GeneralizedMorphismsForCAP

Implementations of generalized morphisms for the CAP project

2024.09-03

4 October 2024

Sebastian Gutsche

Sebastian Posur

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

Contents

Chapter 1

Generalized Morphism Category

Let **A** be an abelian category. We denote its generalized morphism category by $\mathbf{G}(\mathbf{A})$.

1.1 GAP Categories

1.1.1 IsGeneralizedMorphismCategory (for IsCapCategory)

▷ `IsGeneralizedMorphismCategory(`*`object`*`)` (filter)

Returns: `true` or `false`

The GAP category of the category of generalized morphisms.

1.1.2 IsGeneralizedMorphismCategoryObject (for IsCapCategoryObject)

▷ `IsGeneralizedMorphismCategoryObject(`*`object`*`)` (filter)

Returns: `true` or `false`

The GAP category of objects in the generalized morphism category.

1.1.3 IsGeneralizedMorphism (for IsCapCategoryMorphism)

▷ `IsGeneralizedMorphism(`*`object`*`)` (filter)

Returns: `true` or `false`

The GAP category of morphisms in the generalized morphism category.

1.2 Attributes

1.2.1 UnderlyingHonestObject (for IsGeneralizedMorphismCategoryObject)

▷ `UnderlyingHonestObject(a)` (attribute)

Returns: an object in **A**

The argument is an object a in the generalized morphism category. The output is its underlying honest object

1.2.2 DomainEmbedding (for IsGeneralizedMorphism)

▷ `DomainEmbedding(`*`alpha`*`)` (attribute)

Returns: a morphism in $\mathrm{Hom}_{\mathbf{A}}(d,a)$

The argument is a generalized morphism $\alpha : a \to b$. The output is its domain $d \hookrightarrow a \in \mathbf{A}$.

1.2.3 GeneralizedImageEmbedding (for IsGeneralizedMorphism)

▷ `GeneralizedImageEmbedding(`*`alpha`*`)` (attribute)

Returns: a morphism in $\mathrm{Hom}_{\mathbf{A}}(i,b)$

The argument is a generalized morphism $\alpha : a \to b$. The output is its generalized image $i \hookrightarrow b \in \mathbf{A}$.

1.2.4 DefectEmbedding (for IsGeneralizedMorphism)

▷ `DefectEmbedding(`*`alpha`*`)` (attribute)

Returns: a morphism in $\mathrm{Hom}_{\mathbf{A}}(d,b)$

The argument is a generalized morphism $\alpha : a \to b$. The output is its defect $d \hookrightarrow b \in \mathbf{A}$.

1.2.5 GeneralizedKernelEmbedding (for IsGeneralizedMorphism)

▷ `GeneralizedKernelEmbedding(`*`alpha`*`)` (attribute)

Returns: a morphism in $\mathrm{Hom}_{\mathbf{A}}(k,a)$

The argument is a generalized morphism $\alpha : a \to b$. The output is its generalized kernel $k \hookrightarrow a \in \mathbf{A}$.

1.2.6 CodomainProjection (for IsGeneralizedMorphism)

▷ `CodomainProjection(`*`alpha`*`)` (attribute)

Returns: a morphism in $\mathrm{Hom}_{\mathbf{A}}(b,c)$

The argument is a generalized morphism $\alpha : a \to b$. The output is its codomain $b \twoheadrightarrow c \in \mathbf{A}$.

1.2.7 GeneralizedCokernelProjection (for IsGeneralizedMorphism)

▷ `GeneralizedCokernelProjection(`*`alpha`*`)` (attribute)

Returns: a morphism in $\mathrm{Hom}_{\mathbf{A}}(b,c)$

The argument is a generalized morphism $\alpha : a \to b$. The output is its generalized cokernel $b \twoheadrightarrow c \in \mathbf{A}$.

1.2.8 CodefectProjection (for IsGeneralizedMorphism)

▷ `CodefectProjection(`*`alpha`*`)` (attribute)

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

The argument is a generalized morphism $\alpha : a \to b$. The output is its codefect $a \twoheadrightarrow c \in \mathbf{A}$.

1.2.9 GeneralizedCoimageProjection (for IsGeneralizedMorphism)

▷ `GeneralizedCoimageProjection(alpha)` (attribute)

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

The argument is a generalized morphism $\alpha : a \to b$. The output is its generalized coimage $a \twoheadrightarrow c \in \mathbf{A}$.

1.2.10 AssociatedMorphism (for IsGeneralizedMorphism)

▷ `AssociatedMorphism(alpha)` (attribute)

Returns: a morphism in $\mathrm{Hom}_{\mathbf{A}}(d,c)$

The argument is a generalized morphism $\alpha : a \to b$. The output is its associated morphism $d \to c \in \mathbf{A}$.

1.2.11 DomainAssociatedMorphismCodomainTriple (for IsGeneralizedMorphism)

▷ `DomainAssociatedMorphismCodomainTriple(alpha)` (attribute)

Returns: a triple of morphisms in **A**

The argument is a generalized morphism $\alpha : a \to b$. The output is a triple $(d \hookrightarrow a, d \to c, b \twoheadrightarrow c)$ consisting of its domain, associated morphism, and codomain.

1.2.12 HonestRepresentative (for IsGeneralizedMorphism)

▷ `HonestRepresentative(alpha)` (attribute)

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

The argument is a generalized morphism $\alpha : a \to b$. The output is the honest representative in **A** of α, if it exists, otherwise an error is thrown.

1.2.13 GeneralizedInverse (for IsCapCategoryMorphism)

▷ `GeneralizedInverse(alpha)` (operation)

Returns: a morphism in $\mathrm{Hom}_{\mathbf{G}(\mathbf{A})}(b,a)$

The argument is a morphism $\alpha : a \to b \in \mathbf{A}$. The output is its generalized inverse $b \to a$.

1.2.14 IdempotentDefinedBySubobject (for IsCapCategoryMorphism)

▷ `IdempotentDefinedBySubobject(alpha)` (operation)

Returns: a morphism in $\mathrm{Hom}_{\mathbf{G}(\mathbf{A})}(b,b)$

The argument is a subobject $\alpha : a \hookrightarrow b \in \mathbf{A}$. The output is the idempotent $b \to b \in \mathbf{G}(\mathbf{A})$ defined by α.

1.2.15 IdempotentDefinedByFactorobject (for IsCapCategoryMorphism)

▷ `IdempotentDefinedByFactorobject(alpha)` (operation)

Returns: a morphism in $\mathrm{Hom}_{\mathbf{G}(\mathbf{A})}(b,b)$

The argument is a factorobject $\alpha : b \twoheadrightarrow a \in \mathbf{A}$. The output is the idempotent $b \to b \in \mathbf{G}(\mathbf{A})$ defined by α.

1.2.16 UnderlyingHonestCategory (for IsCapCategory)

▷ `UnderlyingHonestCategory(C)` (attribute)

Returns: a category

The argument is a generalized morphism category $C = \mathbf{G}(\mathbf{A})$. The output is $\mathbf{A}$.

1.3 Operations

1.3.1 GeneralizedMorphismFromFactorToSubobject (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

Returns: a morphism in $\mathrm{Hom}_{\mathbf{G}(\mathbf{A})}(c,a)$

The arguments are a a factorobject $\beta : b \twoheadrightarrow c$, and a subobject $\alpha : a \hookrightarrow b$. The output is the generalized morphism from the factorobject to the subobject.

1.3.2 CommonRestriction (for IsList)

▷ `CommonRestriction(L)` (operation)

Returns: a list of generalized morphisms

The argument is a list L of generalized morphisms by three arrows having the same source. The output is a list of generalized morphisms by three arrows which is the comman restriction of L.

1.3.3 ConcatenationProduct (for IsList)

▷ `ConcatenationProduct(L)` (operation)

Returns: a generalized moprhism

The argument is a list $L = (\alpha_1, \dots, \alpha_n)$ of generalized morphisms (with same data structures). The output is their concatenation product, i.e., a generalized morphism α with UnderlyingHonestObject(Source(α)) $= \bigoplus_{i=1}^n$ UnderlyingHonestObject(Source(α_i)), and UnderlyingHonestObject(Range(α)) $= \bigoplus_{i=1}^n$ UnderlyingHonestObject(Range(α_i)), and with morphisms in the representation of α given as the direct sums of the corresponding morphisms of the α_i.

1.4 Properties

1.4.1 IsHonest (for IsGeneralizedMorphism)

▷ `IsHonest(alpha)` (property)

Returns: a boolean

The argument is a generalized morphism α. The output is `true` if α admits an honest representative, otherwise `false`.

1.4.2 HasFullDomain (for IsGeneralizedMorphism)

▷ `HasFullDomain(alpha)` (property)

Returns: a boolean

The argument is a generalized morphism α. The output is `true` if the domain of α is an isomorphism, otherwise `false`.

1.4.3 HasFullCodomain (for IsGeneralizedMorphism)

▷ `HasFullCodomain(`*`alpha`*`)` (property)

Returns: a boolean

The argument is a generalized morphism α. The output is `true` if the codomain of α is an isomorphism, otherwise `false`.

1.4.4 IsSingleValued (for IsGeneralizedMorphism)

▷ `IsSingleValued(`*`alpha`*`)` (property)

Returns: a boolean

The argument is a generalized morphism α. The output is `true` if the codomain of α is an isomorphism, otherwise `false`.

1.4.5 IsTotal (for IsGeneralizedMorphism)

▷ `IsTotal(`*`alpha`*`)` (property)

Returns: a boolean

The argument is a generalized morphism α. The output is `true` if the domain of α is an isomorphism, otherwise `false`.

1.5 Convenience methods

This section contains operations which, depending on the current generalized morphism standard of the system and the category, might point to other Operations. Please use them only as convenience and never in serious code.

