CAP

Categories, Algorithms, Programming

2020.10-01

19 October 2020

Sebastian Gutsche

Sebastian Posur

Øystein Skartsæterhagen
Sebastian Gutsche
Email: 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
Homepage: https://sebastianpos.github.io
Address: Department Mathematik
Universität Siegen
Walter-Flex-Straße 3
57068 Siegen
Germany

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

1 CAP Categories
   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 \texttt{add} .................... 11
   1.11 Performance tweaks ....................................... 11
   1.12 LaTeX ...................................................... 11

2 Objects
   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

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

Index 174
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)

\[\text{IsCapCategory}(\text{object})\]

\textbf{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)

\[\text{IsCapCategoryCell}(\text{object})\]

\textbf{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)

\[\text{IsCapCategoryObject}(\text{object})\]

\textbf{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)

\[ \text{IsCapCategoryMorphism}(\text{object}) \] (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)

\[ \text{IsCapCategoryTwoCell}(\text{object}) \] (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

\[ \text{AddCategoricalProperty}(\text{list}) \] (function)

Adds a categorical property to the list of CAP categorical properties. \text{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

\[ \text{CreateCapCategory()} \] (operation)

**Returns:** a category

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

1.2.2 CreateCapCategory (for IsString)

\[ \text{CreateCapCategory}(s) \] (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)

\[ \text{CategoryFilter}(C) \] (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)

\[ \text{CellFilter}(C) \]

(attribut)

**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)

\[ \text{ObjectFilter}(C) \]

(attribut)

**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)

\[ \text{MorphismFilter}(C) \]

(attribut)

**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)

\[ \text{TwoCellFilter}(C) \]

(attribut)

**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)

\[ \text{CommutativeRingOfLinearCategory}(C) \]

(attribut)

**Returns:** a ring

The argument is a category \( C \) which is expected to lie in the filter \( \text{IsLinearCategoryOverCommutativeRing} \). The output is a commutative ring over which the category is linear.

1.4 Logic switcher

1.4.1 CapCategorySwitchLogicPropagationForObjectsOn

\[ \text{CapCategorySwitchLogicPropagationForObjectsOn}(C) \]

(function)

Activates the predicate logic propagation between equal objects for the category \( C \).

1.4.2 CapCategorySwitchLogicPropagationForObjectsOff

\[ \text{CapCategorySwitchLogicPropagationForObjectsOff}(C) \]

(function)

Deactivates the predicate logic propagation between equal objects for the category \( C \).
1.4.3 CapCategorySwitchLogicPropagationForMorphismsOn

\[ \text{CapCategorySwitchLogicPropagationForMorphismsOn}(C) \] (function)

Activates the predicate logic propagation between equal morphisms for the category \( C \).

1.4.4 CapCategorySwitchLogicPropagationForMorphismsOff

\[ \text{CapCategorySwitchLogicPropagationForMorphismsOff}(C) \] (function)

Deactivates the predicate logic propagation between equal morphisms for the category \( C \).

1.4.5 CapCategorySwitchLogicPropagationOn

\[ \text{CapCategorySwitchLogicPropagationOn}(C) \] (function)

Activates the predicate logic propagation between equal cells for the category \( C \).

1.4.6 CapCategorySwitchLogicPropagationOff

\[ \text{CapCategorySwitchLogicPropagationOff}(C) \] (function)

Deactivates the predicate logic propagation between equal cells for the category \( C \).

1.4.7 CapCategorySwitchLogicOn

\[ \text{CapCategorySwitchLogicOn}(C) \] (function)

Activates the predicate implication logic for the category \( C \).

1.4.8 CapCategorySwitchLogicOff

\[ \text{CapCategorySwitchLogicOff}(C) \] (function)

Deactivates the predicate implication logic for the category \( C \).

1.5 Tool functions

1.5.1 CanCompute (for IsCapCategory, IsString)

\[ \text{CanCompute}(C, s) \] (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)

\[\text{CheckConstructivenessOfCategory}(C, s)\]

(organization)

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)

\[\text{IsWellDefined}(c)\]

(property)

Returns: a boolean

The argument is a cell \( c \). The output is \text{true} if \( c \) is well-defined, otherwise the output is \text{false}.

1.7 Unpacking data structures

1.7.1 Down (for IsObject)

\[\text{Down}(x)\]

(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)

\[\text{DownOnlyMorphismData}(x)\]

(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)

\[\text{DownToBottom}(x)\]

(attribute)

Returns: a GAP object

The argument is a GAP object \( x \). This function iteratively calls \text{Down} until it becomes stable.
1.8 Caching

1.8.1 SetCachingOfCategory

\[\text{SetCachingOfCategory}(\text{category}, \text{type})\] (function)

Sets the caching of \textit{category} to \textit{type}.

1.8.2 SetCachingOfCategoryWeak

\[\text{SetCachingOfCategoryWeak}(\text{category})\] (function)
\[\text{DeactivateCachingOfCategory}(\text{category})\] (function)

Sets the caching of \textit{category} to weak, crisp or none, respectively.

1.8.3 SetDefaultCaching

\[\text{SetDefaultCaching}(\text{type})\] (function)
\[\text{SetDefaultCachingWeak}()\] (function)
\[\text{SetDefaultCachingCrisp}()\] (function)
\[\text{DeactivateDefaultCaching}()\] (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

\[\text{DisableInputSanityChecks}(\text{category})\] (function)
\[\text{DisableOutputSanityChecks}(\text{category})\] (function)
\[\text{EnablePartialInputSanityChecks}(\text{category})\] (function)
\[\text{EnablePartialOutputSanityChecks}(\text{category})\] (function)
\[\text{EnableFullInputSanityChecks}(\text{category})\] (function)
\[\text{EnableFullOutputSanityChecks}(\text{category})\] (function)
\[\text{DisableSanityChecks}(\text{category})\] (function)
\[\text{EnablePartialSanityChecks}(\text{category})\] (function)
\[\text{EnableFullSanityChecks}(\text{category})\] (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

\[
\text{EnableAddForCategoricalOperations}(C) \quad \text{(function)}
\]

\[
\text{DisableAddForCategoricalOperations}(C) \quad \text{(function)}
\]

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

1.11 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)

\[
\text{LaTeXOutput}(c) \quad \text{(operation)}
\]

\textbf{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)

\[ \text{LaTeXOutput}(C) \] 

\textbf{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)

\[ \text{CapCategory}(a) \] (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)

\[ \text{IsEqualForObjects}(a, b) \] (operation)

Returns: a boolean

The arguments are two objects \( a \) and \( b \). The output is \( \text{true} \) if \( a = b \), otherwise the output is \( \text{false} \).

2.2.2 AddIsEqualForObjects (for IsCapCategory, IsFunction)

\[ \text{AddIsEqualForObjects}(C, F) \] (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. \( F : (a, b) \rightarrow \text{IsEqualForObjects}(a, b) \).
2.3 Categorical Properties of Objects

2.3.1 IsProjective (for IsCapCategoryObject)

\[ \text{IsProjective}(a) \]

**Returns**: a boolean

The argument is an object \( a \). The output is \texttt{true} if \( a \) is a projective object, otherwise the output is \texttt{false}.

2.3.2 AddIsProjective (for IsCapCategory, IsFunction)

\[ \text{AddIsProjective}(C, F) \]

**Returns**: nothing

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{IsProjective}. \( F : a \mapsto \text{IsProjective}(a) \).

2.3.3 IsInjective (for IsCapCategoryObject)

\[ \text{IsInjective}(a) \]

**Returns**: a boolean

The argument is an object \( a \). The output is \texttt{true} if \( a \) is an injective object, otherwise the output is \texttt{false}.

2.3.4 AddIsInjective (for IsCapCategory, IsFunction)

\[ \text{AddIsInjective}(C, F) \]

**Returns**: nothing

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{IsInjective}. \( F : a \mapsto \text{IsInjective}(a) \).

2.3.5 IsTerminal (for IsCapCategoryObject)

\[ \text{IsTerminal}(a) \]

**Returns**: a boolean

The argument is an object \( a \) of a category \( C \). The output is \texttt{true} if \( a \) is isomorphic to the terminal object of \( C \), otherwise the output is \texttt{false}.

2.3.6 AddIsTerminal (for IsCapCategory, IsFunction)

\[ \text{AddIsTerminal}(C, F) \]

**Returns**: nothing

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{IsTerminal}. \( F : a \mapsto \text{IsTerminal}(a) \).

2.3.7 IsInitial (for IsCapCategoryObject)

\[ \text{IsInitial}(a) \]

**Returns**: a boolean
The argument is an object \( a \) of a category \( C \). The output is \texttt{true} if \( a \) is isomorphic to the initial object of \( C \), otherwise the output is \texttt{false}.

### 2.3.8 \texttt{AddIsInitial (for IsCapCategory, IsFunction)}

\[
\text{\texttt{AddIsInitial}(C, F)} \quad \text{(operation)}
\]

\textbf{Returns: nothing}

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

### 2.3.9 \texttt{IsZeroForObjects (for IsCapCategoryObject)}

\[
\text{\texttt{IsZeroForObjects}(a)} \quad \text{(property)}
\]

\textbf{Returns: a boolean}

The argument is an object \( a \) of a category \( C \). The output is \texttt{true} if \( a \) is isomorphic to the zero object of \( C \), otherwise the output is \texttt{false}.

### 2.3.10 \texttt{IsZero (for IsCapCategoryObject)}

\[
\text{\texttt{IsZero}(a)} \quad \text{(property)}
\]

\textbf{Returns: a boolean}

The argument is an object \( a \) of a category \( C \). The output is \texttt{true} if \( a \) is isomorphic to the zero object of \( C \), otherwise the output is \texttt{false}.

### 2.3.11 \texttt{AddIsZeroForObjects (for IsCapCategory, IsFunction)}

\[
\text{\texttt{AddIsZeroForObjects}(C, F)} \quad \text{(operation)}
\]

\textbf{Returns: nothing}

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

### 2.4 Random Objects

CAP provides two principal methods to generate random objects:

- \textit{By integers}: The integer is simply a parameter that can be used to create a random object.
- \textit{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 \texttt{RandomObjectByInteger (for IsCapCategory, IsInt)}

\[
\text{\texttt{RandomObjectByInteger}(C, n)} \quad \text{(operation)}
\]

\textbf{Returns: an object or fail}

The arguments are a category \( C \) and an integer \( n \). The output is a random object in \( C \) or fail.
2.4.2 AddRandomObjectByInteger (for IsCapCategory, IsFunction)

- AddRandomObjectByInteger(C, F)
  
  Returns: nothing

  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)
  
  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)
  
  Returns: nothing

  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)
- RandomObject(C, L)
  
  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)
  
  Returns: true or false

  Compares two objects in the cache

2.5.2 AddIsEqualForCacheForObjects (for IsCapCategory, IsFunction)

- AddIsEqualForCacheForObjects(c, F)
  
  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)

$\triangleright$ Add\!(\text{category, object})

(\text{operation})

Adds \textit{object} as an object to \textit{category}.

2.6.2 AddObject (for IsCapCategory, IsAttributeStoringRep)

$\triangleright$ AddObject\!(\text{category, object})

(\text{operation})

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

2.6.3 AddObjectRepresentation (for IsCapCategory, IsObject)

$\triangleright$ AddObjectRepresentation\!(\text{category, filter})

(\text{operation})

The argument \textit{filter} is used to create an object type for the category \textit{category}, which is then used in ObjectifyObjectForCAPWithAttributes to objectify objects for this category.

2.6.4 ObjectifyObjectForCAPWithAttributes

$\triangleright$ ObjectifyObjectForCAPWithAttributes\!(\text{object, category, [attribute1, value1, ...]})

(\text{function})

\textbf{Returns:} an object

Objectifies the object \textit{object} with the type created for objects in the category \textit{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)

$\triangleright$ IsWellDefinedForObjects\!(a)

(\text{operation})

\textbf{Returns:} a boolean

The argument is an object \textit{a}. The output is \textit{true} if \textit{a} is well-defined, otherwise the output is \textit{false}.
2.7.2 AddIsWellDefinedForObjects (for IsCapCategory, IsFunction)

\[ \text{AddIsWellDefinedForObjects}(C, F) \]

\[ \text{Returns: nothing} \]

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 \( \pi : P \to A \).

2.8.1 SomeProjectiveObject (for IsCapCategoryObject)

\[ \text{SomeProjectiveObject}(A) \]

\[ \text{Returns: an object} \]

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

2.8.2 EpimorphismFromSomeProjectiveObject (for IsCapCategoryObject)

\[ \text{EpimorphismFromSomeProjectiveObject}(A) \]

\[ \text{Returns: a morphism in } \text{Hom}(P, A) \]

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

2.8.3 EpimorphismFromSomeProjectiveObjectWithGivenSomeProjectiveObject (for IsCapCategoryObject, IsCapCategoryObject)

\[ \text{EpimorphismFromSomeProjectiveObjectWithGivenSomeProjectiveObject}(A, P) \]

\[ \text{Returns: a morphism in } \text{Hom}(P, A) \]

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

2.8.4 ProjectiveLift (for IsCapCategoryMorphism, IsCapCategoryMorphism)

\[ \text{ProjectiveLift}(\pi, \varepsilon) \]

\[ \text{Returns: a morphism in } \text{Hom}(P, B) \]

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

2.8.5 AddSomeProjectiveObject (for IsCapCategory, IsFunction)

\[ \text{AddSomeProjectiveObject}(C, F) \]

\[ \text{Returns: nothing} \]
The arguments are a category $C$ and a function $F$. This operation adds the given function $F$ to the category for the basic operation $\text{SomeProjectiveObject}. F : A \mapsto P$.

### 2.8.6 AddEpimorphismFromSomeProjectiveObject (for IsCapCategory, IsFunction)

\[ \text{operation} \]

\[ \text{AddEpimorphismFromSomeProjectiveObject}(C, F) \]

**Returns:** nothing

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

### 2.8.7 AddEpimorphismFromSomeProjectiveObjectWithGivenSomeProjectiveObject (for IsCapCategory, IsFunction)

\[ \text{operation} \]

\[ \text{AddEpimorphismFromSomeProjectiveObjectWithGivenSomeProjectiveObject}(C, F) \]

**Returns:** nothing

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

### 2.8.8 AddProjectiveLift (for IsCapCategory, IsFunction)

\[ \text{operation} \]

\[ \text{AddProjectiveLift}(C, F) \]

**Returns:** nothing

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

\[ \text{attribute} \]

\[ \text{SomeInjectiveObject}(A) \]

**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)

\[ \text{attribute} \]

\[ \text{MonomorphismIntoSomeInjectiveObject}(A) \]

**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 $\text{SomeInjectiveObject}(A)$. 
2.9.3 MonomorphismIntoSomeInjectiveObjectWithGivenSomeInjectiveObject (for IsCapCategoryObject, IsCapCategoryObject)

- **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 \( \iota : A \to I \).

2.9.4 InjectiveColift (for IsCapCategoryMorphism, IsCapCategoryMorphism)

- **Returns**: a morphism in \( \text{Hom}(A,I) \)
  
  The arguments are a morphism \( \iota : B \to A \) and \( \beta : B \to I \) where \( I \) is an injective object. The output is a morphism \( \lambda : A \to I \) such that \( \lambda \circ \iota = \beta \).

2.9.5 AddSomeInjectiveObject (for IsCapCategory, IsFunction)

- **Returns**: nothing
  
  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)

- **Returns**: nothing
  
  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)

- **Returns**: nothing
  
  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)

- **Returns**: nothing
  
  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($A$)  
  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($A$, $i$)  
  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($C$, $F$)  
  Returns: nothing  
  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($A$, $i$)  
  Returns: a morphism in 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($C$, $F$)  
  Returns: nothing  
  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)

\[ \text{SimplifyObject}_\text{IsoToInputObject}(A, i) \]

\textbf{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 \((\iota_i A)_{-1}: A_i \to A\) which is the inverse of the output of SimplifyObject_IsoFromInputObject.

2.10.7 AddSimplifyObject_IsoToInputObject (for IsCapCategory, IsFunction)

\[ \text{AddSimplifyObject}_\text{IsoToInputObject}(C, F) \]

\textbf{Returns:} nothing

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 \((\iota_A)^{-1}\)
Chapter 3

Morphisms

Any GAP object satisfying \texttt{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 \texttt{CapCategory} (for \texttt{IsCapCategoryMorphism})

\begin{verbatim}
\texttt{CapCategory(alpha)}
\end{verbatim}

\texttt{Returns:} a category

The argument is a morphism $\alpha$. The output is the category $C$ to which $\alpha$ was added.

3.1.2 \texttt{Source} (for \texttt{IsCapCategoryMorphism})

\begin{verbatim}
\texttt{Source(alpha)}
\end{verbatim}

\texttt{Returns:} an object

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

3.1.3 \texttt{Range} (for \texttt{IsCapCategoryMorphism})

\begin{verbatim}
\texttt{Range(alpha)}
\end{verbatim}

\texttt{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 \texttt{IsMonomorphism} (for \texttt{IsCapCategoryMorphism})

\begin{verbatim}
\texttt{IsMonomorphism(alpha)}
\end{verbatim}

\texttt{Returns:} a boolean

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

23
3.2.2 AddIsMonomorphism (for IsCapCategory, IsFunction)

\( \triangle AddIsMonomorphism(C, F) \) (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 \( IsMonomorphism. F : \alpha \mapsto IsMonomorphism(\alpha). \)

3.2.3 IsEpimorphism (for IsCapCategoryMorphism)

\( \triangle IsEpimorphism(\alpha) \) (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)

\( \triangle AddIsEpimorphism(C, F) \) (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 \( IsEpimorphism. F : \alpha \mapsto IsEpimorphism(\alpha). \)

3.2.5 IsIsomorphism (for IsCapCategoryMorphism)

\( \triangle IsIsomorphism(\alpha) \) (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)

\( \triangle AddIsIsomorphism(C, F) \) (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 \( IsIsomorphism. F : \alpha \mapsto IsIsomorphism(\alpha). \)

3.2.7 IsSplitMonomorphism (for IsCapCategoryMorphism)

\( \triangle IsSplitMonomorphism(\alpha) \) (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)

\( \triangle AddIsSplitMonomorphism(C, F) \) (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 \( IsSplitMonomorphism. F : \alpha \mapsto IsSplitMonomorphism(\alpha). \)
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 $\text{IsSplitEpimorphism}$. $F : \alpha \mapsto \text{IsSplitEpimorphism}(\alpha)$.

3.2.11 IsOne (for IsCapCategoryMorphism)

▷ IsOne(alpha)  (property)

Returns: a boolean

The argument is a morphism $\alpha : a \to a$. 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 $\text{IsOne}$. $F : \alpha \mapsto \text{IsOne}(\alpha)$.

3.2.13 IsIdempotent (for IsCapCategoryMorphism)

▷ IsIdempotent(alpha)  (property)

Returns: a boolean

The argument is a morphism $\alpha : a \to a$. 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 $\text{IsIdempotent}$. $F : \alpha \mapsto \text{IsIdempotent}(\alpha)$.

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

### 3.3.2 AddRandomMorphismWithFixedSourceByInteger (for IsCapCategory, IsFunction)

- **AddRandomMorphismWithFixedSourceByInteger**
  - **Returns**: nothing
  - 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 \( \text{fail} \) or to a random morphism in \( C \) whose source is \( a \).

### 3.3.3 RandomMorphismWithFixedSourceByList (for IsCapCategoryObject, IsList)

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

### 3.3.4 AddRandomMorphismWithFixedSourceByList (for IsCapCategory, IsFunction)

- **AddRandomMorphismWithFixedSourceByList**
  - **Returns**: nothing
  - 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 \( \text{fail} \) or to a random morphism in \( C \) whose source is \( a \).

### 3.3.5 RandomMorphismWithFixedRangeByInteger (for IsCapCategoryObject, IsInt)

- **RandomMorphismWithFixedRangeByInteger**
  - **Returns**: a morphism in \( \text{Hom}(a, b) \) or \( \text{fail} \)
  - 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 \( \text{fail} \).
3.3.6 **AddRandomMorphismWithFixedRangeByInteger** (for IsCapCategory, IsFunction)

▷ AddRandomMorphismWithFixedRangeByInteger\( (C, F) \)

**Returns**: nothing

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\( (b, L) \)

**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\( (C, F) \)

**Returns**: nothing

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\( (a, b, n) \)

**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\( (C, F) \)

**Returns**: nothing

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\( (a, b, L) \)

**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)

\[ \text{AddRandomMorphismWithFixedSourceAndRangeByList}(C, F) \] (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 \text{fail} or to a random morphism in \( C \) from \( a \) to \( b \).

3.3.13 RandomMorphismByInteger (for IsCapCategory, IsInt)

\[ \text{RandomMorphismByInteger}(C, n) \] (operation)

**Returns:** a morphism or \text{fail}

The arguments are a category \( C \) and an integer \( n \). The output is a random morphism in \( C \) or \text{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)

\[ \text{AddRandomMorphismByInteger}(C, F) \] (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 \text{fail} or to a random morphism in \( C \).

3.3.15 RandomMorphismByList (for IsCapCategory, IsList)

\[ \text{RandomMorphismByList}(C, L) \] (operation)

**Returns:** a morphism or \text{fail}

The arguments are a category \( C \) and a list \( L \). The output is a random morphism in \( C \) or \text{fail}.

3.3.16 AddRandomMorphismByList (for IsCapCategory, IsFunction)

\[ \text{AddRandomMorphismByList}(C, F) \] (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 \text{fail} or to a random morphism in \( C \).

3.3.17 RandomMorphismWithFixedSource (for IsCapCategoryObject, IsInt)

\[ \text{RandomMorphismWithFixedSource}(a, n) \] (operation)
\[ \text{RandomMorphismWithFixedSource}(a, L) \] (operation)
\[ \text{RandomMorphismWithFixedRange}(b, n) \] (operation)
\[ \text{RandomMorphismWithFixedRange}(b, L) \] (operation)
\[ \text{RandomMorphismWithFixedSourceAndRange}(a, b, n) \] (operation)
RandomMorphismWithFixedSourceAndRange\((a, b, L)\)

RandomMorphism\((C, n)\)

RandomMorphism\((C, L)\)

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)

\[
\text{IsIdenticalToIdentityMorphism}(\alpha)
\]

Returns: a boolean

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

3.4.2 AddIsIdenticalToIdentityMorphism (for IsCapCategory, IsFunction)

\[
\text{AddIsIdenticalToIdentityMorphism}(C, F)
\]

Returns: nothing

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)

\[
\text{IsIdenticalToZeroMorphism}(\alpha)
\]

Returns: a boolean

The argument is a morphism \(\alpha : a \to b\). The output is true if \(\alpha = 0\), otherwise the output is false.

3.4.4 AddIsIdenticalToZeroMorphism (for IsCapCategory, IsFunction)

\[
\text{AddIsIdenticalToZeroMorphism}(C, F)
\]

Returns: nothing

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)

\[
\text{IsEndomorphism}(\alpha)
\]

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)

\[\text{AddIsEndomorphism}(C, F)\] (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 \(\text{IsEndomorphism}. \ F : \alpha \mapsto \text{IsEndomorphism}(\alpha)\).

3.4.7 IsAutomorphism (for IsCapCategoryMorphism)

\[\text{IsAutomorphism}(\alpha)\] (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)

\[\text{AddIsAutomorphism}(C, F)\] (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 \(\text{IsAutomorphism}. \ F : \alpha \mapsto \text{IsAutomorphism}(\alpha)\).

3.5 Adding Morphisms to a Category

3.5.1 Add (for IsCapCategory, IsCapCategoryMorphism)

\[\text{Add}(\text{category}, \text{morphism})\] (operation)

Adds \(\text{morphism}\) as a morphism to \(\text{category}\).

3.5.2 AddMorphism (for IsCapCategory, IsAttributeStoringRep)

\[\text{AddMorphism}(\text{category}, \text{morphism})\] (operation)

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

3.5.3 AddMorphismRepresentation (for IsCapCategory, IsObject)

\[\text{AddMorphismRepresentation}(\text{category}, \text{filter})\] (operation)

The argument \(\text{filter}\) is used to create a morphism type for the category \(\text{category}\), which is then used in \(\text{ObjectifyMorphismWithSourceAndRangeForCAPWithAttributes}\) to objectify morphisms for this category.

3.5.4 ObjectifyMorphismWithSourceAndRangeForCAPWithAttributes

\[\text{ObjectifyMorphismWithSourceAndRangeForCAPWithAttributes}(\text{morphism}, \text{category}, \text{source}, \text{range}[, \text{attr1}, \text{val1}, \text{attr2}, \text{val2}, ...])\] (function)

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

### 3.6 Equality and Congruence for Morphisms

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

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

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

\( \triangleright \text{AddIsCongruentForMorphisms}(C, F) \)  
**Returns:** nothing  
The arguments are a category \( C \) and a function \( F \). This operation adds the given function \( F \) to the category for the basic operation \( \text{IsCongruentForMorphisms} \). \( F : (\alpha, \beta) \mapsto \text{IsCongruentForMorphisms}(\alpha, \beta) \).

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

\( \triangleright \text{IsEqualForMorphisms}(\alpha, \beta) \)  
**Returns:** a boolean  
The arguments are two morphisms \( \alpha, \beta : a \to b \). The output is \( \text{true} \) if \( \alpha = \beta \), otherwise the output is \( \text{false} \).

