complex

I wear a chain complex now. Chain complexes are cool

27.01.2017

27 January 2017

Kristin Krogh Arnesen

Oystein Skartsaeterhagen

Kamal Saleh

Kristin Krogh Arnesen
Email: kristink@math.ntnu.no
Homepage: http://www.math.ntnu.no/~kristink
Address: Trondheim

Oystein Skartsaeterhagen
Email: oysteini@math.ntnu.no
Homepage: http://www.math.ntnu.no/~oysteini
Address: Trondheim

Kamal Saleh
Email: kamal.saleh@uni-siegen.de
Homepage: https://github.com/kamalsaleh/complex
Address: Siegen

Contents

Chapter 1

Complexes

1.1 Categories and filters

1.1.1 IsChainOrCochainComplex (for IsCapCategoryObject)

▷ `IsChainOrCochainComplex(`*arg*`)` (filter)

Returns: `true` or `false`

bla bla

1.1.2 IsChainComplex (for IsChainOrCochainComplex)

▷ `IsChainComplex(`*arg*`)` (filter)

Returns: `true` or `false`

bla bla

1.1.3 IsCochainComplex (for IsChainOrCochainComplex)

▷ `IsCochainComplex(`*arg*`)` (filter)

Returns: `true` or `false`

bla bla

1.1.4 IsBoundedBelowChainOrCochainComplex (for IsChainOrCochainComplex)

▷ `IsBoundedBelowChainOrCochainComplex(`*arg*`)` (filter)

Returns: `true` or `false`

bla bla

1.1.5 IsBoundedAboveChainOrCochainComplex (for IsChainOrCochainComplex)

▷ `IsBoundedAboveChainOrCochainComplex(`*arg*`)` (filter)

Returns: `true` or `false`

bla bla

1.1.6 IsBoundedChainOrCochainComplex (for IsBoundedBelowChainOrCochainComplex and IsBoundedAboveChainOrCochainComplex)

▷ `IsBoundedChainOrCochainComplex(`*arg*`)` (filter)

Returns: `true` or `false`

bla bla

1.1.7 IsBoundedBelowChainComplex (for IsBoundedBelowChainOrCochainComplex and IsChainComplex)

▷ `IsBoundedBelowChainComplex(`*arg*`)` (filter)

Returns: `true` or `false`

bla bla

1.1.8 IsBoundedBelowCochainComplex (for IsBoundedBelowChainOrCochainComplex and IsCochainComplex)

▷ `IsBoundedBelowCochainComplex(`*arg*`)` (filter)

Returns: `true` or `false`

bla bla

1.1.9 IsBoundedAboveChainComplex (for IsBoundedAboveChainOrCochainComplex and IsChainComplex)

▷ `IsBoundedAboveChainComplex(`*arg*`)` (filter)

Returns: `true` or `false`

bla bla

1.1.10 IsBoundedAboveCochainComplex (for IsBoundedAboveChainOrCochainComplex and IsCochainComplex)

▷ `IsBoundedAboveCochainComplex(`*arg*`)` (filter)

Returns: `true` or `false`

bla bla

1.1.11 IsBoundedChainComplex (for IsBoundedChainOrCochainComplex and IsChainComplex)

▷ `IsBoundedChainComplex(`*arg*`)` (filter)

Returns: `true` or `false`

bla bla

1.1.12 IsBoundedCochainComplex (for IsBoundedChainOrCochainComplex and IsCochainComplex)

▷ `IsBoundedCochainComplex(`*arg*`)` (filter)

Returns: `true` or `false`

bla bla

1.2 Creating chain and cochain complexes

1.2.1 ChainComplex (for IsCapCategory, IsZList)

▷ `ChainComplex(A, diffs)` (operation)
▷ `CochainComplex(A, diffs)` (operation)

Returns: a chain complex

The input is category `A` and an infinite list `diffs`. The output is the chain (resp. cochain) complex $M_\bullet \in \mathrm{Ch}(A)$ ($M^\bullet \in \mathrm{CoCh}(A)$) where $d_i^M = \mathrm{diffs}[i] (d_M^i = \mathrm{diffs}[i])$.