1.5.1 GeneralizedMorphismCategory (for IsCapCategory)

▷ `GeneralizedMorphismCategory(`*`C`*`)` (operation)

Returns: a category

Creates a new category of generalized morphisms. Might point to GeneralizedMorphismCategoryByThreeArrows, GeneralizedMorphismCategoryByCospans, or GeneralizedMorphismCategoryBySpans

1.5.2 GeneralizedMorphismObject (for IsCapCategoryObject)

▷ `GeneralizedMorphismObject(`*`A`*`)` (operation)

Returns: an object in the generalized morphism category

Creates an object in the current generalized morphism category, depending on the standard

1.5.3 AsGeneralizedMorphism (for IsCapCategoryMorphism)

▷ `AsGeneralizedMorphism(`*`phi`*`)` (operation)

Returns: a generalized morphism

Returns the corresponding morphism to phi in the current generalized morphism category.

1.5.4 GeneralizedMorphism (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `GeneralizedMorphism(`*`phi, psi`*`)` (operation)

Returns: a generalized morphism

Returns the corresponding morphism to phi and psi in the current generalized morphism category.

1.5.5 GeneralizedMorphism (for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `GeneralizedMorphism(`*`iota, phi, pi`*`)` (operation)

Returns: a generalized morphism

Returns the corresponding morphism to iota, phi and psi in the current generalized morphism category.

1.5.6 GeneralizedMorphismWithRangeAid (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `GeneralizedMorphismWithRangeAid(`*`arg1, arg2`*`)` (operation)

Returns a generalized morphism with range aid by three arrows or by span, or a generalized morphism by cospan, depending on the standard.

1.5.7 GeneralizedMorphismWithSourceAid (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `GeneralizedMorphismWithSourceAid(`*`arg1, arg2`*`)` (operation)

Returns a generalized morphism with source aid by three arrows or by cospan, or a generalized morphism by span, depending on the standard.

Chapter 2

Generalized Morphism Category by Cospans

2.1 GAP Categories

2.1.1 IsGeneralizedMorphismCategoryByCospans (for IsGeneralizedMorphismCategory)

▷ `IsGeneralizedMorphismCategoryByCospans(`*`object`*`)` (filter)

Returns: `true` or `false`

The GAP category of the category of generalized morphisms by cospans.

2.1.2 IsGeneralizedMorphismCategoryByCospansObject (for IsGeneralizedMorphismCategoryObject)

▷ `IsGeneralizedMorphismCategoryByCospansObject(`*`object`*`)` (filter)

Returns: `true` or `false`

The GAP category of objects in the generalized morphism category by cospans.

2.1.3 IsGeneralizedMorphismByCospan (for IsGeneralizedMorphism)

▷ `IsGeneralizedMorphismByCospan(`*`object`*`)` (filter)

Returns: `true` or `false`

The GAP category of morphisms in the generalized morphism category by cospans.

2.2 Properties

2.2.1 HasIdentityAsReversedArrow (for IsGeneralizedMorphismByCospan)

▷ `HasIdentityAsReversedArrow(`*`alpha`*`)` (property)

Returns: `true` or `false`

The argument is a generalized morphism α by a cospan $a \rightarrow b \leftarrow c$. The output is `true` if $b \leftarrow c$ is congruent to an identity morphism, `false` otherwise.

2.3 Attributes

2.3.1 UnderlyingHonestObject (for IsGeneralizedMorphismCategoryByCospansObject)

▷ `UnderlyingHonestObject(a)` (attribute)

Returns: an object in $\mathbf{A}$

The argument is an object a in the generalized morphism category by cospans. The output is its underlying honest object.

2.3.2 Arrow (for IsGeneralizedMorphismByCospan)

▷ `Arrow(alpha)` (attribute)

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

The argument is a generalized morphism α by a cospan $a \rightarrow b \leftarrow c$. The output is its arrow $a \rightarrow b$.

2.3.3 ReversedArrow (for IsGeneralizedMorphismByCospan)

▷ `ReversedArrow(alpha)` (attribute)

Returns: a morphism in $\mathrm{Hom}_{\mathbf{A}}(c,b)$

The argument is a generalized morphism α by a cospan $a \rightarrow b \leftarrow c$. The output is its reversed arrow $b \leftarrow c$.

2.3.4 NormalizedCospanTuple (for IsGeneralizedMorphismByCospan)

▷ `NormalizedCospanTuple(alpha)` (attribute)

Returns: a pair of morphisms in $\mathbf{A}$.

The argument is a generalized morphism $\alpha : a \rightarrow b$ by a cospan. The output is its normalized cospan pair $(a \rightarrow d, d \leftarrow b)$.

2.3.5 PseudoInverse (for IsGeneralizedMorphismByCospan)

▷ `PseudoInverse(alpha)` (attribute)

Returns: a morphism in $\mathrm{Hom}_{\mathbf{G(A)}}(b,a)$

The argument is a generalized morphism $\alpha : a \rightarrow b$ by a cospan. The output is its pseudo inverse $b \rightarrow a$.

2.3.6 GeneralizedInverseByCospan (for IsCapCategoryMorphism)

▷ `GeneralizedInverseByCospan(alpha)` (attribute)

Returns: a morphism in $\mathrm{Hom}_{\mathbf{G(A)}}(b,a)$

The argument is a morphism $\alpha : a \rightarrow b \in \mathbf{A}$. The output is its generalized inverse $b \rightarrow a$ by cospan.

2.3.7 IdempotentDefinedBySubobjectByCospan (for IsCapCategoryMorphism)

▷ `IdempotentDefinedBySubobjectByCospan(alpha)` (attribute)

Returns: a morphism in $\mathrm{Hom}_{\mathbf{G(A)}}(b,b)$

The argument is a subobject $\alpha : a \hookrightarrow b \in \mathbf{A}$. The output is the idempotent $b \rightarrow b \in \mathbf{G}(\mathbf{A})$ by cospan defined by α.

2.3.8 IdempotentDefinedByFactorobjectByCospan (for IsCapCategoryMorphism)

▷ `IdempotentDefinedByFactorobjectByCospan(alpha)` (attribute)

Returns: a morphism in $\mathrm{Hom}_{\mathbf{G}(\mathbf{A})}(b,b)$

The argument is a factorobject $\alpha : b \twoheadrightarrow a \in \mathbf{A}$. The output is the idempotent $b \to b \in \mathbf{G}(\mathbf{A})$ by cospan defined by α.

2.3.9 NormalizedCospan (for IsGeneralizedMorphismByCospan)

▷ `NormalizedCospan(alpha)` (attribute)

Returns: a morphism in $\mathrm{Hom}_{\mathbf{G}(\mathbf{A})}(a,b)$

The argument is a generalized morphism $\alpha : a \to b$ by a cospan. The output is its normalization by cospan.

2.4 Operations

2.4.1 GeneralizedMorphismFromFactorToSubobjectByCospan (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

Returns: a morphism in $\mathrm{Hom}_{\mathbf{G}(\mathbf{A})}(c,a)$

The arguments are a a factorobject $\beta : b \twoheadrightarrow c$, and a subobject $\alpha : a \hookrightarrow b$. The output is the generalized morphism by cospan from the factorobject to the subobject.

2.5 Constructors

2.5.1 GeneralizedMorphismByCospan (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

Returns: a morphism in $\mathrm{Hom}_{\mathbf{G}(\mathbf{A})}(a,c)$

The arguments are morphisms $\alpha : a \to b$ and $\beta : c \to b$ in $\mathbf{A}$. The output is a generalized morphism by cospan with arrow α and reversed arrow β.

2.5.2 GeneralizedMorphismByCospan (for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism)

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

Returns: a morphism in $\mathrm{Hom}_{\mathbf{G}(\mathbf{A})}(a,d)$

The arguments are morphisms $\alpha : a \leftarrow b$, $\beta : b \to c$, and $\gamma : c \leftarrow d$ in $\mathbf{A}$. The output is a generalized morphism by cospan defined by the composition of the given three arrows regarded as generalized morphisms.

2.5.3 GeneralizedMorphismByCospanWithSourceAid (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

Returns: a morphism in $\mathrm{Hom}_{\mathbf{G}(\mathbf{A})}(a,c)$

The arguments are morphisms $\alpha : a \leftarrow b$, and $\beta : b \rightarrow c$ in $\mathbf{A}$. The output is a generalized morphism by cospan defined by the composition of the given two arrows regarded as generalized morphisms.

2.5.4 AsGeneralizedMorphismByCospan (for IsCapCategoryMorphism)

▷ `AsGeneralizedMorphismByCospan(`*`alpha`*`)` (attribute)

Returns: a morphism in $\mathrm{Hom}_{\mathbf{G(A)}}(a,b)$

The argument is a morphism $\alpha : a \rightarrow b$ in $\mathbf{A}$. The output is the honest generalized morphism by cospan defined by α.

2.5.5 GeneralizedMorphismCategoryByCospans (for IsCapCategory)

▷ `GeneralizedMorphismCategoryByCospans(`*`A`*`)` (attribute)

Returns: a category

The argument is an abelian category $\mathbf{A}$. The output is its generalized morphism category $\mathbf{G}(\mathbf{A})$ by cospans.

2.5.6 GeneralizedMorphismByCospansObject (for IsCapCategoryObject)

▷ `GeneralizedMorphismByCospansObject(a)` (attribute)

Returns: an object in $\mathbf{G}(\mathbf{A})$

The argument is an object a in an abelian category $\mathbf{A}$. The output is the object in the generalized morphism category by cospans whose underlying honest object is a.

2.6 Constructors of lifts of exact functors and natrual (iso)morphisms

2.6.1 AsGeneralizedMorphismByCospan (for IsCapFunctor, IsString)

▷ `AsGeneralizedMorphismByCospan(`*`F, name`*`)` (operation)

Lift the *exact* functor F to a functor $A \rightarrow B$, where $A :=$ `GeneralizedMorphismCategoryByCospans( AsCapCategory( Source( F ) ) )` and $B :=$ `GeneralizedMorphismCategoryByCospans( AsCapCategory( Range( F ) ) )`.