#### 3.6.4 AddIsEqualForMorphisms (for IsCapCategory, IsFunction)

\( \triangleright \text{AddIsEqualForMorphisms}(C, F) \)  
**Returns:** nothing  
The arguments are a category \( C \) and a function \( F \). This operation adds the given function \( F \) to the category for the basic operation \( \text{IsEqualForMorphisms} \). \( F : (\alpha, \beta) \mapsto \text{IsEqualForMorphisms}(\alpha, \beta) \).

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

\( \triangleright \text{IsEqualForMorphismsOnMor}(\alpha, \beta) \)  
**Returns:** a boolean  
The arguments are two morphisms \( \alpha : a \to b, \beta : c \to d \). The output is \( \text{true} \) if \( \alpha = \beta \), otherwise the output is \( \text{false} \).
3.6.6 AddIsEqualForMorphismsOnMor (for IsCapCategory, IsFunction)

\[ \text{AddIsEqualForMorphismsOnMor}(C, F) \]

**Returns:** nothing

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

3.7 Basic Operations for Morphisms in Ab-Categories

3.7.1 IsZeroForMorphisms (for IsCapCategoryMorphism)

\[ \text{IsZeroForMorphisms}(\alpha) \]

**Returns:** a boolean

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

3.7.2 AddIsZeroForMorphisms (for IsCapCategory, IsFunction)

\[ \text{AddIsZeroForMorphisms}(C, F) \]

**Returns:** nothing

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

3.7.3 AdditionForMorphisms (for IsCapCategoryMorphism, IsCapCategoryMorphism)

\[ \text{AdditionForMorphisms}(\alpha, \beta) \]

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

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

3.7.4 AddAdditionForMorphisms (for IsCapCategory, IsFunction)

\[ \text{AddAdditionForMorphisms}(C, F) \]

**Returns:** nothing

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

3.7.5 SubtractionForMorphisms (for IsCapCategoryMorphism, IsCapCategoryMorphism)

\[ \text{SubtractionForMorphisms}(\alpha, \beta) \]

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

The arguments are two morphisms \( \alpha, \beta : a \to b \). The output is the addition \( \alpha - \beta \).
3.7.6 AddSubtractionForMorphisms (for IsCapCategory, IsFunction)

- AddSubtractionForMorphisms(\(C, F\)) (operation)
  - Returns: nothing
  - The arguments are a category \(C\) and a function \(F\). This 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 \(\alpha : a \to b\). The output is its additive inverse \(-\alpha\).

3.7.8 AddAdditiveInverseForMorphisms (for IsCapCategory, IsFunction)

- AddAdditiveInverseForMorphisms(\(C, F\)) (operation)
  - Returns: nothing
  - The arguments are a category \(C\) and a function \(F\). This 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 \(\alpha : a \to b\). 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 \(\alpha : a \to b\) 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 \(R!.\text{interpret_rationals_func}\). 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)

\[
\text{ZeroMorphism}(a, b) \quad \text{(operation)}
\]

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

The arguments are two objects \(a\) and \(b\). The output is the zero morphism \(0 : a \to b\).

3.7.13 AddZeroMorphism (for IsCapCategory, IsFunction)

\[
\text{AddZeroMorphism}(C, F) \quad \text{(operation)}
\]

Returns: nothing

The arguments are a category \(C\) and a function \(F\). This operation adds the given function \(F\) to the category for the basic operation \(\text{ZeroMorphism}\). \(F : (a, b) \mapsto (0 : a \to b)\).

3.8 Subobject and Factorobject Operations

Subobjects of an object \(c\) are monomorphisms with range \(c\) and a special function for 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)

\[
\text{IsEqualAsSubobjects}(\alpha, \beta) \quad \text{(operation)}
\]

Returns: a boolean

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

3.8.2 AddIsEqualAsSubobjects (for IsCapCategory, IsFunction)

\[
\text{AddIsEqualAsSubobjects}(C, F) \quad \text{(operation)}
\]

Returns: nothing

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

3.8.3 IsEqualAsFactorobjects (for IsCapCategoryMorphism, IsCapCategoryMorphism)

\[
\text{IsEqualAsFactorobjects}(\alpha, \beta) \quad \text{(operation)}
\]

Returns: a boolean

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

3.8.4 AddIsEqualAsFactorobjects (for IsCapCategory, IsFunction)

\[
\text{AddIsEqualAsFactorobjects}(C, F) \quad \text{(operation)}
\]

Returns: nothing
The arguments are a category $C$ and a function $F$. This operation adds the given function $F$ to the category for the basic operation $\text{IsEqualAsFactorobjects}$. $F : (\alpha, \beta) \mapsto \text{IsEqualAsFactorobjects}(\alpha, \beta)$.

3.8.5  $\text{IsDominating}$ (for $\text{IsCapCategoryMorphism}$, $\text{IsCapCategoryMorphism}$)

$\triangleright \text{IsDominating}(\text{alpha}, \text{beta})$  
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 $t : a \rightarrow b$ such that $\beta \circ t \sim_{a,c} \alpha$, otherwise the output is false.

3.8.6  $\text{AddIsDominating}$ (for $\text{IsCapCategory}$, $\text{IsFunction}$)

$\triangleright \text{AddIsDominating}(C, F)$  
Returns: nothing

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

3.8.7  $\text{IsCodominating}$ (for $\text{IsCapCategoryMorphism}$, $\text{IsCapCategoryMorphism}$)

$\triangleright \text{IsCodominating}(\text{alpha}, \text{beta})$  
Returns: a boolean

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

3.8.8  $\text{AddIsCodominating}$ (for $\text{IsCapCategory}$, $\text{IsFunction}$)

$\triangleright \text{AddIsCodominating}(C, F)$  
Returns: nothing

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

3.9.1  IdentityMorphism (for IsCapCategoryObject)

\[
\text{IdentityMorphism}(\alpha) \quad \text{attribute}
\]

**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)

\[
\text{AddIdentityMorphism}(C, F) \quad \text{operation}
\]

**Returns:** nothing

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)

\[
\text{PreCompose}(\alpha, \beta) \quad \text{operation}
\]

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

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

3.9.4  PreCompose (for IsList)

\[
\text{PreCompose}(L) \quad \text{operation}
\]

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

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

3.9.5  AddPreCompose (for IsCapCategory, IsFunction)

\[
\text{AddPreCompose}(C, F) \quad \text{operation}
\]

**Returns:** nothing

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)

\[
\text{PostCompose}(\beta, \alpha) \quad \text{operation}
\]

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

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

3.9.7  PostCompose (for IsList)

\[
\text{PostCompose}(L) \quad \text{operation}
\]

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

This is a convenience method. The argument is a list of morphisms \( L = (\alpha_n : a_n \to a_{n+1}, \alpha_{n-1} : a_{n-1} \to a_n, \ldots, \alpha_1 : a_1 \to a_2) \). The output is the composition \( (\alpha_n \circ \alpha_{n-1}) \circ \ldots \circ \alpha_2 \circ \alpha_1 \).
3.9.8  **AddPostCompose (for IsCapCategory, IsFunction)**

▷ AddPostCompose\( (C, F) \)  

*Returns*: nothing

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

3.10  **Well-Definedness of Morphisms**

3.10.1  **IsWellDefinedForMorphisms (for IsCapCategoryMorphism)**

▷ IsWellDefinedForMorphisms\( (\alpha) \)  

*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\( (C, F) \)  

*Returns*: nothing

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

3.11  **Lift/Colift**

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

\[
\begin{array}{ccc}
\alpha/\beta & \alpha & c \\
\downarrow & \downarrow & \downarrow \\
b & \beta & c \\
\end{array}
\]

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

\[
\begin{array}{ccc}
\alpha & \alpha \setminus \beta & c \\
\downarrow & \downarrow & \downarrow \\
a & \beta & b \\
\end{array}
\]

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 \text{Lift} and \text{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 \( \tau: t \to a \) such that there is a morphism \( u: t \to k \) with \( \iota \circ u \sim t, a \tau \). The output is such a \( u \).

3.11.2 AddLiftAlongMonomorphism (for IsCapCategory, IsFunction)

▷ AddLiftAlongMonomorphism(C, F) (operation)

Returns: nothing

The arguments are a category \( C \) and a function \( F \). This operation adds the given function \( F \) to the category for the basic operation LiftAlongMonomorphism. The function \( F \) maps a pair \( (t, \tau) \) 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 \( \varepsilon: a \to c \) and a morphism \( \tau: a \to t \) such that there is a morphism \( u: c \to t \) with \( u \circ \varepsilon \sim_a t, \tau \). The output is such a \( u \).

3.11.4 AddColiftAlongEpimorphism (for IsCapCategory, IsFunction)

▷ AddColiftAlongEpimorphism(C, F) (operation)

Returns: nothing

The arguments are a category \( C \) and a function \( F \). This 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 \( \tau: t \to a \). The output is true if there exists a morphism \( u: t \to k \) with \( t \circ u \sim t, a \tau \). 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. \( F: (t, \tau) \mapsto \text{IsLiftableAlongMonomorphism}(t, \tau) \).
3.11.7 IsColiftableAlongEpimorphism (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ IsColiftableAlongEpimorphism(epsilon, tau)  
(operation)

Returns: a boolean

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

3.11.8 AddIsColiftableAlongEpimorphism (for IsCapCategory, IsFunction)

▷ AddIsColiftableAlongEpimorphism(C, F)  
(operation)

Returns: nothing

The arguments are a category \( C \) and a function \( F \). This 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 \( \alpha : a \to c \), \( \beta : b \to c \). The output is a lift \( \alpha/\beta : a \to b \) of \( \alpha \) along \( \beta \) if such a lift exists or fail if it doesn’t. Recall that a lift \( \alpha/\beta : a \to b \) of \( \alpha \) along \( \beta \) is a morphism such that \( \beta \circ (\alpha/\beta) \sim_{a,b} \alpha \).

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 \( \alpha : a \to c \), \( \beta : a \to b \). The output is a colift \( \alpha \setminus \beta : c \to b \) of \( \beta \) along \( \alpha \) if such a colift exists or fail if it doesn’t. Recall that a colift \( \alpha \setminus \beta : c \to b \) of \( \beta \) along \( \alpha \) is a morphism such that \( (\alpha \setminus \beta) \circ \alpha \sim_{a,b} \beta \).

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 \( \alpha \setminus \beta \) if it exists, and to fail otherwise.
3.11.13 IsLiftable (for IsCapCategoryMorphism, IsCapCategoryMorphism)

\[ \text{returns} : \text{a boolean} \]

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

3.11.14 AddIsLiftable (for IsCapCategory, IsFunction)

\[ \text{returns} : \text{nothing} \]

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)

\[ \text{returns} : \text{a boolean} \]

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

3.11.16 AddIsColiftable (for IsCapCategory, IsFunction)

\[ \text{returns} : \text{nothing} \]

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

\[ \text{id}_a \quad \alpha \quad \text{id}_b \quad \alpha^{-1} \]

3.12.1 Inverse (for IsCapCategoryMorphism)

\[ \text{returns} : \text{a morphism in Hom}(b,a) \]

The argument is an isomorphism \( \alpha : a \to b \). The output is its inverse \( \alpha^{-1} : b \to a \).
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 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 $H : C^{op} \times C \to D$ (when $C$ and $D$ are Ab-categories, $H$ is assumed to be bilinear).
- an object $1 \in D$, called the distinguished object,
- a bijection $\nu : \text{Hom}_C(a, b) \simeq \text{Hom}_D(1, H(a, b))$ 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 $\alpha : a \to a', \beta : b \to b'$ 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 $\alpha : a \to a', \beta : b \to b'$ in $C$, and an object $r = H(a, b')$ in $D$. The output is the value of the homomorphism structure on morphisms $H(\alpha, \beta)$.

3.15.5 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 \( \text{HomomorphismStructureOnMorphismsWithGivenObjects} \). \( F : (s, \alpha : a \to a', \beta : b \to b', r) \mapsto H(\alpha, \beta) \).

### 3.15.6 DistinguishedObjectOfHomomorphismStructure (for IsCapCategory)

\[
\text{DistinguishedObjectOfHomomorphismStructure}(C) \quad \text{(attribute)}
\]

**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)

\[
\text{AddDistinguishedObjectOfHomomorphismStructure}(C, F) \quad \text{(operation)}
\]

**Returns:** nothing

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

### 3.15.8 InterpretMorphismAsMorphismFromDistinguishedObjectToHomomorphismStructure (for IsCapCategoryMorphism)

\[
\text{InterpretMorphismAsMorphismFromDistinguishedObjectToHomomorphismStructure}(\alpha) \quad \text{(attribute)}
\]

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

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

### 3.15.9 AddInterpretMorphismAsMorphismFromDistinguishedObjectToHomomorphismStructure (for IsCapCategory, IsFunction)

\[
\text{AddInterpretMorphismAsMorphismFromDistinguishedObjectToHomomorphismStructure}(C, F) \quad \text{(operation)}
\]

**Returns:** nothing

The arguments are a category \( C \) and a function \( F \). This operation adds the given function \( F \) to the category for the basic operation \( \text{InterpretMorphismAsMorphismFromDistinguishedObjectToHomomorphismStructure} \). \( F : (\alpha : a \to a') \mapsto (\nu(\alpha) : 1 \to H(a, a')) \).

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

\[
\text{InterpretMorphismFromDistinguishedObjectToHomomorphismStructureAsMorphism}(a, a', \iota) \quad \text{(operation)}
\]

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

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

\[
\text{AddInterpretMorphismFromDistinguishedObjectToHomomorphismStructureAsMorphism}(C, F)
\]

Returns: nothing

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

\[
F : (a, a', 1 : 1 \to H(a, a')) \mapsto (v^{-1}(1) : a \to a')
\]

3.15.12 SolveLinearSystemInAbCategory (for IsList, IsList, IsList)

\[
\text{SolveLinearSystemInAbCategory}(\alpha, \beta, \gamma)
\]

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

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

3.15.13 MereExistenceOfSolutionOfLinearSystemInAbCategory (for IsList, IsList, IsList)

\[
\text{MereExistenceOfSolutionOfLinearSystemInAbCategory}(\alpha, \beta, \gamma)
\]

Returns: a boolean

Like \(\text{SolveLinearSystemInAbCategory}\), but the output is simply true if a solution exists, false otherwise.

3.15.14 HomStructure (for IsCapCategoryMorphism, IsCapCategoryMorphism)

\[
\text{HomStructure}(\alpha, \beta)
\]

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

This is a convenience method. The arguments are two morphisms \(\alpha : a \to a', \beta : b \to b'\) in \(C\). The output is \(\text{HomomorphismStructureOnMorphisms}\) called on \(\alpha, \beta\).

3.15.15 HomStructure (for IsCapCategoryMorphism, IsCapCategoryObject)

\[
\text{HomStructure}(\alpha, b)
\]

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

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

3.15.16 HomStructure (for IsCapCategoryObject, IsCapCategoryMorphism)

\[
\text{HomStructure}(a, \beta)
\]

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 \(\beta : b \to b'\) in \(C\). The output is \(\text{HomomorphismStructureOnMorphisms}\) called on \(\text{id}_a, \beta\).

### 3.15.17 HomStructure (for IsCapCategoryObject, IsCapCategoryObject)

\[\text{HomStructure}(a, b)\]  
(returns: an object)

This is a convenience method. The arguments are two objects \(a\) and \(b\) in \(C\). The output is \(\text{HomomorphismStructureOnObjects}\) called on \(a, b\).

### 3.15.18 HomStructure (for IsCapCategoryMorphism)

\[\text{HomStructure}(\text{arg})\]  
(returns:

This is a convenience method for \(\text{InterpretMorphismAsMorphismFromDistinguishedObjectToHomomorphismStructure}\).

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

\[\text{HomStructure}(\text{arg1, arg2, arg3})\]  
(returns:

This is a convenience method for \(\text{InterpretMorphismFromDistinguishedObjectToHomomorphismStructureAsMorphism}\).

### 3.15.20 HomStructure (for IsCapCategory)

\[\text{HomStructure}(\text{arg})\]  
(returns:

This is a convenience method for \(\text{DistinguishedObjectOfHomomorphismStructure}\).

### 3.15.21 BasisOfExternalHom (for IsCapCategoryObject, IsCapCategoryObject)

\[\text{BasisOfExternalHom}(a, b)\]  
(returns:

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 \(\alpha : a \to b\) 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)

\[\text{AddBasisOfExternalHom}(C, F)\]  
(returns:

The arguments are a category \(C\) and a function \(F\). This operation adds the given function \(F\) to the category for the basic operation \(\text{BasisOfExternalHom}\).
3.15.23 CoefficientsOfMorphismWithGivenBasisOfExternalHom (for IsCapCategoryMorphism, IsList)

\[ \text{CoefficientsOfMorphismWithGivenBasisOfExternalHom}(\alpha, L) \] (operation)

Returns: a list of elements in \( k \)

The arguments are a morphism \( \alpha : a \to b \) 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)

\[ \text{AddCoefficientsOfMorphismWithGivenBasisOfExternalHom}(C, F) \] (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 \( \text{CoefficientsOfMorphismWithGivenBasisOfExternalHom} \).

3.15.25 CoefficientsOfMorphism (for IsCapCategoryMorphism)

\[ \text{CoefficientsOfMorphism}(\alpha) \] (attribute)

Returns: a list of elements in \( k \)

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

3.16 Simplified Morphisms

Let \( \phi : A \to B \) 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 \( \text{IsCongruentForMorphisms} \) 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 \( A \to A \), we also have

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

Let \( C \) 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 \( C \), 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-operatations, 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)

\[ \text{SimplifyMorphism}(\phi, i) \] (operation)

Returns: a morphism in 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)

\[ \text{AddSimplifyMorphism}(C, F) \] (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 \( \text{SimplifyMorphism} \). The function \( F \) maps \( (\phi, i \geq 1) \) to \( \phi_i \).

If we regard \( \phi \) as an object in the category \( \sum_i \text{Hom}(A,B) \), 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)

\[ \text{SimplifySource}(\phi, i) \] (operation)

Returns: a morphism in 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)

\[ \text{AddSimplifySource}(C, F) \] (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 \( \text{SimplifySource} \). The function \( F \) maps \( (\phi, i \geq 1) \) to \( \phi_i \).

3.16.5 SimplifySource_IsoToInputObject (for IsCapCategoryMorphism, IsObject)

\[ \text{SimplifySource_IsoToInputObject}(\phi, i) \] (operation)

Returns: a morphism in 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)

\[ \text{AddSimplifySource_IsoToInputObject}(C, F) \] (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 \( \text{SimplifySource_IsoToInputObject} \). The function \( F \) maps \( (\phi, i) \) to \( (\sigma_i)^{-1} \).
3.16.7  \textbf{SimplifySource\_IsoFromInputObject (for IsCapCategoryMorphism, IsObject)}

\begin{verbatim}
▷ SimplifySource\_IsoFromInputObject(phi, i)
    \textbf{Returns:} a morphism in Hom(A,A_i)
    The arguments are a morphism \(\phi : A \to B\) and a non-negative integer \(i\) or infinity. The output
    is the isomorphism \(\sigma_i : A_i \to A_i\).
\end{verbatim}

3.16.8  \textbf{AddSimplifySource\_IsoFromInputObject (for IsCapCategory, IsFunction)}

\begin{verbatim}
▷ AddSimplifySource\_IsoFromInputObject(C, F)
    \textbf{Returns:} nothing
    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 \((\sigma_i)\).
\end{verbatim}

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

3.16.9  \textbf{SimplifyRange (for IsCapCategoryMorphism, IsObject)}

\begin{verbatim}
▷ SimplifyRange(phi, i)
    \textbf{Returns:} a morphism in Hom(A,B_i)
    The arguments are a morphism \(\phi : A \to B\) and a non-negative integer \(i\) or infinity. The output
    is a simplified morphism with simplified range \(\phi_i : A \to B_i\).
\end{verbatim}

3.16.10  \textbf{AddSimplifyRange (for IsCapCategory, IsFunction)}

\begin{verbatim}
▷ AddSimplifyRange(C, F)
    \textbf{Returns:} nothing
    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 \(\phi_i\).
\end{verbatim}

3.16.11  \textbf{SimplifyRange\_IsoToInputObject (for IsCapCategoryMorphism, IsObject)}

\begin{verbatim}
▷ SimplifyRange\_IsoToInputObject(phi, i)
    \textbf{Returns:} a morphism in Hom(B_i,B)
    The arguments are a morphism \(\phi : A \to B\) and a non-negative integer \(i\) or infinity. The output
    is the isomorphism \((\rho_i)^{-1} : B_i \to B\).
\end{verbatim}

3.16.12  \textbf{AddSimplifyRange\_IsoToInputObject (for IsCapCategory, IsFunction)}

\begin{verbatim}
▷ AddSimplifyRange\_IsoToInputObject(C, F)
    \textbf{Returns:} nothing
    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
    \((\rho_i)^{-1}\).
\end{verbatim}
3.16.13 SimplifyRange_IsoFromInputObject (for IsCapCategoryMorphism, IsObject)

▷ SimplifyRange_IsoFromInputObject(\(\phi, i\))

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

The arguments are a morphism \(\phi: A \to B\) and a non-negative integer \(i\) or \(\infty\). The output is the isomorphism \(\rho_i: B \to B_i\).

3.16.14 AddSimplifyRange_IsoFromInputObject (for IsCapCategory, IsFunction)

▷ AddSimplifyRange_IsoFromInputObject(\(C, F\))

**Returns**: nothing

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 \(\Sigma_{A,B} \text{Hom}(A, B)\), then \(\phi_i\) is a morphism of type \(A_i \to B_i\) and there is an isomorphism \(\sigma_i: A \to A_i\) and \(\rho_i: B \to B_i\) 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(\(\phi, i\))

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

The arguments are a morphism \(\phi: A \to B\) and a non-negative integer \(i\) or \(\infty\). The output is a simplified morphism with simplified source and range \(\phi_i: A_i \to B_i\).

3.16.16 AddSimplifySourceAndRange (for IsCapCategory, IsFunction)

▷ AddSimplifySourceAndRange(\(C, F\))

**Returns**: nothing

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(\(\phi, i\))

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

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

3.16.18 AddSimplifySourceAndRange_IsoToInputRange (for IsCapCategory, IsFunction)

▷ AddSimplifySourceAndRange_IsoToInputRange(\(C, F\))

**Returns**: nothing
The arguments are a category $C$ and a function $F$. This operation adds the given function $F$ to the category for the basic operation \texttt{SimplifySourceAndRange\_IsoToInputRange}. The function $F$ maps $(\phi, i)$ to $(\rho_i)^{-1}$.

### 3.16.19 \texttt{SimplifySourceAndRange\_IsoFromInputRange} (for \texttt{IsCapCategoryMorphism, IsObject})

▷ \texttt{SimplifySourceAndRange\_IsoFromInputRange(phi, i)}

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

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 \texttt{AddSimplifySourceAndRange\_IsoFromInputRange} (for \texttt{IsCapCategory, IsFunction})

▷ \texttt{AddSimplifySourceAndRange\_IsoFromInputRange(C, F)}

**Returns:** nothing

The arguments are a category $C$ and a function $F$. This operation adds the given function $F$ to the category for the basic operation \texttt{SimplifySourceAndRange\_IsoFromInputRange}. The function $F$ maps $(\phi, i)$ to $\rho_i$.

### 3.16.21 \texttt{SimplifySourceAndRange\_IsoToInputSource} (for \texttt{IsCapCategoryMorphism, IsObject})

▷ \texttt{SimplifySourceAndRange\_IsoToInputSource(phi, i)}

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

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

### 3.16.22 \texttt{AddSimplifySourceAndRange\_IsoToInputSource} (for \texttt{IsCapCategory, IsFunction})

▷ \texttt{AddSimplifySourceAndRange\_IsoToInputSource(C, F)}

**Returns:** nothing

The arguments are a category $C$ and a function $F$. This operation adds the given function $F$ to the category for the basic operation \texttt{SimplifySourceAndRange\_IsoToInputSource}. The function $F$ maps $(\phi, i)$ to $(\sigma_i)^{-1}$.

### 3.16.23 \texttt{SimplifySourceAndRange\_IsoFromInputSource} (for \texttt{IsCapCategoryMorphism, IsObject})

▷ \texttt{SimplifySourceAndRange\_IsoFromInputSource(phi, i)}

**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)  
  Returns: nothing

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 $\phi : A \to A$ is an endomorphism, we may regard it as an object in the category $\Sigma^A \text{Hom}(A, A)$. In this case $\phi_i$ is a morphism of type $A_i \to A_i$ and there is an isomorphism $\sigma_i : A \to A_i$ 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)  
  Returns: a morphism in $\text{Hom}(A_i, A_i)$

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

3.16.26 AddSimplifyEndo (for IsCapCategory, IsFunction)

▷ AddSimplifyEndo(C, F)  
  Returns: nothing

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 $\phi$.

3.16.27 SimplifyEndo_IsoToInputObject (for IsCapCategoryMorphism, IsObject)

▷ SimplifyEndo_IsoToInputObject(phi, i)  
  Returns: a morphism in $\text{Hom}(A_i, A)$

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

3.16.28 AddSimplifyEndo_IsoToInputObject (for IsCapCategory, IsFunction)

▷ AddSimplifyEndo_IsoToInputObject(C, F)  
  Returns: nothing

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

The arguments are an endomorphism $\phi : A \to A$ and a non-negative integer $i$ or infinity. The output is the isomorphism $\sigma_i : A \to A_i$. 
3.16.30 AddSimplifyEndo_IsoFromInputObject (for IsCapCategory, IsFunction)

\[ \text{AddSimplifyEndo}_\text{IsoFromInputObject} (C, F) \]

(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 \text{SimplifyEndo}_\text{IsoFromInputObject}. The function \( F \) maps \((\phi, i)\) to \((\sigma_i)\).

