# SheafCohomology-OnToricVarieties

## A package to compute sheaf cohomology on toric varieties

2019.01.11

11 January 2019

**Martin Bies**

**Martin Bies**
Email: martin.bies@alumni.uni-heidelberg.de
Homepage: https://www.ulb.ac.be/sciences/ptm/pmif/people.html
Address: Physique Théorique et Mathématique
Université Libre de Bruxelles
Campus Plaine - CP 231
Building NO - Level 6 - Office O.6.111
1050 Brussels
Belgium

# Contents

# Chapter 1

# Introduction

## 1.1 What is the goal of the SheafCohomologyOnToricVarieties package?

*SheafCohomologyOnToricVarieties* provides data structures to compute sheaf cohomology on such varieties. The ultimate goal is to provide high-performance-algorithms for its computation. To this, our main theorem for the computation of sheaf cohomology is based on an idea of Gregory G. Smith (see math/0305214 and DOI: 10.4171/OWR/2013/25), which we combine with the powerful cohomCalg algorithm. Information on the latter can be found at https://arxiv.org/abs/1003.5217v3 and references therein.

# Chapter 2

# Computation of vanishing sets

## 2.1 GAP category for vanishing sets

### 2.1.1 IsVanishingSet (for IsObject)

▷ `IsVanishingSet(`*arg*`)` (filter)

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

The GAP category of vanishing sets formed from affine semigroups.

## 2.2 Constructors

### 2.2.1 VanishingSet (for IsToricVariety, IsList, IsInt)

▷ `VanishingSet(`*variety, L, d, i, or, s*`)` (operation)

**Returns:** a vanishing set

The argument is a toric variety, a list $L$ of AffineSemigroups and the cohomological index $i$. Alternatively a string $s$ can be used instead of $d$ to inform the user for which cohomology classes this set identifies the 'vanishing twists'.

### 2.2.2 VanishingSet (for IsToricVariety, IsList, IsString)

▷ `VanishingSet(`*arg1, arg2, arg3*`)` (operation)

## 2.3 Attributes

### 2.3.1 ListOfUnderlyingAffineSemigroups (for IsVanishingSet)

▷ `ListOfUnderlyingAffineSemigroups(`*V*`)` (attribute)

**Returns:** a list of affine semigroups

The argument is a vanishingSet $V$. We then return the underlying list of semigroups that form this vanishing set.

### 2.3.2 EmbeddingDimension (for IsVanishingSet)

▷ `EmbeddingDimension(`*V*`)` (attribute)

**Returns:** a non-negative integer

The argument is a vanishingSet $V$. We then return the embedding dimension of this vanishing set.

### 2.3.3 CohomologicalIndex (for IsVanishingSet)

▷ `CohomologicalIndex(`*V*`)` (attribute)

**Returns:** an integer between 0 and $dim\left(X_\Sigma\right)$

The argument is a vanishingSet $V$. This vanishing set identifies those $D \in Pic\left(X_\Sigma\right)$ such that $H^i\left(X_\Sigma, \mathcal{O}\left(D\right)\right) = 0$. We return the integer $i$.

### 2.3.4 CohomologicalSpecification (for IsVanishingSet)

▷ `CohomologicalSpecification(`*V*`)` (attribute)

**Returns:** a string

The argument is a vanishingSet $V$. This could for example identify those $D \in Pic\left(X_\Sigma\right)$ such that $H^i\left(X_\Sigma, \mathcal{O}\left(D\right)\right) = 0$ for all $i > 0$. If such a specification is known, it will be returned by this method.

### 2.3.5 AmbientToricVariety (for IsVanishingSet)

▷ `AmbientToricVariety(`*V*`)` (attribute)

The argument is a vanishingSet $V$. We return the toric variety to which this vanishing set belongs.

## 2.4 Property

### 2.4.1 IsFull (for IsVanishingSet)

▷ `IsFull(`*V*`)` (property)

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

The argument is a VanishingSet $V$. We then check if this vanishing set is empty.

## 2.5 Improved vanishing sets via cohomCalg

### 2.5.1 SHEAF_COHOMOLOGY_ON_TORIC_VARIETIES_INTERNAL_COHOMCALG_COMM (for IsToricVariety, IsList)

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

### 2.5.2 SHEAF_COHOMOLOGY_ON_TORIC_VARIETIES_INTERNAL_COHOMCALG_COMM (for IsToricVariety)

▷ `SHEAF_COHOMOLOGY_ON_TORIC_VARIETIES_INTERNAL_COHOMCALG_COMMAND_STRING(`*arg*`)` (operation)

### 2.5.3 ContributingDenominators (for IsToricVariety)

▷ `ContributingDenominators(`*arg*`)` (operation)

### 2.5.4 TurnDenominatorIntoShiftedSemigroup (for IsToricVariety, IsString)

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

### 2.5.5 VanishingSets (for IsToricVariety)

▷ `VanishingSets(`*arg*`)` (attribute)

### 2.5.6 ComputeVanishingSets (for IsToricVariety, IsBool)

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

### 2.5.7 PointContainedInVanishingSet (for IsVanishingSet, IsList)

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

## 2.6 Vanishing sets

We compute thev for a number of examples.