Chapter 3

Generalized Morphism Category by Spans

3.1 GAP Categories

3.1.1 IsGeneralizedMorphismCategoryBySpans (for IsGeneralizedMorphismCategory)

▷ `IsGeneralizedMorphismCategoryBySpans(`*`object`*`)` (filter)

Returns: `true` or `false`

The GAP category of the category of generalized morphisms by spans.

3.1.2 IsGeneralizedMorphismCategoryBySpansObject (for IsGeneralizedMorphismCategoryObject)

▷ `IsGeneralizedMorphismCategoryBySpansObject(`*`object`*`)` (filter)

Returns: `true` or `false`

The GAP category of objects in the generalized morphism category by spans.

3.1.3 IsGeneralizedMorphismBySpan (for IsGeneralizedMorphism)

▷ `IsGeneralizedMorphismBySpan(`*`object`*`)` (filter)

Returns: `true` or `false`

The GAP category of morphisms in the generalized morphism category by spans.

3.2 Properties

3.2.1 HasIdentityAsReversedArrow (for IsGeneralizedMorphismBySpan)

▷ `HasIdentityAsReversedArrow(`*`alpha`*`)` (property)

Returns: `true` or `false`

The argument is a generalized morphism α by a span $a \leftarrow b \rightarrow c$. The output is `true` if $a \leftarrow b$ is congruent to an identity morphism, `false` otherwise.

3.3 Attributes

3.3.1 UnderlyingHonestObject (for IsGeneralizedMorphismCategoryBySpansObject)

▷ `UnderlyingHonestObject(a)` (attribute)

Returns: an object in $\mathbf{A}$

The argument is an object a in the generalized morphism category by spans. The output is its underlying honest object.

3.3.2 Arrow (for IsGeneralizedMorphismBySpan)

▷ `Arrow(alpha)` (attribute)

Returns: a morphism in $\mathrm{Hom}_{\mathbf{A}}(b,c)$

The argument is a generalized morphism α by a span $a \leftarrow b \rightarrow c$. The output is its arrow $b \rightarrow c$.

3.3.3 ReversedArrow (for IsGeneralizedMorphismBySpan)

▷ `ReversedArrow(alpha)` (attribute)

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

The argument is a generalized morphism α by a span $a \leftarrow b \rightarrow c$. The output is its reversed arrow $a \leftarrow b$.

3.3.4 NormalizedSpanTuple (for IsGeneralizedMorphismBySpan)

▷ `NormalizedSpanTuple(alpha)` (attribute)

Returns: a pair of morphisms in $\mathbf{A}$.

The argument is a generalized morphism $\alpha : a \rightarrow b$ by a span. The output is its normalized span pair $(a \leftarrow d, d \rightarrow b)$.

3.3.5 PseudoInverse (for IsGeneralizedMorphismBySpan)

▷ `PseudoInverse(alpha)` (attribute)

Returns: a morphism in $\mathrm{Hom}_{\mathbf{G(A)}}(b,a)$

The argument is a generalized morphism $\alpha : a \rightarrow b$ by a span. The output is its pseudo inverse $b \rightarrow a$.

3.3.6 GeneralizedInverseBySpan (for IsCapCategoryMorphism)

▷ `GeneralizedInverseBySpan(alpha)` (attribute)

Returns: a morphism in $\mathrm{Hom}_{\mathbf{G(A)}}(b,a)$

The argument is a morphism $\alpha : a \rightarrow b \in \mathbf{A}$. The output is its generalized inverse $b \rightarrow a$ by span.

3.3.7 IdempotentDefinedBySubobjectBySpan (for IsCapCategoryMorphism)

▷ `IdempotentDefinedBySubobjectBySpan(alpha)` (attribute)

Returns: a morphism in $\mathrm{Hom}_{\mathbf{G(A)}}(b,b)$

The argument is a subobject $\alpha : a \hookrightarrow b \in \mathbf{A}$. The output is the idempotent $b \rightarrow b \in \mathbf{G}(\mathbf{A})$ by span defined by α.

3.3.8 IdempotentDefinedByFactorobjectBySpan (for IsCapCategoryMorphism)

▷ `IdempotentDefinedByFactorobjectBySpan(alpha)` (attribute)

Returns: a morphism in $\mathrm{Hom}_{\mathbf{G}(\mathbf{A})}(b,b)$

The argument is a factorobject $\alpha : b \twoheadrightarrow a \in \mathbf{A}$. The output is the idempotent $b \rightarrow b \in \mathbf{G}(\mathbf{A})$ by span defined by α.

3.3.9 NormalizedSpan (for IsGeneralizedMorphismBySpan)

▷ `NormalizedSpan(alpha)` (attribute)

Returns: a morphism in $\mathrm{Hom}_{\mathbf{G}(\mathbf{A})}(a,b)$

The argument is a generalized morphism $\alpha : a \rightarrow b$ by a span. The output is its normalization by span.

3.4 Operations

3.4.1 GeneralizedMorphismFromFactorToSubobjectBySpan (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

Returns: a morphism in $\mathrm{Hom}_{\mathbf{G}(\mathbf{A})}(c,a)$

The arguments are a a factorobject $\beta : b \twoheadrightarrow c$, and a subobject $\alpha : a \hookrightarrow b$. The output is the generalized morphism by span from the factorobject to the subobject.

3.5 Constructors

3.5.1 GeneralizedMorphismBySpan (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

Returns: a morphism in $\mathrm{Hom}_{\mathbf{G}(\mathbf{A})}(a,b)$

The arguments are morphisms $\alpha : a \leftarrow c$ and $\beta : c \rightarrow b$ in $\mathbf{A}$. The output is a generalized morphism by span with arrow β and reversed arrow α.

3.5.2 GeneralizedMorphismBySpan (for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism)

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

Returns: a morphism in $\mathrm{Hom}_{\mathbf{G}(\mathbf{A})}(a,d)$

The arguments are morphisms $\alpha : a \leftarrow b$, $\beta : b \rightarrow c$, and $\gamma : c \leftarrow d$ in $\mathbf{A}$. The output is a generalized morphism by span defined by the composition of the given three arrows regarded as generalized morphisms.

3.5.3 GeneralizedMorphismBySpanWithRangeAid (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

Returns: a morphism in $\mathrm{Hom}_{\mathbf{G}(\mathbf{A})}(a,c)$

The arguments are morphisms $\alpha : a \to b$, and $\beta : b \leftarrow c$ in $\mathbf{A}$. The output is a generalized morphism by span defined by the composition of the given two arrows regarded as generalized morphisms.

3.5.4 AsGeneralizedMorphismBySpan (for IsCapCategoryMorphism)

▷ `AsGeneralizedMorphismBySpan(alpha)` (attribute)

Returns: a morphism in $\mathrm{Hom}_{\mathbf{G}(\mathbf{A})}(a,b)$

The argument is a morphism $\alpha : a \to b$ in $\mathbf{A}$. The output is the honest generalized morphism by span defined by α.

3.5.5 GeneralizedMorphismCategoryBySpans (for IsCapCategory)

▷ `GeneralizedMorphismCategoryBySpans(A)` (attribute)

Returns: a category

The argument is an abelian category $\mathbf{A}$. The output is its generalized morphism category $\mathbf{G}(\mathbf{A})$ by spans.

3.5.6 GeneralizedMorphismBySpansObject (for IsCapCategoryObject)

▷ `GeneralizedMorphismBySpansObject(a)` (attribute)

Returns: an object in $\mathbf{G}(\mathbf{A})$

The argument is an object a in an abelian category $\mathbf{A}$. The output is the object in the generalized morphism category by spans whose underlying honest object is a.

Chapter 4

Generalized Morphism Category by Three Arrows

4.1 GAP Categories

4.1.1 IsGeneralizedMorphismCategoryByThreeArrows (for IsGeneralizedMorphismCategory)

▷ `IsGeneralizedMorphismCategoryByThreeArrows(`*`object`*`)` (filter)

Returns: `true` or `false`

The GAP category of the category of generalized morphisms by three arrows.

4.1.2 IsGeneralizedMorphismCategoryByThreeArrowsObject (for IsGeneralizedMorphismCategoryObject)

▷ `IsGeneralizedMorphismCategoryByThreeArrowsObject(`*`object`*`)` (filter)

Returns: `true` or `false`

The GAP category of objects in the generalized morphism category by three arrows.

4.1.3 IsGeneralizedMorphismByThreeArrows (for IsGeneralizedMorphism)

▷ `IsGeneralizedMorphismByThreeArrows(`*`object`*`)` (filter)

Returns: `true` or `false`

The GAP category of morphisms in the generalized morphism category by three arrows.

4.2 Properties

4.2.1 HasIdentitiesAsReversedArrows (for IsGeneralizedMorphismByThreeArrows)

▷ `HasIdentitiesAsReversedArrows(`*`alpha`*`)` (property)

Returns: `true` or `false`

The argument is a generalized morphism α by three arrows $a \leftarrow b \rightarrow c \leftarrow d$. The output is `true` if $a \leftarrow b$ and $c \leftarrow d$ are congruent to identity morphisms, `false` otherwise.

4.2.2 HasIdentityAsSourceAid (for IsGeneralizedMorphismByThreeArrows)

▷ `HasIdentityAsSourceAid(alpha)` (property)

Returns: `true` or `false`

The argument is a generalized morphism α by three arrows $a \leftarrow b \rightarrow c \leftarrow d$. The output is `true` if $a \leftarrow b$ is congruent to an identity morphism, `false` otherwise.