3.16.31 Simplify (for IsCapCategoryMorphism)

\[ \text{Simplify}(\phi) \]

(attribute)

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

This is a convenient method. The argument is a morphism \( \phi : A \to B \). 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 \( (t_A^\infty)^{-1} \circ \phi \circ t_A^\infty \).

3.17 Reduction by split epi summands

Let \( \alpha : A \to B \) be a morphism in an additive category. Suppose we are given direct sum decompositions of \( A \cong A' \oplus A'' \) and \( B \cong B' \oplus B'' \) 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 \( \alpha' : A' \to B' \) some reduction of \( \alpha \) by split epi summands. The inclusions/projections of the decompositions into direct sums induce commutative diagrams

\[
\begin{array}{ccc}
A' & \xrightarrow{\alpha'} & B' \\
\uparrow & & \uparrow \\
A & \xrightarrow{\alpha} & B
\end{array}
\]

and

\[
\begin{array}{ccc}
A' & \xrightarrow{\alpha'} & B' \\
\uparrow & & \uparrow \\
A & \xrightarrow{\alpha} & B
\end{array}
\]
3.17.1 SomeReductionBySplitEpiSummand (for IsCapCategoryMorphism)

- **SomeReductionBySplitEpiSummand** (alpha) (attribute)
  - **Returns:** a morphism in \( \text{Hom}(A', B') \)
  - The argument is a morphism \( \alpha : A \to B \). The output is some reduction of \( \alpha \) by split epi summands \( \alpha' : A' \to B' \).

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 \( \alpha : A \to B \). The output is the morphism \( \beta' : B' \to B \) 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 \( \alpha : A \to B \). The output is the morphism \( \beta : B \to B' \) 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)

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)

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)

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)

Returns: nothing

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

4.2.3 HorizontalPreCompose (for IsCapCategoryTwoCell, IsCapCategoryTwoCell)

> HorizontalPreCompose(c, d)

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 \( d \circ c : (\gamma \circ \alpha) \rightarrow (\delta \circ \beta) \).
4.2.4 AddHorizontalPreCompose (for IsCapCategory, IsFunction)

\[ \text{AddHorizontalPreCompose}(C, F) \text{ (operation)} \]

**Returns:** nothing

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. \( F : (c, d) \mapsto d \circ c. \)

4.2.5 HorizontalPostCompose (for IsCapCategoryTwoCell, IsCapCategoryTwoCell)

\[ \text{HorizontalPostCompose}(d, c) \text{ (operation)} \]

**Returns:** a 2-cell

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

4.2.6 AddHorizontalPostCompose (for IsCapCategory, IsFunction)

\[ \text{AddHorizontalPostCompose}(C, F) \text{ (operation)} \]

**Returns:** nothing

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. \( F : (d, c) \mapsto d \circ c. \)

4.2.7 VerticalPreCompose (for IsCapCategoryTwoCell, IsCapCategoryTwoCell)

\[ \text{VerticalPreCompose}(c, d) \text{ (operation)} \]

**Returns:** a 2-cell

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

4.2.8 AddVerticalPreCompose (for IsCapCategory, IsFunction)

\[ \text{AddVerticalPreCompose}(C, F) \text{ (operation)} \]

**Returns:** nothing

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)

\[ \text{VerticalPostCompose}(d, c) \text{ (operation)} \]

**Returns:** a 2-cell

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

4.2.10 AddVerticalPostCompose (for IsCapCategory, IsFunction)

\[ \text{AddVerticalPostCompose}(C, F) \text{ (operation)} \]

**Returns:** nothing

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 ↦ IsWellDefinedForMorphisms(c).
Chapter 5

Category of Categories

Categories itself with functors as morphisms form a category Cat. So the data structure of \texttt{CapCategory}s is designed to be objects in a category. This category is implemented in \texttt{CapCat}. For every category, the corresponding object in Cat can be obtained via \texttt{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 \texttt{ApplyFunctor}).

5.1 The Category Cat

5.1.1 CapCat

\texttt{CapCat} \hspace{1cm} (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)

\texttt{IsCapCategoryAsCatObject(object)} \hspace{1cm} (filter)

\textbf{Returns:} true or false

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

5.2.2 IsCapFunctor (for IsCapCategoryMorphism)

\texttt{IsCapFunctor(object)} \hspace{1cm} (filter)

\textbf{Returns:} true or false

The GAP category of functors.

5.2.3 IsCapNaturalTransformation (for IsCapCategoryTwoCell)

\texttt{IsCapNaturalTransformation(object)} \hspace{1cm} (filter)

\textbf{Returns:} true or false
The GAP category of natural transformations.

5.3 Constructors

5.3.1 AsCatObject (for IsCapCategory)

\[ \text{AsCatObject}(C) \] (attribute)

Given a CAP category \( C \), this method returns the corresponding object in Cat. For technical reasons, the filter \( \text{IsCapCategory} \) must not imply the filter \( \text{IsCapCategoryObject} \). For example, if \( \text{InitialObject} \) 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 \( \text{AsCapCategory} \).

5.3.2 AsCapCategory (for IsCapCategoryAsCatObject)

\[ \text{AsCapCategory}(C) \] (attribute)

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

5.4 Functors

Functors are morphisms in Cat, thus they have source and target which are categories. A multivariate functor can be constructed via a product category as source, a presheaf is constructed via the opposite category as source. However, the user can explicitly decide the arity of a functor (which will only have technical implications). Thus, it is for example possible to consider a functor \( A \times B \to C \) either as a unary functor with source category \( A \times B \) or as a binary functor. Moreover, an object and a morphism function can be added to a functor, to apply it to objects or morphisms in the source category.

5.4.1 CapFunctor (for IsString, IsCapCategory, IsCapCategory)

\[ \text{CapFunctor}(\text{name}, A, B) \] (operation)
\[ \text{CapFunctor}(\text{name}, A, B) \] (operation)
\[ \text{CapFunctor}(\text{name}, A, B) \] (operation)
\[ \text{CapFunctor}(\text{name}, A, B) \] (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 \( \text{CapCat} \) or as a CAP-category.

5.4.2 CapFunctor (for IsString, IsList, IsCapCategory)

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

5.4.3 SourceOfFunctor (for IsCapFunctor)

\[ \triangleright \text{SourceOfFunctor}(F) \]

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

5.4.4 RangeOfFunctor (for IsCapFunctor)

\[ \triangleright \text{RangeOfFunctor}(F) \]

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

5.4.5 AddObjectFunction (for IsCapFunctor, IsFunction)

\[ \triangleright \text{AddObjectFunction}(F, f) \]

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

5.4.6 FunctorObjectOperation (for IsCapFunctor)

\[ \triangleright \text{FunctorObjectOperation}(F) \]

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)

\[ \triangleright \text{AddMorphismFunction}(F, f) \]

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

5.4.8 FunctorMorphismOperation (for IsCapFunctor)

\[
\text{FunctorMorphismOperation}(F) \quad \text{(attribute)}
\]

\textbf{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

\[
\text{ApplyFunctor}(\texttt{func}, A[,B,\ldots]) \quad \text{(function)}
\]

\textbf{Returns:} \texttt{IsCapCategoryCell}

Applies the functor \(\texttt{func}\) either to

- an object or morphism \(A\) in the source of \(\texttt{func}\) or
- to objects or morphisms belonging to the categories in the input signature of \(\texttt{func}\).

5.4.10 InputSignature (for IsCapFunctor)

\[
\text{InputSignature}(F) \quad \text{(attribute)}
\]

\textbf{Returns:} \texttt{IsList}

The argument is a functor \(F\). The output is a list of pairs \([[A_1, b_1], \ldots, [A_n, b_n]]\) where a pair consists of a CAP-category \(A_i\) and a boolean \(b_i\) for \(i = 1, \ldots, n\). The source of \(F\) is given by the product category \(D_1 \times \ldots \times D_n\), 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)

\[
\text{InstallFunctor}(F, s) \quad \text{(operation)}
\]

\textbf{Returns:} nothing

The arguments are a functor \(F\) and a string \(s\). To simplify the description of this operation, we let \([[A_1, b_1], \ldots, [A_n, b_n]]\) denote the input signature of \(F\). This method tries to install 3 operations: an operation \(\omega_1\) with the name \(s\), an operation \(\omega_2\) with the name \(\texttt{IsObjects}\), and an operation \(\omega_3\) with the name \(\texttt{IsMorphisms}\). The operation \(\omega_1\) takes as input either \(n\) object/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)

\[ \text{IdentityFunctor}(\text{cat}) \]

\textbf{Returns:} a functor

Returns the identity functor of the category \text{cat} viewed as an object in the category of categories.

5.4.13 FunctorCanonicalizeZeroObjects (for IsCapCategory)

\[ \text{FunctorCanonicalizeZeroObjects}(\text{cat}) \]

\textbf{Returns:} a functor

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

5.4.14 NaturalIsomorphismFromIdentityToCanonicalizeZeroObjects (for IsCapCategory)

\[ \text{NaturalIsomorphismFromIdentityToCanonicalizeZeroObjects}(\text{cat}) \]

\textbf{Returns:} a natural transformation

Returns the natural isomorphism from the identity functor to \text{FunctorCanonicalizeZeroObjects}(\text{cat}).

5.4.15 FunctorCanonicalizeZeroMorphisms (for IsCapCategory)

\[ \text{FunctorCanonicalizeZeroMorphisms}(\text{cat}) \]

\textbf{Returns:} a functor

Returns the endofunctor of the category \text{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 \text{ZeroMorphism}(	ext{Source}(\phi), \text{Range}(\phi)). This functor is equivalent to the identity functor.

5.4.16 NaturalIsomorphismFromIdentityToCanonicalizeZeroMorphisms (for IsCapCategory)

\[ \text{NaturalIsomorphismFromIdentityToCanonicalizeZeroMorphisms}(\text{cat}) \]

\textbf{Returns:} a natural transformation

Returns the natural isomorphism from the identity functor to \text{FunctorCanonicalizeZeroMorphisms}(\text{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)

\[ \text{Name}(\text{arg}) \]

\textbf{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)

\[
\text{NaturalTransformation([name, ]F, G)} \quad \text{(operation)}
\]

Returns: a natural transformation

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

5.5.3 AddNaturalTransformationFunction (for IsCapNaturalTransformation, IsFunction)

\[
\text{AddNaturalTransformationFunction(N, func)} \quad \text{(operation)}
\]

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

5.5.4 ApplyNaturalTransformation

\[
\text{ApplyNaturalTransformation(N, A)} \quad \text{(function)}
\]

Returns: a morphism

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

5.5.5 InstallNaturalTransformation (for IsCapNaturalTransformation, IsString)

\[
\text{InstallNaturalTransformation(N, name)} \quad \text{(operation)}
\]

Installs the natural transformation \( N \) as operation with the name \( \text{name} \). Argument for this operation is an object, output is a morphism.

5.5.6 HorizontalPreComposeNaturalTransformationWithFunctor (for IsCapNaturalTransformation, IsCapFunctor)

\[
\text{HorizontalPreComposeNaturalTransformationWithFunctor(N, F)} \quad \text{(operation)}
\]

Returns: a natural transformation

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

5.5.7 HorizontalPreComposeFunctorWithNaturalTransformation (for IsCapFunctor, IsCapNaturalTransformation)

\[
\text{HorizontalPreComposeFunctorWithNaturalTransformation(F, N)} \quad \text{(operation)}
\]

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)

\[ \text{KernelObject}(\alpha) \]

\[ \text{Returns:} \text{ an object} \]

The argument is a morphism \( \alpha \). The output is the kernel \( K \) of \( \alpha \).
6.1.2  KernelEmbedding (for IsCapCategoryMorphism)

\[
\text{KernelEmbedding}(\alpha) \quad \text{(attribute)}
\]

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

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

6.1.3  KernelEmbeddingWithGivenKernelObject (for IsCapCategoryMorphism, IsCapCategoryObject)

\[
\text{KernelEmbeddingWithGivenKernelObject}(\alpha, K) \quad \text{(operation)}
\]

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

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

6.1.4  MorphismFromKernelObjectToSink (for IsCapCategoryMorphism)

\[
\text{MorphismFromKernelObjectToSink}(\alpha) \quad \text{(operation)}
\]

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

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

6.1.5  MorphismFromKernelObjectToSinkWithGivenKernelObject (for IsCapCategoryMorphism, IsCapCategoryObject)

\[
\text{MorphismFromKernelObjectToSinkWithGivenKernelObject}(\alpha, K) \quad \text{(operation)}
\]

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

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

6.1.6  KernelLift (for IsCapCategoryMorphism, IsCapCategoryMorphism)

\[
\text{KernelLift}(\alpha, \tau) \quad \text{(operation)}
\]

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

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

6.1.7  KernelLiftWithGivenKernelObject (for IsCapCategoryMorphism, IsCapCategoryMorphism)

\[
\text{KernelLiftWithGivenKernelObject}(\alpha, \tau, K) \quad \text{(operation)}
\]

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

The arguments are a morphism \(\alpha : A \to B\), a test morphism \(\tau : T \to A\) satisfying \(\alpha \circ \tau \sim T, B 0\), and an object \(K = \text{KernelObject}(\alpha)\). The output is the morphism \(u(\tau) : T \to K\) given by the universal property of the kernel.
6.1.8 AddKernelObject (for IsCapCategory, IsFunction)

\[ \text{AddKernelObject}(C, F) \]

(definition)

**Returns:** nothing

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

6.1.9 AddKernelEmbedding (for IsCapCategory, IsFunction)

\[ \text{AddKernelEmbedding}(C, F) \]

(definition)

**Returns:** nothing

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

6.1.10 AddKernelEmbeddingWithGivenKernelObject (for IsCapCategory, IsFunction)

\[ \text{AddKernelEmbeddingWithGivenKernelObject}(C, F) \]

(definition)

**Returns:** nothing

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

6.1.11 AddMorphismFromKernelObjectToSink (for IsCapCategory, IsFunction)

\[ \text{AddMorphismFromKernelObjectToSink}(C, F) \]

(definition)

**Returns:** nothing

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

6.1.12 AddMorphismFromKernelObjectToSinkWithGivenKernelObject (for IsCapCategory, IsFunction)

\[ \text{AddMorphismFromKernelObjectToSinkWithGivenKernelObject}(C, F) \]

(definition)

**Returns:** nothing

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

6.1.13 AddKernelLift (for IsCapCategory, IsFunction)

\[ \text{AddKernelLift}(C, F) \]

(definition)

**Returns:** nothing

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

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

6.1.16 **KernelObjectFunctorial** (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ KernelObjectFunctorial \((\alpha, \mu, \alpha')\)

( operation )

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

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

6.1.17 **KernelObjectFunctorialWithGivenKernelObjects** (for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject)

▷ KernelObjectFunctorialWithGivenKernelObjects \((s, \alpha, \mu, \alpha', r)\)

( operation )

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

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

6.1.18 **KernelObjectFunctorialWithGivenKernelObjects** (for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject)

▷ KernelObjectFunctorialWithGivenKernelObjects \((s, \alpha, \mu, \nu, \alpha', r)\)

( operation )

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

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

\[ \text{AddKernelObjectFunctorialWithGivenKernelObjects}(C, F) \]

**Returns:** nothing

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

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 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 CokernelObject(\( \alpha \)) \rightarrow CokernelObject(\( \alpha' \)).

6.2.1 CokernelObject (for IsCapCategoryMorphism)

\[ \text{CokernelObject}(\alpha) \]

**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)

\[ \text{CokernelProjection}(\alpha) \]

**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 \( \alpha : A \to B \) and an object \( K = \text{CokernelObject}(\alpha) \). The output is the cokernel projection \( \epsilon : B \to \text{CokernelObject}(\alpha) \).

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 \( \alpha : A \to B \). The output is the zero morphism \( 0 : A \to \text{CokernelObject}(\alpha) \).

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

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 \( \alpha : A \to B \) and a test morphism \( \tau : B \to T \) satisfying \( \tau \circ \alpha \sim_{A,T} 0 \).
   The output is the morphism \( u(\tau) : \text{CokernelObject}(\alpha) \to T \) 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 \( \alpha : A \to B \), a test morphism \( \tau : B \to T \) satisfying \( \tau \circ \alpha \sim_{A,T} 0 \), and an object \( K = \text{CokernelObject}(\alpha) \). The output is the morphism \( u(\tau) : K \to T \) 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\((C, F)\)

Returns: nothing

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. \(F : \alpha \mapsto \epsilon\).

6.2.10 AddCokernelProjectionWithGivenCokernelObject (for IsCapCategory, IsFunction)

▷ AddCokernelProjectionWithGivenCokernelObject\((C, F)\)

Returns: nothing

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. \(F : (\alpha, K) \mapsto \epsilon\).

6.2.11 AddMorphismFromSourceToCokernelObject (for IsCapCategory, IsFunction)

▷ AddMorphismFromSourceToCokernelObject\((C, F)\)

Returns: nothing

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\((C, F)\)

Returns: nothing

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\((C, F)\)

Returns: nothing

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\((C, F)\)

Returns: nothing

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 \textbf{CokernelObjectFunctorial (for IsList)}

\[ \text{CokernelObjectFunctorial}(L) \] (operation)

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

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

6.2.16 \textbf{CokernelObjectFunctorial (for IsCapCategoryMorphism, IsCapCategoryMorphism)}

\[ \text{CokernelObjectFunctorial}(\alpha, \nu, \alpha') \] (operation)

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

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

6.2.17 \textbf{CokernelObjectFunctorialWithGivenCokernelObjects (for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject)}

\[ \text{CokernelObjectFunctorialWithGivenCokernelObjects}(s, \alpha, \nu, \alpha', r) \] (operation)

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

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

6.2.18 \textbf{CokernelObjectFunctorialWithGivenCokernelObjects (for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject)}

\[ \text{CokernelObjectFunctorialWithGivenCokernelObjects}(s, \alpha, \mu, \nu, \alpha', r) \] (operation)

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

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

6.2.19 \textbf{AddCokernelObjectFunctorialWithGivenCokernelObjects (for IsCapCategory, IsFunction)}

\[ \text{AddCokernelObjectFunctorialWithGivenCokernelObjects}(C, F) \] (operation)

\textbf{Returns:} nothing

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. \(F : (\text{CokernelObject}(\alpha), \alpha, \nu, \alpha', \text{CokernelObject}(\alpha')) \to (\text{CokernelObject}(\alpha) \to \text{CokernelObject}(\alpha')).\)
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 $u_{\text{in}}(A) : A \to Z$,
- a function $u_{\text{out}}$ mapping each object $A$ to a morphism $u_{\text{out}}(A) : Z \to A$.

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

6.3.1  ZeroObject (for IsCapCategory)

▷ ZeroObject(C)  

Rules: 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)  

Rules: 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)  

Rules: a morphism in Hom(ZeroObject, A)

This is a convenience method. The argument is an object $A$. It calls UniversalMorphismFromZeroObject on $A$.

6.3.4  MorphismIntoZeroObject (for IsCapCategoryObject)

▷ MorphismIntoZeroObject(A)  

Rules: a morphism in Hom(A, ZeroObject)

This is a convenience method. The argument is an object $A$. It calls UniversalMorphismIntoZeroObject on $A$.

6.3.5  UniversalMorphismFromZeroObject (for IsCapCategoryObject)

▷ UniversalMorphismFromZeroObject(A)  

Rules: a morphism in Hom(ZeroObject, A)

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

6.3.6 UniversalMorphismFromZeroObjectWithGivenZeroObject (for IsCapCategoryObject, IsCapCategoryObject)

\[ \text{UniversalMorphismFromZeroObjectWithGivenZeroObject}(A, Z) \]

(operation)

Returns: a morphism in 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)

\[ \text{UniversalMorphismIntoZeroObject}(A) \]

(attribute)

Returns: a morphism in 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)

\[ \text{UniversalMorphismIntoZeroObjectWithGivenZeroObject}(A, Z) \]

(operation)

Returns: a morphism in 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)

\[ \text{IsomorphismFromZeroObjectToInitialObject}(C) \]

(attribute)

Returns: a morphism in 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)

\[ \text{IsomorphismFromInitialObjectToZeroObject}(C) \]

(attribute)

Returns: a morphism in 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)

\[ \text{IsomorphismFromZeroObjectToTerminalObject}(C) \]

(attribute)

Returns: a morphism in 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)

\[ \text{IsomorphismFromTerminalObjectToZeroObject}(C) \]

(attribute)

Returns: a morphism in 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)

\[ \text{AddZeroObject}(C, F) \]

Returns: nothing

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)

\[ \text{AddUniversalMorphismIntoZeroObject}(C, F) \]

Returns: nothing

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)

\[ \text{AddUniversalMorphismIntoZeroObjectWithGivenZeroObject}(C, F) \]

Returns: nothing

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)

\[ \text{AddUniversalMorphismFromZeroObject}(C, F) \]

Returns: nothing

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)

\[ \text{AddUniversalMorphismFromZeroObjectWithGivenZeroObject}(C, F) \]

Returns: nothing

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)

\[ \text{AddIsomorphismFromZeroObjectToInitialObject}(C, F) \]

Returns: nothing

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:() ↦ (InitialObject → 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:() ↦ (ZeroObject → 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:() ↦ (TerminalObject → ZeroObject).

6.3.22 ZeroObjectFunctorial (for IsCapCategory)

▷ ZeroObjectFunctorial(C) (attribute)

Returns: a morphism in Hom(ZeroObject, ZeroObject)

The argument is a category C. The output is the unique morphism ZeroObject → 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:() ↦ (T → 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 → 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 \(\text{TerminalObject}\). We say that the morphism \(u(A)\) is induced by the universal property of the terminal object. \(\text{TerminalObject}\) 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 \(\text{TerminalObject}\). \(F : () \mapsto T\).

### 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 \(\text{UniversalMorphismIntoTerminalObject}\). \(F : A \mapsto u(A)\).
6.4.7 AddUniversalMorphismIntoTerminalObjectWithGivenTerminalObject (for IsCapCategory, IsFunction)

\[ \text{AddUniversalMorphismIntoTerminalObjectWithGivenTerminalObject}(C, F) \] (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)

\[ \text{TerminalObjectFunctorial}(C) \] (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)

\[ \text{AddTerminalObjectFunctorial}(C, F) \] (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 an 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)

\[ \text{InitialObject}(C) \] (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)

\[ \text{InitialObject}(c) \] (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)  
**Returns:** a morphism in Hom(InitialObject → A). The argument is an object A. The output is the universal morphism u(A) : InitialObject → A.

6.5.4 UniversalMorphismFromInitialObjectWithGivenInitialObject (for IsCapCategoryObject, IsCapCategoryObject)

UniversalMorphismFromInitialObjectWithGivenInitialObject(A, I)  
**Returns:** a morphism in Hom(InitialObject → A). The arguments are an object A, and an object I = InitialObject. The output is the universal morphism u(A) : InitialObject → A.

6.5.5 AddInitialObject (for IsCapCategory, IsFunction)

AddInitialObject(C, F)  
**Returns:** nothing 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 : () ↦ I.

6.5.6 AddUniversalMorphismFromInitialObject (for IsCapCategory, IsFunction)

AddUniversalMorphismFromInitialObject(C, F)  
**Returns:** nothing 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 ↦ u(A).

6.5.7 AddUniversalMorphismFromInitialObjectWithGivenInitialObject (for IsCapCategory, IsFunction)

AddUniversalMorphismFromInitialObjectWithGivenInitialObject(C, F)  
**Returns:** nothing 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) ↦ u(A).

6.5.8 InitialObjectFunctorial (for IsCapCategory)

InitialObjectFunctorial(C)  
**Returns:** a morphism in Hom(InitialObject, InitialObject) The argument is a category C. The output is the unique morphism InitialObject → InitialObject.

6.5.9 AddInitialObjectFunctorial (for IsCapCategory, IsFunction)

AddInitialObjectFunctorial(C, F)  
**Returns:** nothing 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 : () ↦ (I → I).
### 6.6 Direct Sum

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

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

such that

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

where $\delta_{i,j} \in \text{Hom}(S_i, S_j)$ is the identity if $i = j$, and 0 otherwise. The 5-tuple $(S, \pi, t, u_{\text{in}}, u_{\text{out}})$ 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 $(\mu_i : S_i \to S'_i)_{i=1\ldots n}$, we obtain a morphism $\bigoplus_{i=1}^n S_i \to \bigoplus_{i=1}^n S'_i$.

![Diagram of direct sum](image)

#### 6.6.1 DirectSumOp (for IsList, IsCapCategoryObject)

\[ \text{DirectSumOp}(D, \text{method\_selection\_object}) \]

**Returns**: an object

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

![Diagram of DirectSumOp](image)
6.6.2 ProjectionInFactorOfDirectSum (for IsList, IsInt)

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

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

6.6.3 ProjectionInFactorOfDirectSumOp (for IsList, IsInt, IsCapCategoryObject)

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

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

6.6.4 ProjectionInFactorOfDirectSumWithGivenDirectSum (for IsList, IsInt, IsCapCategoryObject)

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

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

6.6.5 InjectionOfCofactorOfDirectSum (for IsList, IsInt)

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

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

6.6.6 InjectionOfCofactorOfDirectSumOp (for IsList, IsInt, IsCapCategoryObject)

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

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

6.6.7 InjectionOfCofactorOfDirectSumWithGivenDirectSum (for IsList, IsInt, IsCapCategoryObject)

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

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

6.6.8 UniversalMorphismIntoDirectSum

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 \( D = (S_1, \ldots, S_n) \) and a list of morphisms \( \tau = (\tau_i : T \to S_i)_{i=1\ldots n} \).