Example
```
gap> F1 := Fan( [[1],[-1]],[[1],[2]] );
<A fan in |R^1>
gap> P1 := ToricVariety( F1 );
<A toric variety of dimension 1>
gap> v1 := VanishingSets( P1 );
rec( 0 := <A non-full vanishing set in Z^1 for cohomological index 0>,
1 := <A non-full vanishing set in Z^1 for cohomological index 1> )
gap> Display( v1.0 );
A non-full vanishing set in Z^1 for cohomological index 0 formed from
the points NOT contained in the following affine semigroup:

A non-trivial affine cone-semigroup in Z^1
Offset: [ 0 ]
Hilbert basis: [ [ 1 ] ]
```

```
gap> Display( v1.1 );
A non-full vanishing set in Z^1 for cohomological index 1 formed from
the points NOT contained in the following affine semigroup:

A non-trivial affine cone-semigroup in Z^1
Offset: [ -2 ]
Hilbert basis: [ [ -1 ] ]
gap> P1xP1 := P1*P1;
<A smooth toric variety of dimension 2 which is a product of
2 toric varieties>
gap> v2 := VanishingSets( P1xP1 );
rec( 0 := <A non-full vanishing set in Z^2 for cohomological index 0>,
     1 := <A non-full vanishing set in Z^2 for cohomological index 1>,
     2 := <A non-full vanishing set in Z^2 for cohomological index 2> )
gap> Display( v2.0 );
A non-full vanishing set in Z^2 for cohomological index 0 formed from
the points NOT contained in the following affine semigroup:

A non-trivial affine cone-semigroup in Z^2
Offset: [ 0, 0 ]
Hilbert basis: [ [ 0, 1 ], [ 1, 0 ] ]
gap> Display( v2.1 );
A non-full vanishing set in Z^2 for cohomological index 1 formed from
the points NOT contained in the following 2 affine semigroups:

Affine semigroup 1:
A non-trivial affine cone-semigroup in Z^2
Offset: [ 0, -2 ]
Hilbert basis: [ [ 0, -1 ], [ 1, 0 ] ]

Affine semigroup 2:
A non-trivial affine cone-semigroup in Z^2
Offset: [ -2, 0 ]
Hilbert basis: [ [ 0, 1 ], [ -1, 0 ] ]
gap> Display( v2.2 );
A non-full vanishing set in Z^2 for cohomological index 2 formed from
the points NOT contained in the following affine semigroup:

A non-trivial affine cone-semigroup in Z^2
Offset: [ -2, -2 ]
Hilbert basis: [ [ 0, -1 ], [ -1, 0 ] ]
gap> P2 := ProjectiveSpace( 2 );
<A projective toric variety of dimension 2>
gap> v3 := VanishingSets( P2 );
rec( 0 := <A non-full vanishing set in Z^1 for cohomological index 0>,
     1 := <A full vanishing set in Z^1 for cohomological index 1>,
     2 := <A non-full vanishing set in Z^1 for cohomological index 2> )
gap> P2xP1xP1 := P2*P1*P1;
<A smooth toric variety of dimension 4 which is a product
of 3 toric varieties>
gap> v4 := VanishingSets( P2xP1xP1 );
rec( 0 := <A non-full vanishing set in Z^3 for cohomological index 0>,
     1 := <A non-full vanishing set in Z^3 for cohomological index 1>,
     2 := <A non-full vanishing set in Z^3 for cohomological index 2>,
     3 := <A non-full vanishing set in Z^3 for cohomological index 3>,
     4 := <A non-full vanishing set in Z^3 for cohomological index 4> )
gap> P := Polytope( [[ -2,2],[1,2],[2,1],[2,-2],[-2,-2]] );
<A polytope in |R^2>
gap> T := ToricVariety( P );
<A projective toric variety of dimension 2>
gap> v5 := VanishingSets( T );
rec( 0 := <A non-full vanishing set in Z^3 for cohomological index 0>,
     1 := <A non-full vanishing set in Z^3 for cohomological index 1>,
     2 := <A non-full vanishing set in Z^3 for cohomological index 2> )
gap> Display( v5.2 );
A non-full vanishing set in Z^3 for cohomological index 2 formed from
the points NOT contained in the following affine semigroup:

A non-trivial affine cone-semigroup in Z^3
Offset: [ -1, -2, -1 ]
Hilbert basis: [ [ 1, 0, -1 ], [ -1, 0, 0 ], [ -1, -1, 1 ], [ 0, -1, 0 ],
[ 0, 0, -1 ] ]
gap> H7 := Fan( [[0,1],[1,0],[0,-1],[-1,7]], [[1,2],[2,3],[3,4],[4,1]] );
<A fan in |R^2>
gap> H7 := ToricVariety( H7 );
<A toric variety of dimension 2>
gap> v6 := VanishingSets( H7 );
rec( 0 := <A non-full vanishing set in Z^2 for cohomological index 0>,
     1 := <A non-full vanishing set in Z^2 for cohomological index 1>,
     2 := <A non-full vanishing set in Z^2 for cohomological index 2> )
gap> H5 := Fan( [[-1,5],[0,1],[1,0],[0,-1]] ,[[1,2],[2,3],[3,4],[4,1]] );
<A fan in |R^2>
gap> H5 := ToricVariety( H5 );
<A toric variety of dimension 2>
gap> v7 := VanishingSets( H5 );
rec( 0 := <A non-full vanishing set in Z^2 for cohomological index 0>,
     1 := <A non-full vanishing set in Z^2 for cohomological index 1>,
     2 := <A non-full vanishing set in Z^2 for cohomological index 2> )
gap> PointContainedInVanishingSet( v1.0, [ 1 ] );
false
gap> PointContainedInVanishingSet( v1.0, [ 0 ] );
false
gap> PointContainedInVanishingSet( v1.0, [ -1 ] );
true
gap> PointContainedInVanishingSet( v1.0, [ -2 ] );
true
gap> rays := [ [1,0,0], [-1,0,0], [0,1,0], [0,-1,0], [0,0,1], [0,0,-1],
>           [2,1,1], [1,2,1], [1,1,2], [1,1,1] ];
[ [ 1, 0, 0 ], [ -1, 0, 0 ], [ 0, 1, 0 ], [ 0, -1, 0 ], [ 0, 0, 1 ], [ 0, 0, -1 ],
[ 2, 1, 1 ], [ 1, 2, 1 ], [ 1, 1, 2 ], [ 1, 1, 1 ] ]
gap> cones := [ [1,3,6], [1,4,6], [1,4,5], [2,3,6], [2,4,6], [2,3,5], [2,4,5],
>            [1,5,9], [3,5,8], [1,3,7], [1,7,9], [5,8,9], [3,7,8],
>            [7,9,10], [8,9,10], [7,8,10] ];
[ [ 1, 3, 6 ], [ 1, 4, 6 ], [ 1, 4, 5 ], [ 2, 3, 6 ], [ 2, 4, 6 ], [ 2, 3, 5 ],
  [ 2, 4, 5 ], [ 1, 5, 9 ], [ 3, 5, 8 ], [ 1, 3, 7 ], [ 1, 7, 9 ], [ 5, 8, 9 ],
  [ 3, 7, 8 ], [ 7, 9, 10 ], [ 8, 9, 10 ], [ 7, 8, 10 ] ]
gap> F := Fan( rays, cones );
<A fan in |R^3>
gap> T := ToricVariety( F );
<A toric variety of dimension 3>
gap> [ IsSmooth( T ), IsComplete( T ), IsProjective( T ) ];
[ true, true, false ]
gap> SRIdeal( T );
<A graded torsion-free (left) ideal given by 23 generators>
gap> v8 := VanishingSets( T );
rec( 0 := <A non-full vanishing set in Z^7 for cohomological index 0>,
     1 := <A non-full vanishing set in Z^7 for cohomological index 1>,
     2 := <A non-full vanishing set in Z^7 for cohomological index 2>,
     3 := <A non-full vanishing set in Z^7 for cohomological index 3> )
gap> Display( v8.3 );
A non-full vanishing set in Z^7 for cohomological index 3 formed from \
the points NOT contained in the following affine semigroup:

A non-trivial affine cone-semigroup in Z^7
Offset: [ -2, -2, -2, -2, -1, -3, -3 ]
Hilbert basis: [ [ 0, 0, -1, -1, -1, -1, -2 ], [ 0, -1, 0, -1, -1, -2,\
 -1 ], [ -1, 0, 0, 0, 0, 0, 0 ], [ -1, 0, 0, 1, 2, 1, 1 ], [ 0, -1, 0,\
 0, 0, 0, 0 ], [ 0, 0, -1, 0, 0, 0, 0 ], [ 0, 0, 0, -1, 0, 0, 0 ], [ 0\
, 0, 0, 0, -1, 0, 0 ], [ 0, 0, 0, 0, 0, -1, 0 ], [ 0, 0, 0, 0, 0, 0, -\
1 ] ]
```