1.2.2 ChainComplex (for IsDenseList, IsInt)

▷ `ChainComplex(diffs, n)` (operation)
▷ `CochainComplex(diffs, n)` (operation)

Returns: a (co)chain complex

The input is a finite dense list `diffs` and an integer `n` . The output is the chain (resp. cochain) complex $M_\bullet \in \mathrm{Ch}(A)$ ($M^\bullet \in \mathrm{CoCh}(A)$) where $d_n^M := \mathrm{diffs}[1] (d_M^n := \mathrm{diffs}[1]), d_{n+1}^M = \mathrm{diffs}[2] (d_M^{n+1} := \mathrm{diffs}[2])$, etc.

1.2.3 ChainComplex (for IsDenseList)

▷ `ChainComplex(diffs)` (operation)
▷ `CochainComplex(diffs)` (operation)

Returns: a (co)chain complex

The same as the previous operations but with $n = 0$.

1.2.4 StalkChainComplex (for IsCapCategoryObject, IsInt)

▷ `StalkChainComplex(diffs, n)` (operation)
▷ `StalkCochainComplex(diffs, n)` (operation)

Returns: a (co)chain complex

The input is an object $M \in A$. The output is chain (resp. cochain) complex $M_\bullet \in \mathrm{Ch(A)} (M^\bullet \in \mathrm{CoCh(A)})$ where $M_n = M (M^n = M)$ and $M_i = 0 (M^i = 0)$ whenever $i \neq n$.

1.2.5 ChainComplexWithInductiveSides (for IsCapCategoryMorphism, IsFunction, IsFunction)

▷ `ChainComplexWithInductiveSides(d, G, F)` (operation)

Returns: a chain complex

The input is a morphism $d \in A$ and two functions F, G. The output is chain complex $M_\bullet \in \mathrm{Ch(A)}$ where $d_0^M = d$ and $d_i^M = G^i(d)$ for all $i \leq -1$ and $d_i^M = F^i(d)$ for all $i \geq 1$.

1.2.6 CochainComplexWithInductiveSides (for IsCapCategoryMorphism, IsFunction, IsFunction)

▷ `CochainComplexWithInductiveSides(d, G, F)` (operation)

Returns: a cochain complex

The input is a morphism $d \in A$ and two functions F, G. The output is cochain complex $M^\bullet \in \mathrm{CoCh(A)}$ where $d_M^0 = d$ and $d_M^i = G^i(d)$ for all $i \leq -1$ and $d_M^i = F^i(d)$ for all $i \geq 1$.

1.2.7 ChainComplexWithInductiveNegativeSide (for IsCapCategoryMorphism, IsFunction)

▷ `ChainComplexWithInductiveNegativeSide(d, G)` (operation)

Returns: a chain complex

The input is a morphism $d \in A$ and a functions G. The output is chain complex $M_\bullet \in \mathrm{Ch(A)}$ where $d_0^M = d$ and $d_i^M = G^i(d)$ for all $i \leq -1$ and $d_i^M = 0$ for all $i \geq 1$.

1.2.8 ChainComplexWithInductivePositiveSide (for IsCapCategoryMorphism, IsFunction)

▷ `ChainComplexWithInductivePositiveSide(d, F)` (operation)

Returns: a chain complex

The input is a morphism $d \in A$ and a functions F. The output is chain complex $M_\bullet \in \mathrm{Ch(A)}$ where $d_0^M = d$ and $d_i^M = F^i(d)$ for all $i \geq 1$ and $d_i^M = 0$ for all $i \leq 1$.

1.2.9 CochainComplexWithInductiveNegativeSide (for IsCapCategoryMorphism, IsFunction)

▷ `CochainComplexWithInductiveNegativeSide(d, G)` (operation)

Returns: a cochain complex

The input is a morphism $d \in A$ and a functions G. The output is cochain complex $M^\bullet \in \mathrm{CoCh(A)}$ where $d_M^0 = d$ and $d_M^i = G^i(d)$ for all $i \leq -1$ and $d_M^i = 0$ for all $i \geq 1$.

