );
<A morphism in Generalized morphism category of VectorSpacesForGeneralizedMorphismsTest>
gap> IsWellDefined( generalized_morphism1 );
true
gap> IsWellDefined( generalized_morphism2 );
true
gap> p := PreCompose( generalized_morphism1, generalized_morphism2 );
<A morphism in Generalized morphism category of VectorSpacesForGeneralizedMorphismsTest>
gap> SourceAid( p );
A rational vector space homomorphism with matrix:
[ [ -1,  1,  0 ],
  [  1,  0,  0 ] ]

gap> Arrow( p );
A rational vector space homomorphism with matrix:
(an empty 2 x 0 matrix)

gap> RangeAid( p );
A rational vector space homomorphism with matrix:
(an empty 3 x 0 matrix)
gap> A := QVectorSpace( 3 );
<A rational vector space of dimension 3>
gap> Asub := QVectorSpace( 2 );
<A rational vector space of dimension 2>
gap> B := QVectorSpace( 3 );
<A rational vector space of dimension 3>
gap> Bfac := QVectorSpace( 1 );
<A rational vector space of dimension 1>
gap> Bsub := QVectorSpace( 2 );
<A rational vector space of dimension 2>
gap> C := QVectorSpace( 3 );
<A rational vector space of dimension 3>
gap> Cfac := QVectorSpace( 2 );
<A rational vector space of dimension 2>
gap> Asub_into_A := VectorSpaceMorphism( Asub, [ [ 1, 0, 0 ], [ 0, 1, 0 ] ], A );
A rational vector space homomorphism with matrix:
[ [ 1,  0,  0 ],
  [ 0,  1,  0 ] ]

gap> Asub_to_Bfac := VectorSpaceMorphism( Asub, [ [ 1 ], [ 1 ] ], Bfac );
A rational vector space homomorphism with matrix:
[ [ 1 ],

```

```

[ 1 ] ]

gap> B_onto_Bfac := VectorSpaceMorphism( B, [ [ 1 ], [ 1 ], [ 1 ] ], Bfac );
A rational vector space homomorphism with matrix:
[ [ 1 ],
  [ 1 ],
  [ 1 ] ]

gap> Bsub_into_B := VectorSpaceMorphism( Bsub, [ [ 2, 2, 0 ], [ 0, 2, 2 ] ], B );
A rational vector space homomorphism with matrix:
[ [ 2, 2, 0 ],
  [ 0, 2, 2 ] ]

gap> Bsub_to_Cfac := VectorSpaceMorphism( Bsub, [ [ 3, 3 ], [ 0, 0 ] ], Cfac );
A rational vector space homomorphism with matrix:
[ [ 3, 3 ],
  [ 0, 0 ] ]

gap> C_onto_Cfac := VectorSpaceMorphism( C, [ [ 1, 0 ], [ 0, 2 ], [ 3, 3 ] ], Cfac );
A rational vector space homomorphism with matrix:
[ [ 1, 0 ],
  [ 0, 2 ],
  [ 3, 3 ] ]

gap> generalized_morphism1 := GeneralizedMorphism( Asub_into_A, Asub_to_Bfac, B_onto_Bfac );
<A morphism in Generalized morphism category of VectorSpacesForGeneralizedMorphismsTest>
gap> generalized_morphism2 := GeneralizedMorphism( Bsub_into_B, Bsub_to_Cfac, C_onto_Cfac );
<A morphism in Generalized morphism category of VectorSpacesForGeneralizedMorphismsTest>
gap> IsWellDefined( generalized_morphism1 );
true
gap> IsWellDefined( generalized_morphism2 );
true
gap> p := PreCompose( generalized_morphism1, generalized_morphism2 );
<A morphism in Generalized morphism category of VectorSpacesForGeneralizedMorphismsTest>
gap> SourceAid( p );
A rational vector space homomorphism with matrix:
[ [ -1, 1, 0 ],
  [ 1, 0, 0 ] ]

gap> Arrow( p );
A rational vector space homomorphism with matrix:
[ [ 0 ],
  [ 0 ] ]

gap> RangeAid( p );
A rational vector space homomorphism with matrix:
[ [ -1 ],
  [ 2 ],
  [ 0 ] ]

```

Honest representative test

Example

```
gap> vecspaces := CreateCapCategory( "VectorSpacesForGeneralizedMorphismsTest" );
```

```

VectorSpacesForGeneralizedMorphismsTest
gap> ReadPackage( "CAP", "examples/VectorSpacesAllMethods.g" );
true
gap> A := QVectorSpace( 1 );
<A rational vector space of dimension 1>
gap> B := QVectorSpace( 2 );
<A rational vector space of dimension 2>
gap> phi_tilde_source_aid := VectorSpaceMorphism( A, [ [ 2 ] ], A );
A rational vector space homomorphism with matrix:
[ [ 2 ] ]

gap> phi_tilde_associated := VectorSpaceMorphism( A, [ [ 1, 1 ] ], B );
A rational vector space homomorphism with matrix:
[ [ 1, 1 ] ]

gap> phi_tilde_range_aid := VectorSpaceMorphism( B, [ [ 1, 2 ], [ 3, 4 ] ], B );
A rational vector space homomorphism with matrix:
[ [ 1, 2 ],
  [ 3, 4 ] ]

gap> phi_tilde := GeneralizedMorphism( phi_tilde_source_aid, phi_tilde_associated, phi_tilde_range_aid );
<A morphism in Generalized morphism category of VectorSpacesForGeneralizedMorphismsTest>
gap> HonestRepresentative( phi_tilde );
A rational vector space homomorphism with matrix:
[ [ -1/4, 1/4 ] ]

gap> IsWellDefined( phi_tilde );
true
gap> IsWellDefined( psi_tilde );
true

```

## 11.10 IsWellDefined

Example

```

gap> vecspaces := CreateCapCategory( "VectorSpacesForIsWellDefinedTest" );
VectorSpacesForIsWellDefinedTest
gap> ReadPackage( "CAP", "examples/VectorSpacesAllMethods.g" );
true
gap> LoadPackage( "GeneralizedMorphismsForCAP" );
true
gap> A := QVectorSpace( 1 );
<A rational vector space of dimension 1>
gap> B := QVectorSpace( 2 );
<A rational vector space of dimension 2>
gap> alpha := VectorSpaceMorphism( A, [ [ 1, 2 ] ], B );
A rational vector space homomorphism with matrix:
[ [ 1, 2 ] ]

gap> g := GeneralizedMorphism( alpha, alpha, alpha );
<A morphism in Generalized morphism category of VectorSpacesForIsWellDefinedTest>
gap> IsWellDefined( alpha );
true

```

```
gap> IsWellDefined( g );
true
```

## 11.11 Kernel

Example

```
gap> vecspaces := CreateCapCategory( "VectorSpaces01" );
VectorSpaces01
gap> ReadPackage( "CAP", "examples/VectorSpacesAddKernel01.g" );
true
gap> V := QVectorSpace( 2 );
<A rational vector space of dimension 2>
gap> W := QVectorSpace( 3 );
<A rational vector space of dimension 3>
gap> alpha := VectorSpaceMorphism( V, [ [ 1, 1, 1 ], [ -1, -1, -1 ] ], W );
A rational vector space homomorphism with matrix:
[ [ 1, 1, 1 ],
  [ -1, -1, -1 ] ]

gap> k := KernelObject( alpha );
<A rational vector space of dimension 1>
gap> T := QVectorSpace( 2 );
<A rational vector space of dimension 2>
gap> tau := VectorSpaceMorphism( T, [ [ 2, 2 ], [ 2, 2 ] ], V );
A rational vector space homomorphism with matrix:
[ [ 2, 2 ],
  [ 2, 2 ] ]

gap> k_lift := Kernellift( alpha, tau );
A rational vector space homomorphism with matrix:
[ [ 2 ],
  [ 2 ] ]

gap> HasKernelEmbedding( alpha );
false
gap> KernelEmbedding( alpha );
A rational vector space homomorphism with matrix:
[ [ 1, 1 ] ]
```

Example

```
gap> vecspaces := CreateCapCategory( "VectorSpaces02" );
VectorSpaces02
gap> ReadPackage( "CAP", "examples/VectorSpacesAddKernel02.g" );
true
gap> V := QVectorSpace( 2 );
<A rational vector space of dimension 2>
gap> W := QVectorSpace( 3 );
<A rational vector space of dimension 3>
gap> alpha := VectorSpaceMorphism( V, [ [ 1, 1, 1 ], [ -1, -1, -1 ] ], W );
A rational vector space homomorphism with matrix:
[ [ 1, 1, 1 ],
  [ -1, -1, -1 ] ]
```

```

gap> k := KernelObject( alpha );
<A rational vector space of dimension 1>
gap> T := QVectorSpace( 2 );
<A rational vector space of dimension 2>
gap> tau := VectorSpaceMorphism( T, [ [ 2, 2 ], [ 2, 2 ] ], V );
A rational vector space homomorphism with matrix:
[ [ 2, 2 ],
  [ 2, 2 ] ]

gap> k_lift := KernelLift( alpha, tau );
A rational vector space homomorphism with matrix:
[ [ 2 ],
  [ 2 ] ]

gap> HasKernelEmbedding( alpha );
false

```

#### Example

```

gap> vecspaces := CreateCapCategory( "VectorSpaces03" );
VectorSpaces03
gap> ReadPackage( "CAP", "examples/VectorSpacesAddKernel03.g" );
true
gap> V := QVectorSpace( 2 );
<A rational vector space of dimension 2>
gap> W := QVectorSpace( 3 );
<A rational vector space of dimension 3>
gap> alpha := VectorSpaceMorphism( V, [ [ 1, 1, 1 ], [ -1, -1, -1 ] ], W );
A rational vector space homomorphism with matrix:
[ [ 1, 1, 1 ],
  [ -1, -1, -1 ] ]

gap> k := KernelObject( alpha );
<A rational vector space of dimension 1>
gap> k_emb := KernelEmbedding( alpha );
A rational vector space homomorphism with matrix:
[ [ 1, 1 ] ]

gap> IsIdenticalObj( Source( k_emb ), k );
true
gap> V := QVectorSpace( 2 );
<A rational vector space of dimension 2>
gap> W := QVectorSpace( 3 );
<A rational vector space of dimension 3>
gap> beta := VectorSpaceMorphism( V, [ [ 1, 1, 1 ], [ -1, -1, -1 ] ], W );
A rational vector space homomorphism with matrix:
[ [ 1, 1, 1 ],
  [ -1, -1, -1 ] ]

gap> k_emb := KernelEmbedding( beta );
A rational vector space homomorphism with matrix:
[ [ 1, 1 ] ]

```