# Chapter 3

# Truncations of Sfpgrmod to single degrees

## 3.1 DegreeXLayers of the Cox ring

### 3.1.1 Exponents (for IsToricVariety, IsList)

▷ `Exponents(`*`vari, degree`*`)` (operation)

**Returns:** a list of lists of integers

Given a smooth and complete toric variety and a list of integers (= degree) corresponding to an element of the class group of the variety, this method return a list of integer valued lists. These lists correspond to the exponents of the monomials of degree in the Cox ring of this toric variety.

### 3.1.2 MonomsOfCoxRingOfDegreeByNormaliz (for IsToricVariety, IsList)

▷ `MonomsOfCoxRingOfDegreeByNormaliz(`*`vari, degree`*`)` (operation)

**Returns:** a list

Given a smooth and complete toric variety and a list of integers (= degree) corresponding to an element of the class group of the variety, this method returns the list of all monomials in the Cox ring of the given degree. This method uses NormalizInterface.

### 3.1.3 DegreeXLayer (for IsToricVariety, IsList)

▷ `DegreeXLayer(`*`vari, degree`*`)` (operation)

**Returns:** a list

Given a smooth and complete toric variety and a list of integers (= degree) corresponding to an element of the class group of the variety, this method returns the list of all monomials in the Cox ring of the given degree. This method uses NormalizInterface.

### 3.1.4 DegreeXLayerVectorsAsColumnMatrices (for IsToricVariety, IsList, IsPosInt, IsPosInt)

▷ `DegreeXLayerVectorsAsColumnMatrices(`*`vari, degree, i, l`*`)` (operation)

**Returns:** a list of matrices

Given a smooth and complete toric variety, a list of integers (= degree) corresponding to an element of the class group of the variety and two non-negative integers i and l, this method returns a list of column matrices. The columns are of length l and have at position i the monoms of the Coxring of degree 'degree'.

## 3.2 GAP category of DegreeXLayerVectorSpaces(Morphisms)

### 3.2.1 IsDegreeXLayerVectorSpace (for IsObject)

▷ `IsDegreeXLayerVectorSpace(`*`object`*`)` (filter)

**Returns:** true or false

The GAP category for vector spaces that represent a degree layer of a f.p. graded module

### 3.2.2 IsDegreeXLayerVectorSpaceMorphism (for IsObject)

▷ `IsDegreeXLayerVectorSpaceMorphism(`*`object`*`)` (filter)

**Returns:** true or false

The GAP category for morphisms between vector spaces that represent a degree layer of a f.p. graded module

### 3.2.3 IsDegreeXLayerVectorSpacePresentation (for IsObject)

▷ `IsDegreeXLayerVectorSpacePresentation(`*`object`*`)` (filter)

**Returns:** true or false

The GAP category for (left) presentations of vector spaces that represent a degree layer of a f.p. graded module

### 3.2.4 IsDegreeXLayerVectorSpacePresentationMorphism (for IsObject)

▷ `IsDegreeXLayerVectorSpacePresentationMorphism(`*`object`*`)` (filter)

**Returns:** true or false

The GAP category for (left) presentation morphisms of vector spaces that represent a degree layer of a f.p. graded module

## 3.3 Constructors for DegreeXLayerVectorSpaces(Morphisms)

### 3.3.1 DegreeXLayerVectorSpace (for IsList, IsHomalgGradedRing, IsVectorSpaceObject, IsInt)

▷ `DegreeXLayerVectorSpace(`*`L, S, V, n`*`)` (operation)

**Returns:** a CAPCategoryObject

The arguments are a list of monomials $L$, a homalg graded ring $S$ (the Coxring of the variety in question)), a vector space $V$ and a non-negative integer $n$. $V$ is to be given as a vector space defined in the package 'LinearAlgebraForCAP'. This method then returns the corresponding DegreeXLayerVectorSpace.

### 3.3.2 DegreeXLayerVectorSpaceMorphism (for IsDegreeXLayerVectorSpace, IsVectorSpaceMorphism, IsDegreeXLayerVectorSpace)

▷ `DegreeXLayerVectorSpaceMorphism(L, S, V)` (operation)

**Returns:** a DegreeXLayerVectorSpaceMorphism

The arguments are a DegreeXLayerVectorSpace *source*, a vector space morphism $\varphi$ (as defined in 'LinearAlgebraForCAP') and a DegreeXLayerVectorSpace *range*. If $\varphi$ is a vector space morphism between the underlying vector spaces of *source* and *range* this method returns the corresponding DegreeXLayerVectorSpaceMorphism.

### 3.3.3 DegreeXLayerVectorSpacePresentation (for IsDegreeXLayerVectorSpaceMorphism)

▷ `DegreeXLayerVectorSpacePresentation(a)` (operation)

**Returns:** a DegreeXLayerVectorSpaceMorphism

The arguments is a DegreeXLayerVectorSpaceMorphism `a`. This method treats this morphism as a presentation, i.e. we are interested in the cokernel of the underlying morphism of vector spaces. The corresponding DegreeXLayerVectorSpacePresentation is returned.

### 3.3.4 DegreeXLayerVectorSpacePresentationMorphism (for IsDegreeXLayerVectorSpacePresentation, IsVectorSpaceMorphism, IsDegreeXLayerVectorSpacePresentation)

▷ `DegreeXLayerVectorSpacePresentationMorphism(source, \varphi, range)` (operation)

**Returns:** a DegreeXLayerVectorSpacePresentationMorphism

The arguments is a DegreeXLayerVectorSpacePresentation *source*, a vector space morphism $\varphi$ and a DegreeXLayerVectorSpacePresentation *range*. The corresponding DegreeXLayerVectorSpacePresentationMorphism is returned.