• The argument is a list of morphisms \( \tau = (\tau_i : T \to S_i)_{i=1\ldots n} \).

• The arguments are morphisms \( \tau_1 : T \to S_1, \ldots, \tau_n : T \to S_n \).

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

6.6.9 UniversalMorphismIntoDirectSumOp (for IsList, IsList, IsCapCategoryObject)

\( \text{UniversalMorphismIntoDirectSumOp}(D, \tau, \text{method_selection_object}) \)

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

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

6.6.10 UniversalMorphismIntoDirectSumWithGivenDirectSum (for IsList, IsList, IsCapCategoryObject)

\( \text{UniversalMorphismIntoDirectSumWithGivenDirectSum}(D, \tau, S) \)

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

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

6.6.11 UniversalMorphismFromDirectSum

\( \text{UniversalMorphismFromDirectSum}(\text{arg}) \)

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 \( D = (S_1, \ldots, S_n) \) and a list of morphisms \( \tau = (\tau_i : S_i \to T)_{i=1\ldots n} \).

• The argument is a list of morphisms \( \tau = (\tau_i : S_i \to T)_{i=1\ldots n} \).

• The arguments are morphisms \( S_1 \to T, \ldots, S_n \to T \).

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

6.6.12 UniversalMorphismFromDirectSumOp (for IsList, IsList, IsCapCategoryObject)

\( \text{UniversalMorphismFromDirectSumOp}(D, \tau, \text{method_selection_object}) \)

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

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

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

**Returns:** a morphism in Hom(S, T)

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

6.6.14 IsomorphismFromDirectSumToDirectProduct (for IsList)

▷ IsomorphismFromDirectSumToDirectProduct(D) (operation)

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

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

6.6.15 IsomorphismFromDirectSumToDirectProductOp (for IsList, IsCapCategoryObject)

▷ IsomorphismFromDirectSumToDirectProductOp(D, method_selection_object) (operation)

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

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

6.6.16 IsomorphismFromDirectProductToDirectSum (for IsList)

▷ IsomorphismFromDirectProductToDirectSum(D) (operation)

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

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

6.6.17 IsomorphismFromDirectProductToDirectSumOp (for IsList, IsCapCategoryObject)

▷ IsomorphismFromDirectProductToDirectSumOp(D, method_selection_object) (operation)

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

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

6.6.18 IsomorphismFromDirectSumToCoproduct (for IsList)

▷ IsomorphismFromDirectSumToCoproduct(D) (operation)

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

The argument is a list of objects \( D = (S_1, \ldots, S_n) \). The output is the canonical isomorphism \( \bigoplus_{i=1}^{n} S_i \to \bigsqcup_{i=1}^{n} S_i \).
6.6.19 IsomorphismFromDirectSumToCoproductOp (for IsList, IsCapCategoryObject)

\[ \text{IsomorphismFromDirectSumToCoproductOp}(D, \text{method_selection_object}) \]

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

6.6.20 IsomorphismFromCoproductToDirectSum (for IsList)

\[ \text{IsomorphismFromCoproductToDirectSum}(D) \]

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

6.6.21 IsomorphismFromCoproductToDirectSumOp (for IsList, IsCapCategoryObject)

\[ \text{IsomorphismFromCoproductToDirectSumOp}(D, \text{method_selection_object}) \]

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

6.6.22 MorphismBetweenDirectSums (for IsList)

\[ \text{MorphismBetweenDirectSums}(M) \]

\[ \text{MorphismBetweenDirectSums}(S, M, T) \]

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

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

6.6.23 AddMorphismBetweenDirectSums (for IsCapCategory, IsFunction)

\[ \text{AddMorphismBetweenDirectSums}(C, F) \]

Returns: nothing

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. \( F : (\bigoplus_{i=1}^{m} A_i, M, \bigoplus_{j=1}^{n} B_j) \to (\bigoplus_{i=1}^{m} A_i \to \bigoplus_{j=1}^{n} B_j) \).


\[ \text{MorphismBetweenDirectSumsOp}(N, m, n, \text{method_selection_morphism}) \]

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

6.6.25 ComponentOfMorphismIntoDirectSum (for IsCapCategoryMorphism, IsList, IsInt)

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

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

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

6.6.26 ComponentOfMorphismFromDirectSum (for IsCapCategoryMorphism, IsList, IsInt)

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

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

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

6.6.27 AddComponentOfMorphismIntoDirectSum (for IsCapCategory, IsFunction)

▷ AddComponentOfMorphismIntoDirectSum(C, F) (operation)

Returns: nothing

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

6.6.28 AddComponentOfMorphismFromDirectSum (for IsCapCategory, IsFunction)

▷ AddComponentOfMorphismFromDirectSum(C, F) (operation)

Returns: nothing

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

6.6.29 AddProjectionInFactorOfDirectSum (for IsCapCategory, IsFunction)

▷ AddProjectionInFactorOfDirectSum(C, F) (operation)

Returns: nothing

The arguments are a category $C$ and a function $F$. This 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 \( \text{ProjectionInFactorOfDirectSumWithGivenDirectSum} \). \( F : (D, k, S) \rightarrow \pi_k \).

### 6.6.31 AddInjectionOfCofactorOfDirectSum (for IsCapCategory, IsFunction)

- \( \triangleright \) 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 \( \text{InjectionOfCofactorOfDirectSum} \). \( F : (D, k) \rightarrow \iota_k \).

### 6.6.32 AddInjectionOfCofactorOfDirectSumWithGivenDirectSum (for IsCapCategory, IsFunction)

- \( \triangleright \) 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 \( \text{InjectionOfCofactorOfDirectSumWithGivenDirectSum} \). \( F : (D, k, S) \rightarrow \iota_k \).

### 6.6.33 AddUniversalMorphismIntoDirectSum (for IsCapCategory, IsFunction)

- \( \triangleright \) 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 \( \text{UniversalMorphismIntoDirectSum} \). \( F : (D, \tau) \rightarrow u_{\text{in}}(\tau) \).

### 6.6.34 AddUniversalMorphismIntoDirectSumWithGivenDirectSum (for IsCapCategory, IsFunction)

- \( \triangleright \) 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 \( \text{UniversalMorphismIntoDirectSumWithGivenDirectSum} \). \( F : (D, \tau, S) \rightarrow u_{\text{in}}(\tau) \).

### 6.6.35 AddUniversalMorphismFromDirectSum (for IsCapCategory, IsFunction)

- \( \triangleright \) 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 \( \text{UniversalMorphismFromDirectSum} \). \( F : (D, \tau) \rightarrow u_{\text{out}}(\tau) \).

### 6.6.36 AddUniversalMorphismFromDirectSumWithGivenDirectSum (for IsCapCategory, IsFunction)

- \( \triangleright \) 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 \( \text{UniversalMorphismFromDirectSumWithGivenDirectSum} \). \( F : (D, \tau, S) \mapsto u_{\text{out}}(\tau) \).

6.6.37 AddIsomorphismFromDirectSumToDirectProduct (for IsCapCategory, IsFunction)

\[ \text{AddIsomorphismFromDirectSumToDirectProduct}(C, F) \]

Returns: nothing

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

6.6.38 AddIsomorphismFromDirectProductToDirectSum (for IsCapCategory, IsFunction)

\[ \text{AddIsomorphismFromDirectProductToDirectSum}(C, F) \]

Returns: nothing

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

6.6.39 AddIsomorphismFromDirectSumToCoproduct (for IsCapCategory, IsFunction)

\[ \text{AddIsomorphismFromDirectSumToCoproduct}(C, F) \]

Returns: nothing

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

6.6.40 AddIsomorphismFromCoproductToDirectSum (for IsCapCategory, IsFunction)

\[ \text{AddIsomorphismFromCoproductToDirectSum}(C, F) \]

Returns: nothing

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

6.6.41 AddDirectSum (for IsCapCategory, IsFunction)

\[ \text{AddDirectSum}(C, F) \]

Returns: nothing

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

▷ DirectSumFunctorial(L)

Returns: a morphism in Hom(⨁_{i=1}^{n} S_i, ⨁_{i=1}^{n} S'_i)

The argument is a list of morphisms L = (µ_1 : S_1 → S'_1, ..., µ_n : S_n → S'_n). The output is a morphism ⨁_{i=1}^{n} S_i → ⨁_{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)

Returns: a morphism in Hom(d_1, d_2)

The arguments are an object d_1 = ⨁_{i=1}^{n} S_i, a list of morphisms L = (µ_1 : S_1 → S'_1, ..., µ_n : S_n → S'_n), and an object d_2 = ⨁_{i=1}^{n} S'_i. The output is a morphism d_1 → d_2 given by the functoriality of the direct sum.

6.6.44 AddDirectSumFunctorialWithGivenDirectSums (for IsCapCategory, IsFunction)

▷ AddDirectSumFunctorialWithGivenDirectSums(C, F)

Returns: nothing

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 : (⨁_{i=1}^{n} S_i, (µ_1, ..., µ_n), ⨁_{i=1}^{n} S'_i) → (⨁_{i=1}^{n} S_i → ⨁_{i=1}^{n} S'_i).

6.7 Coproduct

For an integer n ≥ 1 and a given list of objects D = (I_1, ..., I_n), a coproduct of D consists of three parts:

- an object I,
- a list of morphisms t = (t_i : I_i → I)_{i=1...n},
- a dependent function u mapping each list of morphisms τ = (τ_i : I_i → T) to a morphism u(τ) : I → T such that u(τ) ◦ t_i ∼ τ_i for all i = 1, ..., n.

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

Coproduct is a functorial operation. This means: For (µ_i : I_i → I_i')_{i=1...n}, we obtain a morphism ∐_{i=1}^{n} I_i → ∐_{i=1}^{n} I_i'.
6.7.1 Coproduct (for IsList)

\[\text{Coproduct}(D)\]

**Returns:** an object

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

6.7.2 Coproduct (for IsCapCategoryObject, IsCapCategoryObject)

\[\text{Coproduct}(I_1, I_2)\]

**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)

\[\text{Coproduct}(I_1, I_2, I_3)\]

**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)

\[\text{CoproductOp}(D, \text{method_selection_object})\]

**Returns:** an object

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

6.7.5 InjectionOfCofactorOfCoproduct (for IsList, IsInt)

\[\text{InjectionOfCofactorOfCoproduct}(D, k)\]

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

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

6.7.6 InjectionOfCofactorOfCoproductOp (for IsList, IsInt, IsCapCategoryObject)

\[\text{InjectionOfCofactorOfCoproductOp}(D, k, \text{method_selection_object})\]

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

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

6.7.7 InjectionOfCofactorOfCoproductWithGivenCoproduct (for IsList, IsInt, IsCapCategoryObject)

\[\text{InjectionOfCofactorOfCoproductWithGivenCoproduct}(D, k, I)\]

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

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

\( \text{UniversalMorphismFromCoproduct}(\text{arg}) \)

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

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

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

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

6.7.9 UniversalMorphismFromCoproductOp (for IsList, IsList, IsCapCategoryObject)

\( \text{UniversalMorphismFromCoproductOp}(D, \tau, \text{method\_selection\_object}) \)

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

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

6.7.10 UniversalMorphismFromCoproductWithGivenCoproduct (for IsList, IsList, IsCapCategoryObject)

\( \text{UniversalMorphismFromCoproductWithGivenCoproduct}(D, \tau, I) \)

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

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

6.7.11 AddCoproduct (for IsCapCategory, IsFunction)

\( \text{AddCoproduct}(C, F) \)

\( \text{Returns: nothing} \)

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

6.7.12 AddInjectionOfCofactorOfCoproduct (for IsCapCategory, IsFunction)

\( \text{AddInjectionOfCofactorOfCoproduct}(C, F) \)

\( \text{Returns: nothing} \)

The arguments are a category \( C \) and a function \( F \). This operation adds the given function \( F \) to the category for the basic operation \( \text{InjectionOfCofactorOfCoproduct} \). \( F : ((I_1,\ldots,I_n),i) \mapsto i \).
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, ..., I_n), i) ↦ ι_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, ..., I_n), τ) ↦ u(τ).

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, ..., I_n), τ, I) ↦ u(τ).

6.7.16 CoproductFunctorial (for IsList)

- CoproductFunctorial(L) (operation)
  - Returns: a morphism in Hom(⨆_{i=1}^n I_i, ⨆_{i=1}^n I'_i)
  - The argument is a list L = (μ_1 : I_1 → I'_1, ..., μ_n : I_n → I'_n). The output is a morphism ⨆_{i=1}^n I_i → ⨆_{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 Hom(s, r)
  - The arguments are an object s = ⨆_{i=1}^n I_i, a list L = (μ_1 : I_1 → I'_1, ..., μ_n : I_n → I'_n), and an object r = ⨆_{i=1}^n I'_i. The output is a morphism ⨆_{i=1}^n I_i → ⨆_{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 : (⨆_{i=1}^n I_i, (μ_1, ..., μ_n), ⨆_{i=1}^n I'_i) ↦ (⨆_{i=1}^n I_i → ⨆_{i=1}^n I'_i).
6.8 Direct Product

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

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

The triple $(P, \pi, u)$ 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 $(\mu_i : P_i \to P_i')_{i=1\ldots n}$, we obtain a morphism $\prod_{i=1}^n P_i \to \prod_{i=1}^n P_i'$.

6.8.1 DirectProductOp (for IsList, IsCapCategoryObject)

▷ DirectProductOp($D$) (operation)

Returns: an object

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

6.8.2 ProjectionInFactorOfDirectProduct (for IsList, IsInt)

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

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

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

6.8.3 ProjectionInFactorOfDirectProductOp (for IsList, IsInt, IsCapCategoryObject)

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

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

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

\( \text{ProjectionInFactorOfDirectProductWithGivenDirectProduct}(D, k, P) \) (operation)

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

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

6.8.5 UniversalMorphismIntoDirectProduct

\( \text{UniversalMorphismIntoDirectProduct}(\text{arg}) \) (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 \( D = (P_1, \ldots, P_n) \) and a list of morphisms \( \tau = (\tau_i : T \to P_i)_{i=1,\ldots,n} \).
- The argument is a list of morphisms \( \tau = (\tau_i : T \to P_i)_{i=1,\ldots,n} \).
- The arguments are morphisms \( \tau_1 : T \to P_1, \ldots, \tau_n : T \to P_n. \)

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


\( \text{UniversalMorphismIntoDirectProductOp}(D, \tau, \text{method_selection_object}) \) (operation)

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

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

6.8.7 UniversalMorphismIntoDirectProductWithGivenDirectProduct (for IsList, IsList, IsCapCategoryObject)

\( \text{UniversalMorphismIntoDirectProductWithGivenDirectProduct}(D, \tau, P) \) (operation)

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

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

6.8.8 AddDirectProduct (for IsCapCategory, IsFunction)

\( \text{AddDirectProduct}(C, F) \) (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 \( \text{DirectProduct} \). \( F : ((P_1, \ldots, P_n)) \to P \)
6.8.9  AddProjectionInFactorOfDirectProduct (for IsCapCategory, IsFunction)

▷ AddProjectionInFactorOfDirectProduct(C, F)  
    Returns: nothing  
    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 : ((P₁,...,Pₙ), k) ↦ πₖ

6.8.10 AddProjectionInFactorOfDirectProductWithGivenDirectProduct (for IsCapCategory, IsFunction)

▷ AddProjectionInFactorOfDirectProductWithGivenDirectProduct(C, F)  
    Returns: nothing  
    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 : ((P₁,...,Pₙ), k, P) ↦ πₖ

6.8.11 AddUniversalMorphismIntoDirectProduct (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismIntoDirectProduct(C, F)  
    Returns: nothing  
    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 : ((P₁,...,Pₙ), τ) ↦ u(τ)

6.8.12 AddUniversalMorphismIntoDirectProductWithGivenDirectProduct (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismIntoDirectProductWithGivenDirectProduct(C, F)  
    Returns: nothing  
    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 : ((P₁,...,Pₙ), τ, P) ↦ u(τ)

6.8.13 DirectProductFunctorial (for IsList)

▷ DirectProductFunctorial(L)  
    Returns: a morphism in Hom(∏ᵢ=₁ⁿ Pᵢ, ∏ᵢ=₁ⁿ P'ᵢ)  
    The argument is a list of morphisms L = (µᵢ : Pᵢ → P'ᵢ)ᵢ=₁...n. The output is a morphism ∏ᵢ=₁ⁿ Pᵢ → ∏ᵢ=₁ⁿ P'ᵢ given by the functoriality of the direct product.

6.8.14 DirectProductFunctorialWithGivenDirectProducts (for IsCapCategoryObject, IsList, IsCapCategoryObject)

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

6.8.15 \textbf{AddDirectProductFunctorialWithGivenDirectProducts} \textit{(for IsCapCategory, IsFunction)}

\[ \text{\texttt{AddDirectProductFunctorialWithGivenDirectProducts}}(C, F) \]

\textbf{Returns:} nothing

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

6.9 \textbf{Equalizer}

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

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

The triple \((E, t, u)\) 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 \( D' = (\beta'_i : A' \to B')_{i=1,\ldots,n} \) and a natural morphism between equalizer diagrams (i.e., a collection of morphisms \( \mu : A \to A' \) and \( \beta : B \to B' \) such that \( \beta'_i \circ \mu \sim_{A,B'} \beta_i \) for \( i = 1, \ldots, n \)) we obtain a morphism \( \text{Equalizer}(D) \to \text{Equalizer}(D') \).

6.9.1 \textbf{Equalizer}

\[ \text{\texttt{Equalizer}}(\texttt{arg}) \]

\textbf{Returns:} an object

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

- The argument is a list of morphisms \( D = (\beta_i : A \to B)_{i=1,\ldots,n} \).
• The arguments are morphisms \( \beta_1 : A \to B, \ldots, \beta_n : A \to B \).

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

6.9.2 EqualizerOp (for IsList, IsCapCategoryMorphism)

\[ \text{EqualizerOp}(D, \text{method_selection_morphism}) \]

- **Returns:** an object
  - The arguments are a list of morphisms \( D = (\beta_i : A \to B)_{i=1\ldots n} \) and a morphism for method selection. The output is the equalizer \( \text{Equalizer}(D) \).

6.9.3 EmbeddingOfEqualizer (for IsList)

\[ \text{EmbeddingOfEqualizer}(D) \]

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

6.9.4 EmbeddingOfEqualizerOp (for IsList, IsCapCategoryMorphism)

\[ \text{EmbeddingOfEqualizerOp}(D, \text{method_selection_morphism}) \]

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

6.9.5 EmbeddingOfEqualizerWithGivenEqualizer (for IsList, IsCapCategoryObject)

\[ \text{EmbeddingOfEqualizerWithGivenEqualizer}(D, E) \]

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

6.9.6 MorphismFromEqualizerToSink (for IsList)

\[ \text{MorphismFromEqualizerToSink}(D) \]

- **Returns:** a morphism in \( \text{Hom}(\text{Equalizer}(D), B) \)
  - The arguments are a list of morphisms \( D = (\beta_i : A \to B)_{i=1\ldots n} \). The output is the composition \( \mu : \text{Equalizer}(D) \to B \) of the embedding \( \iota : \text{Equalizer}(D) \to A \) and \( \beta_1 \).

6.9.7 MorphismFromEqualizerToSinkOp (for IsList, IsCapCategoryMorphism)

\[ \text{MorphismFromEqualizerToSinkOp}(D, \text{method_selection_morphism}) \]

- **Returns:** a morphism in \( \text{Hom}(\text{Equalizer}(D), B) \)
  - The arguments are a list of morphisms \( D = (\beta_i : A \to B)_{i=1\ldots n} \) and a morphism for method selection. The output is the composition \( \mu : \text{Equalizer}(D) \to B \) of the embedding \( \iota : \text{Equalizer}(D) \to A \) 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 $D = (\beta_i : A \to B)_{i=1,...,n}$ and an object $E = \text{Equalizer}(D)$. The output is the composition $\mu : E \to B$ of the embedding $\iota : E \to A$ 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 $D = (\beta_i : A \to B)_{i=1,...,n}$ and a morphism $\tau : T \to A$ such that $\beta_i \circ \tau \sim_{T,B} \beta_j \circ \tau$ for all pairs $i, j$. The output is the morphism $u(\tau) : T \to \text{Equalizer}(D)$ 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 $D = (\beta_i : A \to B)_{i=1,...,n}$, a morphism $\tau : T \to A$ such that $\beta_i \circ \tau \sim_{T,B} \beta_j \circ \tau$ for all pairs $i, j$, and an object $E = \text{Equalizer}(D)$. The output is the morphism $u(\tau) : T \to E$ 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 $\text{Equalizer}$. $F : ((\beta_i : A \to B)_{i=1,...,n}) \mapsto E$

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 $\text{EmbeddingOfEqualizer}$. $F : ((\beta_i : A \to B)_{i=1,...,n}, k) \mapsto \iota$

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 $\text{EmbeddingOfEqualizerWithGivenEqualizer}$. $F : ((\beta_i : A \to B)_{i=1,...,n}, E) \mapsto \iota$
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. \( F : \left( \beta_i : P \rightarrow B \right)_{i=1,...,n} \mapsto \mu \)

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. \( F : \left( \beta_i : P \rightarrow B \right)_{i=1,...,n}, E \mapsto \mu \)

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. \( F : \left( \beta_i : A \rightarrow B \right)_{i=1,...,n}, \tau \mapsto u(\tau) \)

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. \( F : \left( \beta_i : A \rightarrow B \right)_{i=1,...,n}, \tau, E \mapsto u(\tau) \)

6.9.18 EqualizerFunctorial (for IsList, IsCapCategoryMorphism, IsList)

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

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

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


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

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

### 6.9.20 AddEqualizerFunctorialWithGivenEqualizers (for IsCapCategory, IsFunction)

\[ \text{AddEqualizerFunctorialWithGivenEqualizers}(C, F) \]  

**Returns:** nothing

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

### 6.10 Coequalizer

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

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

The triple \((C, \pi, u)\) 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 \( D' = (\beta'_i : B' \to A')_{i=1\ldots n} \) and a natural morphism between coequalizer diagrams (i.e., a collection of morphisms \( \mu : A \to A' \) and \( \beta : B \to B' \) such that \( \beta'_i \circ \mu \sim_{B,A} \beta \circ \beta_i \) for \( i = 1, \ldots, n \)) we obtain a morphism \( \text{Coequalizer}(D) \to \text{Coequalizer}(D') \).

### 6.10.1 Coequalizer

\[ \text{Coequalizer}(\text{arg}) \]  

**Returns:** an object

This is a convenience method. There are two different ways to use this method:
• The argument is a list of morphisms $D = (\beta_i : B \to A)_{i=1..n}$.
• The arguments are morphisms $\beta_1 : B \to A, \ldots, \beta_n : B \to A$.

The output is the coequalizer $\text{Coequalizer}(D)$.

### 6.10.2 CoequalizerOp (for IsList, IsCapCategoryMorphism)

\[ \text{CoequalizerOp}(D, \text{method_selection_morphism}) \]

**Returns:** an object

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

### 6.10.3 ProjectionOntoCoequalizer (for IsList)

\[ \text{ProjectionOntoCoequalizer}(D) \]

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

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

### 6.10.4 ProjectionOntoCoequalizerOp (for IsList, IsCapCategoryMorphism)

\[ \text{ProjectionOntoCoequalizerOp}(D, \text{method_selection_morphism}) \]

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

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

### 6.10.5 ProjectionOntoCoequalizerWithGivenCoequalizer (for IsList, IsCapCategoryObject)

\[ \text{ProjectionOntoCoequalizerWithGivenCoequalizer}(D, C) \]

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

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

### 6.10.6 MorphismFromSourceToCoequalizer (for IsList)

\[ \text{MorphismFromSourceToCoequalizer}(D) \]

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

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

### 6.10.7 MorphismFromSourceToCoequalizerOp (for IsList, IsCapCategoryMorphism)

\[ \text{MorphismFromSourceToCoequalizerOp}(D, \text{method_selection_morphism}) \]

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

6.10.8 MorphismFromSourceToCoequalizerWithGivenCoequalizer (for IsList, IsCapCategoryObject)

\[
\text{MorphismFromSourceToCoequalizerWithGivenCoequalizer}(D, C)
\]

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

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

6.10.9 UniversalMorphismFromCoequalizer (for IsList, IsCapCategoryMorphism)

\[
\text{UniversalMorphismFromCoequalizer}(D, \tau)
\]

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

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

6.10.10 UniversalMorphismFromCoequalizerWithGivenCoequalizer (for IsList, IsCapCategoryMorphism, IsCapCategoryObject)

\[
\text{UniversalMorphismFromCoequalizerWithGivenCoequalizer}(D, \tau, C)
\]

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

The arguments are a list of morphisms \( D = (\beta_i : B \to A)_{i=1,...,n} \), a morphism \( \tau : A \to T \) 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 \( u(\tau) : C \to T \) given by the universal property of the coequalizer.

6.10.11 AddCoequalizer (for IsCapCategory, IsFunction)

\[
\text{AddCoequalizer}(C, F)
\]

Returns: nothing.