1.2.10 CochainComplexWithInductivePositiveSide (for IsCapCategoryMorphism, IsFunction)

▷ `CochainComplexWithInductivePositiveSide(d, F)` (operation)

Returns: a cochain complex

The input is a morphism $d \in A$ and a functions F. The output is cochain complex $M^\bullet \in \mathrm{CoCh(A)}$ where $d_M^0 = d$ and $d_M^i = F^i(d)$ for all $i \geq 1$ and $d_M^i = 0$ for all $i \leq 1$.

1.3 Attributes

1.3.1 Differentials (for IsChainOrCochainComplex)

▷ `Differentials(C)` (attribute)

Returns: an infinite list

The command returns the differentials of the chain or cochain complex as an infinite list.

1.3.2 Objects (for IsChainOrCochainComplex)

▷ `Objects(C)` (attribute)

Returns: an infinite list

The command returns the objects of the chain or cochain complex as an infinite list.

1.3.3 CatOfComplex (for IsChainOrCochainComplex)

▷ `CatOfComplex(C)` (attribute)

Returns: a Cap category

The command returns the category in which all objects and differentials of C live.

1.4 Operations

1.4.1 \[\] (for IsChainOrCochainComplex, IsInt)

▷ `\[\](C, i)` (operation)

Returns: an object

The command returns the object of the chain or cochain complex in index i.

1.4.2 \^ (for IsChainOrCochainComplex, IsInt)

▷ `\^(C, i)` (operation)

Returns: a morphism

The command returns the differential of the chain or cochain complex in index i.

1.4.3 CertainCycle (for IsChainOrCochainComplex, IsInt)

▷ `CertainCycle(C, n)` (operation)

Returns: a morphism

The input is a chain or cochain complex C and an integer n. The output is the kernel embedding of the differential in index n.

1.4.4 CertainBoundary (for IsChainOrCochainComplex, IsInt)

▷ `CertainBoundary(C, n)` (operation)

Returns: a morphism

The input is a chain (resp. cochain) complex C and an integer n. The output is the image embeddin of $i+1$'th (resp. $i-1$'th) differential of C.

1.4.5 DefectOfExactness (for IsChainOrCochainComplex, IsInt)

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

Returns: a object

The input is a chain (resp. cochain) complex C and an integer n. The outout is the homology (resp. cohomology) object of C in index n.

1.4.6 IsExactInIndex (for IsChainOrCochainComplex, IsInt)

▷ `IsExactInIndex(C, n)` (operation)

Returns: true or false

The input is a chain or cochain complex C and an integer n. The outout is `true` if C is exact in i. Otherwise the output is `false`.

1.4.7 SetUpperBound (for IsChainOrCochainComplex, IsInt)

▷ `SetUpperBound(C, n)` (operation)

Returns: Side effect

The command sets an upper bound n to the chain (resp. cochain) complex C. This means $C_{i\geq n} = 0 (C^{\geq n} = 0)$. This upper bound will be called *active* upper bound of C. If C already has an active upper bound m, then m will be replaced by n only if n is better upper bound than m, i.e., $n \leq m$. If C has an active lower bound l and $n \leq l$ then the upper bound will set to equal l and as a consequence C will be zeroised.

1.4.8 SetLowerBound (for IsChainOrCochainComplex, IsInt)

▷ `SetLowerBound(C, n)` (operation)

Returns: Side effect

The command sets an lower bound n to the chain (resp. cochain) complex C. This means $C_{i\leq n} = 0 (C^{\leq n} = 0)$. This lower bound will be called *active* lower bound of C. If C already has an active lower bound m, then m will be replaced by n only if n is better lower bound than m, i.e., $n \geq m$. If C has an active upper bound u and $n \geq u$ then the lower bound will set to equal u and as a consequence C will be zeroised.

1.4.9 HasActiveUpperBound (for IsChainOrCochainComplex)

▷ `HasActiveUpperBound(C)` (operation)

Returns: true or false

The input is chain or cochain complex. The output is `true` if an upper bound has been set to C and `false` otherwise.

1.4.10 HasActiveLowerBound (for IsChainOrCochainComplex)

▷ `HasActiveLowerBound(C)` (operation)

Returns: true or false

The input is chain or cochain complex. The output is `true` if a lower bound has been set to C and `false` otherwise.