## 3.4 Attributes for DegreeXLayerVectorSpaces(Morphisms)

### 3.4.1 UnderlyingHomalgGradedRing (for IsDegreeXLayerVectorSpace)

▷ `UnderlyingHomalgGradedRing(V)` (attribute)

**Returns:** a homalg graded ring

The argument is a DegreeXLayerVectorSpace $V$. The output is the Coxring, in which this vector space is embedded via the generators (specified when installing $V$).

### 3.4.2 Generators (for IsDegreeXLayerVectorSpace)

▷ `Generators(V)` (attribute)

**Returns:** a list

The argument is a DegreeXLayerVectorSpace $V$. The output is the list of generators, that embed $V$ into the Coxring in question.

### 3.4.3 UnderlyingVectorSpaceObject (for IsDegreeXLayerVectorSpace)

▷ `UnderlyingVectorSpaceObject(V)` (attribute)

**Returns:** a VectorSpaceObject

The argument is a DegreeXLayerVectorSpace $V$. The output is the underlying vectorspace object (as defined in 'LinearAlgebraForCAP').

### 3.4.4 EmbeddingDimension (for IsDegreeXLayerVectorSpace)

▷ `EmbeddingDimension(V)` (attribute)

**Returns:** a VectorSpaceObject

The argument is a DegreeXLayerVectorSpace $V$. For $S$ its 'UnderlyingHomalgGradedRing' this vector space is embedded (via its generators) into $S^n$. The integer $n$ is the embedding dimension.

### 3.4.5 Source (for IsDegreeXLayerVectorSpaceMorphism)

▷ `Source(a)` (attribute)

**Returns:** a DegreeXLayerVectorSpace

The argument is a DegreeXLayerVectorSpaceMorphism $a$. The output is its source.

### 3.4.6 Range (for IsDegreeXLayerVectorSpaceMorphism)

▷ `Range(a)` (attribute)

**Returns:** a DegreeXLayerVectorSpace

The argument is a DegreeXLayerVectorSpaceMorphism $a$. The output is its range.

### 3.4.7 UnderlyingVectorSpaceMorphism (for IsDegreeXLayerVectorSpaceMorphism)

▷ `UnderlyingVectorSpaceMorphism(a)` (attribute)

**Returns:** a DegreeXLayerVectorSpace

The argument is a DegreeXLayerVectorSpaceMorphism $a$. The output is its range.

### 3.4.8 UnderlyingHomalgGradedRing (for IsDegreeXLayerVectorSpaceMorphism)

▷ `UnderlyingHomalgGradedRing(a)` (attribute)

**Returns:** a homalg graded ring

The argument is a DegreeXLayerVectorSpaceMorphism $a$. The output is the Coxring, in which the source and range of this is morphism are embedded.

### 3.4.9 UnderlyingDegreeXLayerVectorSpaceMorphism (for IsDegreeXLayerVectorSpacePresentation)

▷ `UnderlyingDegreeXLayerVectorSpaceMorphism(a)` (attribute)

**Returns:** a DegreeXLayerVectorSpaceMorphism

The argument is a DegreeXLayerVectorSpacePresentation $a$. The output is the underlying DegreeXLayerVectorSpaceMorphism

### 3.4.10 UnderlyingVectorSpaceObject (for IsDegreeXLayerVectorSpacePresentation)

▷ `UnderlyingVectorSpaceObject(a)` (attribute)

**Returns:** a VectorSpaceObject

The argument is a DegreeXLayerVectorSpacePresentation *a*. The output is the vector space object which is the cokernel of the underlying vector space morphism.

### 3.4.11 UnderlyingVectorSpaceMorphism (for IsDegreeXLayerVectorSpacePresentation)

▷ `UnderlyingVectorSpaceMorphism(a)` (attribute)

**Returns:** a VectorSpaceMorphism

The argument is a DegreeXLayerVectorSpacePresentation *a*. The output is the vector space morphism which defines the underlying morphism of DegreeXLayerVectorSpaces.

### 3.4.12 UnderlyingHomalgGradedRing (for IsDegreeXLayerVectorSpacePresentation)

▷ `UnderlyingHomalgGradedRing(a)` (attribute)

**Returns:** a homalg graded ring

The argument is a DegreeXLayerVectorSpacePresentation *a*. The output is the Coxring, in which the source and range of the underlying morphism of DegreeXLayerVectorSpaces are embedded.

### 3.4.13 UnderlyingVectorSpacePresentation (for IsDegreeXLayerVectorSpacePresentation)

▷ `UnderlyingVectorSpacePresentation(a)` (attribute)

**Returns:** a CAP presentation category object

The argument is a DegreeXLayerVectorSpacePresentation *a*. The output is the underlying vector space presentation.

### 3.4.14 Source (for IsDegreeXLayerVectorSpacePresentationMorphism)

▷ `Source(a)` (attribute)

**Returns:** a DegreeXLayerVectorSpacePresentation

The argument is a DegreeXLayerVectorSpacePresentationMorphism *a*. The output is its source.

### 3.4.15 Range (for IsDegreeXLayerVectorSpacePresentationMorphism)

▷ `Range(a)` (attribute)

**Returns:** a DegreeXLayerVectorSpacePresentation

The argument is a DegreeXLayerVectorSpacePresentationMorphism *a*. The output is its range.

### 3.4.16 UnderlyingHomalgGradedRing (for IsDegreeXLayerVectorSpacePresentationMorphism)

▷ `UnderlyingHomalgGradedRing(a)` (attribute)

**Returns:** a homalg graded ring

The argument is a DegreeXLayerVectorSpacePresentationMorphism $a$. The output is the underlying graded ring of its source.

### 3.4.17 UnderlyingVectorSpacePresentationMorphism (for IsDegreeXLayerVectorSpacePresentationMorphism)

▷ `UnderlyingVectorSpacePresentationMorphism(a)` (attribute)

**Returns:** a CAP presentation category morphism

The argument is a DegreeXLayerVectorSpacePresentationMorphism $a$. The output is the underlying vector space presentation morphism.

## 3.5 Convenient methods to display all information about vector space presentations and morphisms thereof

### 3.5.1 FullInformation (for IsDegreeXLayerVectorSpacePresentation)

▷ `FullInformation(p)` (operation)

**Returns:** detailed information about p

The argument is a DegreeXLayerVectorSpacePresentation $p$. This method displays $p$ in great detail.

### 3.5.2 FullInformation (for IsDegreeXLayerVectorSpacePresentationMorphism)

▷ `FullInformation(p)` (operation)

**Returns:** detailed information about p

The argument is a DegreeXLayerVectorSpacePresentationMorphism $p$. This method displays $p$ in great detail.