```
gap> IsIdenticalObj( Source( k_emb ), KernelObject( beta ) );
true
```

## 11.12 FiberProduct

Example

```
gap> vecspaces := CreateCapCategory( "VectorSpacesForFiberProductTest" );
VectorSpacesForFiberProductTest
gap> ReadPackage( "CAP", "examples/VectorSpacesAllMethods.g" );
true
gap> A := QVectorSpace( 1 );
<A rational vector space of dimension 1>
gap> B := QVectorSpace( 2 );
<A rational vector space of dimension 2>
gap> C := QVectorSpace( 3 );
<A rational vector space of dimension 3>
gap> AtoC := VectorSpaceMorphism( A, [ [ 1, 2, 0 ] ], C );
A rational vector space homomorphism with matrix:
[ [ 1, 2, 0 ] ]

gap> BtoC := VectorSpaceMorphism( B, [ [ 1, 0, 0 ], [ 0, 1, 0 ] ], C );
A rational vector space homomorphism with matrix:
[ [ 1, 0, 0 ],
  [ 0, 1, 0 ] ]

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

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

# Index

- ActivateDerivationInfo, [135](#)
- Add
  - for IsCapCategory, IsCapCategoryMorphism, [30](#)
  - for IsCapCategory, IsCapCategoryObject, [17](#)
  - for IsStringMinHeap, IsString, IsInt, [142](#)
- AddAdditionForMorphisms
  - for IsCapCategory, IsFunction, [32](#)
- AddAdditiveInverseForMorphisms
  - for IsCapCategory, IsFunction, [33](#)
- AddAstrictionToCoimage
  - for IsCapCategory, IsFunction, [124](#)
- AddAstrictionToCoimageWithGivenCoimage
  - for IsCapCategory, IsFunction, [124](#)
- AddBasisOfExternalHom
  - for IsCapCategory, IsFunction, [45](#)
- AddCanonicalIdentificationFromCoimageToImageObject
  - for IsCapCategory, IsFunction, [125](#)
- AddCanonicalIdentificationFromImageObjectToCoimage
  - for IsCapCategory, IsFunction, [125](#)
- AddCategoricalProperty, [6](#)
- AddCoastrictionToImage
  - for IsCapCategory, IsFunction, [119](#)
- AddCoastrictionToImageWithGivenImageObject
  - for IsCapCategory, IsFunction, [119](#)
- AddCoefficientsOfMorphismWithGivenBasisOfExternalHom
  - for IsCapCategory, IsFunction, [46](#)
- AddCoequalizer
  - for IsCapCategory, IsFunction, [99](#)
- AddCoequalizerFunctorialWithGivenCoequalizers
  - for IsCapCategory, IsFunction, [101](#)
- AddCoimage
  - for IsCapCategory, IsFunction, [123](#)
- AddCoimageProjection
  - for IsCapCategory, IsFunction, [124](#)
- AddCoimageProjectionWithGivenCoimage
  - for IsCapCategory, IsFunction, [124](#)
- AddCokernelColift
  - for IsCapCategory, IsFunction, [69](#)
- AddCokernelColiftWithGivenCokernelObject
  - for IsCapCategory, IsFunction, [69](#)
- AddCokernelObject
  - for IsCapCategory, IsFunction, [68](#)
- AddCokernelObjectFunctorialWithGivenCokernelObjects
  - for IsCapCategory, IsFunction, [70](#)
- AddCokernelProjection
  - for IsCapCategory, IsFunction, [69](#)
- AddCokernelProjectionWithGivenCokernelObject
  - for IsCapCategory, IsFunction, [69](#)
- AddColift
  - for IsCapCategory, IsFunction, [39](#)
- AddColiftAlongEpimorphism
  - for IsCapCategory, IsFunction, [38](#)
- AddComponentOfMorphismFromDirectSum
  - for IsCapCategory, IsFunction, [83](#)
- AddComponentOfMorphismIntoDirectSum
  - for IsCapCategory, IsFunction, [83](#)
- AddCoproduct
  - for IsCapCategory, IsFunction, [88](#)
- AddCoproductFunctorialWithGivenCoproducts
  - for IsCapCategory, IsFunction, [89](#)
- AddDerivation
  - for IsDerivedMethodGraph, IsDerivedMethod, [138](#)
  - for IsDerivedMethodGraph, IsFunction, IsDenseList, [138](#)
  - for IsDerivedMethodGraph, IsFunction, Is-

- DenseList, IsObject, [138](#)
- for IsDerivedMethodGraph, IsFunction, IsFunction, [139](#)
- AddDerivationPair
  - for IsDerivedMethodGraph, IsFunction, IsFunction, IsDenseList, IsDenseList, [139](#)
  - for IsDerivedMethodGraph, IsFunction, IsFunction, IsDenseList, IsDenseList, IsDenseList, [139](#)
  - for IsDerivedMethodGraph, IsFunction, IsFunction, IsDenseList, IsFunction, IsFunction, [139](#)
  - for IsDerivedMethodGraph, IsFunction, IsFunction, IsFunction, IsFunction, [139](#)
- AddDerivationPairToCAP, [139](#)
- AddDerivationToCAP, [139](#)
- AddDirectProduct
  - for IsCapCategory, IsFunction, [91](#)
- AddDirectProductFunctorialWithGivenDirectProducts
  - for IsCapCategory, IsFunction, [93](#)
- AddDirectSum
  - for IsCapCategory, IsFunction, [85](#)
- AddDirectSumCodiagonalDifference
  - for IsCapCategory, IsFunction, [112](#)
- AddDirectSumDiagonalDifference
  - for IsCapCategory, IsFunction, [104](#)
- AddDirectSumFunctorialWithGivenDirectSums
  - for IsCapCategory, IsFunction, [86](#)
- AddDirectSumProjectionInPushout
  - for IsCapCategory, IsFunction, [113](#)
- AddDistinguishedObjectOfHomomorphismStructure
  - for IsCapCategory, IsFunction, [43](#)
- AddEmbeddingOfEqualizer
  - for IsCapCategory, IsFunction, [95](#)
- AddEmbeddingOfEqualizerWithGivenEqualizer
  - for IsCapCategory, IsFunction, [95](#)
- AddEpimorphismFromSomeProjectiveObject
  - for IsCapCategory, IsFunction, [19](#)
- AddEpimorphismFromSomeProjectiveObjectWithGivenSomeProjectiveObject
  - for IsCapCategory, IsFunction, [19](#)
- AddEqualizer
  - for IsCapCategory, IsFunction, [95](#)
- AddEqualizerFunctorialWithGivenEqualizers
  - for IsCapCategory, IsFunction, [97](#)
- AddFiberProduct
  - for IsCapCategory, IsFunction, [107](#)
- AddFiberProductEmbeddingInDirectSum
  - for IsCapCategory, IsFunction, [105](#)
- AddFiberProductFunctorialWithGivenFiberProducts
  - for IsCapCategory, IsFunction, [109](#)
- AddHomologyObject
  - for IsCapCategory, IsFunction, [127](#)
- AddHomologyObjectFunctorialWithGivenHomologyObjects
  - for IsCapCategory, IsFunction, [127](#)
- AddHomomorphismStructureOnMorphismsWithGivenObjects
  - for IsCapCategory, IsFunction, [42](#)
- AddHomomorphismStructureOnObjects
  - for IsCapCategory, IsFunction, [42](#)
- AddHorizontalPostCompose
  - for IsCapCategory, IsFunction, [55](#)
- AddHorizontalPreCompose
  - for IsCapCategory, IsFunction, [55](#)
- AddIdentityMorphism
  - for IsCapCategory, IsFunction, [36](#)
- AddIdentityTwoCell
  - for IsCapCategory, IsFunction, [54](#)
- AddImageEmbedding
  - for IsCapCategory, IsFunction, [119](#)
- AddImageEmbeddingWithGivenImageObject
  - for IsCapCategory, IsFunction, [119](#)
- AddImageObject
  - for IsCapCategory, IsFunction, [119](#)
- AddInitialObject
  - for IsCapCategory, IsFunction, [77](#)
- AddInitialObjectFunctorial
  - for IsCapCategory, IsFunction, [77](#)
- AddInjectionOfCofactorOfCoproduct
  - for IsCapCategory, IsFunction, [88](#)
- AddInjectionOfCofactorOfCoproductWithGivenCoproduct
  - for IsCapCategory, IsFunction, [89](#)
- AddInjectionOfCofactorOfDirectSum