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

6.10.12 AddProjectionOntoCoequalizer (for IsCapCategory, IsFunction)

\[
\text{AddProjectionOntoCoequalizer}(C, F)
\]

Returns: nothing.

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

6.10.13 AddProjectionOntoCoequalizerWithGivenCoequalizer (for IsCapCategory, IsFunction)

\[
\text{AddProjectionOntoCoequalizerWithGivenCoequalizer}(C, F)
\]

Returns: nothing.
The arguments are a category $C$ and a function $F$. This operation adds the given function $F$ to the category for the basic operation $\text{ProjectionOntoCoequalizerWithGivenCoequalizer}$. $F : ((\beta_i : B \to A)_{i=1...n}, C) \mapsto \pi$

6.10.14 AddMorphismFromSourceToCoequalizer (for IsCapCategory, IsFunction)

\[ \text{operation} \]

Returns: nothing

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

6.10.15 AddMorphismFromSourceToCoequalizerWithGivenCoequalizer (for IsCapCategory, IsFunction)

\[ \text{operation} \]

Returns: nothing

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

6.10.16 AddUniversalMorphismFromCoequalizer (for IsCapCategory, IsFunction)

\[ \text{operation} \]

Returns: nothing

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

6.10.17 AddUniversalMorphismFromCoequalizerWithGivenCoequalizer (for IsCapCategory, IsFunction)

\[ \text{operation} \]

Returns: nothing

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

6.10.18 CoequalizerFunctorial (for IsList, IsCapCategoryMorphism, IsList)

\[ \text{operation} \]

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

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

- CoequalizerFunctorialWithGivenCoequalizers\( (s, Ls, mu, Lr, r) \) (operation)

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

  The arguments are an object \( s = \text{Coequalizer}(\{\beta\}_i^{n=1}) \), a list of morphisms \( L_s = (\beta_i : B \to A)_i^{1...n} \), a morphism \( \mu : A \to A' \), and a list of morphisms \( L_r = (\beta'_i : B' \to A')_i^{1...n} \) such that there exists a morphism \( \beta : B \to B' \) such that \( \beta'_i \circ \beta \sim \beta_i \circ \beta_i \) for \( i = 1, \ldots, n \), and an object \( r = \text{Coequalizer}(\{\beta'_i\}_i^{n=1}) \).

The output is the morphism \( s \to r \) given by the functorality of the coequalizer.

6.10.20 AddCoequalizerFunctorialWithGivenCoequalizers (for IsCapCategory, IsFunction)

- AddCoequalizerFunctorialWithGivenCoequalizers\( (C, F) \) (operation)

  Returns: nothing

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

6.11 Fiber Product

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

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

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 FiberProduct\( (D) \). We say that the morphism \( u(\tau) \) is induced by the universal property of the fiber product.

FiberProduct is a functorial operation. This means: For a second diagram \( D' = (\beta'_i : P'_i \to B')_i^{1...n} \) and a natural morphism between pullback diagrams (i.e., a collection of morphisms \( (\mu_i : P_i \to P'_i)_i^{1...n} \) and \( \beta : B \to B' \) such that \( \beta'_i \circ \mu_i \sim_{P, B'} \beta \circ \beta_i \) for \( i = 1, \ldots, n \)) we obtain a morphism \( \text{FiberProduct}(D) \to \text{FiberProduct}(D') \).
6.11.1 IsomorphismFromFiberProductToKernelOfDiagonalDifference (for IsList)

▷ IsomorphismFromFiberProductToKernelOfDiagonalDifference(D)

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

Returns: a morphism in Hom(FiberProduct(D), Δ)
The arguments are a list of morphisms $D = (\beta_i : P_i \rightarrow B)_{i=1..n}$ and a morphism for method selection. The output is 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)

Returns: nothing
The arguments are a category $C$ and a function $F$. This operation adds the given function $F$ to the category for the basic operation $\text{IsomorphismFromFiberProductToKernelOfDiagonalDifference}$. $F : ((\beta_i : P_i \rightarrow B)_{i=1..n}) \mapsto \text{FiberProduct}(D) \rightarrow \Delta$

6.11.4 IsomorphismFromKernelOfDiagonalDifferenceToFiberProduct (for IsList)

▷ IsomorphismFromKernelOfDiagonalDifferenceToFiberProduct(D)

Returns: a morphism in Hom(Δ, FiberProduct(D))
The argument is a list of morphisms $D = (\beta_i : P_i \rightarrow B)_{i=1..n}$. 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 \( D = (\beta_i : P_i \to B)_{i=1,...,n} \) and a morphism for method selection. The output is a morphism \( \Delta \to \text{FiberProduct}(D) \), 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 \( \text{IsomorphismFromKernelOfDiagonalDifferenceToFiberProduct} \). \( F : ((\beta_i : P_i \to B)_{i=1,...,n}) \mapsto \Delta \to \text{FiberProduct}(D) \)

6.11.7 IsomorphismFromFiberProductToEqualizerOfDirectProductDiagram (for IsList)

- **IsomorphismFromFiberProductToEqualizerOfDirectProductDiagram (D)** (operation)
  - **Returns:** a morphism in \( \text{Hom}(\text{FiberProduct}(D), \Delta) \)
  - The argument is a list of morphisms \( D = (\beta_i : P_i \to B)_{i=1,...,n} \). The output is a morphism \( \text{FiberProduct}(D) \to \Delta \), where \( \Delta \) denotes the equalizer of the product diagram of the morphisms \( \beta_i \).

6.11.8 IsomorphismFromFiberProductToEqualizerOfDirectProductDiagramOp (for IsList, IsCapCategoryMorphism)

- **IsomorphismFromFiberProductToEqualizerOfDirectProductDiagramOp (D, method_selection_morphism)** (operation)
  - **Returns:** a morphism in \( \text{Hom}(\text{FiberProduct}(D), \Delta) \)
  - The arguments are a list of morphisms \( D = (\beta_i : P_i \to B)_{i=1,...,n} \) and a morphism for method selection. The output is a morphism \( \text{FiberProduct}(D) \to \Delta \), where \( \Delta \) denotes the equalizer of the product diagram of the morphisms \( \beta_i \).

6.11.9 AddIsomorphismFromFiberProductToEqualizerOfDirectProductDiagram (for IsList, 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 \( \text{IsomorphismFromFiberProductToEqualizerOfDirectProductDiagram} \). \( F : ((\beta_i : P_i \to B)_{i=1,...,n}) \mapsto \text{FiberProduct}(D) \to \Delta \)
6.11.10 IsomorphismFromEqualizerOfDirectProductDiagramToFiberProduct \(\text{(for IsList)}\)

\(\triangleright\) IsomorphismFromEqualizerOfDirectProductDiagramToFiberProduct(D) \(\text{(operation)}\)

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

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

6.11.11 IsomorphismFromEqualizerOfDirectProductDiagramToFiberProductOp \(\text{(for IsList, IsCapCategoryMorphism)}\)

\(\triangleright\) IsomorphismFromEqualizerOfDirectProductDiagramToFiberProductOp(D) \(\text{(operation)}\)

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

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

6.11.12 AddIsomorphismFromEqualizerOfDirectProductDiagramToFiberProduct \(\text{(for IsCapCategory, IsFunction)}\)

\(\triangleright\) AddIsomorphismFromEqualizerOfDirectProductDiagramToFiberProduct(C, F) \(\text{(operation)}\)

**Returns:** nothing

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

6.11.13 DirectSumDiagonalDifference \(\text{(for IsList)}\)

\(\triangleright\) DirectSumDiagonalDifference(D) \(\text{(operation)}\)

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

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

6.11.14 DirectSumDiagonalDifferenceOp \(\text{(for IsList, IsCapCategoryMorphism)}\)

\(\triangleright\) DirectSumDiagonalDifferenceOp(D, method_selection_morphism) \(\text{(operation)}\)

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

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

6.11.15 AddDirectSumDiagonalDifference \(\text{(for IsCapCategory, IsFunction)}\)

\(\triangleright\) AddDirectSumDiagonalDifference(C, F) \(\text{(operation)}\)

**Returns:** nothing

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

\[ \text{FiberProductEmbeddingInDirectSum}(D) \]

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

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

6.11.17 FiberProductEmbeddingInDirectSumOp (for IsList, IsCapCategoryMorphism)

\[ \text{FiberProductEmbeddingInDirectSumOp}(D, \text{method_selection_morphism}) \]

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

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

6.11.18 AddFiberProductEmbeddingInDirectSum (for IsCapCategory, IsFunction)

\[ \text{AddFiberProductEmbeddingInDirectSum}(C, F) \]

**Returns:** nothing

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

6.11.19 FiberProduct

\[ \text{FiberProduct}(\text{arg}) \]

**Returns:** an object

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

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

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

6.11.20 FiberProductOp (for IsList, IsCapCategoryMorphism)

\[ \text{FiberProductOp}(D, \text{method_selection_morphism}) \]

**Returns:** an object

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

6.11.21 ProjectionInFactorOfFiberProduct (for IsList, IsInt)

\[ \text{ProjectionInFactorOfFiberProduct}(D, k) \]

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

The arguments are a list of morphisms \( D = (\beta_i : P_i \to B)_{i=1...n} \) and an integer \( k \). The output is the \( k \)-th projection \( \pi_k : \text{FiberProduct}(D) \to P_k \).
6.11.22 ProjectionInFactorOfFiberProductOp (for IsList, IsInt, IsCapCategoryMorphism)

\[ \text{ProjectionInFactorOfFiberProductOp}(D, k, \text{method_selection_morphism}) \] (operation)

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

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

6.11.23 ProjectionInFactorOfFiberProductWithGivenFiberProduct (for IsList, IsInt, IsCapCategoryObject)

\[ \text{ProjectionInFactorOfFiberProductWithGivenFiberProduct}(D, k, P) \] (operation)

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

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

6.11.24 MorphismFromFiberProductToSink (for IsList)

\[ \text{MorphismFromFiberProductToSink}(D) \] (operation)

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

The arguments are a list of morphisms \( D = (\beta_i : P_i \to B)_{i=1,...,n} \). The output is the composition \( \mu : \text{FiberProduct}(D) \to B \) of the 1-st projection \( \pi_1 : \text{FiberProduct}(D) \to P_1 \) and \( \beta_1 \).

6.11.25 MorphismFromFiberProductToSinkOp (for IsList, IsCapCategoryMorphism)

\[ \text{MorphismFromFiberProductToSinkOp}(D, \text{method_selection_morphism}) \] (operation)

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

The arguments are a list of morphisms \( D = (\beta_i : P_i \to B)_{i=1,...,n} \) and a morphism for method selection. The output is the composition \( \mu : \text{FiberProduct}(D) \to B \) of the 1-st projection \( \pi_1 : \text{FiberProduct}(D) \to P_1 \) and \( \beta_1 \).

6.11.26 MorphismFromFiberProductToSinkWithGivenFiberProduct (for IsList, IsCapCategoryObject)

\[ \text{MorphismFromFiberProductToSinkWithGivenFiberProduct}(D, P) \] (operation)

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

The arguments are a list of morphisms \( D = (\beta_i : P_i \to B)_{i=1,...,n} \) and an object \( P = \text{FiberProduct}(D) \). The output is the composition \( \mu : P \to B \) of the 1-st projection \( \pi_1 : P \to P_1 \) and \( \beta_1 \).

6.11.27 UniversalMorphismIntoFiberProduct

\[ \text{UniversalMorphismIntoFiberProduct}(\text{arg}) \] (function)

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

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

6.11.28 UniversalMorphismIntoFiberProductOp (for IsList, IsList, IsCapCategoryMorphism)

\[ \text{UniversalMorphismIntoFiberProductOp}(D, \tau, \text{method_selection_morphism}) \]

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

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

6.11.29 UniversalMorphismIntoFiberProductWithGivenFiberProduct (for IsList, IsList, IsCapCategoryObject)

\[ \text{UniversalMorphismIntoFiberProductWithGivenFiberProduct}(D, \tau, P) \]

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

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

6.11.30 AddFiberProduct (for IsCapCategory, IsFunction)

\[ \text{AddFiberProduct}(C, F) \]

\text{Returns:} nothing

The arguments are a category \( C \) and a function \( F \). This operation adds the given function \( F \) to the category for the basic operation \( \text{FiberProduct}. \ F : ((\beta_i : P_i \to B)_{i=1,...,n}) \to P \)

6.11.31 AddProjectionInFactorOfFiberProduct (for IsCapCategory, IsFunction)

\[ \text{AddProjectionInFactorOfFiberProduct}(C, F) \]

\text{Returns:} nothing

The arguments are a category \( C \) and a function \( F \). This operation adds the given function \( F \) to the category for the basic operation \( \text{ProjectionInFactorOfFiberProduct}. \ F : ((\beta_i : P_i \to B)_{i=1,...,n}) \to \pi_k \)

6.11.32 AddProjectionInFactorOfFiberProductWithGivenFiberProduct (for IsCapCategory, IsFunction)

\[ \text{AddProjectionInFactorOfFiberProductWithGivenFiberProduct}(C, F) \]

\text{Returns:} nothing

The arguments are a category \( C \) and a function \( F \). This operation adds the given function \( F \) to the category for the basic operation \( \text{ProjectionInFactorOfFiberProductWithGivenFiberProduct}. \ F : ((\beta_i : P_i \to B)_{i=1,...,n}) \to \pi_k \)
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 : (((β_i : P_i → B)_{i=1...n}) ↦ μ

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 : (((β_i : P_i → B)_{i=1...n}, P) ↦ μ

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 : (((β_i : P_i → B)_{i=1...n}, τ) ↦ u(τ)

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 : (((β_i : P_i → B)_{i=1...n}, τ, P) ↦ u(τ)

6.11.37 FiberProductFunctorial (for IsList, IsList, IsList)

▶ FiberProductFunctorial(Ls, Lm, Lr) (operation)

**Returns:** a morphism in Hom(FiberProduct(((β_i)_{i=1...n}), FiberProduct((β'_i)_{i=1...n})))

The arguments are three lists of morphisms L_s = (β_i : P_i → B)_{i=1...n}, L_m = (μ_i : P_i → P'_i)_{i=1...n}, L_r = (β'_i : P'_i → B')_{i=1...n} having the same length n such that there exists a morphism β : B → B' such that β'_i ° μ_i ∼ β_i for i = 1, ..., n. The output is the morphism FiberProduct(((β_i)_{i=1...n}) → FiberProduct(((β'_i)_{i=1...n}) given by the functoriality of the fiber product.


▶ FiberProductFunctorialWithGivenFiberProducts(s, Ls, Lm, Lr, r) (operation)

**Returns:** a morphism in Hom(s,r)
The arguments are an object \( s = \text{FiberProduct}((\beta_i)_{i=1...n}) \), three lists of morphisms \( L_k = (\beta_i : P_i \to B)_{i=1...n} \), \( L_m = (\mu_i : P_i \to P'_i)_{i=1...n} \), \( L_r = (\beta'_i : P'_i \to B')_{i=1...n} \) having the same length \( n \) such that there exists a morphism \( \beta : B \to B' \) such that \( \beta'_i \circ \mu_i \sim_{B,B'} \beta \circ \beta_i \) for \( i = 1, \ldots, n \), and an object \( r = \text{FiberProduct}((\beta'_i)_{i=1...n}) \). The output is the morphism \( s \to r \) given by the functoriality of the fiber product.

6.11.39 AddFiberProductFunctorialWithGivenFiberProducts (for IsCapCategory, IsFunction)

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

**Returns:** nothing

The arguments are a category \( C \) and a function \( F \). This operation adds the given function \( F \) to the category for the basic operation AddFiberProductFunctorialWithGivenFiberProducts. \( \text{FiberProductFunctorialWithGivenFiberProducts}((\beta_i)_{i=1...n},(\beta'_i : P'_i \to B')_{i=1...n},\text{FiberProduct}((\beta'_i)_{i=1...n})) \to \text{FiberProduct}((\beta_i)_{i=1...n}) \to \text{FiberProduct}((\beta'_i)_{i=1...n})) \)

6.12 Pushout

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

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

The triple \((I,t,u)\) 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 Pushout\((D)\). We say that the morphism \( u(\tau) \) is induced by the universal property of the pushout.

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

- IsomorphismFromPushoutToCokernelOfDiagonalDifference(D)
  
  **Returns:** a morphism in Hom(Pushout(D), \( \Delta \))

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

  **Returns:** a morphism in Hom(Pushout(D), \( \Delta \))

  The argument is a list of morphisms \( D = (\beta_i : B \rightarrow I_i)_{i=1...n} \) and a morphism for method selection. The output is a morphism 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)

  **Returns:** nothing

  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. \( F : ((\beta_i : B \rightarrow I_i)_{i=1...n}) \rightarrow \) (Pushout(D) \( \rightarrow \Delta \))

6.12.4 IsomorphismFromCokernelOfDiagonalDifferenceToPushout (for IsList)

- IsomorphismFromCokernelOfDiagonalDifferenceToPushout(D)

  **Returns:** a morphism in Hom(\( \Delta \), Pushout(D))

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

  **Returns:** a morphism in Hom(\( \Delta \), Pushout(D))

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

6.12.6 AddIsomorphismFromCokernelOfDiagonalDifferenceToPushout (for IsCapCategory, IsFunction)

- AddIsomorphismFromCokernelOfDiagonalDifferenceToPushout(C, F)

  **Returns:** nothing
The arguments are a category $C$ and a function $F$. This operation adds the given function $F$ to the
category for the basic operation $\text{IsomorphismFromCokernelOfDiagonalDifferenceToPushout}$. $F : ((β_i : B \to I_i)_{i=1...n}) \mapsto (Δ \to \text{Pushout}(D))$

6.12.7 $\text{IsomorphismFromPushoutToCoequalizerOfCoproductDiagram (for IsList)}$

$\triangleright$ $\text{IsomorphismFromPushoutToCoequalizerOfCoproductDiagram}(D)$ (operation)

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

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

6.12.8 $\text{IsomorphismFromPushoutToCoequalizerOfCoproductDiagramOp (for IsList, IsCapCategoryMorphism)}$

$\triangleright$ $\text{IsomorphismFromPushoutToCoequalizerOfCoproductDiagramOp}(D, \text{method_selection_morphism})$ (operation)

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

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

6.12.9 $\text{AddIsomorphismFromPushoutToCoequalizerOfCoproductDiagram (for IsCapCategory, IsFunction)}$

$\triangleright$ $\text{AddIsomorphismFromPushoutToCoequalizerOfCoproductDiagram}(C, F)$ (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 $\text{IsomorphismFromPushoutToCoequalizerOfCoproductDiagram}$. $F : ((β_i : B \to I_i)_{i=1...n}) \mapsto (\text{Pushout}(D) \to Δ)$

6.12.10 $\text{IsomorphismFromCoequalizerOfCoproductDiagramToPushout (for IsList)}$

$\triangleright$ $\text{IsomorphismFromCoequalizerOfCoproductDiagramToPushout}(D)$ (operation)

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

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

6.12.11 $\text{IsomorphismFromCoequalizerOfCoproductDiagramToPushoutOp (for IsList, IsCapCategoryMorphism)}$

$\triangleright$ $\text{IsomorphismFromCoequalizerOfCoproductDiagramToPushoutOp}(D, \text{method_selection_morphism})$ (operation)

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

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

\[ \text{AddIsomorphismFromCoequalizerOfCoproductDiagramToPushout}(C, F) \] (operation)

\textbf{Returns:} nothing

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{IsomorphismFromCoequalizerOfCoproductDiagramToPushout}. \( F : ((\beta_i : B \to I_i)_{i=1...n}) \mapsto (\Delta \to \text{Pushout}(D)) \)

6.12.13 DirectSumCodiagonalDifference (for IsList)

\[ \text{DirectSumCodiagonalDifference}(D) \] (operation)

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

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


\[ \text{DirectSumCodiagonalDifferenceOp}(D, \text{method_selection_morphism}) \] (operation)

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

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

6.12.15 AddDirectSumCodiagonalDifference (for IsCapCategory, IsFunction)

\[ \text{AddDirectSumCodiagonalDifference}(C, F) \] (operation)

\textbf{Returns:} nothing

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{DirectSumCodiagonalDifference}. \( F : (D) \mapsto \text{DirectSumCodiagonalDifference}(D) \)

6.12.16 DirectSumProjectionInPushout (for IsList)

\[ \text{DirectSumProjectionInPushout}(D) \] (operation)

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

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


\[ \text{DirectSumProjectionInPushoutOp}(D, \text{method_selection_morphism}) \] (operation)

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

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

6.12.19 Pushout (for IsList)

\[ \text{Pushout}(D) \]

(operation)

Returns: an object

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

6.12.20 Pushout (for IsCapCategoryMorphism, IsCapCategoryMorphism)

\[ \text{Pushout}(D) \]

(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)

\[ \text{PushoutOp}(D) \]

(operation)

Returns: an object

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

6.12.22 InjectionOfCofactorOfPushout (for IsList, IsInt)

\[ \text{InjectionOfCofactorOfPushout}(D, k) \]

(operation)

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

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


\[ \text{InjectionOfCofactorOfPushoutOp}(D, k, \text{method_selection_morphism}) \]

(operation)

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

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


\[ \text{InjectionOfCofactorOfPushoutWithGivenPushout}(D, k, I) \]

(operation)

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

### 6.12.25 MorphismFromSourceToPushout (for IsList)

[Definition](operation) \[ \text{MorphismFromSourceToPushout}(D) \]

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

The arguments are a list of morphisms \( D = (\beta_i : B \to I_i)_{i=1, \ldots, n} \). The output is the composition \( \mu : B \to \text{Pushout}(D) \) of \( \beta_1 \) and the 1-st injection \( \iota_1 : I_1 \to \text{Pushout}(D) \).

#### 6.12.26 MorphismFromSourceToPushoutOp (for IsList, IsCapCategoryMorphism)

[Definition](operation) \[ \text{MorphismFromSourceToPushoutOp}(D, \text{method_selection_morphism}) \]

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

The arguments are a list of morphisms \( D = (\beta_i : B \to I_i)_{i=1, \ldots, n} \) and a morphism for method selection. The output is the composition \( \mu : B \to \text{Pushout}(D) \) of \( \beta_1 \) and the 1-st injection \( \iota_1 : I_1 \to \text{Pushout}(D) \).

#### 6.12.27 MorphismFromSourceToPushoutWithGivenPushout (for IsList, IsCapCategoryObject)

[Definition](operation) \[ \text{MorphismFromSourceToPushoutWithGivenPushout}(D, I) \]

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

The arguments are a list of morphisms \( D = (\beta_i : B \to I_i)_{i=1, \ldots, n} \) and an object \( I = \text{Pushout}(D) \). The output is the composition \( \mu : B \to I \) of \( \beta_1 \) and the 1-st injection \( \iota_1 : I_1 \to I \).

### 6.12.28 UniversalMorphismFromPushout

[Definition](function) \[ \text{UniversalMorphismFromPushout}(\text{arg}) \]

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

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

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


[Definition](operation) \[ \text{UniversalMorphismFromPushoutOp}(D, \tau, \text{method_selection_morphism}) \]

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

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

\[ \text{UniversalMorphismFromPushoutWithGivenPushout}(D, \tau, I) \] (operation)

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

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

6.12.31 AddPushout (for IsCapCategory, IsFunction)

\[ \text{AddPushout}(C, F) \] (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. \( F : ((\beta_i : B \to I_i)_{i=1,...,n}) \mapsto I \)

6.12.32 AddInjectionOfCofactorOfPushout (for IsCapCategory, IsFunction)

\[ \text{AddInjectionOfCofactorOfPushout}(C, F) \] (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. \( F : ((\beta_i : B \to I_i)_{i=1,...,n}, k) \mapsto \iota_k \)

6.12.33 AddInjectionOfCofactorOfPushoutWithGivenPushout (for IsCapCategory, IsFunction)

\[ \text{AddInjectionOfCofactorOfPushoutWithGivenPushout}(C, F) \] (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. \( F : ((\beta_i : B \to I_i)_{i=1,...,n}, k, I) \mapsto \iota_k \)

6.12.34 AddMorphismFromSourceToPushout (for IsCapCategory, IsFunction)

\[ \text{AddMorphismFromSourceToPushout}(C, F) \] (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. \( F : ((\beta_i : B \to I_i)_{i=1,...,n}) \mapsto \mu \)

6.12.35 AddMorphismFromSourceToPushoutWithGivenPushout (for IsCapCategory, IsFunction)

\[ \text{AddMorphismFromSourceToPushoutWithGivenPushout}(C, F) \] (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. \( F : ((\beta_i : B \to I_i)_{i=1,...,n}, I) \mapsto \mu \)
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 : ((β_i : B → I_i)_{i=1..n}, τ) ↦ u(τ)

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 : ((β_i : B → I_i)_{i=1..n}, τ, I) ↦ u(τ)


▷ PushoutFunctorial(Ls, Lm, Lr)

(operation)

Returns: a morphism in Hom(Pushout((β)|_{i=1}, Pushout((β)|_{i=1})))

The arguments are three lists of morphisms L_s = (β_i : B → I_i)_{i=1..n}, L_m = (μ_i : I_i → I'_i)_{i=1..n}, L_r = (β'_i : B' → I'_i)_{i=1..n} having the same length n such that there exists a morphism B : B → B' such that β'_i ∼ β_i. for i = 1..n. The output is the morphism Pushout((β)|_{i=1}) → Pushout((β)|_{i=1}) given by the functoriality of the pushout.