4.2.3 HasIdentityAsRangeAid (for IsGeneralizedMorphismByThreeArrows)

▷ `HasIdentityAsRangeAid(alpha)` (property)

Returns: `true` or `false`

The argument is a generalized morphism α by three arrows $a \leftarrow b \rightarrow c \leftarrow d$. The output is `true` if $c \leftarrow d$ is congruent to an identity morphism, `false` otherwise.

4.3 Attributes

4.3.1 UnderlyingHonestObject (for IsGeneralizedMorphismCategoryByThreeArrowsObject)

▷ `UnderlyingHonestObject(a)` (attribute)

Returns: an object in **A**

The argument is an object a in the generalized morphism category by three arrows. The output is its underlying honest object.

4.3.2 SourceAid (for IsGeneralizedMorphismByThreeArrows)

▷ `SourceAid(alpha)` (attribute)

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

The argument is a generalized morphism α by three arrows $a \leftarrow b \rightarrow c \leftarrow d$. The output is its source aid $a \leftarrow b$.

4.3.3 RangeAid (for IsGeneralizedMorphismByThreeArrows)

▷ `RangeAid(alpha)` (attribute)

Returns: a morphism in $\mathrm{Hom}_{\mathbf{A}}(d,c)$

The argument is a generalized morphism α by three arrows $a \leftarrow b \rightarrow c \leftarrow d$. The output is its range aid $c \leftarrow d$.

4.3.4 Arrow (for IsGeneralizedMorphismByThreeArrows)

▷ `Arrow(alpha)` (attribute)

Returns: a morphism in $\mathrm{Hom}_{\mathbf{A}}(b,c)$

The argument is a generalized morphism α by three arrows $a \leftarrow b \rightarrow c \leftarrow d$. The output is its range aid $b \rightarrow c$.

4.3.5 PseudoInverse (for IsGeneralizedMorphismByThreeArrows)

▷ `PseudoInverse(alpha)` (attribute)

Returns: a morphism in $\mathrm{Hom}_{\mathbf{G}(\mathbf{A})}(b,a)$

The argument is a generalized morphism $\alpha : a \to b$ by three arrows. The output is its pseudo inverse $b \to a$.

4.3.6 GeneralizedInverseByThreeArrows (for IsCapCategoryMorphism)

▷ `GeneralizedInverseByThreeArrows(alpha)` (attribute)

Returns: a morphism in $\mathrm{Hom}_{\mathbf{G}(\mathbf{A})}(b,a)$

The argument is a morphism $\alpha : a \to b \in \mathbf{A}$. The output is its generalized inverse $b \to a$ by three arrows.

4.3.7 IdempotentDefinedBySubobjectByThreeArrows (for IsCapCategoryMorphism)

▷ `IdempotentDefinedBySubobjectByThreeArrows(alpha)` (attribute)

Returns: a morphism in $\mathrm{Hom}_{\mathbf{G}(\mathbf{A})}(b,b)$

The argument is a subobject $\alpha : a \hookrightarrow b \in \mathbf{A}$. The output is the idempotent $b \to b \in \mathbf{G}(\mathbf{A})$ by three arrows defined by α.

4.3.8 IdempotentDefinedByFactorobjectByThreeArrows (for IsCapCategoryMorphism)

▷ `IdempotentDefinedByFactorobjectByThreeArrows(alpha)` (attribute)

Returns: a morphism in $\mathrm{Hom}_{\mathbf{G}(\mathbf{A})}(b,b)$

The argument is a factorobject $\alpha : b \twoheadrightarrow a \in \mathbf{A}$. The output is the idempotent $b \to b \in \mathbf{G}(\mathbf{A})$ by three arrows defined by α.

4.4 Operations

4.4.1 GeneralizedMorphismFromFactorToSubobjectByThreeArrows (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

Returns: a morphism in $\mathrm{Hom}_{\mathbf{G}(\mathbf{A})}(c,a)$

The arguments are a a factorobject $\beta : b \twoheadrightarrow c$, and a subobject $\alpha : a \hookrightarrow b$. The output is the generalized morphism by three arrows from the factorobject to the subobject.

4.4.2 CommonCoastriction (for IsList)

▷ `CommonCoastriction(L)` (operation)

Returns: a list of generalized morphisms

The argument is a list L of generalized morphisms by three arrows having the same range. The output is a list of generalized morphisms by three arrows which is the comman coastriction of L.

4.5 Constructors

4.5.1 GeneralizedMorphismByThreeArrows (for IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism)

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

Returns: a morphism in $\mathrm{Hom}_{\mathbf{G(A)}}(a,d)$

The arguments are morphisms $\alpha : a \leftarrow b$, $\beta : b \rightarrow c$, and $\gamma : c \leftarrow d$ in $\mathbf{A}$. The output is a generalized morphism by three arrows with source aid α, arrow β, and range aid γ.

4.5.2 GeneralizedMorphismByThreeArrowsWithSourceAid (for IsCapCategoryMorphism, IsCapCategoryMorphism)

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

Returns: a morphism in $\mathrm{Hom}_{\mathbf{G(A)}}(a,c)$

The arguments are morphisms $\alpha : a \leftarrow b$, and $\beta : b \rightarrow c$ in $\mathbf{A}$. The output is a generalized morphism by three arrows defined by the composition of the given two arrows regarded as generalized morphisms.

4.5.3 GeneralizedMorphismByThreeArrowsWithRangeAid (for IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `GeneralizedMorphismByThreeArrowsWithRangeAid(beta, gamma)` (operation)

Returns: a morphism in $\mathrm{Hom}_{\mathbf{G(A)}}(b,d)$

The arguments are morphisms $\beta : b \rightarrow c$, and $\gamma : c \leftarrow d$ in $\mathbf{A}$. The output is a generalized morphism by three arrows defined by the composition of the given two arrows regarded as generalized morphisms.

4.5.4 AsGeneralizedMorphismByThreeArrows (for IsCapCategoryMorphism)

▷ `AsGeneralizedMorphismByThreeArrows(alpha)` (attribute)

Returns: a morphism in $\mathrm{Hom}_{\mathbf{G(A)}}(a,b)$

The argument is a morphism $\alpha : a \rightarrow b$ in $\mathbf{A}$. The output is the honest generalized morphism by three arrows defined by α.

4.5.5 GeneralizedMorphismCategoryByThreeArrows (for IsCapCategory)

▷ `GeneralizedMorphismCategoryByThreeArrows(A)` (attribute)

Returns: a category

The argument is an abelian category $\mathbf{A}$. The output is its generalized morphism category $\mathbf{G}(\mathbf{A})$ by three arrows.

4.5.6 GeneralizedMorphismByThreeArrowsObject (for IsCapCategoryObject)

▷ `GeneralizedMorphismByThreeArrowsObject(a)` (attribute)

Returns: an object in $\mathbf{G}(\mathbf{A})$

The argument is an object a in an abelian category $\mathbf{A}$. The output is the object in the generalized morphism category by three arrows whose underlying honest object is a.

Chapter 5

Conversion functors for generalized morphisms

5.1 Functors from Cospans

5.1.1 FunctorFromCospansToThreeArrows (for IsCapCategory)

▷ `FunctorFromCospansToThreeArrows(C)` (attribute)

Returns: a functor

For the given category *C*, this attribute is the functor from the cospan generalized morphism category of *C* to the generalized morphism category modeled by three arrows.

5.1.2 FunctorFromCospansToSpans (for IsCapCategory)

▷ `FunctorFromCospansToSpans(C)` (attribute)

Returns: a functor

For the given category *C*, this attribute is the functor from the cospan generalized morphism category of *C* to the generalized morphism category modeled by spans.

5.2 Functors from Spans

5.2.1 FunctorFromSpansToThreeArrows (for IsCapCategory)

▷ `FunctorFromSpansToThreeArrows(C)` (attribute)

Returns: a functor

For the given category *C*, this attribute is the functor from the span generalized morphism category of *C* to the generalized morphism category modeled by three arrows.

5.2.2 FunctorFromSpansToCospans (for IsCapCategory)

▷ `FunctorFromSpansToCospans(C)` (attribute)

Returns: a functor

For the given category *C*, this attribute is the functor from the span generalized morphism category of *C* to the generalized morphism category modeled by cospans.

5.3 Functors from Three Arrows

5.3.1 FunctorFromThreeArrowsToCospans (for IsCapCategory)

▷ `FunctorFromThreeArrowsToCospans(C)` (attribute)

Returns: a functor

For the given category `C`, this attribute is the functor from the three arrow generalized morphism category of `C` to the generalized morphism category modeled by cospans.

5.3.2 FunctorFromThreeArrowsToSpans (for IsCapCategory)

▷ `FunctorFromThreeArrowsToSpans(C)` (attribute)

Returns: a functor

For the given category `C`, this attribute is the functor from the three arrow generalized morphism category of `C` to the generalized morphism category modeled by spans.

Chapter 6

Serre Quotients Subcategory

This is an implementation of a convenience object for Serre quotients. The Subcategory implemented in this file is not a CAP category at all, but only a handler for the serre quotient construction. It does not contain objects or anything useful, it is just a wrapper for the function the Serre quotient is based upon.

6.1 implementation

6.1.1 FullSubcategoryByMembershipFunction (for IsCapCategory, IsFunction)

▷ `FullSubcategoryByMembershipFunction(`*C, func*`)` (operation)

Returns: a function handler

Creates an object which handles the function to create a Serre quotient category. It can be used instead of the function for Serre quotients. Note that the result is *NOT A CATEGORY* and can not be seen as one.

Chapter 7

Serre Quotients

Serre quotients are implemented using generalized morphisms. A Serre quotient category is the quotient of an abelian category A by a thick subcategory C. The objects of the quotient are the objects from A, the morphisms are a limit construction. In the implementation those morphisms are modeled by generalized morphisms, and therefore there are, like in the generalized morphism case, three types of Serre quotients.

7.1 General operations

As in the generalized morphism case, the generic constructors depend on the generalized morphism standard. Please note that for implementations the specialized constructors should be used.