- for IsCapCategory, IsFunction, [84](#)
- AddInjectionOfCofactorOfDirectSumWith-  
GivenDirectSum  
for IsCapCategory, IsFunction, [84](#)
- AddInjectionOfCofactorOfPushout  
for IsCapCategory, IsFunction, [115](#)
- AddInjectionOfCofactorOfPushoutWith-  
GivenPushout  
for IsCapCategory, IsFunction, [115](#)
- AddInjectiveColift  
for IsCapCategory, IsFunction, [20](#)
- AddInterpretMorphismAsMorphism-  
FromDistinguishedObjectTo-  
HomomorphismStructure  
for IsCapCategory, IsFunction, [43](#)
- AddInterpretMorphismFromDistinguished-  
ObjectToHomomorphismStructure-  
AsMorphism  
for IsCapCategory, IsFunction, [44](#)
- AddInverse  
for IsCapCategory, IsFunction, [41](#)
- AddInverseMorphismFromCoimageToImage-  
WithGivenObjects  
for IsCapCategory, IsFunction, [121](#)
- AddIsAutomorphism  
for IsCapCategory, IsFunction, [30](#)
- AddIsCodominating  
for IsCapCategory, IsFunction, [35](#)
- AddIsColiftable  
for IsCapCategory, IsFunction, [40](#)
- AddIsColiftableAlongEpimorphism  
for IsCapCategory, IsFunction, [39](#)
- AddIsCongruentForMorphisms  
for IsCapCategory, IsFunction, [31](#)
- AddIsDominating  
for IsCapCategory, IsFunction, [35](#)
- AddIsEndomorphism  
for IsCapCategory, IsFunction, [30](#)
- AddIsEpimorphism  
for IsCapCategory, IsFunction, [24](#)
- AddIsEqualAsFactorobjects  
for IsCapCategory, IsFunction, [34](#)
- AddIsEqualAsSubobjects  
for IsCapCategory, IsFunction, [34](#)
- AddIsEqualForCacheForMorphisms  
for IsCapCategory, IsFunction, [41](#)
- AddIsEqualForCacheForObjects  
for IsCapCategory, IsFunction, [16](#)
- AddIsEqualForMorphisms  
for IsCapCategory, IsFunction, [31](#)
- AddIsEqualForMorphismsOnMor  
for IsCapCategory, IsFunction, [32](#)
- AddIsEqualForObjects  
for IsCapCategory, IsFunction, [13](#)
- AddIsHomSetInhabited  
for IsCapCategory, IsFunction, [41](#)
- AddIsIdempotent  
for IsCapCategory, IsFunction, [25](#)
- AddIsIdenticalToIdentityMorphism  
for IsCapCategory, IsFunction, [29](#)
- AddIsIdenticalToZeroMorphism  
for IsCapCategory, IsFunction, [29](#)
- AddIsInitial  
for IsCapCategory, IsFunction, [15](#)
- AddIsInjective  
for IsCapCategory, IsFunction, [14](#)
- AddIsIsomorphism  
for IsCapCategory, IsFunction, [24](#)
- AddIsLiftable  
for IsCapCategory, IsFunction, [40](#)
- AddIsLiftableAlongMonomorphism  
for IsCapCategory, IsFunction, [38](#)
- AddIsMonomorphism  
for IsCapCategory, IsFunction, [24](#)
- AddIsomorphismFromCoequalizerOf-  
CoproductDiagramToPushout  
for IsCapCategory, IsFunction, [112](#)
- AddIsomorphismFromCoimageToCokernelOf-  
Kernel  
for IsCapCategory, IsFunction, [122](#)
- AddIsomorphismFromCokernelOfDiagonal-  
DifferenceToPushout  
for IsCapCategory, IsFunction, [110](#)
- AddIsomorphismFromCokernelOfKernelTo-  
Coimage  
for IsCapCategory, IsFunction, [122](#)
- AddIsomorphismFromCoproductToDirectSum  
for IsCapCategory, IsFunction, [85](#)
- AddIsomorphismFromDirectProductTo-  
DirectSum  
for IsCapCategory, IsFunction, [85](#)
- AddIsomorphismFromDirectSumToCoproduct

- for IsCapCategory, IsFunction, [85](#)
- AddIsomorphismFromDirectSumToDirect-Product
  - for IsCapCategory, IsFunction, [85](#)
- AddIsomorphismFromEqualizerOfDirect-ProductDiagramToFiberProduct
  - for IsCapCategory, IsFunction, [104](#)
- AddIsomorphismFromFiberProductTo-EqualizerOfDirectProductDiagram
  - for IsCapCategory, IsFunction, [103](#)
- AddIsomorphismFromFiberProductTo-KernelOfDiagonalDifference
  - for IsCapCategory, IsFunction, [102](#)
- AddIsomorphismFromImageObjectToKernel-OfCokernel
  - for IsCapCategory, IsFunction, [117](#)
- AddIsomorphismFromInitialObjectToZero-Object
  - for IsCapCategory, IsFunction, [74](#)
- AddIsomorphismFromKernelOfCokernelTo-ImageObject
  - for IsCapCategory, IsFunction, [118](#)
- AddIsomorphismFromKernelOfDiagonal-DifferenceToFiberProduct
  - for IsCapCategory, IsFunction, [103](#)
- AddIsomorphismFromPushoutTo-CoequalizerOfCoproductDiagram
  - for IsCapCategory, IsFunction, [111](#)
- AddIsomorphismFromPushoutToCokernelOf-DiagonalDifference
  - for IsCapCategory, IsFunction, [110](#)
- AddIsomorphismFromTerminalObjectTo-ZeroObject
  - for IsCapCategory, IsFunction, [74](#)
- AddIsomorphismFromZeroObjectToInitial-Object
  - for IsCapCategory, IsFunction, [73](#)
- AddIsomorphismFromZeroObjectTo-TerminalObject
  - for IsCapCategory, IsFunction, [74](#)
- AddIsOne
  - for IsCapCategory, IsFunction, [25](#)
- AddIsProjective
  - for IsCapCategory, IsFunction, [14](#)
- AddIsSplitEpimorphism
  - for IsCapCategory, IsFunction, [25](#)
- AddIsSplitMonomorphism
  - for IsCapCategory, IsFunction, [24](#)
- AddIsTerminal
  - for IsCapCategory, IsFunction, [14](#)
- AddIsWellDefinedForMorphisms
  - for IsCapCategory, IsFunction, [37](#)
- AddIsWellDefinedForObjects
  - for IsCapCategory, IsFunction, [18](#)
- AddIsWellDefinedForTwoCells
  - for IsCapCategory, IsFunction, [56](#)
- AddIsZeroForMorphisms
  - for IsCapCategory, IsFunction, [32](#)
- AddIsZeroForObjects
  - for IsCapCategory, IsFunction, [15](#)
- AdditionForMorphisms
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, [32](#)
- AdditiveInverseForMorphisms
  - for IsCapCategoryMorphism, [33](#)
- AddKernelEmbedding
  - for IsCapCategory, IsFunction, [65](#)
- AddKernelEmbeddingWithGivenKernel-Object
  - for IsCapCategory, IsFunction, [65](#)
- AddKernelLift
  - for IsCapCategory, IsFunction, [65](#)
- AddKernelLiftWithGivenKernelObject
  - for IsCapCategory, IsFunction, [66](#)
- AddKernelObject
  - for IsCapCategory, IsFunction, [65](#)
- AddKernelObjectFunctorialWithGiven-KernelObjects
  - for IsCapCategory, IsFunction, [67](#)
- AddLift
  - for IsCapCategory, IsFunction, [39](#)
- AddLiftAlongMonomorphism
  - for IsCapCategory, IsFunction, [38](#)
- AddMonomorphismIntoSomeInjectiveObject
  - for IsCapCategory, IsFunction, [20](#)
- AddMonomorphismIntoSomeInjective-ObjectWithGivenSomeInjective-Object
  - for IsCapCategory, IsFunction, [20](#)
- AddMorphism
  - for IsCapCategory, IsAttributeStoringRep, [30](#)

- AddMorphismBetweenDirectSums  
for IsCapCategory, IsFunction, [82](#)
- AddMorphismFromCoimageToImageWith-  
GivenObjects  
for IsCapCategory, IsFunction, [121](#)
- AddMorphismFromEqualizerToSink  
for IsCapCategory, IsFunction, [96](#)
- AddMorphismFromEqualizerToSinkWith-  
GivenEqualizer  
for IsCapCategory, IsFunction, [96](#)
- AddMorphismFromFiberProductToSink  
for IsCapCategory, IsFunction, [108](#)
- AddMorphismFromFiberProductToSinkWith-  
GivenFiberProduct  
for IsCapCategory, IsFunction, [108](#)
- AddMorphismFromKernelObjectToSink  
for IsCapCategory, IsFunction, [65](#)
- AddMorphismFromKernelObjectToSinkWith-  
GivenKernelObject  
for IsCapCategory, IsFunction, [65](#)
- AddMorphismFromSourceToCoequalizer  
for IsCapCategory, IsFunction, [100](#)
- AddMorphismFromSourceToCoequalizer-  
WithGivenCoequalizer  
for IsCapCategory, IsFunction, [100](#)
- AddMorphismFromSourceToCokernelObject  
for IsCapCategory, IsFunction, [69](#)
- AddMorphismFromSourceToCokernelObject-  
WithGivenCokernelObject  
for IsCapCategory, IsFunction, [69](#)
- AddMorphismFromSourceToPushout  
for IsCapCategory, IsFunction, [115](#)
- AddMorphismFromSourceToPushoutWith-  
GivenPushout  
for IsCapCategory, IsFunction, [115](#)
- AddMorphismFunction  
for IsCapFunctor, IsFunction, [59](#)
- AddMorphismRepresentation  
for IsCapCategory, IsObject, [30](#)
- AddMultiplyWithElementOfCommutative-  
RingForMorphisms  
for IsCapCategory, IsFunction, [33](#)
- AddNaturalTransformationFunction  
for IsCapNaturalTransformation, IsFunction,  
[62](#)
- AddObject  
for IsCapCategory, IsAttributeStoringRep,  
[17](#)
- AddObjectFunction  
for IsCapFunctor, IsFunction, [59](#)
- AddObjectRepresentation  
for IsCapCategory, IsObject, [17](#)
- AddOperationsToDerivationGraph  
for IsDerivedMethodGraph, IsDenseList,  
[138](#)
- AddPostCompose  
for IsCapCategory, IsFunction, [37](#)
- AddPreCompose  
for IsCapCategory, IsFunction, [36](#)
- AddPrimitiveOperation  
for IsOperationWeightList, IsString, IsInt,  
[142](#)
- AddProjectionInFactorOfDirectProduct  
for IsCapCategory, IsFunction, [92](#)
- AddProjectionInFactorOfDirectProduct-  
WithGivenDirectProduct  
for IsCapCategory, IsFunction, [92](#)
- AddProjectionInFactorOfDirectSum  
for IsCapCategory, IsFunction, [83](#)
- AddProjectionInFactorOfDirectSumWith-  
GivenDirectSum  
for IsCapCategory, IsFunction, [83](#)
- AddProjectionInFactorOfFiberProduct  
for IsCapCategory, IsFunction, [107](#)
- AddProjectionInFactorOfFiberProduct-  
WithGivenFiberProduct  
for IsCapCategory, IsFunction, [107](#)
- AddProjectionOntoCoequalizer  
for IsCapCategory, IsFunction, [99](#)
- AddProjectionOntoCoequalizerWithGiven-  
Coequalizer  
for IsCapCategory, IsFunction, [99](#)
- AddProjectiveLift  
for IsCapCategory, IsFunction, [19](#)
- AddPushout  
for IsCapCategory, IsFunction, [115](#)
- AddPushoutFunctorialWithGivenPushouts  
for IsCapCategory, IsFunction, [116](#)
- AddRandomMorphismByInteger  
for IsCapCategory, IsFunction, [28](#)
- AddRandomMorphismByList  
for IsCapCategory, IsFunction, [28](#)