1.4.11 ActiveUpperBound (for IsChainOrCochainComplex)

▷ `ActiveUpperBound(C)` (operation)

Returns: an integer

The input is chain or cochain complex. The output is its active upper bound if such has been set to C. Otherwise we get error.

1.4.12 ActiveLowerBound (for IsChainOrCochainComplex)

▷ `ActiveLowerBound(C)` (operation)

Returns: an integer

The input is chain or cochain complex. The output is its active lower bound if such has been set to C. Otherwise we get error.

1.4.13 Display (for IsChainOrCochainComplex, IsInt, IsInt)

▷ `Display(C, m, n)` (operation)

Returns: nothing

The input is chain or cochain complex C and two integers m and n. The command displays all components of C between the indices m,n.

1.5 Truncations

1.5.1 GoodTruncationBelow (for IsChainComplex, IsInt)

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

Returns: chain complex

Let $C_\bullet$ be chain complex. A good truncation of $C_\bullet$ below n is the chain complex $\tau_{\geq n}C_\bullet$ whose differentials are defined by

$$d_i^{\tau_{\geq n}C_\bullet} = \begin{cases} 0 : 0 \leftarrow 0 & \text{if} \quad i < n, \\ 0 : 0 \leftarrow Z_n & \text{if} \quad i = n, \\ \mathrm{KernelLift}(d_n^C, d_{n+1}^C) : Z_n \leftarrow C_{n+1} & \text{if} \quad i = n+1, \\ d_i^C : C_{i-1} \leftarrow C_i & \text{if} \quad i > n+1. \end{cases}$$

where Z_n is the cycle in index n. It can be shown that $H_i(\tau_{\geq n}C_\bullet) = 0$ for $i < n$ and $H_i(\tau_{\geq n}C_\bullet) = H_i(C_\bullet)$ for $i \geq n$.

$$\begin{tikzcd}
C_\bullet & \cdots & C_{n-1} \arrow[l] & C_n \arrow[l] & C_{n+1} \arrow[l] \arrow[dl] & C_{n+2} \arrow[l] & \cdots \arrow[l] \\
\tau_{\geq n}C_\bullet & \cdots & 0 \arrow[l] & Z_n \arrow[l] \arrow[u, hook] & & &
\end{tikzcd}$$

1.5.2 GoodTruncationAbove (for IsChainComplex, IsInt)

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

Returns: chain complex

Let $C_\bullet$ be chain complex. A good truncation of $C_\bullet$ above n is the quotient chain complex $\tau_{<n}C_\bullet = C_\bullet / \tau_{\geq n}C_\bullet$. It can be shown that $H_i(\tau_{<n}C_\bullet) = 0$ for $i \geq n$ and $H_i(\tau_{<n}C_\bullet) = H_i(C_\bullet)$ for $i < n$.

1.5.3 GoodTruncationAbove (for IsCochainComplex, IsInt)

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

Returns:

Let $C^\bullet$ be cochain complex. A good truncation of $C^\bullet$ above n is the cochain complex $\tau_{\leq n}C^\bullet$ whose differentials are defined by

$$d^i_{\tau_{\leq n}C^\bullet} = \begin{cases} 0 : 0 \rightarrow 0 & \text{if} \quad i > n, \\ 0 : Z_n \rightarrow 0 & \text{if} \quad i = n, \\ \mathrm{KernelLift}(d_C^n, d_C^{n-1}) : C_{n-1} \rightarrow Z_n & \text{if} \quad i = n-1, \\ d_C^i : C_i \rightarrow C_{i+1} & \text{if} \quad i < n-1. \end{cases}$$

where Z_n is the cycle in index n. It can be shown that $H^i(\tau_{\leq n}C^\bullet) = 0$ for $i > n$ and $H^i(\tau_{\leq n}C^\bullet) = H_i(C^\bullet)$ for $i \leq n$.