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

(operation)

Returns: a morphism in Hom(s, r)

The arguments are an object s = Pushout((β)|_{i=1}, three lists of morphisms L_s = (β_i : B → I_i)_{i=1..n}, L_m = (μ_i : I_i → I'_i)_{i=1..n}, L_r = (β'_i : B' → I'_i)_{i=1..n} having the same length n such that there exists a morphism B : B → B' such that β'_i ∼ β_i. for i = 1..n, and an object r = Pushout((β)|_{i=1}). The output is the morphism s → 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 : (Pushout((β)|_{i=1}), (β : B → I_i)_{i=1..n}, (μ_i : I_i → I'_i)_{i=1..n}, (β'_i : B' → I'_i)_{i=1..n}, Pushout((β)|_{i=1})) ↦ (Pushout((β)|_{i=1}) → Pushout((β)|_{i=1}))

6.13 Image

For a given morphism α : A → B, an image of α consists of four parts:
• an object \( I \),
• a morphism \( c : A \to I \),
• a monomorphism \( \iota : I \hookrightarrow B \) such that \( \iota \circ c \sim_{A,B} \alpha \),
• a dependent function \( u \) mapping each pair of morphisms \( \tau = (\tau_1 : A \to T, \tau_2 : T \hookrightarrow B) \) where \( \tau_2 \) is a monomorphism such that \( \tau_2 \circ \tau_1 \sim_{A,B} \alpha \) to a morphism \( u(\tau) : I \to T \) such that \( \tau_2 \circ u(\tau) \sim_{I,B} \iota \) and \( u(\tau) \circ c \sim_{A,T} \tau_1 \).

The 4-tuple \((I, c, \iota, u)\) 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.

\[\text{Im}(\alpha) \]

6.13.1 IsomorphismFromImageObjectToKernelOfCokernel (for IsCapCategoryMorphism)

\[\text{IsomorphismFromImageObjectToKernelOfCokernel(} \alpha \text{)} \quad \text{(attribute)}\]

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 \( \text{im}(\alpha) \to \text{KernelObject}(\text{CokernelProjection}(\alpha)) \).

6.13.2 AddIsomorphismFromImageObjectToKernelOfCokernel (for IsCapCategory, IsFunction)

\[\text{AddIsomorphismFromImageObjectToKernelOfCokernel(} C, F \text{)} \quad \text{(operation)}\]

Returns: nothing

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{IsomorphismFromImageObjectToKernelOfCokernel}. \( F : \alpha \mapsto (\text{im}(\alpha) \to \text{KernelObject}(\text{CokernelProjection}(\alpha))) \)

6.13.3 IsomorphismFromKernelOfCokernelToImageObject (for IsCapCategoryMorphism)

\[\text{IsomorphismFromKernelOfCokernelToImageObject(} \alpha \text{)} \quad \text{(attribute)}\]

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 \( \text{KernelObject}(\text{CokernelProjection}(\alpha)) \to \text{im}(\alpha) \).
6.13.4 AddIsomorphismFromKernelOfCokernelToImageObject (for IsCapCategory, IsFunction)

▷ AddIsomorphismFromKernelOfCokernelToImageObject(C, F) (operation)

Returns: nothing

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

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 CoastrictionToImage (for IsCapCategoryMorphism)

▷ CoastrictionToImage(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 coastriction to image $c : A \rightarrow \text{im}(\alpha)$.

6.13.9 CoastrictionToImageWithGivenImageObject (for IsCapCategoryMorphism, IsCapCategoryObject)

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

6.13.11 UniversalMorphismFromImageWithGivenImageObject (for IsCapCategoryMorphism, IsList, IsCapCategoryObject)

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

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

The arguments are a morphism $\alpha : A \to B$, a pair of morphisms $\tau = (\tau_1 : A \to T, \tau_2 : T \hookrightarrow B)$ where $\tau_2$ 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 $u(\tau) : \text{im}(\alpha) \to T$ given by the universal property of the image.

6.13.12 AddImageObject (for IsCapCategory, IsFunction)

▷ AddImageObject(C, F) (operation)

Returns: nothing

The arguments are a category $C$ and a function $F$. This 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. $F : \alpha \mapsto i$.

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. $F : (\alpha, I) \mapsto i$.

6.13.15 AddCoastrictionToImage (for IsCapCategory, IsFunction)

▷ AddCoastrictionToImage(C, F) (operation)

Returns: nothing

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

6.13.16 AddCoastrictionToImageWithGivenImageObject (for IsCapCategory, IsFunction)

▷ AddCoastrictionToImageWithGivenImageObject(C, F) (operation)

Returns: nothing
The arguments are a category \( C \) and a function \( F \). This 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) \hspace{1cm} (operation)

Returns: nothing

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) \hspace{1cm} (operation)

Returns: nothing

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 \( \alpha : A \to B \), a coimage of \( \alpha \) consists of four parts:

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

The 4-tuple \((C, \pi, a, u)\) 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.

\[
\begin{array}{c}
A \\
\pi \\
\alpha \\
\downarrow \\
C \\
\tau_1 \\
\downarrow \\
\text{coim(\alpha)} \\
\tau_2 \\
\downarrow \\
F \\
\end{array}
\]
6.14.1 MorphismFromCoimageToImage (for IsCapCategoryMorphism)

▷ MorphismFromCoimageToImage(alpha)

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

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

6.14.2 MorphismFromCoimageToImageWithGivenObjects (for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryObject)

▷ MorphismFromCoimageToImageWithGivenObjects(alpha)

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

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

6.14.3 AddMorphismFromCoimageToImageWithGivenObjects (for IsCapCategory, IsFunction)

▷ AddMorphismFromCoimageToImageWithGivenObjects(C, F)

Returns: nothing

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

6.14.4 InverseMorphismFromCoimageToImage (for IsCapCategoryMorphism)

▷ InverseMorphismFromCoimageToImage(alpha)

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

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

6.14.5 InverseMorphismFromCoimageToImageWithGivenObjects (for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryObject)

▷ InverseMorphismFromCoimageToImageWithGivenObjects(C, alpha, I)

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

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

6.14.6 AddInverseMorphismFromCoimageToImageWithGivenObjects (for IsCapCategory, IsFunction)

▷ AddInverseMorphismFromCoimageToImageWithGivenObjects(C, F)

Returns: nothing

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. $F : (C, \alpha, I) \mapsto (I \to C)$.
6.14.7 IsomorphismFromCoimageToCokernelOfKernel (for IsCapCategoryMorphism)

▶ IsomorphismFromCoimageToCokernelOfKernel(alpha)

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

The argument is a morphism α: A → B. The output is the canonical morphism coim(α) → CokernelObject(KernelEmbedding(α)).

6.14.8 AddIsomorphismFromCoimageToCokernelOfKernel (for IsCapCategory, IsFunction)

▶ AddIsomorphismFromCoimageToCokernelOfKernel(C, F)

Returns: nothing

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: α ↦ (coim(α) → CokernelObject(KernelEmbedding(α))).

6.14.9 IsomorphismFromCokernelOfKernelToCoimage (for IsCapCategoryMorphism)

▶ IsomorphismFromCokernelOfKernelToCoimage(alpha)

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

The argument is a morphism α: A → B. The output is the canonical morphism CokernelObject(KernelEmbedding(α)) → coim(α).

6.14.10 AddIsomorphismFromCokernelOfKernelToCoimage (for IsCapCategory, IsFunction)

▶ AddIsomorphismFromCokernelOfKernelToCoimage(C, F)

Returns: nothing

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: α ↦ (CokernelObject(KernelEmbedding(α)) → coim(α)).

6.14.11 Coimage (for IsCapCategoryMorphism)

▶ Coimage(alpha)

Returns: an object

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

6.14.12 CoimageProjection (for IsCapCategoryMorphism)

▶ CoimageProjection(alpha)

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

The argument is a morphism α: A → B. The output is the coimage projection π: A → coim(α).
6.14.13 CoimageProjectionWithGivenCoimage (for IsCapCategoryMorphism, IsCapCategoryObject)

▷ CoimageProjectionWithGivenCoimage(alpha, C) (operation)

Returns: a morphism in Hom(A, C)

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

6.14.14 AstrictionToCoimage (for IsCapCategoryMorphism)

▷ AstrictionToCoimage(alpha) (attribute)

Returns: a morphism in Hom(coim(\alpha), B)

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

6.14.15 AstrictionToCoimageWithGivenCoimage (for IsCapCategoryMorphism, IsCapCategoryObject)

▷ AstrictionToCoimageWithGivenCoimage(alpha, C) (operation)

Returns: a morphism in Hom(C, B)

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

6.14.16 UniversalMorphismIntoCoimage (for IsCapCategoryMorphism, IsList)

▷ UniversalMorphismIntoCoimage(alpha, tau) (operation)

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

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


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

Returns: a morphism in Hom(T, C)

The arguments are a morphism \( \alpha : A \to B \), a pair of morphisms \( \tau = (\tau_1 : A \to T, \tau_2 : T \to B) \) 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 \( u(\tau) : T \to C \) 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. \( F : \alpha \to C \).

▷ AddCoimageProjection(C, F)  
  Returns: nothing
  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)  
  Returns: nothing
  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)  
  Returns: nothing
  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)  
  Returns: nothing
  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. \( F : (\alpha, \tau) \mapsto a \)

6.14.23  AddUniversalMorphismIntoCoimage (for IsCapCategory, IsFunction)

▷ AddUniversalMorphismIntoCoimage(C, F)  
  Returns: nothing
  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. \( F : (\alpha, \tau) \mapsto u(\tau) \)


▷ AddUniversalMorphismIntoCoimageWithGivenCoimage(C, F)  
  Returns: nothing
  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 Hom(im(alpha), coim(alpha))  
   
   The argument is a morphism alpha : A → B. The output is the canonical identification c : im(alpha) → 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 ↦ c

6.14.27 **CanonicalIdentificationFromCoimageToImageObject** (for **IsCapCategory-Morphism**)

▷ **CanonicalIdentificationFromCoimageToImageObject**(alpha)  

   *(attribute)*  
   
   Returns: a morphism in Hom(coim(alpha), im(alpha))  
   
   The argument is a morphism alpha : A → B. The output is the canonical identification c : coim(alpha) → 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 ↦ 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 → B, beta : B → C and outputs the subquotient of B given by

* H := KernelObject(beta) / (KernelObject(beta) ∩ 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 → E, delta : E → F of morphisms, and a morphism epsilon : B → E such that there exists omega_1 : A → D, omega_2 : C → F with epsilon ∘ alpha ~_{A,E} gamma ∘ omega_1 and omega_2 ∘ beta ~_{B,F} delta ∘ epsilon 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 \( \alpha : A \to B, \beta : B \to C \). The output is the homology object \( H \) of this pair.

6.15.2 HomologyObjectFunctorial (for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ HomologyObjectFunctorial(alpha, beta, epsilon, gamma, delta) (operation)

Returns: a morphism in \( \operatorname{Hom}(H_1, H_2) \)

The arguments are five morphisms \( \alpha : A \to B, \beta : B \to C, \epsilon : B \to E, \gamma : D \to E, \delta : E \to F \) such that there exists \( \omega_1 : A \to D, \omega_2 : C \to F \) with \( \epsilon \circ \alpha \sim A, E \gamma \circ \omega_1 \) and \( \omega_2 \circ \beta \sim B, F \delta \circ \epsilon \). The output is the functorial morphism induced by \( \epsilon \) 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 \( \operatorname{Hom}(H_1, H_2) \)

The arguments are an object \( H_1 \), a list \( L \) consisting of five morphisms \( \alpha : A \to B, \beta : B \to C, \epsilon : B \to E, \gamma : D \to E, \delta : E \to F \), and an object \( H_2 \). The output is the natural isomorphism from the homology object \( H_1 \) to the construction of the homology object as \( \text{ImageObject}(\text{PreCompose}(\text{KernelEmbedding}(\beta), \text{CokernelProjection}(\alpha))) \), denoted by \( I \).

6.15.4 IsomorphismFromHomologyObjectToItsConstructionAsAnImageObject (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ IsomorphismFromHomologyObjectToItsConstructionAsAnImageObject(alpha, beta) (operation)

Returns: a morphism in \( \operatorname{Hom}(\text{HomologyObject}(\alpha, \beta), I) \)

The arguments are two morphisms \( \alpha : A \to B, \beta : B \to C \). 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 \( \operatorname{Hom}(I, \text{HomologyObject}(\alpha, \beta)) \)

The arguments are two morphisms \( \alpha : A \to B, \beta : B \to C \). The output is the natural isomorphism from the construction of the homology object as
ImageObject(PreCompose(KernelEmbedding(β), CokernelProjection(α))), denoted by \( I \), to the homology object \( H \) of \( α \) and \( β \).

### 6.15.6 AddHomologyObject (for IsCapCategory, IsFunction)

\[ \triangleleft \text{AddHomologyObject}(C, F) \]  

(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 \( \text{HomologyObject} \). \( F : (α, β) \mapsto \text{HomologyObject}(α, β) \).

### 6.15.7 AddHomologyObjectFunctorialWithGivenHomologyObjects (for IsCapCategory, IsFunction)

\[ \triangleleft \text{AddHomologyObjectFunctorialWithGivenHomologyObjects}(C, F) \]  

(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 \( \text{AddHomologyObjectFunctorialWithGivenHomologyObjects} \). \( F : (H_1, α, β, ε, γ, δ, H_2) \mapsto (H_1 \rightarrow H_2) \).
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 depend on the name SomeFunc. It is called with the same input as the function itself, and should return either [true], which continues the computation, or [false, "message"], which will cause an error with message "message" and some additional information.

- **Full prefunction**: The full 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:

```plaintext
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 wether there should be to do list entries which propagate well definedness from the input of the installed methods to their output. Please note that true only makes sense if at least one argument and the output of the installed method is a cell.

• cache_name (optional): The name of the cache which is used for the installed methods. If no cache name is given, the caching for the operation is deactivated completely.

• argument_list (optional): A list containing integers, which defines which arguments should be used for the additional functions, (e.g redirect, pre, ...). This is important for the Op method contructions. If no argument list is given, all arguments are used. Please note that if you have a method selection argument for your function, you need to give the argument_list to explicitly state which argument is the method selection argument.

• return_type (optional): The return type can be one of the following:
  - object or object_or_fail,
  - morphism or morphism_or_fail,
  - twocell,
  - bool,
  - other_object,
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

\[\text{CAPOperationPrepareFunction}(\text{prepare\_function},\ \text{category},\ \text{func})\]

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

\[\text{CAPAddPrepareFunction}(\text{prepare\_function},\ \text{name},\ \text{doc\_string}[,\ \text{precondition\_list}])\]

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)

  Lists all prepare functions.
Chapter 8

Managing Derived Methods

8.1 Info Class

8.1.1 DerivationInfo

\[ \text{DerivationInfo} \]

Info class for derivations.

8.1.2 ActivateDerivationInfo

\[ \text{ActivateDerivationInfo}(\text{arg}) \]

8.1.3 DeactivateDerivationInfo

\[ \text{DeactivateDerivationInfo}(\text{arg}) \]

8.2 Derivation Objects

8.2.1 IsDerivedMethod (for IsObject)

\[ \text{IsDerivedMethod}(\text{arg}) \]

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)

\[ \text{MakeDerivation}(\text{name, target_op, used_ops_with_multiples, weight, implementations_with_extra_filters, category_filter}) \]

135
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)

\[ \text{IsApplicableToCategory}(d, C) \]

\textbf{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)

\[ \text{TargetOperation}(d) \]

\textbf{Returns:} The name (as a string) of the operation implemented by the derivation \( d \)

8.2.9 UsedOperations (for IsDerivedMethod)

\[ \text{UsedOperations}(d) \]

\textbf{Returns:} The names (as strings) of the operations used by the derivation \( d \)

8.2.10 UsedOperationMultiples (for IsDerivedMethod)

\[ \text{UsedOperationMultiples}(d) \]

\textbf{Returns:} Multiplicities of each operation used by the derivation \( d \), in order corresponding to the operation names returned by \text{UsedOperations}(d).

8.2.11 UsedOperationsWithMultiples (for IsDerivedMethod)

\[ \text{UsedOperationsWithMultiples}(d) \]

\textbf{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 \([\text{op\_name}, \text{mult}]\), where \text{op\_name} is a string and \text{mult} a positive integer.

8.2.12 InstallDerivationForCategory (for IsDerivedMethod, IsPosInt, IsCapCategory)

\[ \text{InstallDerivationForCategory}(d, \text{weight}, C) \]

Install the derived method \( d \) for the category \( C \). The integer \text{weight} is the computed weight of the operation implemented by this derivation.

8.2.13 DerivationResultWeight (for IsDerivedMethod, IsDenseList)

\[ \text{DerivationResultWeight}(d, \text{op\_weights}) \]

Computes the resulting weight of the target operation of this derivation given a list of weights for the operations it uses. The argument \text{op\_weights} should be a list of integers specifying weights for the operations given by \text{UsedOperations}(d), in the same order.

8.2.14 FunctionCalledBeforeInstallation (for IsDerivedMethod)

\[ \text{FunctionCalledBeforeInstallation}(d) \]

\textbf{Returns:}
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)

\[ \text{IsDerivedMethodGraph}(\text{arg}) \]

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)

\[ \text{MakeDerivationGraph} (\text{operations}) \]

Make a derivation graph containing the given set of operations and no derivations. The argument \text{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 \text{AddDerivation}.

8.3.3 AddOperationsToDerivationGraph (for IsDerivedMethodGraph, IsDenseList)

\[ \text{AddOperationsToDerivationGraph} (\text{graph}, \text{operations}) \]

Adds a list of operation names \text{operations} to a given derivation graph \text{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)

\[ \text{AddDerivation}(G, d) \]

Add a derivation to a derivation graph.

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

\[ \text{AddDerivation}(\text{arg1}, \text{arg2}, \text{arg3}, \text{arg4}) \]

8.3.6 AddDerivation (for IsDerivedMethodGraph, IsFunction, IsDenseList)

\[ \text{AddDerivation}(\text{arg1}, \text{arg2}, \text{arg3}) \]
8.3.7 AddDerivation (for IsDerivedMethodGraph, IsFunction, IsFunction)

\[ \text{AddDerivation}(\text{arg1}, \text{arg2}, \text{arg3}) \] (operation)

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

\[ \text{AddDerivationPair}(\text{arg1}, \text{arg2}, \text{arg3}, \text{arg4}, \text{arg5}, \text{arg6}) \] (operation)

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

\[ \text{AddDerivationPair}(\text{arg1}, \text{arg2}, \text{arg3}, \text{arg4}, \text{arg5}) \] (operation)

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

\[ \text{AddDerivationPair}(\text{arg1}, \text{arg2}, \text{arg3}, \text{arg4}, \text{arg5}, \text{arg6}) \] (operation)

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

\[ \text{AddDerivationPair}(\text{arg1}, \text{arg2}, \text{arg3}, \text{arg4}, \text{arg5}) \] (operation)

8.3.12 AddDerivationToCAP

\[ \text{AddDerivationToCAP}(\text{arg}) \] (function)

8.3.13 AddDerivationPairToCAP

\[ \text{AddDerivationPairToCAP}(\text{arg}) \] (function)

8.3.14 AddWithGivenDerivationPairToCAP

\[ \text{AddWithGivenDerivationPairToCAP}(\text{arg}) \] (function)

8.3.15 Operations (for IsDerivedMethodGraph)

\[ \text{Operations}(G) \] (attribute)

Gives the operations in the graph \( G \), as a list of strings.
8.3.16 DerivationsUsingOperation (for IsDerivedMethodGraph, IsString)

\[ \text{DerivationsUsingOperation}(G, \text{op\_name}) \]

Finds all the derivations in the graph \( G \) that use the operation named \( \text{op\_name} \), and returns them as a list.

8.3.17 DerivationsOfOperation (for IsDerivedMethodGraph, IsString)

\[ \text{DerivationsOfOperation}(G, \text{op\_name}) \]

Finds all the derivations in the graph \( G \) targeting the operation named \( \text{op\_name} \) (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)

\[ \text{IsOperationWeightList}(\text{arg}) \]

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)

\[ \text{MakeOperationWeightList}(C, G) \]

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)

\[ \text{DerivationGraph}(|\text{owl}|) \]

Returns the derivation graph used by the operation weight list \( |\text{owl}| \).

8.4.4 CategoryOfOperationWeightList (for IsOperationWeightList)

\[ \text{CategoryOfOperationWeightList}(|\text{owl}|) \]

Returns the CAP category associated to the operation weight list \( |\text{owl}| \).
8.4.5 CurrentOperationWeight (for IsOperationWeightList, IsString)

\[\text{CurrentOperationWeight}(\text{owl}, \text{op\_name})\]

Returns the current weight of the operation named \text{op\_name}.

8.4.6 OperationWeightUsingDerivation (for IsOperationWeightList, IsDerivedMethod)

\[\text{OperationWeightUsingDerivation}(\text{owl}, \text{d})\]

Finds out what the weight of the operation implemented by the derivation \text{d} would be if we had used that derivation.

8.4.7 DerivationOfOperation (for IsOperationWeightList, IsString)

\[\text{DerivationOfOperation}(\text{owl}, \text{op\_name})\]

Returns the derivation which is currently used to implement the operation named \text{op\_name}. If the operation is not implemented by a derivation (that is, either implemented directly or not implemented at all), then \text{fail} is returned.

8.4.8 InstallDerivationsUsingOperation (for IsOperationWeightList, IsString)

\[\text{InstallDerivationsUsingOperation}(\text{owl}, \text{op\_name})\]

Performs a search from the operation \text{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)

\[\text{Reevaluate}(\text{owl})\]

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)

\[\text{Saturate}(\text{owl})\]

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)

\( \text{AddPrimitiveOperation}(\text{owl}, \text{op\_name}, \text{weight}) \)

Add the operation named \text{op\_name} to the operation weight list \text{owl} with weight \text{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)

\( \text{PrintDerivationTree}(\text{owl}, \text{op\_name}) \)

Print a tree representation of the way the operation named \text{op\_name} is implemented in the category of the operation weight list \text{owl}.

8.4.13 PrintTree (for IsObject, IsFunction, IsFunction)

\( \text{PrintTree}(\text{arg1}, \text{arg2}, \text{arg3}) \)

Prints a tree structure.

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

\( \text{PrintTreeRec}(\text{arg1}, \text{arg2}, \text{arg3}, \text{arg4}) \)

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)

\( \text{IsStringMinHeap}(\text{arg}) \)

\textbf{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

\( \text{StringMinHeap}(\text{arg}) \)

Create an empty string min heap.

8.5.3 Add (for IsStringMinHeap, IsString, IsInt)

\( \text{Add}(\text{H}, \text{string}, \text{key}) \)

Add a new node containing the label \text{string} and the key \text{key} to the heap \text{H}. 
8.5.4 ExtractMin (for IsStringMinHeap)

\[
\text{ExtractMin}(H)
\]

(operation)

Remove a node with minimal key value from the heap \(H\), and return it. The return value is a list \([\text{label}, \text{key}]\), where \text{label} is the extracted node’s label (a string) and \text{key} is the node’s key (an integer).

8.5.5 DecreaseKey (for IsStringMinHeap, IsString, IsInt)

\[
\text{DecreaseKey}(H, \text{string}, \text{key})
\]

(operation)

Decrease the key value for the node with label \text{string} in the heap \(H\). The new key value is given by \text{key} and must be smaller than the node’s current value.

8.5.6 IsEmptyHeap (for IsStringMinHeap)

\[
\text{IsEmptyHeap}(H)
\]

(operation)

Returns \text{true} if the heap \(H\) is empty, \text{false} otherwise.

8.5.7 HeapSize (for IsStringMinHeap)

\[
\text{HeapSize}(H)
\]

(operation)

Returns the number of nodes in the heap \(H\).

8.5.8 Contains (for IsStringMinHeap, IsString)

\[
\text{Contains}(H, \text{string})
\]

(operation)

Returns \text{true} if the heap \(H\) contains a node with label \text{string}, and \text{false} otherwise.

8.5.9 Swap (for IsStringMinHeap, IsPosInt, IsPosInt)

\[
\text{Swap}(H, i, j)
\]

(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)

\[
\text{Heapify}(H, i)
\]

(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)

\[ \text{ObjectCache(functor)} \]

(returns: IsCachingObject)

Returns the caching object which stores the results of the functor \( \text{functor} \) applied to objects.

9.1.2 MorphismCache (for IsCapFunctor)

\[ \text{MorphismCache(functor)} \]

(returns: IsCachingObject)

Returns the caching object which stores the results of the functor \( \text{functor} \) applied to morphisms.

9.2 Install Functions for IsWellDefined

9.2.1 InstallMethodWithToDoForIsWellDefined

\[ \text{InstallMethodWithToDoForIsWellDefined(arg)} \]

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 auxiliary function which computes the output from the function given to InstallMethodWithToDoForIsWellDefined, then installs the ToDoListEntries, and then installs the auxiliary function instead of the original one. This is normally done with InstallMethod. However, one can change this via the option InstallMethod, which can be set to any other function which is then used instead of InstallMethod. This is used for the caching functions.
9.2.2 **InstallSetWithToDoForIsWellDefined** (for IsObject, IsString, IsList)

```
InstallSetWithToDoForIsWellDefined(arg1, arg2, arg3) (operation)
```

For the caching one needs the possibility to install setters for functions with multiple arguments. This is a setter function which automatically adds ToDoListEntries for IsWellDefined like described above for the manually setted result of a method.

9.2.3 **DeclareAttributeWithToDoForIsWellDefined**