AddRandomMorphismWithFixedRangeByInteger  
     for IsCapCategory, IsFunction, 27  
 AddRandomMorphismWithFixedRangeByList  
     for IsCapCategory, IsFunction, 27  
 AddRandomMorphismWithFixedSourceAndRangeByInteger  
     for IsCapCategory, IsFunction, 27  
 AddRandomMorphismWithFixedSourceAndRangeByList  
     for IsCapCategory, IsFunction, 28  
 AddRandomMorphismWithFixedSourceByInteger  
     for IsCapCategory, IsFunction, 26  
 AddRandomMorphismWithFixedSourceByList  
     for IsCapCategory, IsFunction, 26  
 AddRandomObjectByInteger  
     for IsCapCategory, IsFunction, 16  
 AddRandomObjectByList  
     for IsCapCategory, IsFunction, 16  
 AddSimplifyEndo  
     for IsCapCategory, IsFunction, 51  
 AddSimplifyEndo\_IsoFromInputObject  
     for IsCapCategory, IsFunction, 52  
 AddSimplifyEndo\_IsoToInputObject  
     for IsCapCategory, IsFunction, 51  
 AddSimplifyMorphism  
     for IsCapCategory, IsFunction, 47  
 AddSimplifyObject  
     for IsCapCategory, IsFunction, 21  
 AddSimplifyObject\_IsoFromInputObject  
     for IsCapCategory, IsFunction, 21  
 AddSimplifyObject\_IsoToInputObject  
     for IsCapCategory, IsFunction, 22  
 AddSimplifyRange  
     for IsCapCategory, IsFunction, 48  
 AddSimplifyRange\_IsoFromInputObject  
     for IsCapCategory, IsFunction, 49  
 AddSimplifyRange\_IsoToInputObject  
     for IsCapCategory, IsFunction, 48  
 AddSimplifySource  
     for IsCapCategory, IsFunction, 47  
 AddSimplifySourceAndRange  
     for IsCapCategory, IsFunction, 49  
 AddSimplifySourceAndRange\_IsoFromInputRange  
     for IsCapCategory, IsFunction, 50  
 AddSimplifySourceAndRange\_IsoFromInputSource  
     for IsCapCategory, IsFunction, 51  
 AddSimplifySourceAndRange\_IsoToInputRange  
     for IsCapCategory, IsFunction, 49  
 AddSimplifySourceAndRange\_IsoToInputSource  
     for IsCapCategory, IsFunction, 50  
 AddSimplifySource\_IsoFromInputObject  
     for IsCapCategory, IsFunction, 48  
 AddSimplifySource\_IsoToInputObject  
     for IsCapCategory, IsFunction, 47  
 AddSomeInjectiveObject  
     for IsCapCategory, IsFunction, 20  
 AddSomeProjectiveObject  
     for IsCapCategory, IsFunction, 18  
 AddSomeReductionBySplitEpiSummand  
     for IsCapCategory, IsFunction, 53  
 AddSomeReductionBySplitEpiSummand\_MorphismFromInputRange  
     for IsCapCategory, IsFunction, 53  
 AddSomeReductionBySplitEpiSummand\_MorphismToInputRange  
     for IsCapCategory, IsFunction, 53  
 AddSubtractionForMorphisms  
     for IsCapCategory, IsFunction, 33  
 AddTerminalObject  
     for IsCapCategory, IsFunction, 75  
 AddTerminalObjectFunctorial  
     for IsCapCategory, IsFunction, 76  
 AddUniversalMorphismFromCoequalizer  
     for IsCapCategory, IsFunction, 100  
 AddUniversalMorphismFromCoequalizer-WithGivenCoequalizer  
     for IsCapCategory, IsFunction, 100  
 AddUniversalMorphismFromCoproduct  
     for IsCapCategory, IsFunction, 89  
 AddUniversalMorphismFromCoproductWithGivenCoproduct  
     for IsCapCategory, IsFunction, 89  
 AddUniversalMorphismFromDirectSum  
     for IsCapCategory, IsFunction, 84  
 AddUniversalMorphismFromDirectSumWithGivenDirectSum

- for IsCapCategory, IsFunction, [84](#)
- AddUniversalMorphismFromImage
  - for IsCapCategory, IsFunction, [120](#)
- AddUniversalMorphismFromImageWith-
  - GivenImageObject
    - for IsCapCategory, IsFunction, [120](#)
- AddUniversalMorphismFromInitialObject
  - for IsCapCategory, IsFunction, [77](#)
- AddUniversalMorphismFromInitialObject-
  - WithGivenInitialObject
    - for IsCapCategory, IsFunction, [77](#)
- AddUniversalMorphismFromPushout
  - for IsCapCategory, IsFunction, [116](#)
- AddUniversalMorphismFromPushoutWith-
  - GivenPushout
    - for IsCapCategory, IsFunction, [116](#)
- AddUniversalMorphismFromZeroObject
  - for IsCapCategory, IsFunction, [73](#)
- AddUniversalMorphismFromZeroObject-
  - WithGivenZeroObject
    - for IsCapCategory, IsFunction, [73](#)
- AddUniversalMorphismIntoCoimage
  - for IsCapCategory, IsFunction, [124](#)
- AddUniversalMorphismIntoCoimageWith-
  - GivenCoimage
    - for IsCapCategory, IsFunction, [124](#)
- AddUniversalMorphismIntoDirectProduct
  - for IsCapCategory, IsFunction, [92](#)
- AddUniversalMorphismIntoDirectProduct-
  - WithGivenDirectProduct
    - for IsCapCategory, IsFunction, [92](#)
- AddUniversalMorphismIntoDirectSum
  - for IsCapCategory, IsFunction, [84](#)
- AddUniversalMorphismIntoDirectSumWith-
  - GivenDirectSum
    - for IsCapCategory, IsFunction, [84](#)
- AddUniversalMorphismIntoEqualizer
  - for IsCapCategory, IsFunction, [96](#)
- AddUniversalMorphismIntoEqualizerWith-
  - GivenEqualizer
    - for IsCapCategory, IsFunction, [96](#)
- AddUniversalMorphismIntoFiberProduct
  - for IsCapCategory, IsFunction, [108](#)
- AddUniversalMorphismIntoFiberProduct-
  - WithGivenFiberProduct
    - for IsCapCategory, IsFunction, [108](#)
- AddUniversalMorphismIntoTerminalObject
  - for IsCapCategory, IsFunction, [75](#)
- AddUniversalMorphismIntoTerminal-
  - ObjectWithGivenTerminalObject
    - for IsCapCategory, IsFunction, [76](#)
- AddUniversalMorphismIntoZeroObject
  - for IsCapCategory, IsFunction, [73](#)
- AddUniversalMorphismIntoZeroObject-
  - WithGivenZeroObject
    - for IsCapCategory, IsFunction, [73](#)
- AddVerticalPostCompose
  - for IsCapCategory, IsFunction, [55](#)
- AddVerticalPreCompose
  - for IsCapCategory, IsFunction, [55](#)
- AddWithGivenDerivationPairToCAP, [139](#)
- AddZeroMorphism
  - for IsCapCategory, IsFunction, [34](#)
- AddZeroObject
  - for IsCapCategory, IsFunction, [73](#)
- AddZeroObjectFunctorial
  - for IsCapCategory, IsFunction, [74](#)
- ApplyFunctor, [60](#)
- ApplyNaturalTransformation, [62](#)
- AsCapCategory
  - for IsCapCategoryAsCatObject, [58](#)
- AsCatObject
  - for IsCapCategory, [58](#)
- AstrictionToCoimage
  - for IsCapCategoryMorphism, [123](#)
- AstrictionToCoimageWithGivenCoimage
  - for IsCapCategoryMorphism, IsCapCategoryObject, [123](#)
- BasisOfExternalHom
  - for IsCapCategoryObject, IsCapCategoryObject, [45](#)
- BrowseCachingStatistic, [147](#)
- \\*
  - for IsRingElement, IsCapCategoryMorphism, [33](#)
- CachingStatistic, [147](#)
- CanCompute
  - for IsCapCategory, IsString, [8](#)
- CanonicalIdentificationFromCoimageTo-
  - ImageObject
    - for IsCapCategoryMorphism, [125](#)