$$
\begin{tikzcd}
\cdots \arrow[r] & C_{n-2} \arrow[r] & C_{n-1} \arrow[r] \arrow[dr] & C_n \arrow[r] & C_{n+1} \arrow[r] & \cdots & C_\bullet \\
 & & & Z_n \arrow[u, hook] \arrow[r] & 0 \arrow[r] & \cdots & \tau_{\leq n}C^\bullet
\end{tikzcd}
$$

1.5.4 GoodTruncationBelow (for IsCochainComplex, IsInt)

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

Returns: cochain complex

Let $C^\bullet$ be cochain complex. A good truncation of $C^\bullet$ above n is the quotient cochain complex $\tau_{>n}C^\bullet = C^\bullet/\tau_{\leq n}C^\bullet$. It can be shown that $H^i(\tau_{>n}C^\bullet) = 0$ for $i \leq n$ and $H^i(\tau_{>n}C^\bullet) = H_i(C^\bullet)$ for $i > n$.

1.5.5 BrutalTruncationBelow (for IsChainComplex, IsInt)

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

Returns: chain complex

Let $C_\bullet$ be chain complex. A brutal truncation of $C_\bullet$ below n is the chain complex $\sigma_{\geq n}C_\bullet$ where $(\sigma_{\geq n}C_\bullet)_i = C_i$ when $i \geq n$ and $(\sigma_{\geq n}C_\bullet)_i = 0$ otherwise.

1.5.6 BrutalTruncationAbove (for IsChainComplex, IsInt)

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

Returns: chain complex

Let $C_\bullet$ be chain complex. A brutal truncation of $C_\bullet$ above n is the chain quotient chain complex $\sigma_{<n}C_\bullet := C_\bullet/\sigma_{\geq n}C_\bullet$. Hence $(\sigma_{<n}C_\bullet)_i = C_i$ when $i < n$ and $(\sigma_{<n}C_\bullet)_i = 0$ otherwise.

1.5.7 BrutalTruncationAbove (for IsCochainComplex, IsInt)

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

Returns: chain complex

Let $C^\bullet$ be cochain complex. A brutal truncation of $C_\bullet$ above n is the cochain complex $\sigma_{\leq n}C^\bullet$ where $(\sigma_{\leq n}C^\bullet)_i = C_i$ when $i \leq n$ and $(\sigma_{\leq n}C^\bullet)_i = 0$ otherwise.

1.5.8 BrutalTruncationBelow (for IsCochainComplex, IsInt)

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

Returns: chain complex

Let $C^\bullet$ be cochain complex. A brutal truncation of $C^\bullet$ bellow n is the quotient cochain complex $\sigma_{>n}C^\bullet := C^\bullet/\sigma_{\leq n}C_\bullet$. Hence $(\sigma_{>n}C^\bullet)_i = C_i$ when $i > n$ and $(\sigma_{<n}C^\bullet)_i = 0$ otherwise.

1.6 Examples

bla latex code here.