## 3.6 Truncations of projective graded modules (as defined in CAP) to a single degree

### 3.6.1 DegreeXLayerOfProjectiveGradedLeftOrRightModule (for IsToricVariety, IsCAPCategoryOfProjectiveGradedLeftOrRightModulesObject, IsList, IsHomalgRing)

▷ `DegreeXLayerOfProjectiveGradedLeftOrRightModule(V, M, degree_list)` (operation)

**Returns:** a DegreeXLayerVectorSpace

The arguments are a toric variety $V$, a projective graded $S$-module $M$ ($S$ being the Cox ring of $V$) and a `degree_list` specifying an element of the degree group of the toric variety $V$. The latter can either be specified by a list of integers or a HomalgModuleElement. Based on this input, the method computes the truncation of $M$ to the specified degree. We expect that $V$ is smooth and compact. Under these circumstances the truncation is a finite dimensional vector space and we return the corresponding DegreeXLayerVectorSpace.

### 3.6.2 DegreeXLayerOfProjectiveGradedLeftOrRightModuleGeneratorsAsListOfColumnMatrices (for IsToricVariety, IsCAPCategoryOfProjectiveGradedLeftOrRightModulesObject, IsList)

▷ `DegreeXLayerOfProjectiveGradedLeftOrRightModuleGeneratorsAsListOfColumnMatrices(`*V, M, degree_list*`)` (operation)

**Returns:** a formated DegreeXLayerVectorSpace

The arguments are as before, but will format the DegreeXLayer a bit more.

### 3.6.3 DegreeXLayerOfProjectiveGradedLeftOrRightModuleMorphism (for IsToricVariety, IsCAPCategoryOfProjectiveGradedLeftOrRightModulesMorphism, IsList, IsHomalgRing, IsBool)

▷ `DegreeXLayerOfProjectiveGradedLeftOrRightModuleMorphism(`*V, \varphi, degree_list*`)` (operation)

**Returns:** a DegreeXLayerVectorSpaceMorphism

The arguments are a toric variety $V$, a projective graded $S$-module morphism $\varphi$ ($S$ being the Cox ring of $V$) and a `degree_list` specifying an element of the degree group of the toric variety $V$. The latter can either be a list of integers or a HomalgModuleElement. Based on this input, the method returns the truncation of $\varphi$ to the specified degree. We expect that $V$ is smooth and compact. Under these circumstances the truncation is a morphism of finite dimensional vector spaces. We return the corresponding DegreeXLayerVectorSpaceMorphism.

## 3.7 DegreeXLayer of projective module morphism saved in file

### 3.7.1 ComputeDegreeXLayerOfProjectiveGradedLeftOrRightModuleMorphismMinimally (for IsList, IsBool)

▷ `ComputeDegreeXLayerOfProjectiveGradedLeftOrRightModuleMorphismMinimally(`*V, \varphi, degree_list, file_name*`)` (operation)

**Returns:** true or false

The arguments are a toric variety $V$, a projective graded $S$-module morphism $\varphi$ ($S$ being the Cox ring of $V$) and a `degree_list` specifying an element of the degree group of the toric variety $V$ and string 'file_name'. This method then computes the matrix encoding the DegreeXLayer of the given morphism of projective modules and saves it to the file 'file_name'. This file is prepared to be used with gap.

## 3.8 Truncation functor of projective graded modules (as defined in CAP) to a single degree

### 3.8.1 DegreeXLayerOfProjectiveGradedLeftModulesFunctor (for IsToricVariety, IsList)

▷ `DegreeXLayerOfProjectiveGradedLeftModulesFunctor(`*V, degree_list*`)` (operation)

**Returns:** a functor

The arguments are a toric variety $V$ and `degree_list` specifying an element of the degree group of the toric variety $V$. The latter can either be a list of integers or a HomalgModuleElement. Based on this input, the method returns the functor for the truncation of projective graded left-$S$-modules to *degree_list*.

### 3.8.2 DegreeXLayerOfProjectiveGradedRightModulesFunctor (for IsToricVariety, IsList)

▷ DegreeXLayerOfProjectiveGradedRightModulesFunctor(*V*, *degree_list*) (operation)

**Returns:** a functor

The arguments are a toric variety $V$ and *degree_list* specifying an element of the degree group of the toric variety $V$. The latter can either be a list of integers or a HomalgModuleElement. Based on this input, the method returns the functor for the truncation of projective graded right-$S$-modules to *degree_list*.

### 3.8.3 DegreeXLayerOfGradedLeftModulePresentationFunctor (for IsToricVariety, IsList, IsBool)

▷ DegreeXLayerOfGradedLeftModulePresentationFunctor(*V*, *degree_list*) (operation)

**Returns:** a functor

The arguments are a toric variety $V$ and *degree_list* specifying an element of the degree group of the toric variety $V$. The latter can either be a list of integers or a HomalgModuleElement. Based on this input, the method returns the functor for the truncation of graded left-$S$-module presentations to *degree_list*. Optionally, a boolean $b$ can be provided as fourth argument. It will display/suppress information on the status of the computation. $b = true$ will print information on the status of the computation, which might be useful in case the calculation takes several hours and the user wants to stay informed on the status of the computation. $b = false$ will suppress this output. The defaul value is false.

### 3.8.4 DegreeXLayerOfGradedRightModulePresentationFunctor (for IsToricVariety, IsList, IsBool)

▷ DegreeXLayerOfGradedRightModulePresentationFunctor(*V*, *degree_list*) (operation)

**Returns:** a functor

The arguments are a toric variety $V$ and *degree_list* specifying an element of the degree group of the toric variety $V$. The latter can either be a list of integers or a HomalgModuleElement. Based on this input, the method returns the functor for the truncation of graded right-$S$-module presentations to *degree_list*.

## 3.9 Truncations of graded module presentations (as defined in CAP) to a single degree

### 3.9.1 DegreeXLayerOfGradedLeftOrRightModulePresentation (for IsToricVariety, IsGradedLeftOrRightModulePresentationForCAP, IsList, IsBool)

▷ DegreeXLayerOfGradedLeftOrRightModulePresentation(*V*, *alpha*, *degree_list*) (operation)

**Returns:** a DegreeXLayerVectorSpacePresentation

The arguments are a toric variety $V$, a graded module presentation $\alpha$ and `degree_list` specifying an element of the degree group of the toric variety $V$. The latter can either be a list of integers or a HomalgModuleElement. Based on this input, the method returns the truncation of $\alpha$ to the specified degree. We expect that $V$ is smooth and compact. Under these circumstances the truncation is a finite dimensional vector space presentation. We return the corresponding DegreeXLayerVectorSpacePresentation. Optionally, a boolean $b$ can be provided as fourth argument. It will display/suppress information on the status of the computation. $b = true$ will print information on the status of the computation, which might be useful in case the calculation takes several hours and the user wants to stay informed on the status of the computation. $b = false$ will suppress this output. The defaul value is false.