```
DeclareAttributeWithToDoForIsWellDefined(arg) (function)
```

Since attributes install their setters themselfes, one needs to declare attributes in another way to ensure ToDoListEntries for IsWellDefined in the setter of an attribute. This function works like DeclareAttribute, but installs ToDoListEntries for the setter of the attribute. Please note that implementations still need to be done with InstallMethodWithToDoForIsWellDefined.

9.2.4 **DeclareFamilyProperty**

```
DeclareFamilyProperty(arg) (function)
```

9.2.5 **CAP_INTERNAL_REPLACE_STRINGS_WITH_FILTERS**

```
CAP_INTERNAL_REPLACE_STRINGS_WITH_FILTERS(list, category) (function)

Returns: Replaced list
```

The function takes a list (of lists) of filters or strings, where the strings can be category, cell, object, morphism, or twocell. The strings are then recursively replaced by the corresponding filters of the category. The replaced list is returned.

9.2.6 **CAP_INTERNAL_MERGE_FILTER_LISTS**

```
CAP_INTERNAL_MERGE_FILTER_LISTS(list, additional, list) (function)

Returns: merged lists
```

The first argument should be a dense list with filters, the second a sparse list containing filters not longer then the first one. The filters of the second list are then appended (via and) to the filters in the first list at the corresponding position, and the resulting list is returned.

9.2.7 **CAP_INTERNAL_RETURN_OPTION_OR_DEFAULT**

```
CAP_INTERNAL_RETURN_OPTION_OR_DEFAULT(string, value) (function)

Returns: option value
```

Returns the value of the option with name string, or, if this value is fail, the object value.

9.2.8 **CAP_INTERNAL_FIND_APPEARANCE_OF_SYMBOL_IN_FUNCTION**

```
CAP_INTERNAL_FIND_APPEARANCE_OF_SYMBOL_IN_FUNCTION(function, symbol_list, loop_multiple, replacement_record) (function)

Returns: a list of symbols with multiples
```
The function searches for the appearance of the strings in symbol list on the function function and returns a list of pairs, containing the name of the symbol and the number of appearance. If the symbol appears in a loop, the number of appearance is counted times the loop multiple. Moreover, if appearances of found strings should be replaced by collections of other strings, then these can be specified in the replacement record.

9.2.9 CAP_INTERNAL_MERGE_PRECONDITIONS_LIST

```plaintext
function CAP_INTERNAL_MERGE_PRECONDITIONS_LIST(list1, list2)

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

```plaintext
function CAP_INTERNAL_ASSERT_IS_CELL_OF_CATEGORY(cell, category, human_readable_identifier_getter)

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

```plaintext
function CAP_INTERNAL_ASSERT_IS_OBJECT_OF_CATEGORY(object, category, human_readable_identifier_getter)

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

```plaintext
function CAP_INTERNAL_ASSERT_IS_MORPHISM_OF_CATEGORY(morphism, category, human_readable_identifier_getter)

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

\[\text{\textbackslash{}CAP\textunderscore{}INTERNAL\textunderscore{}ASSERT\textunderscore{}IS\textunderscore{}TWO\textunderscore{}CELL\textunderscore{}OF\textunderscore{}CATEGORY}(\texttt{two\_cell}, \texttt{category}, \texttt{human\_readable\_identifier\_getter})\]

The function throws an error if \texttt{two\_cell} is not a 2-cell of \texttt{category}. If \texttt{category} is the boolean \texttt{false}, only general checks not specific to a concrete category are performed. \texttt{human\_readable\_identifier\_getter} is a 0-ary function returning a string which is used to refer to \texttt{two\_cell} in the error message.

9.2.14 CachingStatistic

\[\text{\textbackslash{}CachingStatistic}(\texttt{category}[\texttt{operation}])\]

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

9.2.15 BrowseCachingStatistic

\[\text{\textbackslash{}BrowseCachingStatistic}(\texttt{category})\]

Displays statistics for all caches in \texttt{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:

```haskell
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 TensorProductOnMorphismsWithGivenTensorProducts( source, alpha, DualOnMorphisms( beta ), range ); end;;
gap> AddObjectFunction( F, obj_func );
gap> AddMorphismFunction( F, mor_func );
```

CAP regards $F$ as a binary functor on a technical level, as we can see by looking at its input signature:

```
Example

gap> InputSignature( F );
[ [ Category of left presentations of Q, false ], [ Category of left presentations of Q, true ] ]
```

We can see that \texttt{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> p11 := ProjectionInFactorOfDirectSum( [ V1, V1 ], 1 );;
gap> p12 := ProjectionInFactorOfDirectSum( [ V3, V1 ], 1 );;
gap> value1 := ApplyFunctor( F, p11, p12 );;
gap> input := Product( p11, Opposite( p12 ) );;
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],
```
CAP regards $F^2$ as a unary functor on a technical level, as we can see by looking at its input signature:

```gap
gap> InputSignature( F2 );
[ [ Product of: Category of left presentations of Q, Opposite of Category of left presentations of Q, false ] ]
```

Installation of the first functor as a GAP-operation. It will be installed both as a unary and binary version.

```gap
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.

```gap
gap> InstallFunctor( F2, "F_installation2" );
gap> F_installation2( input );
gap> F_installation2OnObjects( Product( V1, Opposite( V1 ) ) );
gap> F_installation2OnMorphisms( input );
```

### 11.2 Homomorphism structure

```gap
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 ) ) );
gap> true;
gap> a = HomStructure( u, u, HomStructure( a ) );
gap> true;
gap> IsEqualForObjects( HomStructure( Qoid ), DistinguishedObjectOfHomomorphismStructure( Qoid ) );
gap> true;
gap> c := FieldAsCategoryMorphism( 3, Qoid );
gap> d := FieldAsCategoryMorphism( 0, Qoid );
```
```gap
gap> left_coeffs := [[a, b], [c, d]];;
gap> right_coeffs := [[PreCompose(a, b), PreCompose(b, c)], [c, PreCompose(a, a)]];;
gap> right_side := [a, b];;
gap> MereExistenceOfSolutionOfLinearSystemInAbCategory(left_coeffs, right_coeffs, right_side);;
true
gap> solution :=
  SolveLinearSystemInAbCategory(
    left_coeffs,
    right_coeffs,
    right_side
  );;
gap> ForAll([1, 2], i ->
    IsCongruentForMorphisms(
      Sum(List([1, 2], j -> PreCompose([left_coeffs[i][j], solution[j], right_coeffs[i][j]]))),
      right_side[i]
    );
true

gap> Lift(c, d);
fail

gap> Colift(c, d);
0

gap> Colift(d, c);
fail
```

Example

```gap
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 ) ) );
false

gap> ForAll( [5,6,7,8], i -> IsEqualForObjects( SimplifyObject( obj1, i ), SimplifyObject( obj1 ) ) );
true

gap> ## Test SimplifyMorphism
> IsEqualForMorphisms( SimplifyMorphism( mor, 0 ), mor );
true

gap> IsEqualForMorphisms( SimplifyMorphism( mor, 1 ), mor );
false

gap> ForAny( [0,1], i -> IsEqualForMorphisms( SimplifyMorphism( mor, i ), SimplifyMorphism( mor, i ) ) );
false

gap> ForAll( [2,3,4,5], i -> IsEqualForMorphisms( SimplifyMorphism( mor, i ), SimplifyMorphism( mor, i ) ) );
true
```
gap> ## Test SimplifySource
> IsEqualForMorphismsOnMor( SimplifySource( mor, 0 ), mor );
true
gap> IsEqualForMorphismsOnMor( SimplifySource( mor, 1 ), mor );
false
gap> ForAny( [0,1,2,3,4], i -> IsEqualForMorphismsOnMor( SimplifySource( mor, i ), SimplifySource( mor, i + 1 ) );
false
gap> ForAll( [5,6,7,8,9], i -> IsEqualForMorphismsOnMor( SimplifySource( mor, i ), SimplifySource( mor, i + 1 ) );
true
gap> IsCongruentForMorphisms( PreCompose( SimplifySource_IsFromInputObject( mor, infinity ), SimplifySource( mor, infinity ) ), SimplifySource( mor, infinity ) );
true
gap> IsCongruentForMorphisms( PreCompose( SimplifySource_IsToInputObject( 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, i + 1 ) );
false
gap> ForAll( [5,6,7,8,9], i -> IsEqualForMorphismsOnMor( SimplifyRange( mor, i ), SimplifyRange( mor, i + 1 ) );
true
gap> IsCongruentForMorphisms( PreCompose( SimplifyRange( mor, infinity ), SimplifyRange_IsToInputObject( 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, i + 1 ) );
false
gap> ForAll( [5,6,7,8,9], i -> IsEqualForMorphismsOnMor( SimplifySourceAndRange( mor, i ), SimplifySourceAndRange( mor, i + 1 ) );
true
gap> IsCongruentForMorphisms( PreCompose( SimplifySourceAndRange_IsFromInputSource( mor, infinity ), SimplifySourceAndRange_IsToInputObject( mor, infinity ) ), SimplifySourceAndRange( mor, infinity ) );
true

gap> IsCongruentForMorphisms( SimplifySourceAndRange( mor, infinity ),
[ SimplifySourceAndRange_IsFromInputSource( mor, infinity ),
  SimplifySourceAndRange( mor, infinity ),
  SimplifySourceAndRange_IsToInputRange( mor, infinity ) ] );
true

> PreCompose( [ SimplifySourceAndRange_IsoToInputSource( mor, infinity ),
> mor, SimplifySourceAndRange_IsoFromInputRange( mor, infinity ) ] )
> );
true
gap> ## Test SimplifyEndo
> endo := StringsAsCategoryMorphism( obj1, "uoiea", obj1 );;
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 ), SimplifySourceAndRange( endo, i + 1 ) ) );
false
 gap> ForAll( [5,6,7,8,9], i -> IsEqualForMorphismsOnMor( SimplifySourceAndRange( endo, i ), SimplifySourceAndRange( endo, i + 1 ) ) );
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
\texttt{gap> CH1 := CohomologyFunctor( Zmod, 1 );}
1-th cohomology functor of Category of left presentations of \( Z \)
\texttt{gap> cmor1 := ApplyFunctor( CH1, cocomplex_mor );}
<A morphism in Category of left presentations of \( Z \)>
\texttt{gap> Display( UnderlyingMatrix( cmor1 ) );}

4
\texttt{gap> ToComplex := CocomplexToComplexFunctor( Zmod );}
Cocomplex to complex functor of Category of left presentations of \( Z \)
\texttt{gap> complex_mor := ApplyFunctor( ToComplex, cocomplex_mor );}
<A morphism in Complex category of Category of left presentations of \( Z \)>
\texttt{gap> H0 := HomologyFunctor( Zmod, 0 );}
0-th homology functor of Category of left presentations of \( Z \)
\texttt{gap> mor0 := ApplyFunctor( H0, complex_mor );}
<A morphism in Category of left presentations of \( Z \)>
\texttt{gap> Display( UnderlyingMatrix( mor0 ) );}

Example
\texttt{gap> QQ := HomalgFieldOfRationalsInSingular( );;}
\texttt{gap> R := QQ * "x,y";}
\( \mathbb{Q}[x,y] \)
\texttt{gap> SetRecursionTrapInterval( 10000 );}
\texttt{gap> category := LeftPresentations( R );}
Category of left presentations of \( \mathbb{Q}[x,y] \)
\texttt{gap> S := FreeLeftPresentation( 1, R );}
<An object in Category of left presentations of \( \mathbb{Q}[x,y] \)>
\texttt{gap> object_func := function( i ) return S; end;}
\texttt{function( i ) ... end}
\texttt{gap> morphism_func := function( i ) return IdentityMorphism( S ); end;}
\texttt{function( i ) ... end}
\texttt{gap> C0 := ZFunctorObjectExtendedByInitialAndIdentity( object_func, morphism_func, category, 0, 4 );}
<An object in Functors from integers into Category of left presentations of \( \mathbb{Q}[x,y] \)>
\texttt{gap> S2 := FreeLeftPresentation( 2, R );}
<An object in Category of left presentations of \( \mathbb{Q}[x,y] \)>
\texttt{gap> C1 := ZFunctorObjectFromMorphismList( [ InjectionOfCofactorOfDirectSum( [ S2, S ], 1 ) ], 2 );}
<An object in Functors from integers into Category of left presentations of \( \mathbb{Q}[x,y] \)>
\texttt{gap> C1 := ZFunctorObjectExtendedByInitialAndIdentity( C1, 2, 3 );}
<An object in Functors from integers into Category of left presentations of \( \mathbb{Q}[x,y] \)>
\texttt{gap> C2 := ZFunctorObjectFromMorphismList( [ InjectionOfCofactorOfDirectSum( [ S, S ], 1 ) ], 3 );}
<An object in Functors from integers into Category of left presentations of \( \mathbb{Q}[x,y] \)>
\texttt{gap> C2 := ZFunctorObjectExtendedByInitialAndIdentity( C2, 3, 4 );}
<An object in Functors from integers into Category of left presentations of \( \mathbb{Q}[x,y] \)>
\texttt{gap> delta_1_3 := PresentationMorphism( C1[3], HomalgMatrix( [ [ "x^-2" ], [ "xy" ], [ "y^-3" ] ], 3 );}
<A morphism in Category of left presentations of \( \mathbb{Q}[x,y] \)>
\texttt{gap> delta_1_2 := PresentationMorphism( C1[2], HomalgMatrix( [ [ "x^-2" ], [ "xy" ] ], 2, 1, R ),}
<A morphism in Category of left presentations of \( \mathbb{Q}[x,y] \)>
gap> delta1 := ZFunctorMorphism( C1, [ UniversalMorphismFromInitialObject( C0[1] ), UniversalMorphismFromInitialObject( C0[1] ), delta_1_2, delta_1_3 ], 0, C0 );
<A morphism in Functors from integers into Category of left presentations of \text{Q}[x,y]>

gap> delta1 := ZFunctorMorphismExtendedByInitialAndIdentity( delta1, 0, 3 );
<A morphism in Category of left presentations of \text{Q}[x,y]>

gap> delta1 := AsAscendingFilteredMorphism( delta1 );
<A morphism in Ascending filtered object category of Category of left presentations of \text{Q}[x,y]>

gap> delta_2_3 := PresentationMorphism( C2[3], HomalgMatrix( [ [ "y", "-x", "0" ] ] ), 1, 3, R );
<A morphism in Category of left presentations of \text{Q}[x,y]>

gap> delta_2_4 := PresentationMorphism( C2[4], HomalgMatrix( [ [ "y", "-x", "0" ], [ "0", "y^2", "-x" ] ] ), 2, 3, R );
<A morphism in Category of left presentations of \text{Q}[x,y]>

gap> delta2 := ZFunctorMorphism( C2, [ UniversalMorphismFromInitialObject( C1[2] ), delta_2_3, delta_2_4 ], 2, C1 );
<A morphism in Functors from integers into Category of left presentations of \text{Q}[x,y]>

gap> delta2 := ZFunctorMorphismExtendedByInitialAndIdentity( delta2, 2, 4 );
<A morphism in Functors from integers into Category of left presentations of \text{Q}[x,y]>

gap> delta2 := AsAscendingFilteredMorphism( delta2 );
<A morphism in Ascending filtered object category of Category of left presentations of \text{Q}[x,y]>

gap> SetIsAdditiveCategory( CategoryOfAscendingFilteredObjects( category ), true );

gap> complex := ZFunctorObjectFromMorphismList( [ delta2, delta1 ] );
<An object in Functors from integers into Ascending filtered object category of Category of left presentations of \text{Q}[x,y]>

gap> complex := AsComplex( complex );
<An object in Complex category of Ascending filtered object category of Category of left presentations of \text{Q}[x,y]>

gap> LessGenFunctor := FunctorLessGeneratorsLeft( R );
Less generators for Category of left presentations of \text{Q}[x,y]

gap> s := SpectralSequenceEntryOfAscendingFilteredComplex( complex, 0, 0, 0 );
<A morphism in Generalized morphism category of Category of left presentations of \text{Q}[x,y]>

gap> Display( UnderlyingMatrix( ApplyFunctor( LessGenFunctor, UnderlyingHonestObject( Source( s ) ) ) ) );
(an empty 0 x 1 matrix)

gap> s := SpectralSequenceEntryOfAscendingFilteredComplex( complex, 1, 0, 0 );
<A morphism in Generalized morphism category of Category of left presentations of \text{Q}[x,y]>

gap> Display( UnderlyingMatrix( ApplyFunctor( LessGenFunctor, UnderlyingHonestObject( Source( s ) ) ) ) );
(an empty 0 x 1 matrix)

gap> s := SpectralSequenceEntryOfAscendingFilteredComplex( complex, 2, 0, 0 );
<A morphism in Generalized morphism category of Category of left presentations of \text{Q}[x,y]>

gap> Display( UnderlyingMatrix( ApplyFunctor( LessGenFunctor, UnderlyingHonestObject( Source( s ) ) ) ) );
(an empty 0 x 1 matrix)

gap> s := SpectralSequenceEntryOfAscendingFilteredComplex( complex, 3, 0, 0 );
<A morphism in Generalized morphism category of Category of left presentations of \text{Q}[x,y]>

gap> Display( UnderlyingMatrix( ApplyFunctor( LessGenFunctor, UnderlyingHonestObject( Source( s ) ) ) ) );
\text{x}*\text{y}, \text{x}^2

gap> s := SpectralSequenceEntryOfAscendingFilteredComplex( complex, 4, 0, 0 );
<A morphism in Generalized morphism category of Category of left presentations of \text{Q}[x,y]>

gap> Display( UnderlyingMatrix( ApplyFunctor( LessGenFunctor, UnderlyingHonestObject( Source( s ) ) ) ) );
\text{x}*\text{y}, \text{x}^2, \text{y}^3

gap> s := SpectralSequenceEntryOfAscendingFilteredComplex( complex, 5, 0, 0 );
<A morphism in Generalized morphism category of Category of left presentations of \text{Q}[x,y]>

gap> Display( UnderlyingMatrix( ApplyFunctor( LessGenFunctor, UnderlyingHonestObject( Source( s ) ) ) ) );
\text{x}*\text{y}, \text{x}^2, \text{y}^3
11.4 Homology object

Example

```gap

gap> field := HomalgFieldOfRationals( );
gap> A := VectorSpaceObject( 1, field );
gap> B := VectorSpaceObject( 2, field );
gap> C := VectorSpaceObject( 3, field );
gap> alpha := VectorSpaceMorphism( A, HomalgMatrix( [[ 1, 0, 0 ]], 1, 3, field ), C );
gap> beta := VectorSpaceMorphism( C, HomalgMatrix( [[ 1, 0 ], [ 1, 1 ], [ 1, 2 ]], 3, 2, field ), B );

false

gap> IsCongruentForMorphisms( IdentityMorphism( HomologyObject( alpha, beta ) ),
                              HomologyObjectFunctorial( alpha, beta, IdentityMorphism( C ), alpha, beta ) );
true

gap> kernel_beta := KernelEmbedding( beta );

```

gap> 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, alpha_op )
> );
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

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

true
11.5 Liftable

Example

```gap
gap> field := HomalgFieldOfRationals();;
gap> V := VectorSpaceObject( 1, field );;
gap> W := VectorSpaceObject( 2, field );;
gap> alpha := VectorSpaceMorphism( V, HomalgMatrix( [[ 1, -1 ]], 1, 2, field ), W );;
gap> beta := VectorSpaceMorphism( W, HomalgMatrix( [[ 1, 2 ], [ 3, 4 ]], 2, 2, field ), W );;
gap> IsLiftable( alpha, beta );
true
gap> IsLiftable( beta, alpha );
false
gap> IsLiftableAlongMonomorphism( beta, alpha );
true
gap> gamma := VectorSpaceMorphism( W, HomalgMatrix( [[ 1 ], [ 1 ]], 2, 1, field ), V );;
gap> IsColiftable( beta, gamma );
true
gap> IsColiftable( gamma, beta );
false
gap> IsColiftableAlongEpimorphism( beta, gamma );
true
```

11.6 Monoidal Categories

Example

```gap
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> Tl := TensorProductOnObjects( M1, N1 );
<An object in Category of left presentations of Z>
gap> Display( UnderlyingMatrix( Tl ) );
[ [ 3 ],
  [ 2 ] ]
gap> IsZeroForObjects( Tl );
true
gap> Bl := Braiding( DirectSum( M1, N1 ), DirectSum( M1, M1 ) );
<A morphism in Category of left presentations of Z>
gap> Display( UnderlyingMatrix( Bl ) );
[ [ 1, 0, 0, 0 ],
  [ 0, 0, 1, 0 ],
  [ 0, 1, 0, 0 ],
  [ 0, 0, 0, 1 ] ]
gap> IsWellDefined( Bl );
true
gap> Ul := TensorUnit( CapCategory( M1 ) );
<An object in Category of left presentations of Z>
gap> IntHoml := InternalHomOnObjects( DirectSum( M1, Ul ), N1 );
<An object in Category of left presentations of Z>
gap> Display( UnderlyingMatrix( IntHoml ) );
[ [ -2, -1 ],
  [ 1, -1 ] ]```
gap> generator_l1 := StandardGeneratorMorphism( IntHoml, 1 );
<A morphism in Category of left presentations of Z>
gap> morphism_l1 := LambdaElimination( DirectSum( Ml, Ul ), Nl, generator_l1 );
<A morphism in Category of left presentations of Z>
gap> Display( UnderlyingMatrix( morphism_l1 ) );
\[
\begin{bmatrix}
0 \\
2
\end{bmatrix}
\]
gap> generator_l2 := StandardGeneratorMorphism( IntHoml, 2 );
<A morphism in Category of left presentations of Z>
gap> morphism_l2 := LambdaElimination( DirectSum( Ml, Ul ), Nl, generator_l2 );
<A morphism in Category of left presentations of Z>
gap> Display( UnderlyingMatrix( morphism_l2 ) );
\[
\begin{bmatrix}
0 \\
2
\end{bmatrix}
\]
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 ) );
\[
\begin{bmatrix}
3 & 2
\end{bmatrix}
\]
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 ) );
\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]
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 ) );
\[
\begin{bmatrix}
-2 & 1 \\
-1 & -1
\end{bmatrix}
\]
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>
\begin{verbatim}
gap> Display( UnderlyingMatrix( morphism_r1 ) );
[ [ 0, 2 ] ]
gap> generator_r2 := StandardGeneratorMorphism( IntHoml, 2 );
<A morphism in Category of left presentations of Z>
gap> morphism_r2 := LambdaElimination( DirectSum( Ml, Ul ), Nl, generator_r2 );
<A morphism in Category of left presentations of Z>
gap> Display( UnderlyingMatrix( morphism_r2 ) );
[ [ 0 ],
  [ 2 ] ]
gap> IsEqualForMorphisms( LambdaIntroduction( morphism_r1 ), generator_r1 );
false
gap> IsCongruentForMorphisms( LambdaIntroduction( morphism_r1 ), generator_r1 );
true
gap> IsEqualForMorphisms( LambdaIntroduction( morphism_r2 ), generator_r2 );
false
gap> IsCongruentForMorphisms( LambdaIntroduction( morphism_r2 ), generator_r2 );
true

11.7 MorphismFromSourceToPushout and MorphismFromFiberProductToSink
\end{verbatim}

--- Example ---

\begin{verbatim}
gap> field := HomalgFieldOfRationals( );;
gap> A := VectorSpaceObject( 3, field );;
gap> B := VectorSpaceObject( 2, field );;
gap> alpha := VectorSpaceMorphism( B, HomalgMatrix( [ [ 1, -1, 1 ], [ 1, 1, 1 ] ], 2, 3, field ), A );;
gap> beta := VectorSpaceMorphism( B, HomalgMatrix( [ [ 1, 2, 1 ], [ 2, 1, 1 ] ], 2, 3, field ), A );;
gap> m := MorphismFromFiberProductToSink( [ alpha, beta ] );;
gap> IsCongruentForMorphisms( m, PreCompose( ProjectionInFactorOfFiberProduct( [ alpha, beta ], 1 ), alpha ) );
true
gap> IsCongruentForMorphisms( m, PreCompose( ProjectionInFactorOfFiberProduct( [ alpha, beta ], 2 ), beta ) );
true
gap> IsCongruentForMorphisms( MorphismFromKernelObjectToSink( alpha ), PreCompose( KernelEmbedding( alpha ), alpha ) );
true
gap> alpha_p := DualOnMorphisms( alpha );;
gap> beta_p := DualOnMorphisms( beta );;
gap> m_p := MorphismFromSourceToPushout( [ alpha_p, beta_p ] );;
gap> IsCongruentForMorphisms( m_p, PreCompose( alpha_p, InjectionOfCofactorOfPushout( [ alpha_p, beta_p ], 1 ) ) );
true
gap> IsCongruentForMorphisms( m_p, PreCompose( beta_p, InjectionOfCofactorOfPushout( [ alpha_p, beta_p ], 2 ) ) );
true
\end{verbatim}
gap> 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> IsOne( UniversalMorphismFromImage( mor, [ CoastrictionToImage( 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
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>
Arrow( c1 + c2);
A rational vector space homomorphism with matrix:
[ [ 12, 12 ] ]
```

First composition test:

```gap
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 ] ]
```
Second composition test