Example

```
gap> Q := HomalgFieldOfRationals( );;
gap> matrix_category := MatrixCategory( Q );
Category of matrices over Q
gap> cochain_cat := CochainComplexCategory( matrix_category );
Cochain complexes category over category of matrices over 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> f := VectorSpaceMorphism( A, HomalgMatrix( [ [ 1, 3 ] ], 1, 2, Q ), B );
<A morphism in Category of matrices over Q>
gap> g := VectorSpaceMorphism( B, HomalgMatrix( [ [ 0 ], [ 0 ] ], 2, 1, Q ), A );
<A morphism in Category of matrices over Q>
gap> C := CochainComplex( [ f,g,f ], 3 );
<A bounded object in cochain complexes category over category of matrices over Q
with active lower bound 2 and active upper bound 7.>
gap> ActiveUpperBound( C );
7
gap> ActiveLowerBound( C );
2
gap> C[ 1 ];
<A vector space object over Q of dimension 0>
gap> C[ 3 ];
<A vector space object over Q of dimension 1>
gap> C^3;
<A morphism in Category of matrices over Q>
gap> C^3 = f;
true
gap> Display( CertainCycle( C, 4 ) );
[ [  1,  0 ],
  [  0,  1 ] ]

A split monomorphism in Category of matrices over Q
gap> diffs := Differentials( C );
<An infinite list>
gap> diffs[ 1 ];
<A zero, isomorphism in Category of matrices over Q>
gap> diffs[ 10000 ];
<A zero, isomorphism in Category of matrices over Q>
gap> objs := Objects( C );
<An infinite list>
gap> DefectOfExactness( C, 4 );
<A vector space object over Q of dimension 1>
gap> DefectOfExactness( C, 3 );
<A vector space object over Q of dimension 0>
gap> IsExactInIndex( C, 4 );
false
gap> IsExactInIndex( C, 3 );
true
gap> C;
<A not cyclic, bounded object in cochain complexes category over category of
matrices over Q with active lower bound 2 and active upper bound 7.>
```

```
gap> T := ShiftFunctor( cochain_cat, 3 );
Shift (3 times to the left) functor in cochain complexes category over category
 of matrices over Q
gap> C_3 := ApplyFunctor( T, C );
<A not cyclic, bounded object in cochain complexes category over category of
matrices over Q with active lower bound -1 and active upper bound 4.>
gap> P := CochainComplex( matrix_category, diffs );
<An object in Cochain complexes category over category of matrices over Q>
gap> SetUpperBound( P, 15 );
gap> P;
<A bounded from above object in cochain complexes category over category of
matrices over Q with active upper bound 15.>
gap> SetUpperBound( P, 20 );
gap> P;
<A bounded from above object in cochain complexes category over category of
matrices over Q with active upper bound 15.>
gap> ActiveUpperBound( P );
15
gap> SetUpperBound( P, 7 );
gap> P;
<A bounded from above object in cochain complexes category over category of
matrices over Q with active upper bound 7.>
gap> ActiveUpperBound( P );
7
```

bla latex code here.

Example

```
gap> S := KoszulDualRing( HomalgFieldOfRationalsInSingular()*"x,y,z" );;
gap> right_pre_category := RightPresentations( S );;
gap> m := HomalgMatrix( "[ [ e0, e1, e2 ],[ 0, 0, e0 ] ]", 2, 3, S );;
gap> M := AsRightPresentation( m );;
gap> F := FreeRightPresentation( 2, S );;
gap> f_matrix := HomalgMatrix( "[ [ e1, 0 ], [ 0, 1 ] ]",2, 2, S );;
gap> f := PresentationMorphism( F, f_matrix, M );;
gap> g := KernelEmbedding( f );;
gap> K := Source( g );;
gap> h := ZeroMorphism( M, K );;
gap> l := RepeatListZ( [ h, f, g ] );;
gap> C := ChainComplex( right_pre_category, l );;
gap> Display( C, 0, 1 );

-----------------------------------------------------------------
In index 0

Object is
e0,e1,e2,
0, 0, e0

An object in Category of right presentations of Q{e0,e1,e2}

Differential is
0,0,
0,0,
```

```
0,0

A zero morphism in Category of right presentations of Q{e0,e1,e2}

------------------------------------------------------------------
In index 1

Object is
(an empty 2 x 0 matrix)

An object in Category of right presentations of Q{e0,e1,e2}

Differential is
e1,0,
0, 1

A morphism in Category of right presentations of Q{e0,e1,e2}

----------------------------------
gap> C[ 2 ];;
gap> C^2;;
```

Chapter 2

Complexes morphisms

2.1 Categories and filters

2.1.1 IsChainOrCochainMorphism (for IsCapCategoryMorphism)

▷ `IsChainOrCochainMorphism(phi)` (filter)

Returns: `true` or `false`

bla bla

2.1.2 IsBoundedBelowChainOrCochainMorphism (for IsChainOrCochainMorphism)

▷ `IsBoundedBelowChainOrCochainMorphism(phi)` (filter)

Returns: `true` or `false`

bla bla

2.1.3 IsBoundedAboveChainOrCochainMorphism (for IsChainOrCochainMorphism)

▷ `IsBoundedAboveChainOrCochainMorphism(phi)` (filter)

Returns: `true` or `false`

bla bla

2.1.4 IsBoundedChainOrCochainMorphism (for IsBoundedBelowChainOrCochainMorphism and IsBoundedAboveChainOrCochainMorphism)