### 3.9.2 DegreeXLayer (for IsToricVariety, IsGradedLeftOrRightModulePresentationForCAP, IsList, IsBool)

▷ `DegreeXLayer(V, alpha, degree_list)` (operation)

**Returns:** a CAPPresentationCategoryObject

The same as DegreeXLayerOfGradedLeftOrRightModulePresentation, but immediately returns the underlying vector space presentation.

### 3.9.3 DegreeXLayerOfGradedLeftOrRightModulePresentationMorphism (for IsToricVariety, IsGradedLeftOrRightModulePresentationMorphismForCAP, IsList, IsBool)

▷ `DegreeXLayerOfGradedLeftOrRightModulePresentationMorphism(V, alpha, degree_list)` (operation)

**Returns:** a DegreeXLayerVectorSpacePresentationMorphism

The arguments are a toric variety $V$, a graded module presentation morphism $\alpha$ and `degree_list` specifying an element of the degree group of the toric variety $V$. The latter can either be a list of integers or a HomalgModuleElement. Based on this input, the method returns the truncation of $\alpha$ to the specified degree. We expect that $V$ is smooth and compact. Under these circumstances the truncation is a morphism of finite dimensional vector space presentations. We return the corresponding DegreeXLayerVectorSpacePresentationMorphism. Optionally, a boolean $b$ can be provided as fourth argument. It will display/suppress information on the status of the computation. $b = true$ will print information on the status of the computation, which might be useful in case the calculation takes several hours and the user wants to stay informed on the status of the computation. $b = false$ will suppress this output. The defaul value is false.

### 3.9.4 DegreeXLayerOfGradedLeftOrRightModulePresentationMorphismWithGivenSourceAndRan (for IsToricVariety, IsGradedLeftOrRightModulePresentationMorphismForCAP, IsList,IsCAPPresentationCategoryObject, IsCAPPresentationCategoryObject, IsBool)

▷ `DegreeXLayerOfGradedLeftOrRightModulePresentationMorphismWithGivenSourceAndRange(V, alpha, degree_list, truncated_source, truncated_range)` (operation)

**Returns:** a DegreeXLayerVectorSpacePresentationMorphism

As 'DegreeXLayerOfGradedLeftOrRightModulePresentationMorphism', but takes the truncations of source and range as input.

### 3.9.5 DegreeXLayer (for IsToricVariety, IsGradedLeftOrRightModulePresentationMorphismForCAP, IsList, IsBool)

▷ `DegreeXLayer(`*`V, alpha, degree_list`*`)` (operation)

**Returns:** a CAPPresentationCategoryMorphism

The same as DegreeXLayerOfGradedLeftOrRightModulePresentationMorphism, but immediately returns the underlying vector space presentation morphism.

## 3.10 Truncations of graded module presentations as defined in the package GradedModules

### 3.10.1 DegreeXLayerOfProjectiveGradedLeftOrRightModuleGeneratorsAsListOfColumnMatrices (for IsToricVariety, IsGradedModuleOrGradedSubmoduleRep, IsList)

▷ `DegreeXLayerOfProjectiveGradedLeftOrRightModuleGeneratorsAsListOfColumnMatrices(`*`vari, module, degree`*`)` (operation)

**Returns:** a list of lists

Given a smooth and complete toric variety with Coxring S, a graded free S-module 'module' and list of integers (=degree) corresponding to an element of the class group of the toric variety, this method returns a list of generators of the degree layer of the 'module'.

### 3.10.2 DegreeXLayerOfFPGradedModuleForGradedModules (for IsToricVariety, IsGradedModuleOrGradedSubmoduleRep, IsList)

▷ `DegreeXLayerOfFPGradedModuleForGradedModules(`*`vari, module, degree`*`)` (operation)

**Returns:** a vector space

Given a smooth and complete toric variety with Coxring S, a f. p. graded S-module 'module' and list of integers (=degree) corresponding to an element of the class group of the toric variety, this method computes the degree 'degree' layer of (a) presentation morphisms of 'module' and returns the cokernel object of this homomorphism of vector spaces.