- CanonicalIdentificationFromImage-ObjectToCoimage  
for IsCapCategoryMorphism, [125](#)
- CAPAddPrepareFunction, [133](#)
- CapCat, [57](#)
- CapCategory  
for IsCapCategoryMorphism, [23](#)  
for IsCapCategoryObject, [13](#)
- CapCategorySwitchLogicOff, [8](#)
- CapCategorySwitchLogicOn, [8](#)
- CapCategorySwitchLogicPropagationForMorphismsOff, [8](#)
- CapCategorySwitchLogicPropagationForMorphismsOn, [8](#)
- CapCategorySwitchLogicPropagationForObjectsOff, [7](#)
- CapCategorySwitchLogicPropagationForObjectsOn, [7](#)
- CapCategorySwitchLogicPropagationOff, [8](#)
- CapCategorySwitchLogicPropagationOn, [8](#)
- CapFunctor  
for IsString, IsCapCategory, IsCapCategory, [58](#)  
for IsString, IsCapCategory, IsCapCategoryAsCatObject, [58](#)  
for IsString, IsCapCategoryAsCatObject, IsCapCategory, [58](#)  
for IsString, IsCapCategoryAsCatObject, IsCapCategoryAsCatObject, [58](#)  
for IsString, IsList, IsCapCategory, [58](#)  
for IsString, IsList, IsCapCategoryAsCatObject, [58](#)
- CAPOperationPrepareFunction, [133](#)
- CAP\_INTERNAL\_ASSERT\_IS\_CELL\_OF\_CATEGORY, [146](#)
- CAP\_INTERNAL\_ASSERT\_IS\_MORPHISM\_OF\_CATEGORY, [146](#)
- CAP\_INTERNAL\_ASSERT\_IS\_OBJECT\_OF\_CATEGORY, [146](#)
- CAP\_INTERNAL\_ASSERT\_IS\_TWO\_CELL\_OF\_CATEGORY, [147](#)
- CAP\_INTERNAL\_FIND\_APPEARANCE\_OF\_SYMBOL\_IN\_FUNCTION, [145](#)
- CAP\_INTERNAL\_MERGE\_FILTER\_LISTS, [145](#)
- CAP\_INTERNAL\_MERGE\_PRECONDITIONS\_LIST, [146](#)
- CAP\_INTERNAL\_REPLACE\_STRINGS\_WITH\_FILTERS, [145](#)
- CAP\_INTERNAL\_RETURN\_OPTION\_OR\_DEFAULT, [145](#)
- CategoryFilter  
for IsCapCategory, [6](#)  
for IsDerivedMethod, [136](#)
- CategoryOfOperationWeightList  
for IsOperationWeightList, [140](#)
- CellFilter  
for IsCapCategory, [7](#)
- CheckConstructivenessOfCategory  
for IsCapCategory, IsString, [9](#)
- CostrictionToImage  
for IsCapCategoryMorphism, [118](#)
- CostrictionToImageWithGivenImageObject  
for IsCapCategoryMorphism, IsCapCategoryObject, [118](#)
- CoefficientsOfMorphism  
for IsCapCategoryMorphism, [46](#)
- CoefficientsOfMorphismWithGivenBasisOfExternalHom  
for IsCapCategoryMorphism, IsList, [46](#)
- Coequalizer, [97](#)
- CoequalizerFunctorial  
for IsList, IsCapCategoryMorphism, IsList, [100](#)
- CoequalizerFunctorialWithGivenCoequalizers  
for IsCapCategoryObject, IsList, IsCapCategoryMorphism, IsList, IsCapCategoryObject, [101](#)
- CoequalizerOp  
for IsList, IsCapCategoryMorphism, [98](#)
- Coimage  
for IsCapCategoryMorphism, [122](#)
- CoimageProjection  
for IsCapCategoryMorphism, [122](#)
- CoimageProjectionWithGivenCoimage  
for IsCapCategoryMorphism, IsCapCategoryObject, [123](#)
- CokernelColift  
for IsCapCategoryMorphism, IsCapCategoryMorphism, [68](#)
- CokernelColiftWithGivenCokernelObject

- for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject, 68
- CokernelObject
  - for IsCapCategoryMorphism, 67
- CokernelObjectFunctorial
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, 70
  - for IsList, 70
- CokernelObjectFunctorialWithGivenCokernelObjects
  - for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject, 70
  - for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject, 70
- CokernelProjection
  - for IsCapCategoryMorphism, 67
- CokernelProjectionWithGivenCokernelObject
  - for IsCapCategoryMorphism, IsCapCategoryObject, 68
- Colift
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, 39
- ColiftAlongEpimorphism
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, 38
- CommutativeRingOfLinearCategory
  - for IsCapCategory, 7
- ComponentOfMorphismFromDirectSum
  - for IsCapCategoryMorphism, IsList, IsInt, 83
- ComponentOfMorphismIntoDirectSum
  - for IsCapCategoryMorphism, IsList, IsInt, 83
- Contains
  - for IsStringMinHeap, IsString, 143
- Coproduct
  - for IsCapCategoryObject, IsCapCategoryObject, 87
  - for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryObject, 87
  - for IsList, 87
- CoproductFunctorial
  - for IsList, 89
- CoproductFunctorialWithGivenCoproducts
  - for IsCapCategoryObject, IsList, IsCapCategoryObject, 89
- CoproductOp
  - for IsList, IsCapCategoryObject, 87
- CreateCapCategory, 6
  - for IsString, 6
- CurrentOperationWeight
  - for IsOperationWeightList, IsString, 141
- DeactivateCachingOfCategory, 10
- DeactivateDefaultCaching, 10
- DeactivateDerivationInfo, 135
- DeclareAttributeWithToDoForIsWellDefined, 145
- DeclareFamilyProperty, 145
- DecreaseKey
  - for IsStringMinHeap, IsString, IsInt, 143
- DerivationFunctionsWithExtraFilters
  - for IsDerivedMethod, 136
- DerivationGraph
  - for IsOperationWeightList, 140
- DerivationInfo, 135
- DerivationName
  - for IsDerivedMethod, 136
- DerivationOfOperation
  - for IsOperationWeightList, IsString, 141
- DerivationResultWeight
  - for IsDerivedMethod, IsDenseList, 137
- DerivationsOfOperation
  - for IsDerivedMethodGraph, IsString, 140
- DerivationsUsingOperation
  - for IsDerivedMethodGraph, IsString, 140
- DerivationWeight
  - for IsDerivedMethod, 136
- DirectProductFunctorial
  - for IsList, 92
- DirectProductFunctorialWithGivenDirectProducts
  - for IsCapCategoryObject, IsList, IsCapCategoryObject, 92
- DirectProductOp
  - for IsList, IsCapCategoryObject, 90
- DirectSumCodiagonalDifference
  - for IsList, 112



- DirectSumCodiagonalDifferenceOp
  - for IsList, IsCapCategoryMorphism, [112](#)
- DirectSumDiagonalDifference
  - for IsList, [104](#)
- DirectSumDiagonalDifferenceOp
  - for IsList, IsCapCategoryMorphism, [104](#)
- DirectSumFunctorial
  - for IsList, [86](#)
- DirectSumFunctorialWithGivenDirectSums
  - for IsCapCategoryObject, IsList, IsCapCategoryObject, [86](#)
- DirectSumOp
  - for IsList, IsCapCategoryObject, [78](#)
- DirectSumProjectionInPushout
  - for IsList, [112](#)
- DirectSumProjectionInPushoutOp
  - for IsList, IsCapCategoryMorphism, [112](#)
- DisableAddForCategoricalOperations, [11](#)
- DisableInputSanityChecks, [10](#)
- DisableOutputSanityChecks, [10](#)
- DisableSanityChecks, [10](#)
- DistinguishedObjectOfHomomorphism-Structure
  - for IsCapCategory, [43](#)
- Down
  - for IsObject, [9](#)
- DownOnlyMorphismData
  - for IsCapCategoryMorphism, [9](#)
- DownToBottom
  - for IsObject, [9](#)
- EmbeddingOfEqualizer
  - for IsList, [94](#)
- EmbeddingOfEqualizerOp
  - for IsList, IsCapCategoryMorphism, [94](#)
- EmbeddingOfEqualizerWithGivenEqualizer
  - for IsList, IsCapCategoryObject, [94](#)
- EnableAddForCategoricalOperations, [11](#)
- EnableFullInputSanityChecks, [10](#)
- EnableFullOutputSanityChecks, [10](#)
- EnableFullSanityChecks, [10](#)
- EnablePartialInputSanityChecks, [10](#)
- EnablePartialOutputSanityChecks, [10](#)
- EnablePartialSanityChecks, [10](#)
- EpimorphismFromSomeProjectiveObject
  - for IsCapCategoryObject, [18](#)
- EpimorphismFromSomeProjectiveObject-
  - WithGivenSomeProjectiveObject
  - for IsCapCategoryObject, IsCapCategory-Object, [18](#)
- Equalizer, [93](#)
- EqualizerFunctorial
  - for IsList, IsCapCategoryMorphism, IsList, [96](#)
- EqualizerFunctorialWithGivenEqualizers
  - for IsCapCategoryObject, IsList, IsCapCategoryMorphism, IsList, IsCapCategory-Object, [96](#)
- EqualizerOp
  - for IsList, IsCapCategoryMorphism, [94](#)
- ExtractMin
  - for IsStringMinHeap, [143](#)
- FiberProduct, [105](#)
- FiberProductEmbeddingInDirectSum
  - for IsList, [105](#)
- FiberProductEmbeddingInDirectSumOp
  - for IsList, IsCapCategoryMorphism, [105](#)
- FiberProductFunctorial
  - for IsList, IsList, IsList, [108](#)
- FiberProductFunctorialWithGivenFiber-Products
  - for IsCapCategoryObject, IsList, IsList, IsList, IsCapCategoryObject, [108](#)
- FiberProductOp
  - for IsList, IsCapCategoryMorphism, [105](#)
- FunctionCalledBeforeInstallation
  - for IsDerivedMethod, [137](#)
- FunctorCanonicalizeZeroMorphisms
  - for IsCapCategory, [61](#)
- FunctorCanonicalizeZeroObjects
  - for IsCapCategory, [61](#)
- FunctorMorphismOperation
  - for IsCapFunctor, [60](#)
- FunctorObjectOperation
  - for IsCapFunctor, [59](#)
- Heapify
  - for IsStringMinHeap, IsPosInt, [143](#)
- HeapSize
  - for IsStringMinHeap, [143](#)
- HomologyObject