▷ `IsBoundedChainOrCochainMorphism(phi)` (filter)

Returns: `true` or `false`

bla bla

2.1.5 IsChainMorphism (for IsChainOrCochainMorphism)

▷ `IsChainMorphism(phi)` (filter)

Returns: `true` or `false`

bla bla

2.1.6 IsBoundedBelowChainMorphism (for IsBoundedBelowChainOrCochainMorphism and IsChainMorphism)

▷ `IsBoundedBelowChainMorphism(phi)` (filter)

Returns: `true` or `false`

bla bla

2.1.7 IsBoundedAboveChainMorphism (for IsBoundedAboveChainOrCochainMorphism and IsChainMorphism)

▷ `IsBoundedAboveChainMorphism(phi)` (filter)

Returns: `true` or `false`

bla bla

2.1.8 IsBoundedChainMorphism (for IsBoundedChainOrCochainMorphism and IsChainMorphism)

▷ `IsBoundedChainMorphism(phi)` (filter)

Returns: `true` or `false`

bla bla

2.1.9 IsCochainMorphism (for IsChainOrCochainMorphism)

▷ `IsCochainMorphism(phi)` (filter)

Returns: `true` or `false`

bla bla

2.1.10 IsBoundedBelowCochainMorphism (for IsBoundedBelowChainOrCochainMorphism and IsCochainMorphism)

▷ `IsBoundedBelowCochainMorphism(phi)` (filter)

Returns: `true` or `false`

bla bla

2.1.11 IsBoundedAboveCochainMorphism (for IsBoundedAboveChainOrCochainMorphism and IsCochainMorphism)

▷ `IsBoundedAboveCochainMorphism(phi)` (filter)

Returns: `true` or `false`

bla bla

2.1.12 IsBoundedCochainMorphism (for IsBoundedChainOrCochainMorphism and IsCochainMorphism)

▷ `IsBoundedCochainMorphism(phi)` (filter)

Returns: `true` or `false`

bla bla

2.2 Creating chain and cochain morphisms

2.2.1 ChainMorphism (for IsChainComplex, IsChainComplex, IsZList)

▷ `ChainMorphism(C, D, l)` (operation)

Returns: a chain morphism

The input is two chain complexes C,D and an infinite list l. The output is the chain morphism $\phi : C \to D$ defined by $\phi_i := l[i]$.

2.2.2 ChainMorphism (for IsChainComplex, IsChainComplex, IsDenseList, IsInt)

▷ `ChainMorphism(C, D, l, k)` (operation)

Returns: a chain morphism

The input is two chain complexes C,D, dense list l and an integer k. The output is the chain morphism $\phi : C \to D$ such that $\phi_k = l[1]$, $\phi_{k+1} = l[2]$, etc.

2.2.3 ChainMorphism (for IsDenseList, IsInt, IsDenseList, IsInt, IsDenseList, IsInt)

▷ `ChainMorphism(c, m, d, n, l, k)` (operation)

Returns: a chain morphism

The output is the chain morphism $\phi : C \to D$, where $C_m = c[1], C_{m+1} = c[2]$, etc. $D_n = d[1], D_{n+1} = d[2]$, etc. and $\phi_k = l[1]$, $\phi_{k+1} = l[2]$, etc.

2.2.4 CochainMorphism (for IsCochainComplex, IsCochainComplex, IsZList)

▷ `CochainMorphism(C, D, l)` (operation)

Returns: a cochain morphism

The input is two cochain complexes C,D and an infinite list l. The output is the cochain morphism $\phi : C \to D$ defined by $\phi_i := l[i]$.

2.2.5 CochainMorphism (for IsCochainComplex, IsCochainComplex, IsDenseList, IsInt)

▷ `CochainMorphism(C, D, l, k)` (operation)

Returns: a chain morphism

The input is two cochain complexes C,D, dense list l and an integer k. The output is the cochain morphism $\phi : C \to D$ such that $\phi^k = l[1]$, $\phi^{k+1} = l[2]$, etc.