7.1.1 IsSerreQuotientCategoryObject (for IsCapCategoryObject)

▷ `IsSerreQuotientCategoryObject(`*arg*`)` (filter)

Returns: `true` or `false`

The category of objects in the category of Serre quotients. For actual objects this needs to be specialized.

7.1.2 IsSerreQuotientCategoryMorphism (for IsCapCategoryMorphism)

▷ `IsSerreQuotientCategoryMorphism(`*arg*`)` (filter)

Returns: `true` or `false`

The category of morphisms in the category of Serre quotients. For actual morphisms this needs to be specialized.

7.1.3 SerreQuotientCategory (for IsCapCategory, IsFunction, IsString)

▷ `SerreQuotientCategory(`*A, func[, name]*`)` (operation)

Returns: a CAP category

Creates a Serre quotient category `S` with name `name` out of an Abelian category `A`. If `name` is not given, a generic name is constructed out of the name of `A`. The argument `func` must be a unary function on the objects of `A` deciding the membership in the thick subcategory C mentioned above.

7.1.4 AsSerreQuotientCategoryObject (for IsCapCategory, IsCapCategoryObject)

▷ `AsSerreQuotientCategoryObject(A/C, M)` (operation)

Returns: an object

Given a Serre quotient category `A/C` and an object `M` in `A`, this constructor returns the corresponding object in the Serre quotient category.

7.1.5 SerreQuotientCategoryMorphism (for IsCapCategory, IsGeneralizedMorphism)

▷ `SerreQuotientCategoryMorphism(A/C, phi)` (operation)

Returns: a morphism

Given a Serre quotient category `A/C` and a generalized morphism `phi` in the generalized morphism category `A/C` is modeled upon, this constructor returns the corresponding morphism in the Serre quotient category.

7.1.6 SerreQuotientCategoryMorphism (for IsCapCategory, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `SerreQuotientCategoryMorphism(A/C, iota, phi, pi)` (operation)

Returns: a morphism

Given a Serre quotient category `A/C` and three morphisms $\iota : M' \rightarrow M$, $\phi : M' \rightarrow N'$ and $\pi : N \rightarrow N'$ this operation contructs a morphism in the Serre quotient category.

7.1.7 SerreQuotientCategoryMorphism (for IsCapCategory, IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `SerreQuotientCategoryMorphism(A/C, alpha, beta)` (operation)

Returns: a morphism

Given a Serre quotient category `A/C` and two morphisms of the form $\alpha : X \rightarrow M$ and $\beta : X \rightarrow N$ or $\alpha : M \rightarrow X$ and $\beta : N \rightarrow X$, this operation constructs the corresponding morphism in the Serre quotient category. This operation is only implemented if `A/C` is modeled upon a span generalized morphism category in the first option or upon a cospan category in the second.

7.1.8 SerreQuotientCategoryMorphismWithSourceAid (for IsCapCategory, IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `SerreQuotientCategoryMorphismWithSourceAid(A/C, alpha, beta)` (operation)

Returns: a morphism

Given a Serre quotient category `A/C` and two morphisms $\alpha : M \rightarrow X$ and $\beta : X \rightarrow N$ this operation constructs the corresponding morphism in the Serre quotient category.

7.1.9 SerreQuotientCategoryMorphismWithRangeAid (for IsCapCategory, IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `SerreQuotientCategoryMorphismWithRangeAid(A/C, alpha, beta)` (operation)

Returns: a morphism

Given a Serre quotient category `A/C` and two morphisms $\alpha : X \to M$ and $\beta : X \to N$ this operation constructs the corresponding morphism in the Serre quotient category.

7.1.10 AsSerreQuotientCategoryMorphism (for IsCapCategory, IsCapCategoryMorphism)

▷ `AsSerreQuotientCategoryMorphism(A/C, phi)` (operation)

Returns: a morphism

Given a Serre quotient category `A/C` and a morphism `phi` in `A`, this constructor returns the corresponding morphism in the Serre quotient category.

7.1.11 SubcategoryMembershipTestFunctionForSerreQuotient (for IsCapCategory)

▷ `SubcategoryMembershipTestFunctionForSerreQuotient(C)` (attribute)

Returns: a function

When a Serre quotient category is created, a membership function for the subcategory is given. This attribute stores and returns this function

7.1.12 UnderlyingHonestCategory (for IsCapCategory)

▷ `UnderlyingHonestCategory(A/C)` (attribute)

Returns: a category

For a Serre quotient category `A/C` this attribute returns the category `A`.

7.1.13 UnderlyingGeneralizedMorphismCategory (for IsCapCategory)

▷ `UnderlyingGeneralizedMorphismCategory(A/C)` (attribute)

Returns: a category

For a Serre quotient category `A/C` this attribute returns generalized morphism category the quotient is modelled upon.

7.1.14 UnderlyingGeneralizedObject (for IsSerreQuotientCategoryObject)

▷ `UnderlyingGeneralizedObject(M)` (attribute)

Returns: an object

For an object `M` in the Serre quotient category A/C this attribute returns the corresponding object in the generalized morphism category the quotient is modelled upon.

7.1.15 UnderlyingHonestObject (for IsSerreQuotientCategoryObject)

▷ `UnderlyingHonestObject(M)` (attribute)

Returns: an object

For an object `M` in the Serre quotient category A/C this attribute returns the corresponding object in `A`.

7.1.16 UnderlyingGeneralizedMorphism (for IsSerreQuotientCategoryMorphism)

▷ `UnderlyingGeneralizedMorphism(phi)` (attribute)

Returns: a morphism

For a morphism `phi` in the Serre quotient category A/C this attribute returns the corresponding generalized morphism in the generalized morphism category the quotient is modelled upon.

7.1.17 CanonicalProjection (for IsCapCategory)

▷ `CanonicalProjection(A/C)` (attribute)

Returns: a functor

Given a Serre quotient category `A/C`, this operation returns the canonical projection functor $A \rightarrow A/C$.

7.2 Serre quotients by cospans

7.2.1 SerreQuotientCategoryByCospans (for IsCapCategory, IsFunction, IsString)

▷ `SerreQuotientCategoryByCospans(A, func[, name])` (operation)

Returns: a CAP category

Creates a Serre quotient category S with name `name` out of an Abelian category `A`. The Serre quotient category will be modeled upon the generalized morphisms by cospans category of `A` If `name` is not given, a generic name is constructed out of the name of `A`. The argument `func` must be a unary function on the objects of `A` deciding the membership in the thick subcategory C mentioned above.

7.2.2 AsSerreQuotientCategoryByCospansObject (for IsCapCategory, IsCapCategoryObject)

▷ `AsSerreQuotientCategoryByCospansObject(A/C, M)` (operation)

Returns: an object

Given a Serre quotient category `A/C` modeled by cospans and an object `M` in `A`, this constructor returns the corresponding object in the Serre quotient category.

7.2.3 SerreQuotientCategoryByCospansMorphism (for IsCapCategory, IsGeneralizedMorphismByCospan)

▷ `SerreQuotientCategoryByCospansMorphism(A/C, phi)` (operation)

Returns: a morphism

Given a Serre quotient category `A/C` modeled by cospans and a generalized morphism `phi` in the generalized morphism category `A/C` is modeled upon, this constructor returns the corresponding morphism in the Serre quotient category.

7.2.4 SerreQuotientCategoryByCospansMorphism (for IsCapCategory, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `SerreQuotientCategoryByCospansMorphism(A/C, iota, phi, pi)` (operation)

Returns: a morphism

Given a Serre quotient category `A/C` modeled by cospans and three morphisms $\iota : M' \rightarrow M$, $\phi : M' \rightarrow N'$ and $\pi : N \rightarrow N'$ this operation contructs a morphism in the Serre quotient category.

7.2.5 SerreQuotientCategoryByCospansMorphismWithSourceAid (for IsCapCategory, IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `SerreQuotientCategoryByCospansMorphismWithSourceAid(A/C, alpha, beta)` (operation)

Returns: a morphism

Given a Serre quotient category `A/C` modeled by cospans and two morphisms $\alpha : M \rightarrow X$ and $\beta : X \rightarrow N$ this operation constructs the corresponding morphism in the Serre quotient category.

7.2.6 SerreQuotientCategoryByCospansMorphism (for IsCapCategory, IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `SerreQuotientCategoryByCospansMorphism(A/C, alpha, beta)` (operation)

Returns: a morphism

Given a Serre quotient category `A/C` modeled by cospans and two morphisms $\alpha : X \rightarrow M$ and $\beta : X \rightarrow N$ this operation constructs the corresponding morphism in the Serre quotient category.

7.2.7 AsSerreQuotientCategoryByCospansMorphism (for IsCapCategory, IsCapCategoryMorphism)

▷ `AsSerreQuotientCategoryByCospansMorphism(A/C, phi)` (operation)

Returns: a morphism

Given a Serre quotient category `A/C` modeled by cospans and a morphism `phi` in `A`, this constructor returns the corresponding morphism in the Serre quotient category.

7.3 Serre Quotients by Spans

7.3.1 SerreQuotientCategoryBySpans (for IsCapCategory, IsFunction, IsString)

▷ `SerreQuotientCategoryBySpans(A, func[, name])` (operation)

Returns: a CAP category

Creates a Serre quotient category S with name `name` out of an Abelian category `A`. The Serre quotient category will be modeled upon the generalized morphisms by spans category of `A` If `name` is not given, a generic name is constructed out of the name of `A`. The argument `func` must be a unary function on the objects of `A` deciding the membership in the thick subcategory C mentioned above.

7.3.2 AsSerreQuotientCategoryBySpansObject (for IsCapCategory, IsCapCategoryObject)

▷ `AsSerreQuotientCategoryBySpansObject(A/C, M)` (operation)

Returns: an object

Given a Serre quotient category `A/C` modeled by spans and an object `M` in `A`, this constructor returns the corresponding object in the Serre quotient category.