- for IsCapCategoryMorphism, IsCapCategoryMorphism, [126](#)
- HomologyObjectFunctorial
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, [126](#)
- HomologyObjectFunctorialWithGivenHomologyObjects
  - for IsCapCategoryObject, IsList, IsCapCategoryObject, [126](#)
- HomomorphismStructureOnMorphisms
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, [42](#)
- HomomorphismStructureOnMorphismsWithGivenObjects
  - for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject, [42](#)
- HomomorphismStructureOnObjects
  - for IsCapCategoryObject, IsCapCategoryObject, [42](#)
- HomStructure
  - for IsCapCategory, [45](#)
  - for IsCapCategoryMorphism, [45](#)
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, [44](#)
  - for IsCapCategoryMorphism, IsCapCategoryObject, [44](#)
  - for IsCapCategoryObject, IsCapCategoryMorphism, [44](#)
  - for IsCapCategoryObject, IsCapCategoryObject, [45](#)
  - for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryMorphism, [45](#)
- HorizontalPostCompose
  - for IsCapCategoryTwoCell, IsCapCategoryTwoCell, [55](#)
- HorizontalPreCompose
  - for IsCapCategoryTwoCell, IsCapCategoryTwoCell, [54](#)
- HorizontalPreComposeFunctorWithNaturalTransformation
  - for IsCapFunctor, IsCapNaturalTransformation, [62](#)
- HorizontalPreComposeNaturalTransformationWithFunctor
  - for IsCapNaturalTransformation, IsCapFunctor, [62](#)
- IdentityFunctor
  - for IsCapCategory, [61](#)
- IdentityMorphism
  - for IsCapCategoryObject, [36](#)
- IdentityTwoCell
  - for IsCapCategoryMorphism, [54](#)
- ImageEmbedding
  - for IsCapCategoryMorphism, [118](#)
- ImageEmbeddingWithGivenImageObject
  - for IsCapCategoryMorphism, IsCapCategoryObject, [118](#)
- ImageObject
  - for IsCapCategoryMorphism, [118](#)
- InitialObject
  - for IsCapCategory, [76](#)
  - for IsCapCategoryCell, [76](#)
- InitialObjectFunctorial
  - for IsCapCategory, [77](#)
- InjectionOfCofactorOfCoproduct
  - for IsList, IsInt, [87](#)
- InjectionOfCofactorOfCoproductOp
  - for IsList, IsInt, IsCapCategoryObject, [87](#)
- InjectionOfCofactorOfCoproductWithGivenCoproduct
  - for IsList, IsInt, IsCapCategoryObject, [87](#)
- InjectionOfCofactorOfDirectSum
  - for IsList, IsInt, [79](#)
- InjectionOfCofactorOfDirectSumOp
  - for IsList, IsInt, IsCapCategoryObject, [79](#)
- InjectionOfCofactorOfDirectSumWithGivenDirectSum
  - for IsList, IsInt, IsCapCategoryObject, [79](#)
- InjectionOfCofactorOfPushout
  - for IsList, IsInt, [113](#)
- InjectionOfCofactorOfPushoutOp
  - for IsList, IsInt, IsCapCategoryMorphism, [113](#)
- InjectionOfCofactorOfPushoutWithGivenPushout
  - for IsList, IsInt, IsCapCategoryObject, [113](#)
- InjectiveColift
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, [20](#)

- InputSignature
  - for IsCapFunctor, [60](#)
- InstallDerivationForCategory
  - for IsDerivedMethod, IsPosInt, IsCapCategory, [137](#)
- InstallDerivationsUsingOperation
  - for IsOperationWeightList, IsString, [141](#)
- InstallFunctor
  - for IsCapFunctor, IsString, [60](#)
- InstallMethodWithToDoForIsWellDefined, [144](#)
- InstallNaturalTransformation
  - for IsCapNaturalTransformation, IsString, [62](#)
- InstallSetWithToDoForIsWellDefined
  - for IsObject, IsString, IsList, [145](#)
- InterpretMorphismAsMorphismFromDistinguishedObjectToHomomorphismStructure
  - for IsCapCategoryMorphism, [43](#)
- InterpretMorphismFromDistinguishedObjectToHomomorphismStructureAsMorphism
  - for IsCapCategoryObject, IsCapCategoryObject, IsCapCategoryMorphism, [43](#)
- Inverse
  - for IsCapCategoryMorphism, [40](#)
- InverseMorphismFromCoimageToImage
  - for IsCapCategoryMorphism, [121](#)
- InverseMorphismFromCoimageToImageWithGivenObjects
  - for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryObject, [121](#)
- IsApplicableToCategory
  - for IsDerivedMethod, IsCapCategory, [137](#)
- IsAutomorphism
  - for IsCapCategoryMorphism, [30](#)
- IsCapCategory
  - for IsObject, [5](#)
- IsCapCategoryAsCatObject
  - for IsCapCategoryObject, [57](#)
- IsCapCategoryCell
  - for IsAttributeStoringRep, [5](#)
- IsCapCategoryMorphism
  - for IsCapCategoryCell, [6](#)
- IsCapCategoryObject
  - for IsCapCategoryCell, [5](#)
- IsCapCategoryTwoCell
  - for IsCapCategoryCell, [6](#)
- IsCapFunctor
  - for IsCapCategoryMorphism, [57](#)
- IsCapNaturalTransformation
  - for IsCapCategoryTwoCell, [57](#)
- IsCodominating
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, [35](#)
- IsColiftable
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, [40](#)
- IsColiftableAlongEpimorphism
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, [39](#)
- IsCongruentForMorphisms
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, [31](#)
- IsDerivedMethod
  - for IsObject, [135](#)
- IsDerivedMethodGraph
  - for IsObject, [138](#)
- IsDominating
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, [35](#)
- IsEmptyHeap
  - for IsStringMinHeap, [143](#)
- IsEndomorphism
  - for IsCapCategoryMorphism, [29](#)
- IsEpimorphism
  - for IsCapCategoryMorphism, [24](#)
- IsEqualAsFactorobjects
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, [34](#)
- IsEqualAsSubobjects
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, [34](#)
- IsEqualForCacheForMorphisms
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, [41](#)
- IsEqualForCacheForObjects
  - for IsCapCategoryObject, IsCapCategoryObject, [16](#)
- IsEqualForMorphisms
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, [31](#)

- IsEqualForMorphismsOnMor
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, [31](#)
- IsEqualForObjects
  - for IsCapCategoryObject, IsCapCategoryObject, [13](#)
- IsHomSetInhabited
  - for IsCapCategoryObject, IsCapCategoryObject, [41](#)
- IsIdempotent
  - for IsCapCategoryMorphism, [25](#)
- IsIdenticalToIdentityMorphism
  - for IsCapCategoryMorphism, [29](#)
- IsIdenticalToZeroMorphism
  - for IsCapCategoryMorphism, [29](#)
- IsInitial
  - for IsCapCategoryObject, [14](#)
- IsInjective
  - for IsCapCategoryObject, [14](#)
- IsIsomorphism
  - for IsCapCategoryMorphism, [24](#)
- IsLiftable
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, [40](#)
- IsLiftableAlongMonomorphism
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, [38](#)
- IsMonomorphism
  - for IsCapCategoryMorphism, [23](#)
- IsomorphismFromCoequalizerOfCoproductDiagramToPushout
  - for IsList, [111](#)
- IsomorphismFromCoequalizerOfCoproductDiagramToPushoutOp
  - for IsList, IsCapCategoryMorphism, [111](#)
- IsomorphismFromCoimageToCokernelOfKernel
  - for IsCapCategoryMorphism, [122](#)
- IsomorphismFromCokernelOfDiagonalDifferenceToPushout
  - for IsList, [110](#)
- IsomorphismFromCokernelOfDiagonalDifferenceToPushoutOp
  - for IsList, IsCapCategoryMorphism, [110](#)
- IsomorphismFromCokernelOfKernelToCoimage
  - for IsCapCategoryMorphism, [122](#)
- IsomorphismFromCoproductToDirectSum
  - for IsList, [82](#)
- IsomorphismFromCoproductToDirectSumOp
  - for IsList, IsCapCategoryObject, [82](#)
- IsomorphismFromDirectProductToDirectSum
  - for IsList, [81](#)
- IsomorphismFromDirectProductToDirectSumOp
  - for IsList, IsCapCategoryObject, [81](#)
- IsomorphismFromDirectSumToCoproduct
  - for IsList, [81](#)
- IsomorphismFromDirectSumToCoproductOp
  - for IsList, IsCapCategoryObject, [82](#)
- IsomorphismFromDirectSumToDirectProduct
  - for IsList, [81](#)
- IsomorphismFromDirectSumToDirectProductOp
  - for IsList, IsCapCategoryObject, [81](#)
- IsomorphismFromEqualizerOfDirectProductDiagramToFiberProduct
  - for IsList, [104](#)
- IsomorphismFromEqualizerOfDirectProductDiagramToFiberProductOp
  - for IsList, IsCapCategoryMorphism, [104](#)
- IsomorphismFromFiberProductToEqualizerOfDirectProductDiagram
  - for IsList, [103](#)
- IsomorphismFromFiberProductToEqualizerOfDirectProductDiagramOp
  - for IsList, IsCapCategoryMorphism, [103](#)
- IsomorphismFromFiberProductToKernelOfDiagonalDifference
  - for IsList, [102](#)
- IsomorphismFromFiberProductToKernelOfDiagonalDifferenceOp
  - for IsList, IsCapCategoryMorphism, [102](#)
- IsomorphismFromHomologyObjectToItsConstructionAsAnImageObject
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, [126](#)
- IsomorphismFromImageObjectToKernelOfCokernel