2.2.6 CochainMorphism (for IsDenseList, IsInt, IsDenseList, IsInt, IsDenseList, IsInt)

▷ `CochainMorphism(c, m, d, n, l, k)` (operation)

Returns: a cochain morphism

The output is the cochain morphism $\phi : C \to D$, where $C^m = c[1], C^{m+1} = c[2]$, etc. $D^n = d[1], D^{n+1} = d[2]$, etc. and $\phi^k = l[1]$, $\phi^{k+1} = l[2]$, etc.

2.3 Attributes

2.3.1 Morphisms (for IsChainOrCochainMorphism)

▷ `Morphisms(phi)` (attribute)

Returns: infinite list

The output is morphisms of the chain or cochain morphism as an infinite list.

2.3.2 MappingCone (for IsChainOrCochainMorphism)

▷ `MappingCone(phi)` (attribute)

Returns: complex

The input a chain (resp. cochain) morphism $\phi : C \to D$. The output is its mapping cone chain (resp. cochain) complex $\mathrm{Cone}(\phi)$.

2.3.3 NaturalInjectionInMappingCone (for IsChainOrCochainMorphism)

▷ `NaturalInjectionInMappingCone(phi)` (attribute)

Returns: chain (resp. cochain) morphism

The input a chain (resp. cochain) morphism $\phi : C \to D$. The output is the natural injection $i : D \to \mathrm{Cone}\phi)$.

2.3.4 NaturalProjectionFromMappingCone (for IsChainOrCochainMorphism)

▷ `NaturalProjectionFromMappingCone(phi)` (attribute)

Returns: chain (resp. cochain) morphism

The input a chain (resp. cochain) morphism $\phi : C \to D$. The output is the natural projection $\pi : \mathrm{Cone}(\phi) \to C[u]$ where $u = -1$ if ϕ is chain morphism and $u = 1$ if ϕ is cochain morphism.

2.4 Operations

2.4.1 SetUpperBound (for IsChainOrCochainMorphism, IsInt)

▷ `SetUpperBound(phi, n)` (operation)

Returns: a side effect

The command sets an upper bound to the morphism ϕ. An upper bound of ϕ is an integer u with $\phi_{i \geq u} = 0$. The integer u will be called *active* upper bound of ϕ. If ϕ already has an active upper bound, say u', then u' will be replaced by u only if $u \leq u'$.

2.4.2 SetLowerBound (for IsChainOrCochainMorphism, IsInt)

▷ `SetLowerBound(phi, n)` (operation)

Returns: a side effect

The command sets an lower bound to the morphism ϕ. A lower bound of ϕ is an integer l with $\phi_{i \leq l} = 0$. The integer l will be called *active* lower bound of ϕ. If ϕ already has an active lower bound, say l', then l' will be replaced by l only if $l \geq l'$.

2.4.3 HasActiveUpperBound (for IsChainOrCochainMorphism)

▷ `HasActiveUpperBound(phi)` (operation)

Returns: true or false

The input is chain or cochain morphism ϕ. The output is `true` if an upper bound has been set to ϕ and `false` otherwise.

2.4.4 HasActiveLowerBound (for IsChainOrCochainMorphism)

▷ `HasActiveLowerBound(phi)` (operation)

Returns: true or false

The input is chain or cochain morphism ϕ. The output is `true` if a lower bound has been set to ϕ and `false` otherwise.

2.4.5 ActiveUpperBound (for IsChainOrCochainMorphism)

▷ `ActiveUpperBound(phi)` (operation)

Returns: an integer

The input is chain or cochain morphism. The output is its active upper bound if such has been set to ϕ. Otherwise we get error.

2.4.6 ActiveLowerBound (for IsChainOrCochainMorphism)

▷ `ActiveLowerBound(phi)` (operation)

Returns: an integer

The input is chain or cochain morphism. The output is its active lower bound if such has been set to ϕ. Otherwise we get error.

2.4.7 \[\] (for IsChainOrCochainMorphism, IsInt)

▷ `\[\](phi, n)` (operation)

Returns: an integer

The input is chain (resp. cochain) morphism and an integer n. The output is the component of ϕ in index n, i.e., ϕ_n(resp. ϕ^n).

Index