7.3.3 SerreQuotientCategoryBySpansMorphism (for IsCapCategory, IsGeneralizedMorphismBySpan)

▷ `SerreQuotientCategoryBySpansMorphism(A/C, phi)` (operation)

Returns: a morphism

Given a Serre quotient category `A/C` modeled by spans and a generalized morphism `phi` in the generalized morphism category `A/C` is modeled upon, this constructor returns the corresponding morphism in the Serre quotient category.

7.3.4 SerreQuotientCategoryBySpansMorphism (for IsCapCategory, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `SerreQuotientCategoryBySpansMorphism(A/C, iota, phi, pi)` (operation)

Returns: a morphism

Given a Serre quotient category `A/C` modeled by spans and three morphisms $\iota : M' \rightarrow M$, $\phi : M' \rightarrow N'$ and $\pi : N \rightarrow N'$ this operation contructs a morphism in the Serre quotient category.

7.3.5 SerreQuotientCategoryBySpansMorphism (for IsCapCategory, IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `SerreQuotientCategoryBySpansMorphism(A/C, alpha, beta)` (operation)

Returns: a morphism

Given a Serre quotient category `A/C` modeled by spans and two morphisms $\alpha : M \rightarrow X$ and $\beta : X \rightarrow N$ this operation constructs the corresponding morphism in the Serre quotient category.

7.3.6 SerreQuotientCategoryBySpansMorphismWithRangeAid (for IsCapCategory, IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `SerreQuotientCategoryBySpansMorphismWithRangeAid(A/C, alpha, beta)` (operation)

Returns: a morphism

Given a Serre quotient category `A/C` modeled by spans and two morphisms $\alpha : X \rightarrow M$ and $\beta : X \rightarrow N$ this operation constructs the corresponding morphism in the Serre quotient category.

7.3.7 AsSerreQuotientCategoryBySpansMorphism (for IsCapCategory, IsCapCategoryMorphism)

▷ `AsSerreQuotientCategoryBySpansMorphism(A/C, phi)` (operation)

Returns: a morphism

Given a Serre quotient category `A/C` modeled by spans and a morphism `phi` in `A`, this constructor returns the corresponding morphism in the Serre quotient category.

7.4 Serre Quotients modeled by three arrows

7.4.1 SerreQuotientCategoryByThreeArrows (for IsCapCategory, IsFunction, IsString)

▷ `SerreQuotientCategoryByThreeArrows(A, func[, name])` (operation)

Returns: a CAP category

Creates a Serre quotient category S with name `name` out of an Abelian category `A`. The Serre quotient category will be modeled upon the generalized morphisms by three arrows category of `A` If `name` is not given, a generic name is constructed out of the name of `A`. The argument `func` must be a unary function on the objects of `A` deciding the membership in the thick subcategory C mentioned above.

7.4.2 AsSerreQuotientCategoryByThreeArrowsObject (for IsCapCategory, IsCapCategoryObject)

▷ `AsSerreQuotientCategoryByThreeArrowsObject(A/C, M)` (operation)

Returns: an object

Given a Serre quotient category `A/C` modeled by three arrows and an object `M` in `A`, this constructor returns the corresponding object in the Serre quotient category.

7.4.3 SerreQuotientCategoryByThreeArrowsMorphism (for IsCapCategory, IsGeneralizedMorphismByThreeArrows)

▷ `SerreQuotientCategoryByThreeArrowsMorphism(A/C, phi)` (operation)

Returns: a morphism

Given a Serre quotient category `A/C` modeled by three arrows and a generalized morphism `phi` in the generalized morphism category `A/C` is modeled upon, this constructor returns the corresponding morphism in the Serre quotient category.

7.4.4 SerreQuotientCategoryByThreeArrowsMorphism (for IsCapCategory, IsCapCategoryMorphism, IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `SerreQuotientCategoryByThreeArrowsMorphism(A/C, iota, phi, pi)` (operation)

Returns: a morphism

Given a Serre quotient category `A/C` modeled by three arrows and three morphisms $\iota : M' \rightarrow M$, $\phi : M' \rightarrow N'$ and $\pi : N \rightarrow N'$ this operation contructs a morphism in the Serre quotient category.

7.4.5 SerreQuotientCategoryByThreeArrowsMorphismWithSourceAid (for IsCapCategory, IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `SerreQuotientCategoryByThreeArrowsMorphismWithSourceAid(A/C, alpha, beta)` (operation)

Returns: a morphism

Given a Serre quotient category `A/C` modeled by three arrows and two morphisms $\alpha : M \rightarrow X$ and $\beta : X \rightarrow N$ this operation constructs the corresponding morphism in the Serre quotient category.

7.4.6 SerreQuotientCategoryByThreeArrowsMorphismWithRangeAid (for IsCapCategory, IsCapCategoryMorphism, IsCapCategoryMorphism)

▷ `SerreQuotientCategoryByThreeArrowsMorphismWithRangeAid(A/C, alpha, beta)` (operation)

Returns: a morphism

Given a Serre quotient category `A/C` modeled by three arrows and two morphisms $\alpha : X \rightarrow M$ and $\beta : X \rightarrow N$ this operation constructs the corresponding morphism in the Serre quotient category.

7.4.7 AsSerreQuotientCategoryByThreeArrowsMorphism (for IsCapCategory, IsCapCategoryMorphism)

▷ `AsSerreQuotientCategoryByThreeArrowsMorphism(A/C, phi)` (operation)

Returns: a morphism

Given a Serre quotient category `A/C` modeled by three arrows and a morphism `phi` in `A`, this constructor returns the corresponding morphism in the Serre quotient category.

Chapter 8

Serre Quotients Functors

The operations in this chapter construct conversion functors between two Serre quotients of different standards. Please note that all the operations DO NOT CARE if the Serre quotients are modeling the same category, they only care wether the categories are based upon the same underlying honest category.

8.1 Functors

8.1.1 SerreQuotientConversionFunctor (for IsCapCategory, IsCapCategory)

▷ `SerreQuotientConversionFunctor(A, B)` (operation)

Returns: a functor

Given two Serre quotients `A` and `B` of the same category `C` by equivalent functions, this operation returns the conversion functor between the two categories. Please note that this function does not check the quotients in any sense, i.e., wether the functions are equivalent.

Chapter 9

Examples and Tests

9.1 Basic Commands

Example

```
gap> Q := HomalgFieldOfRationals();;
gap> A := VectorSpaceObject( 4, Q );;
gap> B := VectorSpaceObject( 3, Q );;
gap> C := VectorSpaceObject( 2, Q );;
gap> alpha := VectorSpaceMorphism( A,
> HomalgMatrix( [ [ 1, 1, 1 ], [ 0, 1, 1 ],
> [ 1, 0, 1 ], [ 1, 1, 0 ] ], 4, 3, Q ), B );;
gap> gamma := VectorSpaceMorphism( C,
> HomalgMatrix( [ [ -1, 1, -1 ], [ 1, 0, -1 ] ], 2, 3, Q ), B );;
gap> p := ProjectionInFactorOfFiberProduct( [ alpha, gamma ], 1 );;
gap> q := ProjectionInFactorOfFiberProduct( [ alpha, gamma ], 2 );;
gap> PreCompose( AsGeneralizedMorphism( alpha ), GeneralizedInverse( gamma ) );
<A morphism in Generalized morphism category of Category of matrices over Q>
gap> gen1 := PreCompose( AsGeneralizedMorphism( alpha ),
>                        GeneralizedInverse( gamma ) );
<A morphism in Generalized morphism category of Category of matrices over Q>
gap> gen2 := PreCompose( GeneralizedInverse( p ), AsGeneralizedMorphism( q ) );
<A morphism in Generalized morphism category of Category of matrices over Q>
gap> IsCongruentForMorphisms( gen1, gen2 );
true
```

9.2 Intersection of Nodal Curve and Cusp

We are going to intersect the nodal curve $f = y^2 - x^2(x+1)$ and the cusp $g = (x+y)^2 - (y-x)^3$. The two curves are arranged in a way such that they intersect at $(0,0)$ with intersection number as high as possible. We are going to compute this intersection number using the definition of the intersection number as the length of the module $R/(f,g)$ localized at $(0,0)$. In order to model modules over the localization of $Q[x,y]$ at $(0,0)$, we use a suitable Serre quotient category. 1 2 1 1 true We are going to intersect the nodal curve $f = y^2 - x^2(x+1)$ and the cusp $g = (x+y)^2 - (y-x)^3$. The two curves are arranged in a way such that they intersect at $(0,0)$ with intersection number as high as possible. We are going to compute this intersection number using the definition of the intersection number as the length of the module $R/(f,g)$ localized at $(0,0)$. In order to model modules over the localization of $Q[x,y]$ at $(0,0)$, we use a suitable Serre quotient category. 1 2 1 1 true We are going to intersect the nodal curve $f = y^2 - x^2(x+1)$ and the cusp $g = (x+y)^2 - (y-x)^3$. The two curves are arranged in a way such that they intersect at $(0,0)$ with intersection number as high as possible. We are going to compute this intersection number using the definition of the intersection number as the length of the module $R/(f,g)$ localized at $(0,0)$. In order to model modules over the localization of $Q[x,y]$ at $(0,0)$, we use a suitable Serre quotient category. 1 2 1 1 true