- for IsCapCategoryMorphism, [117](#)
- IsomorphismFromInitialObjectToZero-Object
  - for IsCapCategory, [72](#)
- IsomorphismFromItsConstructionAsAn-ImageObjectToHomologyObject
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, [126](#)
- IsomorphismFromKernelOfCokernelTo-ImageObject
  - for IsCapCategoryMorphism, [117](#)
- IsomorphismFromKernelOfDiagonal-DifferenceToFiberProduct
  - for IsList, [102](#)
- IsomorphismFromKernelOfDiagonal-DifferenceToFiberProductOp
  - for IsList, IsCapCategoryMorphism, [103](#)
- IsomorphismFromPushoutToCoequalizerOf-CoproductDiagram
  - for IsList, [111](#)
- IsomorphismFromPushoutToCoequalizerOf-CoproductDiagramOp
  - for IsList, IsCapCategoryMorphism, [111](#)
- IsomorphismFromPushoutToCokernelOf-DiagonalDifference
  - for IsList, [110](#)
- IsomorphismFromPushoutToCokernelOf-DiagonalDifferenceOp
  - for IsList, IsCapCategoryMorphism, [110](#)
- IsomorphismFromTerminalObjectToZero-Object
  - for IsCapCategory, [72](#)
- IsomorphismFromZeroObjectToInitial-Object
  - for IsCapCategory, [72](#)
- IsomorphismFromZeroObjectToTerminal-Object
  - for IsCapCategory, [72](#)
- IsOne
  - for IsCapCategoryMorphism, [25](#)
- IsOperationWeightList
  - for IsObject, [140](#)
- IsProjective
  - for IsCapCategoryObject, [14](#)
- IsSplitEpimorphism
  - for IsCapCategoryMorphism, [25](#)
- IsSplitMonomorphism
  - for IsCapCategoryMorphism, [24](#)
- IsStringMinHeap
  - for IsObject, [142](#)
- IsTerminal
  - for IsCapCategoryObject, [14](#)
- IsWellDefined
  - for IsCapCategoryCell, [9](#)
- IsWellDefinedForMorphisms
  - for IsCapCategoryMorphism, [37](#)
- IsWellDefinedForObjects
  - for IsCapCategoryObject, [17](#)
- IsWellDefinedForTwoCells
  - for IsCapCategoryTwoCell, [56](#)
- IsZero
  - for IsCapCategoryObject, [15](#)
- IsZeroForMorphisms
  - for IsCapCategoryMorphism, [32](#)
- IsZeroForObjects
  - for IsCapCategoryObject, [15](#)
- KernelEmbedding
  - for IsCapCategoryMorphism, [64](#)
- KernelEmbeddingWithGivenKernelObject
  - for IsCapCategoryMorphism, IsCapCategoryObject, [64](#)
- KernelLift
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, [64](#)
- KernelLiftWithGivenKernelObject
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject, [64](#)
- KernelObject
  - for IsCapCategoryMorphism, [63](#)
- KernelObjectFunctorial
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, [66](#)
  - for IsList, [66](#)
- KernelObjectFunctorialWithGivenKernel-Objects
  - for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject, [66](#)

- for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject, 66
- LaTeXOutput
  - for IsCapCategory, 12
  - for IsCapCategoryCell, 11
- Lift
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, 39
- LiftAlongMonomorphism
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, 38
- ListCAPPrepareFunctions, 134
- MakeDerivation
  - for IsString, IsFunction, IsDenseList, IsPosInt, IsDenseList, IsFunction, 135
- MakeDerivationGraph
  - for IsDenseList, 138
- MakeOperationWeightList
  - for IsCapCategory, IsDerivedMethodGraph, 140
- MereExistenceOfSolutionOfLinearSystemInAbCategory
  - for IsList, IsList, IsList, 44
- MonomorphismIntoSomeInjectiveObject
  - for IsCapCategoryObject, 19
- MonomorphismIntoSomeInjectiveObjectWithGivenSomeInjectiveObject
  - for IsCapCategoryObject, IsCapCategoryObject, 20
- MorphismBetweenDirectSums
  - for IsCapCategoryObject, IsList, IsCapCategoryObject, 82
  - for IsList, 82
- MorphismBetweenDirectSumsOp
  - for IsList, IsInt, IsInt, IsCapCategoryMorphism, 82
- MorphismCache
  - for IsCapFunctor, 144
- MorphismFilter
  - for IsCapCategory, 7
- MorphismFromCoimageToImage
  - for IsCapCategoryMorphism, 121
- MorphismFromCoimageToImageWithGivenObjects
  - for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryObject, 121
- MorphismFromEqualizerToSink
  - for IsList, 94
- MorphismFromEqualizerToSinkOp
  - for IsList, IsCapCategoryMorphism, 94
- MorphismFromEqualizerToSinkWithGivenEqualizer
  - for IsList, IsCapCategoryObject, 95
- MorphismFromFiberProductToSink
  - for IsList, 106
- MorphismFromFiberProductToSinkOp
  - for IsList, IsCapCategoryMorphism, 106
- MorphismFromFiberProductToSinkWithGivenFiberProduct
  - for IsList, IsCapCategoryObject, 106
- MorphismFromKernelObjectToSink
  - for IsCapCategoryMorphism, 64
- MorphismFromKernelObjectToSinkWithGivenKernelObject
  - for IsCapCategoryMorphism, IsCapCategoryObject, 64
- MorphismFromSourceToCoequalizer
  - for IsList, 98
- MorphismFromSourceToCoequalizerOp
  - for IsList, IsCapCategoryMorphism, 98
- MorphismFromSourceToCoequalizerWithGivenCoequalizer
  - for IsList, IsCapCategoryObject, 99
- MorphismFromSourceToCokernelObject
  - for IsCapCategoryMorphism, 68
- MorphismFromSourceToCokernelObjectWithGivenCokernelObject
  - for IsCapCategoryMorphism, IsCapCategoryObject, 68
- MorphismFromSourceToPushout
  - for IsList, 114
- MorphismFromSourceToPushoutOp
  - for IsList, IsCapCategoryMorphism, 114
- MorphismFromSourceToPushoutWithGivenPushout
  - for IsList, IsCapCategoryObject, 114
- MorphismFromZeroObject
  - for IsCapCategoryObject, 71

MorphismIntoZeroObject  
     for IsCapCategoryObject, [71](#)  
 MultiplyWithElementOfCommutativeRing-  
     ForMorphisms  
     for IsRingElement, IsCapCategoryMor-  
     phism, [33](#)  
 Name  
     for IsCapNaturalTransformation, [61](#)  
 NaturalIsomorphismFromIdentityTo-  
     CanonicalizeZeroMorphisms  
     for IsCapCategory, [61](#)  
 NaturalIsomorphismFromIdentityTo-  
     CanonicalizeZeroObjects  
     for IsCapCategory, [61](#)  
 NaturalTransformation  
     for IsCapFunctor, IsCapFunctor, [62](#)  
 ObjectCache  
     for IsCapFunctor, [144](#)  
 ObjectFilter  
     for IsCapCategory, [7](#)  
 ObjectifyMorphismWithSourceAndRange-  
     ForCAPWithAttributes, [30](#)  
 ObjectifyObjectForCAPWithAttributes, [17](#)  
 Operations  
     for IsDerivedMethodGraph, [139](#)  
 OperationWeightUsingDerivation  
     for IsOperationWeightList, IsDerived-  
     Method, [141](#)  
 PostCompose  
     for IsCapCategoryMorphism, IsCapCatego-  
     ryMorphism, [36](#)  
     for IsList, [36](#)  
 PreCompose  
     for IsCapCategoryMorphism, IsCapCatego-  
     ryMorphism, [36](#)  
     for IsList, [36](#)  
 PrintDerivationTree  
     for IsOperationWeightList, IsString, [142](#)  
 PrintTree  
     for IsObject, IsFunction, IsFunction, [142](#)  
 PrintTreeRec  
     for IsObject, IsFunction, IsFunction, IsInt,  
     [142](#)  
 ProjectionInFactorOfDirectProduct  
     for IsList, IsInt, [90](#)  
 ProjectionInFactorOfDirectProductOp  
     for IsList, IsInt, IsCapCategoryObject, [90](#)  
 ProjectionInFactorOfDirectProductWith-  
     GivenDirectProduct  
     for IsList, IsInt, IsCapCategoryObject, [91](#)  
 ProjectionInFactorOfDirectSum  
     for IsList, IsInt, [79](#)  
 ProjectionInFactorOfDirectSumOp  
     for IsList, IsInt, IsCapCategoryObject, [79](#)  
 ProjectionInFactorOfDirectSumWith-  
     GivenDirectSum  
     for IsList, IsInt, IsCapCategoryObject, [79](#)  
 ProjectionInFactorOfFiberProduct  
     for IsList, IsInt, [105](#)  
 ProjectionInFactorOfFiberProductOp  
     for IsList, IsInt, IsCapCategoryMorphism,  
     [106](#)  
 ProjectionInFactorOfFiberProductWith-  
     GivenFiberProduct  
     for IsList, IsInt, IsCapCategoryObject, [106](#)  
 ProjectionOntoCoequalizer  
     for IsList, [98](#)  
 ProjectionOntoCoequalizerOp  
     for IsList, IsCapCategoryMorphism, [98](#)  
 ProjectionOntoCoequalizerWithGiven-  
     Coequalizer  
     for IsList, IsCapCategoryObject, [98](#)  
 ProjectiveLift  
     for IsCapCategoryMorphism, IsCapCatego-  
     ryMorphism, [18](#)  
 Pushout  
     for IsCapCategoryMorphism, IsCapCatego-  
     ryMorphism, [113](#)  
     for IsList, [113](#)  
 PushoutFunctorial  
     for IsList, IsList, IsList, [116](#)  
 PushoutFunctorialWithGivenPushouts  
     for IsCapCategoryObject, IsList, IsList, Is-  
     List, IsCapCategoryObject, [116](#)  
 PushoutOp  
     for IsList, IsCapCategoryMorphism, [113](#)  
 RandomMorphism  
     for IsCapCategory, IsInt, [29](#)  
     for IsCapCategory, IsList, [29](#)  
 RandomMorphismByInteger