## 3.11 DegreeXLayerVectorSpaces: Examples

### 3.11.1 DegreeXLayerVectorSpaces of graded projective modules (and their morphisms) on P1xP1

Example

```
gap> P1 := ProjectiveSpace( 1 );
<A projective toric variety of dimension 1>
gap> P1xP1 := P1*P1;
<A projective toric variety of dimension 2 which is a
product of 2 toric varieties>
gap> ByASmallerPresentation( ClassGroup( P1xP1 ) );
<A free left module of rank 2 on free generators>
gap> S := CoxRing( P1xP1 );
Q[x_1,x_2,x_3,x_4]
(weights: [ ( 0, 1 ), ( 1, 0 ), ( 1, 0 ), ( 0, 1 ) ])
gap> sourceL := CAPCategoryOfProjectiveGradedLeftModulesObject(
>             [[[0,0],1]], S );
```

```
<A projective graded left module of rank 1>
gap> rangeL := CAPCategoryOfProjectiveGradedLeftModulesObject(
>           [[[-1,0],1]], S );
<A projective graded left module of rank 1>
gap> rangeL2 := CAPCategoryOfProjectiveGradedLeftModulesObject(
>            [[[-1,0],2]], S );
<A projective graded left module of rank 2>
gap> mappingL := CAPCategoryOfProjectiveGradedLeftOrRightModulesMorphism(
>             sourceL, HomalgMatrix( [[ "x_2" ]], S ), rangeL );
<A morphism in the category of projective graded left modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ], [ 0, 1 ] ])>
gap> mappingL2 := CAPCategoryOfProjectiveGradedLeftOrRightModulesMorphism(
>              sourceL, HomalgMatrix( [[ "x_2", "x_3" ]], S ), rangeL2 );
<A morphism in the category of projective graded left modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ], [ 0, 1 ] ])>
gap> res1L := DegreeXLayerOfProjectiveGradedLeftOrRightModule( P1xP1, sourceL, [0,0] );
<A vector space embedded into (Q[x_1,x_2,x_3,x_4] (with weights [ [ 0, 1 ],
[ 1, 0 ], [ 1, 0 ], [ 0, 1 ] ]))^1>
gap> Length( Generators( res1L ) );
1
gap> res2L := DegreeXLayerOfProjectiveGradedLeftOrRightModule( P1xP1, sourceL, [1,2] );
<A vector space embedded into (Q[x_1,x_2,x_3,x_4]
(with weights [ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ], [ 0, 1 ] ]))^1>
gap> Length( Generators( res2L ) );
6
gap> res3L := DegreeXLayerOfProjectiveGradedLeftOrRightModule( P1xP1, rangeL, [0,0] );
<A vector space embedded into (Q[x_1,x_2,x_3,x_4]
(with weights [ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ], [ 0, 1 ] ]))^1>
gap> Length( Generators( res3L ) );
2
gap> res4L := DegreeXLayerOfProjectiveGradedLeftOrRightModule( P1xP1, rangeL, [1,2] );
<A vector space embedded into (Q[x_1,x_2,x_3,x_4]
(with weights [ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ], [ 0, 1 ] ]))^1>
gap> Length( Generators( res4L ) );
9
gap> res5L := DegreeXLayerOfProjectiveGradedLeftOrRightModule( P1xP1, rangeL2, [0,0] );
<A vector space embedded into (Q[x_1,x_2,x_3,x_4]
(with weights [ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ], [ 0, 1 ] ]))^2>
gap> Length( Generators( res5L ) );
4
gap> res6L := DegreeXLayerOfProjectiveGradedLeftOrRightModule( P1xP1, rangeL2, [1,2] );
<A vector space embedded into (Q[x_1,x_2,x_3,x_4]
(with weights [ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ], [ 0, 1 ] ]))^2>
gap> Length( Generators( res6L ) );
18
gap> mor1L := UnderlyingVectorSpaceMorphism(
>          DegreeXLayerOfProjectiveGradedLeftOrRightModuleMorphism( P1xP1, mappingL, [ 0,0 ], tru
Starting the matrix computation now...

NrRows: 2
NrColumns: 1
Have to go until i = 1
```

```
100% done...
matrix created...
<A morphism in Category of matrices over Q>
gap> CokernelObject( mor1L );
<A vector space object over Q of dimension 1>
gap> mor2L := UnderlyingVectorSpaceMorphism(
>           DegreeXLayerOfProjectiveGradedLeftOrRightModuleMorphism( P1xP1, mappingL, [ 1,0 ], tru
Starting the matrix computation now...

NrRows: 3
NrColumns: 2
Have to go until i = 2
50% done...
100% done...
matrix created...
<A morphism in Category of matrices over Q>
gap> CokernelObject( mor2L );
<A vector space object over Q of dimension 1>
gap> mor3L := UnderlyingVectorSpaceMorphism(
>           DegreeXLayerOfProjectiveGradedLeftOrRightModuleMorphism( P1xP1, mappingL, [ 1,2 ], tru
Starting the matrix computation now...

NrRows: 9
NrColumns: 6
Have to go until i = 6
10% done...
30% done...
50% done...
60% done...
80% done...
100% done...
matrix created...
<A morphism in Category of matrices over Q>
gap> CokernelObject( mor3L );
<A vector space object over Q of dimension 3>
gap> mor4L := UnderlyingVectorSpaceMorphism(
>           DegreeXLayerOfProjectiveGradedLeftOrRightModuleMorphism( P1xP1, mappingL2, [ 0,0 ], tr
Starting the matrix computation now...

NrRows: 4
NrColumns: 1
Have to go until i = 1
100% done...
matrix created...
<A morphism in Category of matrices over Q>
gap> CokernelObject( mor4L );
<A vector space object over Q of dimension 3>
gap> mor5L := UnderlyingVectorSpaceMorphism(
>           DegreeXLayerOfProjectiveGradedLeftOrRightModuleMorphism( P1xP1, mappingL2, [ 1,0 ], tr
Starting the matrix computation now...

NrRows: 6
```

```
NrColumns: 2
Have to go until i = 2
50% done...
100% done...
matrix created...
<A morphism in Category of matrices over Q>
gap> CokernelObject( mor5L );
<A vector space object over Q of dimension 4>
gap> mor6L := UnderlyingVectorSpaceMorphism(
>          DegreeXLayerOfProjectiveGradedLeftOrRightModuleMorphism( P1xP1, mappingL2, [ 1,2 ], tr
Starting the matrix computation now...

NrRows: 18
NrColumns: 6
Have to go until i = 6
10% done...
30% done...
50% done...
60% done...
80% done...
100% done...
matrix created...
<A morphism in Category of matrices over Q>
gap> CokernelObject( mor6L );
<A vector space object over Q of dimension 12>
gap> sourceR := CAPCategoryOfProjectiveGradedRightModulesObject( [[[0,0],1]], S );
<A projective graded right module of rank 1>
gap> rangeR := CAPCategoryOfProjectiveGradedRightModulesObject( [[[-1,0],1]], S );
<A projective graded right module of rank 1>
gap> rangeR2 := CAPCategoryOfProjectiveGradedRightModulesObject( [[[-1,0],2]], S );
<A projective graded right module of rank 2>
gap> mappingR := CAPCategoryOfProjectiveGradedLeftOrRightModulesMorphism( sourceR, HomalgMatrix(
<A morphism in the category of projective graded right modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ], [ 0, 1 ] ])>
gap> mappingR2 := CAPCategoryOfProjectiveGradedLeftOrRightModulesMorphism( sourceR, HomalgMatrix(
<A morphism in the category of projective graded right modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ], [ 0, 1 ] ])>
gap> res1R := DegreeXLayerOfProjectiveGradedLeftOrRightModule( P1xP1, sourceL, [0,0] );
<A vector space embedded into (Q[x_1,x_2,x_3,x_4]
(with weights [ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ], [ 0, 1 ] ]))^1>
gap> Length( Generators( res1R ) );
1
gap> res2R := DegreeXLayerOfProjectiveGradedLeftOrRightModule( P1xP1, sourceL, [1,2] );
<A vector space embedded into (Q[x_1,x_2,x_3,x_4]
(with weights [ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ], [ 0, 1 ] ]))^1>
gap> Length( Generators( res2R ) );
6
gap> res3R := DegreeXLayerOfProjectiveGradedLeftOrRightModule( P1xP1, rangeL, [0,0] );
<A vector space embedded into (Q[x_1,x_2,x_3,x_4]
(with weights [ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ], [ 0, 1 ] ]))^1>
gap> Length( Generators( res3R ) );
2
```

```
gap> res4R := DegreeXLayerOfProjectiveGradedLeftOrRightModule( P1xP1, rangeL, [1,2] );
<A vector space embedded into (Q[x_1,x_2,x_3,x_4]
(with weights [ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ], [ 0, 1 ] ]))^1>
gap> Length( Generators( res4R ) );
9
gap> res5R := DegreeXLayerOfProjectiveGradedLeftOrRightModule( P1xP1, rangeL2, [0,0] );
<A vector space embedded into (Q[x_1,x_2,x_3,x_4]
(with weights [ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ], [ 0, 1 ] ]))^2>
gap> Length( Generators( res5R ) );
4
gap> res6R := DegreeXLayerOfProjectiveGradedLeftOrRightModule( P1xP1, rangeL2, [1,2] );
<A vector space embedded into (Q[x_1,x_2,x_3,x_4]
(with weights [ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ], [ 0, 1 ] ]))^2>
gap> Length( Generators( res6R ) );
18
gap> mor1R := UnderlyingVectorSpaceMorphism(
>           DegreeXLayerOfProjectiveGradedLeftOrRightModuleMorphism( P1xP1, mappingL, [ 0,0 ], tru
Starting the matrix computation now...

NrRows: 2
NrColumns: 1
Have to go until i = 1
100% done...
matrix created...
<A morphism in Category of matrices over Q>
gap> CokernelObject( mor1R );
<A vector space object over Q of dimension 1>
gap> mor2R := UnderlyingVectorSpaceMorphism(
>           DegreeXLayerOfProjectiveGradedLeftOrRightModuleMorphism( P1xP1, mappingL, [ 1,0 ], tru
Starting the matrix computation now...

NrRows: 3
NrColumns: 2
Have to go until i = 2
50% done...
100% done...
matrix created...
<A morphism in Category of matrices over Q>
gap> CokernelObject( mor2R );
<A vector space object over Q of dimension 1>
gap> mor3R := UnderlyingVectorSpaceMorphism(
>           DegreeXLayerOfProjectiveGradedLeftOrRightModuleMorphism( P1xP1, mappingL, [ 1,2 ], tru
Starting the matrix computation now...

NrRows: 9
NrColumns: 6
Have to go until i = 6
10% done...
30% done...
50% done...
60% done...
80% done...
```

```
100% done...
matrix created...
<A morphism in Category of matrices over Q>
gap> CokernelObject( mor3R );
<A vector space object over Q of dimension 3>
gap> mor4R := UnderlyingVectorSpaceMorphism(
>           DegreeXLayerOfProjectiveGradedLeftOrRightModuleMorphism( P1xP1, mappingL2, [ 0,0 ], tr
Starting the matrix computation now...

NrRows: 4
NrColumns: 1
Have to go until i = 1
100% done...
matrix created...
<A morphism in Category of matrices over Q>
gap> CokernelObject( mor4R );
<A vector space object over Q of dimension 3>
gap> mor5R := UnderlyingVectorSpaceMorphism(
>           DegreeXLayerOfProjectiveGradedLeftOrRightModuleMorphism( P1xP1, mappingL2, [ 1,0 ], tr
Starting the matrix computation now...

NrRows: 6
NrColumns: 2
Have to go until i = 2
50% done...
100% done...
matrix created...
<A morphism in Category of matrices over Q>
gap> CokernelObject( mor5R );
<A vector space object over Q of dimension 4>
gap> mor6R := UnderlyingVectorSpaceMorphism(
>           DegreeXLayerOfProjectiveGradedLeftOrRightModuleMorphism( P1xP1, mappingL2, [ 1,2 ], tr
Starting the matrix computation now...

NrRows: 18
NrColumns: 6
Have to go until i = 6
10% done...
30% done...
50% done...
60% done...
80% done...
100% done...
matrix created...
<A morphism in Category of matrices over Q>
gap> CokernelObject( mor6R );
<A vector space object over Q of dimension 12>
```