9.3 WrapperCategory

Example

```
gap> LoadPackage( "LinearAlgebraForCAP" );
true
gap> LoadPackage( "GeneralizedMorphismsForCAP", false );
true
gap> old_generalized_morphism_standard := CAP_INTERNAL!.generalized_morphism_standard;;
gap> SwitchGeneralizedMorphismStandard( "cospan" );
gap> Q := HomalgFieldOfRationals( );
Q
gap> id := HomalgIdentityMatrix( 8, Q );
<An unevaluated 8 x 8 identity matrix over an internal ring>
gap> a := CertainColumns( CertainRows( id, [ 1, 2, 3 ] ), [ 2, 3, 4, 5 ] );
<An unevaluated non-zero 3 x 4 matrix over an internal ring>
gap> b := CertainColumns( CertainRows( id, [ 1, 2, 3, 4 ] ), [ 2, 3, 4, 5, 6 ] );
<An unevaluated non-zero 4 x 5 matrix over an internal ring>
gap> c := CertainColumns( CertainRows( id, [ 1, 2, 3, 4, 5 ] ), [ 3, 4, 5, 6, 7, 8 ] );
<An unevaluated non-zero 5 x 6 matrix over an internal ring>
gap> IsZero( a * b );
false
gap> IsZero( b * c );
false
gap> IsZero( a * b * c );
true
gap> Qmat := MatrixCategory( Q );
Category of matrices over Q
gap> Wrapper := WrapperCategory( Qmat, rec( ) );
WrapperCategory( Category of matrices over Q )
gap> a := a / Wrapper;
<A morphism in WrapperCategory( Category of matrices over Q )>
gap> b := b / Wrapper;
<A morphism in WrapperCategory( Category of matrices over Q )>
gap> c := c / Wrapper;
<A morphism in WrapperCategory( Category of matrices over Q )>
gap> d := CokernelProjection( a );
<An epimorphism in WrapperCategory( Category of matrices over Q )>
gap> e := CokernelColift( a, PreCompose( b, c ) );
<A morphism in WrapperCategory( Category of matrices over Q )>
gap> f := KernelEmbedding( e );
<A monomorphism in WrapperCategory( Category of matrices over Q )>
gap> g := KernelEmbedding( c );
<A monomorphism in WrapperCategory( Category of matrices over Q )>
gap> h := KernelLift( c, PreCompose( a, b ) );
<A morphism in WrapperCategory( Category of matrices over Q )>
gap> i := CokernelProjection( h );
```

```
<An epi morphism in WrapperCategory( Category of matrices over Q )>
gap> ff := AsGeneralizedMorphism( f );
<A morphism in Generalized morphism category of
 WrapperCategory( Category of matrices over Q ) by cospan>
gap> dd := AsGeneralizedMorphism( d );
<A morphism in Generalized morphism category of
 WrapperCategory( Category of matrices over Q ) by cospan>
gap> bb := AsGeneralizedMorphism( b );
<A morphism in Generalized morphism category of
 WrapperCategory( Category of matrices over Q ) by cospan>
gap> gg := AsGeneralizedMorphism( g );
<A morphism in Generalized morphism category of
 WrapperCategory( Category of matrices over Q ) by cospan>
gap> ii := AsGeneralizedMorphism( i );
<A morphism in Generalized morphism category of
 WrapperCategory( Category of matrices over Q ) by cospan>
gap> ss := PreCompose( [ ff, PseudoInverse( dd ), bb, PseudoInverse( gg ), ii ] );
<A morphism in Generalized morphism category of
 WrapperCategory( Category of matrices over Q ) by cospan>
gap> s := HonestRepresentative( ss );
<A morphism in WrapperCategory( Category of matrices over Q )>
gap> j := KernelObjectFunctorial( b, d, e );
<A morphism in WrapperCategory( Category of matrices over Q )>
gap> k := CokernelObjectFunctorial( h, g, b );
<A morphism in WrapperCategory( Category of matrices over Q )>
gap> HK := HomologyObject( j, s );
<An object in WrapperCategory( Category of matrices over Q )>
gap> HC := HomologyObject( s, k );
<An object in WrapperCategory( Category of matrices over Q )>
gap> SwitchGeneralizedMorphismStandard( old_generalized_morphism_standard );
```

9.4 Sweep

Geometric interpretation of sweeping a matrix by Terence Tao.

Example

```
gap> Q := HomalgFieldOfRationals();;
gap> V := VectorSpaceObject( 3, Q );;
gap> mat := HomalgMatrix( [ [ 9, 8, 7 ], [ 6, 5, 4 ], [ 3, 2, 1 ] ], 3, 3, Q );;
gap> alpha := VectorSpaceMorphism( V, mat, V );;
gap> graph := FiberProductEmbeddingInDirectSum(
>              [ alpha, IdentityMorphism( V ) ] );;
gap> Display( graph );
[ [     1,    -2,     1,     0,     0,     0 ],
  [  -4/3,   7/3,     0,     2,     1,     0 ],
  [   5/3,  -8/3,     0,    -1,     0,     1 ] ]

A morphism in Category of matrices over Q
gap> D := DirectSum( V, V );;
gap> rotmat := HomalgMatrix( [ [ 0, 0, 0, -1, 0, 0 ],
>                              [ 0, 1, 0, 0, 0, 0 ],
>                              [ 0, 0, 1, 0, 0, 0 ],
>                               [ 1, 0, 0, 0, 0, 0 ],
>                               [ 0, 0, 0, 0, 1, 0 ],
>                               [ 0, 0, 0, 0, 0, 1 ] ],
>                               6, 6, Q );;
gap> rot := VectorSpaceMorphism( D, rotmat, D );;
gap> p := PreCompose( graph, rot );;
gap> Display( p );
[ [     0,    -2,     1,    -1,     0,     0 ],
  [     2,   7/3,     0,   4/3,     1,     0 ],
  [    -1,  -8/3,     0,  -5/3,     0,     1 ] ]

A morphism in Category of matrices over Q
gap> pi1 := ProjectionInFactorOfDirectSum( [ V, V ], 1 );;
gap> pi2 := ProjectionInFactorOfDirectSum( [ V, V ], 2 );;
gap> reversed_arrow := PreCompose( p, pi1 );;
gap> arrow := PreCompose( p, pi2 );;
gap> g := GeneralizedMorphismBySpan( reversed_arrow, arrow );;
gap> IsHonest( g );
true
gap> sweep_1_alpha := HonestRepresentative( g );;
gap> Display( sweep_1_alpha );
[ [  -1/9,   8/9,   7/9 ],
  [   2/3,  -1/3,  -2/3 ],
  [   1/3,  -2/3,  -4/3 ] ]

A morphism in Category of matrices over Q
gap> Display( alpha );
[ [  9,  8,  7 ],
  [  6,  5,  4 ],
  [  3,  2,  1 ] ]

A morphism in Category of matrices over Q
```

9.5 Generalized Morphisms Category

Example

```
gap> Q := HomalgFieldOfRationals();
Q
gap> B := VectorSpaceObject( 2, Q );
<A vector space object over Q of dimension 2>
gap> C := VectorSpaceObject( 3, Q );
<A vector space object over Q of dimension 3>
gap> B_1 := VectorSpaceObject( 1, Q );
<A vector space object over Q of dimension 1>
gap> C_1 := VectorSpaceObject( 2, Q );
<A vector space object over Q of dimension 2>
gap> c1_source_aid := VectorSpaceMorphism( B_1, [ [ 1, 0 ] ], B );
<A morphism in Category of matrices over Q>
gap> SetIsSubobject( c1_source_aid, true );
gap> c1_range_aid := VectorSpaceMorphism( C, [ [ 1, 0 ], [ 0, 1 ], [ 0, 0 ] ], C_1 );
<A morphism in Category of matrices over Q>
gap> SetIsFactorobject( c1_range_aid, true );
gap> c1_associated := VectorSpaceMorphism( B_1, [ [ 1, 1 ] ], C_1 );
<A morphism in Category of matrices over Q>
gap> c1 := GeneralizedMorphism( c1_source_aid, c1_associated, c1_range_aid );
<A morphism in Generalized morphism category of Category of matrices over Q>
gap> B_2 := VectorSpaceObject( 1, Q );
<A vector space object over Q of dimension 1>
gap> C_2 := VectorSpaceObject( 2, Q );
<A vector space object over Q of dimension 2>
gap> c2_source_aid := VectorSpaceMorphism( B_2, [ [ 2, 0 ] ], B );
<A morphism in Category of matrices over Q>
gap> SetIsSubobject( c2_source_aid, true );
gap> c2_range_aid := VectorSpaceMorphism( C, [ [ 3, 0 ], [ 0, 3 ], [ 0, 0 ] ], C_2 );
<A morphism in Category of matrices over Q>
gap> SetIsFactorobject( c2_range_aid, true );
gap> c2_associated := VectorSpaceMorphism( B_2, [ [ 6, 6 ] ], C_2 );
<A morphism in Category of matrices over Q>
gap> c2 := GeneralizedMorphism( c2_source_aid, c2_associated, c2_range_aid );
<A morphism in Generalized morphism category of Category of matrices over Q>
gap> IsCongruentForMorphisms( c1, c2 );
true
gap> IsCongruentForMorphisms( c1, c1 );
true
gap> c3_associated := VectorSpaceMorphism( B_1, [ [ 2, 2 ] ], C_1 );
<A morphism in Category of matrices over Q>
gap> c3 := GeneralizedMorphism( c1_source_aid, c3_associated, c1_range_aid );
<A morphism in Generalized morphism category of Category of matrices over Q>
gap> IsCongruentForMorphisms( c1, c3 );
false
gap> IsCongruentForMorphisms( c2, c3 );
false
gap> c1 + c2;
<A morphism in Generalized morphism category of Category of matrices over Q>
gap> Arrow( c1 + c2 );
<A morphism in Category of matrices over Q>
```

First composition test:

Example
```
gap> Q := HomalgFieldOfRationals();
Q
gap> A := VectorSpaceObject( 1, Q );
<A vector space object over Q of dimension 1>
gap> B := VectorSpaceObject( 2, Q );
<A vector space object over Q of dimension 2>
gap> C := VectorSpaceObject( 3, Q );
<A vector space object over Q of dimension 3>
gap> phi_tilde_associated := VectorSpaceMorphism( A, [ [ 1, 2, 0 ] ], C );
<A morphism in Category of matrices over Q>
gap> phi_tilde_source_aid := VectorSpaceMorphism( A, [ [ 1, 2 ] ], B );
<A morphism in Category of matrices over Q>
gap> phi_tilde := GeneralizedMorphismWithSourceAid( phi_tilde_source_aid, phi_tilde_associated );
<A morphism in Generalized morphism category of Category of matrices over Q>
gap> psi_tilde_associated := IdentityMorphism( B );
```

```
<An identity morphism in Category of matrices over Q>
gap> psi_tilde_source_aid := VectorSpaceMorphism( B, [ [ 1, 0, 0 ], [ 0, 1, 0 ] ], C );
<A morphism in Category of matrices over Q>
gap> psi_tilde := GeneralizedMorphismWithSourceAid( psi_tilde_source_aid, psi_tilde_associated );
<A morphism in Generalized morphism category of Category of matrices over Q>
gap> composition := PreCompose( phi_tilde, psi_tilde );
<A morphism in Generalized morphism category of Category of matrices over Q>
gap> Arrow( composition );
<A morphism in Category of matrices over Q>
gap> SourceAid( composition );
<A morphism in Category of matrices over Q>
gap> RangeAid( composition );
<An identity morphism in Category of matrices over Q>
```

Second composition test

Example

```
gap> Q := HomalgFieldOfRationals();
Q
gap> A := VectorSpaceObject( 1, Q );
<A vector space object over Q of dimension 1>
gap> B := VectorSpaceObject( 2, Q );
<A vector space object over Q of dimension 2>
gap> C := VectorSpaceObject( 3, Q );
<A vector space object over Q of dimension 3>
gap> phi2_tilde_associated := VectorSpaceMorphism( A, [ [ 1, 5 ] ], B );
<A morphism in Category of matrices over Q>
gap> phi2_tilde_range_aid := VectorSpaceMorphism( C, [ [ 1, 0 ], [ 0, 1 ], [ 1, 1 ] ], B );
<A morphism in Category of matrices over Q>
gap> phi2_tilde := GeneralizedMorphismWithRangeAid( phi2_tilde_associated, phi2_tilde_range_aid )
<A morphism in Generalized morphism category of Category of matrices over Q>
gap> psi2_tilde_associated := VectorSpaceMorphism( C, [ [ 1 ], [ 3 ], [ 4 ] ], A );
<A morphism in Category of matrices over Q>
gap> psi2_tilde_range_aid := VectorSpaceMorphism( B, [ [ 1 ], [ 1 ] ], A );
<A morphism in Category of matrices over Q>
gap> psi2_tilde := GeneralizedMorphismWithRangeAid( psi2_tilde_associated, psi2_tilde_range_aid )
<A morphism in Generalized morphism category of Category of matrices over Q>
gap> composition2 := PreCompose( phi2_tilde, psi2_tilde );
<A morphism in Generalized morphism category of Category of matrices over Q>
gap> Arrow( composition2 );
<A morphism in Category of matrices over Q>
gap> RangeAid( composition2 );
<A morphism in Category of matrices over Q>
gap> SourceAid( composition2 );
<An identity morphism in Category of matrices over Q>
```

Third composition test

Example

```
gap> Q := HomalgFieldOfRationals();
Q
gap> A := VectorSpaceObject( 3, Q );
<A vector space object over Q of dimension 3>
gap> Asub := VectorSpaceObject( 2, Q );
```

```
<A vector space object over Q of dimension 2>
gap> B := VectorSpaceObject( 3, Q );
<A vector space object over Q of dimension 3>
gap> Bfac := VectorSpaceObject( 1, Q );
<A vector space object over Q of dimension 1>
gap> Bsub := VectorSpaceObject( 2, Q );
<A vector space object over Q of dimension 2>
gap> C := VectorSpaceObject( 3, Q );
<A vector space object over Q of dimension 3>
gap> Cfac := VectorSpaceObject( 1, Q );
<A vector space object over Q of dimension 1>
gap> Asub_into_A := VectorSpaceMorphism( Asub, [ [ 1, 0, 0 ], [ 0, 1, 0 ] ], A );
<A morphism in Category of matrices over Q>
gap> Asub_to_Bfac := VectorSpaceMorphism( Asub, [ [ 1 ], [ 1 ] ], Bfac );
<A morphism in Category of matrices over Q>
gap> B_onto_Bfac := VectorSpaceMorphism( B, [ [ 1 ], [ 1 ], [ 1 ] ], Bfac );
<A morphism in Category of matrices over Q>
gap> Bsub_into_B := VectorSpaceMorphism( Bsub, [ [ 2, 2, 0 ], [ 0, 2, 2 ] ], B );
<A morphism in Category of matrices over Q>
gap> Bsub_to_Cfac := VectorSpaceMorphism( Bsub, [ [ 3 ], [ 0 ] ], Cfac );
<A morphism in Category of matrices over Q>
gap> C_onto_Cfac := VectorSpaceMorphism( C, [ [ 1 ], [ 2 ], [ 3 ] ], Cfac );
<A morphism in Category of matrices over Q>
gap> generalized_morphism1 := GeneralizedMorphism( Asub_into_A, Asub_to_Bfac, B_onto_Bfac );
<A morphism in Generalized morphism category of Category of matrices over Q>
gap> generalized_morphism2 := GeneralizedMorphism( Bsub_into_B, Bsub_to_Cfac, C_onto_Cfac );
<A morphism in Generalized morphism category of Category of matrices over Q>
gap> IsWellDefined( generalized_morphism1 );
true
gap> IsWellDefined( generalized_morphism2 );
true
gap> p := PreCompose( generalized_morphism1, generalized_morphism2 );
<A morphism in Generalized morphism category of Category of matrices over Q>
gap> SourceAid( p );
<A morphism in Category of matrices over Q>
gap> Arrow( p );
<A morphism in Category of matrices over Q>
gap> RangeAid( p );
<A morphism in Category of matrices over Q>
gap> A := VectorSpaceObject( 3, Q );
<A vector space object over Q of dimension 3>
gap> Asub := VectorSpaceObject( 2, Q );
<A vector space object over Q of dimension 2>
gap> B := VectorSpaceObject( 3, Q );
<A vector space object over Q of dimension 3>
gap> Bfac := VectorSpaceObject( 1, Q );
<A vector space object over Q of dimension 1>
gap> Bsub := VectorSpaceObject( 2, Q );
<A vector space object over Q of dimension 2>
gap> C := VectorSpaceObject( 3, Q );
<A vector space object over Q of dimension 3>
gap> Cfac := VectorSpaceObject( 2, Q );
```

```
<A vector space object over Q of dimension 2>
gap> Bsub_to_Cfac := VectorSpaceMorphism( Bsub, [ [ 3, 3 ], [ 0, 0 ] ], Cfac );
<A morphism in Category of matrices over Q>
gap> C_onto_Cfac := VectorSpaceMorphism( C, [ [ 1, 0 ], [ 0, 2 ], [ 3, 3 ] ], Cfac );
<A morphism in Category of matrices over Q>
gap> generalized_morphism1 := GeneralizedMorphism( Asub_into_A, Asub_to_Bfac, B_onto_Bfac );
<A morphism in Generalized morphism category of Category of matrices over Q>
gap> generalized_morphism2 := GeneralizedMorphism( Bsub_into_B, Bsub_to_Cfac, C_onto_Cfac );
<A morphism in Generalized morphism category of Category of matrices over Q>
gap> IsWellDefined( generalized_morphism1 );
true
gap> IsWellDefined( generalized_morphism2 );
true
gap> p := PreCompose( generalized_morphism1, generalized_morphism2 );
<A morphism in Generalized morphism category of Category of matrices over Q>
gap> SourceAid( p );
<A morphism in Category of matrices over Q>
gap> Arrow( p );
<A morphism in Category of matrices over Q>
gap> RangeAid( p );
<A morphism in Category of matrices over Q>
```

Honest representative test

Example

```
gap> Q := HomalgFieldOfRationals();
Q
gap> A := VectorSpaceObject( 1, Q );
<A vector space object over Q of dimension 1>
gap> B := VectorSpaceObject( 2, Q );
<A vector space object over Q of dimension 2>
gap> phi_tilde_source_aid := VectorSpaceMorphism( A, [ [ 2 ] ], A );
<A morphism in Category of matrices over Q>
gap> phi_tilde_associated := VectorSpaceMorphism( A, [ [ 1, 1 ] ], B );
<A morphism in Category of matrices over Q>
gap> phi_tilde_range_aid := VectorSpaceMorphism( B, [ [ 1, 2 ], [ 3, 4 ] ], B );
<A morphism in Category of matrices over Q>
gap> phi_tilde := GeneralizedMorphism( phi_tilde_source_aid, phi_tilde_associated, phi_tilde_rang
<A morphism in Generalized morphism category of Category of matrices over Q>
gap> HonestRepresentative( phi_tilde );
<A morphism in Category of matrices over Q>
gap> IsWellDefined( phi_tilde );
true
gap> IsWellDefined( psi_tilde );
true
```

9.6 IsWellDefined

Example

```
gap> Q := HomalgFieldOfRationals();
Q
gap> A := VectorSpaceObject( 1, Q );
<A vector space object over Q of dimension 1>
```

```
gap> B := VectorSpaceObject( 2, Q );
<A vector space object over Q of dimension 2>
gap> alpha := VectorSpaceMorphism( A, [ [ 1, 2 ] ], B );
<A morphism in Category of matrices over Q>
gap> g := GeneralizedMorphism( alpha, alpha, alpha );
<A morphism in Generalized morphism category of Category of matrices over Q>
gap> IsWellDefined( alpha );
true
gap> IsWellDefined( g );
true
gap> IsEqualForObjects( A, B );
false
```

Index