- for IsCapCategory, IsInt, 28
- RandomMorphismByList
  - for IsCapCategory, IsList, 28
- RandomMorphismWithFixedRange
  - for IsCapCategoryObject, IsInt, 28
  - for IsCapCategoryObject, IsList, 28
- RandomMorphismWithFixedRangeByInteger
  - for IsCapCategoryObject, IsInt, 26
- RandomMorphismWithFixedRangeByList
  - for IsCapCategoryObject, IsList, 27
- RandomMorphismWithFixedSource
  - for IsCapCategoryObject, IsInt, 28
  - for IsCapCategoryObject, IsList, 28
- RandomMorphismWithFixedSourceAndRange
  - for IsCapCategoryObject, IsCapCategory-Object, IsInt, 28
  - for IsCapCategoryObject, IsCapCategory-Object, IsList, 29
- RandomMorphismWithFixedSourceAndRange-ByInteger
  - for IsCapCategoryObject, IsCapCategory-Object, IsInt, 27
- RandomMorphismWithFixedSourceAndRange-ByList
  - for IsCapCategoryObject, IsCapCategory-Object, IsList, 27
- RandomMorphismWithFixedSourceByInteger
  - for IsCapCategoryObject, IsInt, 26
- RandomMorphismWithFixedSourceByList
  - for IsCapCategoryObject, IsList, 26
- RandomObject
  - for IsCapCategory, IsInt, 16
  - for IsCapCategory, IsList, 16
- RandomObjectByInteger
  - for IsCapCategory, IsInt, 15
- RandomObjectByList
  - for IsCapCategory, IsList, 16
- Range
  - for IsCapCategoryMorphism, 23
  - for IsCapCategoryTwoCell, 54
- RangeOfFunctor
  - for IsCapFunctor, 59
- Reevaluate
  - for IsOperationWeightList, 141
- Saturate
  - for IsOperationWeightList, 141
- SetCachingOfCategory, 10
- SetCachingOfCategoryCrisp, 10
- SetCachingOfCategoryWeak, 10
- SetDefaultCaching, 10
- SetDefaultCachingCrisp, 10
- SetDefaultCachingWeak, 10
- Simplify
  - for IsCapCategoryMorphism, 52
  - for IsCapCategoryObject, 21
- SimplifyEndo
  - for IsCapCategoryMorphism, IsObject, 51
- SimplifyEndo\_IsoFromInputObject
  - for IsCapCategoryMorphism, IsObject, 51
- SimplifyEndo\_IsoToInputObject
  - for IsCapCategoryMorphism, IsObject, 51
- SimplifyMorphism
  - for IsCapCategoryMorphism, IsObject, 47
- SimplifyObject
  - for IsCapCategoryObject, IsObject, 21
- SimplifyObject\_IsoFromInputObject
  - for IsCapCategoryObject, IsObject, 21
- SimplifyObject\_IsoToInputObject
  - for IsCapCategoryObject, IsObject, 22
- SimplifyRange
  - for IsCapCategoryMorphism, IsObject, 48
- SimplifyRange\_IsoFromInputObject
  - for IsCapCategoryMorphism, IsObject, 49
- SimplifyRange\_IsoToInputObject
  - for IsCapCategoryMorphism, IsObject, 48
- SimplifySource
  - for IsCapCategoryMorphism, IsObject, 47
- SimplifySourceAndRange
  - for IsCapCategoryMorphism, IsObject, 49
- SimplifySourceAndRange\_IsoFromInput-Range
  - for IsCapCategoryMorphism, IsObject, 50
- SimplifySourceAndRange\_IsoFromInput-Source
  - for IsCapCategoryMorphism, IsObject, 50
- SimplifySourceAndRange\_IsoToInputRange
  - for IsCapCategoryMorphism, IsObject, 49
- SimplifySourceAndRange\_IsoToInput-Source
  - for IsCapCategoryMorphism, IsObject, 50
- SimplifySource\_IsoFromInputObject
  - for IsCapCategoryMorphism, IsObject, 48



- SimplifySource\_IsoToInputObject
  - for IsCapCategoryMorphism, IsObject, [47](#)
- SolveLinearSystemInAbCategory
  - for IsList, IsList, IsList, [44](#)
- SomeInjectiveObject
  - for IsCapCategoryObject, [19](#)
- SomeProjectiveObject
  - for IsCapCategoryObject, [18](#)
- SomeReductionBySplitEpiSummand
  - for IsCapCategoryMorphism, [53](#)
- SomeReductionBySplitEpiSummand\_-
  - MorphismFromInputRange
  - for IsCapCategoryMorphism, [53](#)
- SomeReductionBySplitEpiSummand\_-
  - MorphismToInputRange
  - for IsCapCategoryMorphism, [53](#)
- Source
  - for IsCapCategoryMorphism, [23](#)
  - for IsCapCategoryTwoCell, [54](#)
- SourceOfFunctor
  - for IsCapFunctor, [59](#)
- StringMinHeap, [142](#)
- SubtractionForMorphisms
  - for IsCapCategoryMorphism, IsCapCategoryMorphism, [32](#)
- Swap
  - for IsStringMinHeap, IsPosInt, IsPosInt, [143](#)
- TargetOperation
  - for IsDerivedMethod, [137](#)
- TerminalObject
  - for IsCapCategory, [75](#)
  - for IsCapCategoryCell, [75](#)
- TerminalObjectFunctorial
  - for IsCapCategory, [76](#)
- TwoCellFilter
  - for IsCapCategory, [7](#)
- UniversalMorphismFromCoequalizer
  - for IsList, IsCapCategoryMorphism, [99](#)
- UniversalMorphismFromCoequalizerWith-
  - GivenCoequalizer
  - for IsList, IsCapCategoryMorphism, IsCapCategoryObject, [99](#)
- UniversalMorphismFromCoproduct, [88](#)
- UniversalMorphismFromCoproductOp
  - for IsList, IsList, IsCapCategoryObject, [88](#)
- UniversalMorphismFromCoproductWith-
  - GivenCoproduct
  - for IsList, IsList, IsCapCategoryObject, [88](#)
- UniversalMorphismFromDirectSum, [80](#)
- UniversalMorphismFromDirectSumOp
  - for IsList, IsList, IsCapCategoryObject, [80](#)
- UniversalMorphismFromDirectSumWith-
  - GivenDirectSum
  - for IsList, IsList, IsCapCategoryObject, [81](#)
- UniversalMorphismFromImage
  - for IsCapCategoryMorphism, IsList, [118](#)
- UniversalMorphismFromImageWithGiven-
  - ImageObject
  - for IsCapCategoryMorphism, IsList, IsCapCategoryObject, [119](#)
- UniversalMorphismFromInitialObject
  - for IsCapCategoryObject, [77](#)
- UniversalMorphismFromInitialObject-
  - WithGivenInitialObject
  - for IsCapCategoryObject, IsCapCategoryObject, [77](#)
- UniversalMorphismFromPushout, [114](#)
- UniversalMorphismFromPushoutOp
  - for IsList, IsList, IsCapCategoryMorphism, [114](#)
- UniversalMorphismFromPushoutWithGiven-
  - Pushout
  - for IsList, IsList, IsCapCategoryObject, [115](#)
- UniversalMorphismFromZeroObject
  - for IsCapCategoryObject, [71](#)
- UniversalMorphismFromZeroObjectWith-
  - GivenZeroObject
  - for IsCapCategoryObject, IsCapCategoryObject, [72](#)
- UniversalMorphismIntoCoimage
  - for IsCapCategoryMorphism, IsList, [123](#)
- UniversalMorphismIntoCoimageWithGiven-
  - Coimage
  - for IsCapCategoryMorphism, IsList, IsCapCategoryObject, [123](#)
- UniversalMorphismIntoDirectProduct, [91](#)
- UniversalMorphismIntoDirectProductOp
  - for IsList, IsList, IsCapCategoryObject, [91](#)
- UniversalMorphismIntoDirectProduct-
  - WithGivenDirectProduct
  - for IsList, IsList, IsCapCategoryObject, [91](#)



UniversalMorphismIntoDirectSum, [79](#)  
 UniversalMorphismIntoDirectSumOp  
     for IsList, IsList, IsCapCategoryObject, [80](#)  
 UniversalMorphismIntoDirectSumWith-  
     GivenDirectSum  
     for IsList, IsList, IsCapCategoryObject, [80](#)  
 UniversalMorphismIntoEqualizer  
     for IsList, IsCapCategoryMorphism, [95](#)  
 UniversalMorphismIntoEqualizerWith-  
     GivenEqualizer  
     for IsList, IsCapCategoryMorphism, IsCap-  
     CategoryObject, [95](#)  
 UniversalMorphismIntoFiberProduct, [106](#)  
 UniversalMorphismIntoFiberProductOp  
     for IsList, IsList, IsCapCategoryMorphism,  
     [107](#)  
 UniversalMorphismIntoFiberProductWith-  
     GivenFiberProduct  
     for IsList, IsList, IsCapCategoryObject, [107](#)  
 UniversalMorphismIntoTerminalObject  
     for IsCapCategoryObject, [75](#)  
 UniversalMorphismIntoTerminalObject-  
     WithGivenTerminalObject  
     for IsCapCategoryObject, IsCapCategory-  
     Object, [75](#)  
 UniversalMorphismIntoZeroObject  
     for IsCapCategoryObject, [72](#)  
 UniversalMorphismIntoZeroObjectWith-  
     GivenZeroObject  
     for IsCapCategoryObject, IsCapCategory-  
     Object, [72](#)  
 UsedOperationMultiples  
     for IsDerivedMethod, [137](#)  
 UsedOperations  
     for IsDerivedMethod, [137](#)  
 UsedOperationsWithMultiples  
     for IsDerivedMethod, [137](#)  
  
 VerticalPostCompose  
     for IsCapCategoryTwoCell, IsCapCatego-  
     ryTwoCell, [55](#)  
 VerticalPreCompose  
     for IsCapCategoryTwoCell, IsCapCatego-  
     ryTwoCell, [55](#)  
  
 ZeroMorphism  
     for IsCapCategoryObject, IsCapCategory-  
     Object, [34](#)  
 ZeroObject  
     for IsCapCategory, [71](#)  
     for IsCapCategoryCell, [71](#)  
 ZeroObjectFunctorial  
     for IsCapCategory, [74](#)