```gap
vecspaces := CreateCapCategory( "VectorSpacesForGeneralizedMorphismsTest" );
true
A := QVectorSpace( 1 );
B := QVectorSpace( 2 );
C := QVectorSpace( 3 );
phi2_tilde_associated := VectorSpaceMorphism( A, [ [ 1, 5 ] ], B );
phi2_tilde_range_aid := VectorSpaceMorphism( C, [ [ 1, 0 ], [ 0, 1 ], [ 1, 1 ] ], B );
phi2_tilde := GeneralizedMorphismWithRangeAid( phi2_tilde_associated, phi2_tilde_range_aid );
psi2_tilde_associated := VectorSpaceMorphism( C, [ [ 1 ], [ 3 ], [ 4 ] ], A );
psi2_tilde_range_aid := VectorSpaceMorphism( B, [ [ 1 ], [ 1 ] ], A );
psi2_tilde := GeneralizedMorphismWithRangeAid( psi2_tilde_associated, psi2_tilde_range_aid );
composition2 := PreCompose( phi2_tilde, psi2_tilde );
```
Third composition test

```
gap> vecspaces := CreateCapCategory( "VectorSpacesForGeneralizedMorphismsTest" );
VectorSpacesForGeneralizedMorphismsTest
true

gap> A := QVectorSpace( 3 );
<A rational vector space of dimension 3>

gap> A := QVectorSpace( 3 );
<A rational vector space of dimension 3>

gap> A := QVectorSpace( 3 );
<A rational vector space of dimension 3>

gap> A := QVectorSpace( 3 );
<A rational vector space of dimension 3>

gap> A := QVectorSpace( 3 );
<A rational vector space of dimension 3>

gap> A := QVectorSpace( 3 );
<A rational vector space of dimension 3>

```

Example

```
gap> A := QVectorSpace( 3 );
<A rational vector space of dimension 3>

gap> A := QVectorSpace( 3 );
<A rational vector space of dimension 3>

gap> A := QVectorSpace( 3 );
<A rational vector space of dimension 3>

gap> A := QVectorSpace( 3 );
<A rational vector space of dimension 3>

gap> A := QVectorSpace( 3 );
<A rational vector space of dimension 3>

```
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>

A rational vector space homomorphism with matrix:
[ [ -1, 1, 0 ],
[ 1, 0, 0 ] ]

A rational vector space homomorphism with matrix:
(an empty 2 x 0 matrix)

A rational vector space homomorphism with matrix:
(an empty 3 x 0 matrix)

A rational vector space of dimension 3

A rational vector space of dimension 2

A rational vector space of dimension 3

A rational vector space of dimension 1

A rational vector space of dimension 2

A rational vector space of dimension 3

A rational vector space of dimension 2

A rational vector space of dimension 2

A rational vector space homomorphism with matrix:
[ [ 1, 0, 0 ],
[ 0, 1, 0 ] ]

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
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
```

1.10 IsWellDefined

```gap
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
gap> g := GeneralizedMorphism(alpha, alpha, alpha);
<A morphism in Generalized morphism category of VectorSpacesForIsWellDefinedTest>

gap> IsWellDefined(alpha);
true
```

true
11.11 Kernel

Example

```gap
gap> IsWellDefined( g );
true

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
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

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 ] ]
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
AddCanonicalIdentificationFromCoimageToObject
for IsCapCategory, IsFunction, 125
AddCanonicalIdentificationFromImageObjectToCoimage
for IsCapCategory, IsFunction, 125
AddCategoricalProperty, 6
AddCoastrictionToObject
for IsCapCategory, IsFunction, 119
AddCoastrictionToObjectWithGivenImageObject
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, 99
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
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
AddIsEqualForCacheForObject
for IsCapCategory, IsFunction, 16
AddIsEqualForMorphisms
for IsCapCategory, IsFunction, 31
AddIsEqualForMorphismsOnMor
for IsCapCategory, IsFunction, 32
AddIsEqualForObject
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
AddIsIsomorphismFromDirectSumToDirect-Product
for IsCapCategory, IsFunction, 85
AddIsIsomorphismFromEqualizerOfDirect-ProductDiagramToFiberProduct
for IsCapCategory, IsFunction, 104
AddIsIsomorphismFromFiberProductToEqualizerOfDirectProductDiagram
for IsCapCategory, IsFunction, 103
AddIsIsomorphismFromFiberProductToKernelOfDiagonalDifference
for IsCapCategory, IsFunction, 103
AddIsIsomorphismFromImageObjectToKernelOfCokernel
for IsCapCategory, IsFunction, 117
AddIsIsomorphismFromInitialObjectToZero-Object
for IsCapCategory, IsFunction, 74
AddIsIsomorphismFromKernelOfCokernelToObject
for IsCapCategory, IsFunction, 118
AddIsIsomorphismFromKernelOfDiagonal-DifferenceToFiberProduct
for IsCapCategory, IsFunction, 103
AddIsIsomorphismFromPushoutToCoequalizerOfCoproductDiagram
for IsCapCategory, IsFunction, 111
AddIsIsomorphismFromPushoutToCokernelOfDiagonalDifference
for IsCapCategory, IsFunction, 110
AddIsIsomorphismFromTerminalObjectToZeroObject
for IsCapCategory, IsFunction, 74
AddIsIsomorphismFromZeroObjectToInitial-Object
for IsCapCategory, IsFunction, 73
AddIsIsOne
for IsCapCategory, IsFunction, 25
AddIsIsProjective
for IsCapCategory, IsFunction, 14
AddIsIsSplitEpimorphism
for IsCapCategory, IsFunction, 25
AddIsIsSplitMonomorphism
for IsCapCategory, IsFunction, 24
AddIsTerminal
for IsCapCategory, IsFunction, 14
AddIsIsWellDefinedForMorPhisms
for IsCapCategory, IsFunction, 37
AddIsIsWellDefinedForObjectS
for IsCapCategory, IsFunction, 18
AddIsIsWellDefinedForTwoCells
for IsCapCategory, IsFunction, 56
AddIsIsZeroForMorphisms
for IsCapCategory, IsFunction, 32
AddIsIsZeroForObjectS
for IsCapCategory, IsFunction, 15
AddIsAdditionForMorphisms
for IsCapCategoryMorphism, IsCapCategoryMorphism, 32
AddIsAdditiveInverseForMorphisms
for IsCapCategoryMorphism, 33
AddIsIsKernelEmbedding
for IsCapCategory, IsFunction, 65
AddIsIsKernelEmbeddingWithGivenKernelObject
for IsCapCategory, IsFunction, 65
AddIsIsKernelLift
for IsCapCategory, IsFunction, 65
AddIsIsKernelLiftWithGivenKernelObject
for IsCapCategory, IsFunction, 66
AddIsIsKernelObject
for IsCapCategory, IsFunction, 65
AddIsIsKernelObjectFunctorialWithGivenKernelObjects
for IsCapCategory, IsFunction, 67
AddIsIsLift
for IsCapCategory, IsFunction, 39
AddIsIsLiftAlongMonomorphism
for IsCapCategory, IsFunction, 38
AddIsIsMonomorphismIntoSomeInjectiveObject
for IsCapCategory, IsFunction, 20
AddIsIsMonomorphismIntoSomeInjective-ObjectWithGivenSomeInjectiveObject
for IsCapCategory, IsFunction, 20
AddIsIsMorphism
for IsCapCategory, IsAttributeStoringRep, 30
AddMorphismBetweenDirectSums for IsCapCategory, IsFunction, 82
AddMorphismFromCokernelToObjectToImageWith-GivenObjects for IsCapCategory, IsFunction, 121
AddMorphismFromEqualizerToSink for IsCapCategory, IsFunction, 96
AddMorphismFromEqualizerToSinkWith-GivenEqualizer for IsCapCategory, IsFunction, 96
AddMorphismFromFiberProductToObjectToSink for IsCapCategory, IsFunction, 108
AddMorphismFromFiberProductToObjectToSinkWith-GivenFiberProduct for IsCapCategory, IsFunction, 108
AddMorphismFromKernelToObjectToSink for IsCapCategory, IsFunction, 65
AddMorphismFromKernelToObjectToSinkWith-GivenKernelObject for IsCapCategory, IsFunction, 65
AddMorphismFromSourceToObjectToCoequalizer for IsCapCategory, IsFunction, 100
AddMorphismFromSourceToObjectToCoequalizerWith-GivenCoequalizer for IsCapCategory, IsFunction, 100
AddMorphismFromSourceToObjectToCokernelObject for IsCapCategory, IsFunction, 69
AddMorphismFromSourceToObjectToCokernelObjectWith-GivenCokernelObject 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
AddProjectionInFactorOfDirectProductWithGivenDirectProduct for IsCapCategory, IsFunction, 92
AddProjectionInFactorOfDirectSum for IsCapCategory, IsFunction, 83
AddProjectionInFactorOfDirectSumWith-GivenDirectSum for IsCapCategory, IsFunction, 83
AddProjectionInFactorOfFiberProduct for IsCapCategory, IsFunction, 107
AddProjectionInFactorOfFiberProductWithGivenFiberProduct 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, 26
AddRandomObjectByList
for IsCapCategory, IsFunction, 26
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_IsoToInputRange
for IsCapCategory, IsFunction, 51
AddSimplifySourceAndRange_IsoToInputSource
for IsCapCategory, IsFunction, 49
AddSimplifySourceAndRange_IsoToInputSource
for IsCapCategory, IsFunction, 50
AddSomeInjectiveObject
for IsCapCategory, IsFunction, 47
AddSomeProjectiveObject
for IsCapCategory, IsFunction, 20
AddSomeProjectiveObject
for IsCapCategory, IsFunction, 18
AddSomeReductionBySplitEpiSummand
for IsCapCategory, IsFunction, 53
AddSomeReductionBySplitEpiSummand_IsoFromInputRange
for IsCapCategory, IsFunction, 53
AddSomeReductionBySplitEpiSummand_IsoToInputRange
for IsCapCategory, IsFunction, 53
AddSimplificationForMorphisms
for IsCapCategory, IsFunction, 33
AddTerminalObject
for IsCapCategory, IsFunction, 75
AddTerminalObjectFunctorial
for IsCapCategory, IsFunction, 76
AddUniversalMorphismFromCoequalizer
for IsCapCategory, IsFunction, 100
AddUniversalMorphismFromCoequalizer_IsoFromGivenCoequalizer
for IsCapCategory, IsFunction, 100
AddUniversalMorphismFromCoproduct
for IsCapCategory, IsFunction, 89
AddUniversalMorphismFromCoproduct_IsoFromGivenCoproduct
for IsCapCategory, IsFunction, 89
AddUniversalMorphismFromDirectSum
for IsCapCategory, IsFunction, 84
AddUniversalMorphismFromDirectSum_IsoFromGivenDirectSum
AddUniversalMorphismFromImage for IsCapCategory, IsFunction, 84
AddUniversalMorphismFromImageWithGivenImageObject for IsCapCategory, IsFunction, 120
AddUniversalMorphismFromInitialObject for IsCapCategory, IsFunction, 77
AddUniversalMorphismFromInitialObjectWithGivenInitialObject for IsCapCategory, IsFunction, 77
AddUniversalMorphismFromPushout for IsCapCategory, IsFunction, 116
AddUniversalMorphismFromPushoutWithGivenPushout for IsCapCategory, IsFunction, 116
AddUniversalMorphismFromZeroObject for IsCapCategory, IsFunction, 73
AddUniversalMorphismIntoCoimage for IsCapCategory, IsFunction, 124
AddUniversalMorphismIntoCoimageWithGivenCoimage for IsCapCategory, IsFunction, 124
AddUniversalMorphismIntoDirectProduct for IsCapCategory, IsFunction, 92
AddUniversalMorphismIntoDirectProductWithGivenDirectProduct for IsCapCategory, IsFunction, 92
AddUniversalMorphismIntoDirectSum for IsCapCategory, IsFunction, 84
AddUniversalMorphismIntoDirectSumWithGivenDirectSum for IsCapCategory, IsFunction, 84
AddUniversalMorphismIntoEqualizer for IsCapCategory, IsFunction, 96
AddUniversalMorphismIntoEqualizerWithGivenEqualizer for IsCapCategory, IsFunction, 96
AddUniversalMorphismIntoFiberProduct for IsCapCategory, IsFunction, 108
AddUniversalMorphismIntoZeroObject for IsCapCategory, IsFunction, 73
AddUniversalMorphismIntoTerminalObject for IsCapCategory, IsFunction, 75
AddUniversalMorphismIntoTerminalObjectWithGivenTerminalObject for IsCapCategory, IsFunction, 76
AddUniversalMorphismIntoZeroObject for IsCapCategory, IsFunction, 73
AddUniversalMorphismIntoZeroObjectWithGivenZeroObject 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 IsCatCategoryAsCatObject, 58
AsCatObject for IsCapCategory, 58
AstrictionToCoimage for IsCapCategoryMorphism, 123
AstrictionToCoimageWithGivenCoimage for IsCapCategoryMorphism, IsCatCategoryObject, 123
BasisOfExternalHom for IsCapCategoryObject, IsCatCategoryObject, 45
BrowseCachingStatistic, 147
/* for IsRingElement, IsCapCategoryMorphism, 33
CachingStatistic, 147
CanCompute for IsCapCategory, IsString, 8
CanonicalIdentificationFromCoimageToImageObject 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
for IsString, IsList, IsCapCategoryAsCatObjectAsCatObject, 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, 145
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
CoastrictionToImage
for IsCapCategoryMorphism, 118
CoastrictionToImageWithGivenImageObject
for IsCapCategoryMorphism, IsCapCategoryObject, 118
CoefficientsOfMorphism
for IsCapCategoryMorphism, 46
CoefficientsOfMorphismWithGivenBasisOfExternalHom
for IsCapCategoryMorphism, IsList, 46
Coequalizer, 97
CoequalizerFunctorial
for IsList, IsCapCategoryMorphism, IsList, 100
CoequalizerFunctorialWithGivenCoequalizers
CoequalizerUp
for IsList, IsCapCategoryMorphism, 98
Coimage
for IsCapCategoryMorphism, 122
CoimageProjection
for IsCapCategoryMorphism, 122
CoimageProjectionWithGivenCoimage
for IsCapCategoryMorphism, IsCapCategoryObject, 123
CokernelColift
for IsCapCategoryMorphism, IsCapCategoryMorphism, 68
CokernelColiftWithGivenCokernelObject
DirectSumCodiagonalDifferenceOp
  for IsList, IsCapCategoryMorphism, 112
DirectSumDiagonalDifference
  for IsList, 104
DirectSumDiagonalDifferenceOp
  for IsList, IsCapCategoryMorphism, 112
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
DistinguishedObjectOfHomomorphismStructure
  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
  with GivenSomeProjectiveObject
  for IsCapCategoryObject, IsCapCategoryObject, 18
EpimorphismFromSomeProjectiveObjectWithGivenSomeProjectiveObject
  for IsCapCategoryObject, IsCapCategoryObject, 18
Equalizer, 93
EqualizerFunctorial
  for IsList, IsCapCategoryMorphism, IsList, 96
EqualizerFunctorialWithGivenEqualizers
  for IsCapCategoryObject, IsList, IsCapCategoryMorphism, IsList, IsCapCategoryObject, 96
EqualizerOp
  for IsList, IsCapCategoryMorphism, 94
ExtractMin
  for IsStringMinHeap, 143
FiberProduct, 105
FiberProductEmbeddingInDirectSum
  for IsList, 105
FiberProductEmbeddingInDirectSumOp
  for IsList, IsCapCategoryMorphism, 105
FiberProductFunctorial
FiberProductFunctorialWithGivenFiberProducts
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, IsCapCategoryMorphism, 126
HomologyObjectFunctorialWithGivenHomologyObjects
for IsCapCategoryObject, IsList, IsCapCategoryObject, 126
HomomorphismStructureOnMorphisms
for IsCapCategoryMorphism, IsCapCategoryMorphism, 42
HomomorphismStructureOnMorphismsWithGivenObjects
for IsCapCategoryObject, 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,
for IsObject, IsString, IsList, 145
InstallNaturalTransformation
for IsCapNaturalTransformation, IsString, 62
InstallSetWithToDoForIsWellDefined
for IsObject, IsString, IsList, 145
InterpretMorphismAsMorphismFrom-
DistinguishedObjectTo-
HomomorphismStructure
for IsCapCategoryMorphism, 43
InterpretMorphismFromDistinguished-
ObjectToHomomorphismStructure-
AsMorphism
for IsCapCategoryObject, IsCapCategory-
Object, IsCapCategoryMorphism, 43
Inverse
for IsCapCategoryMorphism, 40
InverseMorphismFromCoimageToImage
for IsCapCategoryMorphism, 121
InverseMorphismFromCoimageToImageWith-
GivenObjects
for IsCapCategoryObject, IsCapCategory-
Morphism, 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
IsCodomaining
for IsCapCategoryMorphism, IsCapCategory-
Morphism, 35
IsCollifting
for IsCapCategoryMorphism, IsCapCategory-
Morphism, 40
IsColiftingAlongEpimorphism
for IsCapCategoryMorphism, IsCapCategory-
Morphism, 39
IsCongruentForMorphisms
for IsCapCategoryMorphism, IsCapCategory-
Morphism, 31
IsDerivedMethod
for IsObject, 135
IsDerivedMethodGraph
for IsObject, 138
IsDominating
for IsCapCategoryMorphism, IsCapCategory-
Morphism, 35
IsEmptyHeap
for IsStringMinHeap, 143
IsEndomorphism
for IsCapCategoryMorphism, 29
IsEpimorphism
for IsCapCategoryMorphism, 24
IsEqualAsFactorobjects
for IsCapCategoryMorphism, IsCapCategory-
Morphism, 34
IsEqualAsSubobjects
for IsCapCategoryMorphism, IsCapCategory-
Morphism, 34
IsEqualForCacheForMorphisms
for IsCapCategoryMorphism, IsCapCategory-
Morphism, 41
IsEqualForCacheForObject
for IsCapCategoryObject, IsCapCategory-
Object, 16
IsEqualForMorphisms
for IsCapCategoryMorphism, IsCapCategory-
Morphism, 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
IsomorphismFromCoprodToDirectSum
for IsList, 82
IsomorphismFromCoprodToDirectSumOp
for IsList, IsCapCategoryObject, 82
IsomorphismFromDirectProductToDirectSum
for IsList, 81
IsomorphismFromDirectProductToDirectSumOp
for IsList, IsCapCategoryObject, 81
IsomorphismFromDirectSumToCoproduct
for IsList, IsCapCategoryObject, 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
IsomorphismFromInitialObjectToZeroObject
for IsCapCategory, 72
IsomorphismFromItsConstructionAsAnImageObjectToHomologyObject
for IsCapCategoryMorphism, IsCapCategoryMorphism, 126
IsomorphismFromKernelOfCokernelToImageObject
for IsCapCategoryMorphism, 117
IsomorphismFromKernelOfDiagonalDifferenceToFiberProduct
for IsList, 102
IsomorphismFromKernelOfDiagonalDifferenceToFiberProductOp
for IsList, IsCapCategoryMorphism, 103
IsomorphismFromPushoutToCoequalizerOfCoproductDiagram
for IsList, 111
IsomorphismFromPushoutToCoequalizerOfCoproductDiagramOp
for IsList, IsCapCategoryMorphism, 111
IsomorphismFromPushoutToCokernelOfDiagonalDifference
for IsList, 110
IsomorphismFromPushoutToCokernelOfDiagonalDifferenceOp
for IsList, IsCapCategoryMorphism, 110
IsomorphismFromTerminalObjectToZeroObject
for IsCapCategory, 72
IsomorphismFromZeroObjectToInitialObject
for IsCapCategory, 72
IsomorphismFromZeroObjectToTerminalObject
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
IsWellDefinedForObject
for IsCapCategoryObject, 17
IsWellDefinedForTwoCells
for IsCapCategoryTwoCell, 56
IsZero
for IsCapCategoryObject, 15
IsZeroForMorphisms
for IsCapCategoryMorphism, 32
IsZeroForObject
for IsCapCategoryObject, 15
KernelEmbedding
for IsCapCategoryMorphism, 64
KernelEmbeddingGivenKernelObject
for IsCapCategoryMorphism, IsCapCategoryObject, 64
KernelLift
for IsCapCategoryMorphism, IsCapCategoryMorphism, 64
KernelLiftGivenKernelObject
for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject, 64
KernelObject
for IsCapCategoryMorphism, 63
KernelObjectFunctorial
for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism, 66
for IsList, 66
KernelObjectFunctorialGivenKernelObject
for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryMorphism, 66
for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject, 66
for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryObject, 66

\text{LaTeXOutput}
\begin{itemize}
\item for IsCapCategory, 12
\item for IsCapCategoryCell, 11
\end{itemize}

\text{Lift}
\begin{itemize}
\item for IsCapCategoryMorphism, IsCapCategoryMorphism, 39
\item LiftAlongMonomorphism
\begin{itemize}
\item for IsCapCategoryMorphism, IsCapCategoryMorphism, 38
\end{itemize}
\end{itemize}

\text{ListCAPPrepareFunctions}, 134

\text{MakeDerivation}
\begin{itemize}
\item for IsString, IsFunction, IsDenseList, IsPosInt, IsDenseList, IsFunction, 135
\item MakeDerivationGraph
\begin{itemize}
\item for IsDenseList, 138
\end{itemize}
\item MakeOperationWeightList
\begin{itemize}
\item for IsCapCategory, IsDerivedMethodGraph, 140
\end{itemize}
\item MereExistenceOfSolutionOfLinearSystemInAbCategory
\begin{itemize}
\item for IsList, IsList, IsList, 44
\end{itemize}
\item MonomorphismIntoSomeInjectiveObject
\begin{itemize}
\item for IsCapCategoryObject, 19
\item MonomorphismIntoSomeInjectiveObjectWithGivenSomeInjectiveObject
\begin{itemize}
\item for IsCapCategoryObject, IsCapCategoryObject, 20
\end{itemize}
\end{itemize}
\item MorphismBetweenDirectSums
\begin{itemize}
\item for IsCapCategoryObject, IsList, IsCapCategoryObject, 82
\item for IsList, 82
\item MorphismBetweenDirectSumsOp
\begin{itemize}
\item for IsList, IsInt, IsInt, IsCapCategoryMorphism, 82
\end{itemize}
\end{itemize}
\item MorphismCache
\begin{itemize}
\item for IsCapFunctor, 144
\end{itemize}
\item MorphismFilter
\begin{itemize}
\item for IsCapCategory, 7
\end{itemize}
\item MorphismFromCoimageToImage
\begin{itemize}
\item for IsCapCategoryMorphism, 121
\end{itemize}
\item MorphismFromCoimageToImageWithGivenObjects
\begin{itemize}
\item for IsCapCategoryObject, IsCapCategoryMorphism, IsCapCategoryObject, 121
\end{itemize}
\item MorphismFromEqualizerToSink
\begin{itemize}
\item for IsList, 94
\item MorphismFromEqualizerToSinkOp
\begin{itemize}
\item for IsList, IsCapCategoryMorphism, 94
\end{itemize}
\item MorphismFromEqualizerToSinkWithGivenEqualizer
\begin{itemize}
\item for IsList, IsCapCategoryObject, 95
\end{itemize}
\item MorphismFromFiberProductToSink
\begin{itemize}
\item for IsList, 106
\item MorphismFromFiberProductToSinkOp
\begin{itemize}
\item for IsList, IsCapCategoryMorphism, 106
\end{itemize}
\item MorphismFromFiberProductToSinkWithGivenFiberProduct
\begin{itemize}
\item for IsList, IsCapCategoryObject, 106
\end{itemize}
\item MorphismFromKernelObjectToSink
\begin{itemize}
\item for IsCapCategoryMorphism, 64
\item MorphismFromKernelObjectToSinkWithGivenKernelObject
\begin{itemize}
\item for IsCapCategoryMorphism, IsCapCategoryObject, 64
\end{itemize}
\item MorphismFromSourceToCoequalizer
\begin{itemize}
\item for IsList, 98
\item MorphismFromSourceToCoequalizerOp
\begin{itemize}
\item for IsList, IsCapCategoryMorphism, 98
\end{itemize}
\item MorphismFromSourceToCoequalizerWithGivenCoequalizer
\begin{itemize}
\item for IsList, IsCapCategoryObject, 99
\end{itemize}
\item MorphismFromSourceToCokernelObject
\begin{itemize}
\item for IsCapCategoryMorphism, 68
\item MorphismFromSourceToCokernelObjectWithGivenCokernelObject
\begin{itemize}
\item for IsCapCategoryMorphism, IsCapCategoryObject, 68
\end{itemize}
\item MorphismFromSourceToPushout
\begin{itemize}
\item for IsList, 114
\item MorphismFromSourceToPushoutOp
\begin{itemize}
\item for IsList, IsCapCategoryMorphism, 114
\end{itemize}
\item MorphismFromSourceToPushoutWithGivenPushout
\begin{itemize}
\item for IsList, IsCapCategoryObject, 114
\end{itemize}
\item MorphismFromZeroObject
\begin{itemize}
\item for IsCapCategoryObject, 71
MorphismIntoZeroObject
   for IsCapCategoryObject, 71
MultiplyWithElementOfCommutativeRing-
   ForMorphisms
   for IsRingElement, IsCapCategoryMorphism, 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
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
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
UniversalMorphismIntoDirectProductWith-GivenDirectProduct
for IsList, IsList, IsCapCategoryObject, 91
UniversalMorphismIntoDirectSum, 79
for IsList, IsList, IsCapCategoryObject, 80
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