### 3.11.2 DegreeXLayerVectorSpaces of f.p. graded modules (and their morphisms) on P1xP1

Example

```
gap> obj1L := CAPPresentationCategoryObject( mappingL );
<A graded left module presentation over the ring Q[x_1,x_2,x_3,x_4]
(with weights [ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ], [ 0, 1 ] ])>
gap> deg1L := DegreeXLayerOfGradedLeftOrRightModulePresentation( P1xP1, obj1L, [ 0,0 ], true );
Starting the matrix computation now...

NrRows: 2
NrColumns: 1
Have to go until i = 1
100% done...
matrix created...
<A vector space embedded into (a suitable power of) Q[x_1,x_2,x_3,x_4]
 (with weights [ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ], [ 0, 1 ] ]) given as the
 cokernel of a vector space morphism>
gap> IsEqualForMorphisms( UnderlyingVectorSpaceMorphism( deg1L ), mor1L );
true
gap> UnderlyingVectorSpaceObject( deg1L );
<A vector space object over Q of dimension 1>
gap> obj2L := CAPPresentationCategoryObject( mappingL2 );
<A graded left module presentation over the ring Q[x_1,x_2,x_3,x_4]
(with weights [ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ], [ 0, 1 ] ])>
gap> deg2L := DegreeXLayerOfGradedLeftOrRightModulePresentation( P1xP1, obj2L, [ 3,4 ], true );
Starting the matrix computation now...

NrRows: 50
NrColumns: 20
Have to go until i = 20
0% done...
10% done...
20% done...
30% done...
40% done...
50% done...
60% done...
70% done...
80% done...
90% done...
100% done...
matrix created...
<A vector space embedded into (a suitable power of) Q[x_1,x_2,x_3,x_4]
(with weights [ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ], [ 0, 1 ] ]) given as the
cokernel of a vector space morphism>
gap> IsMonomorphism( UnderlyingVectorSpaceMorphism( deg2L ) );
true
gap> IsEpimorphism( UnderlyingVectorSpaceMorphism( deg2L ) );
false
gap> UnderlyingVectorSpaceObject( deg2L );
<A vector space object over Q of dimension 30>
gap> mappingL3 := CAPCategoryOfProjectiveGradedLeftOrRightModulesMorphism(
>               rangeL, HomalgMatrix( [ [ 1,1 ]], S ), rangeL2 );
```

```
<A morphism in the category of projective graded left modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ], [ 0, 1 ] ])>
gap> mappingL4 := CAPCategoryOfProjectiveGradedLeftOrRightModulesMorphism(
>               sourceL, HomalgMatrix( [ [ "x_2", "x_2" ]], S ), rangeL2 );
<A morphism in the category of projective graded left modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ], [ 0, 1 ] ])>
gap> presentation_morphismL := CAPPresentationCategoryMorphism(
>                            CAPPresentationCategoryObject( mappingL ), mappingL3,
>                            CAPPresentationCategoryObject( mappingL4 ) );
<A morphism of graded left module presentations over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ], [ 0, 1 ] ])>
gap> deg3L := DegreeXLayerOfGradedLeftOrRightModulePresentationMorphism(
>           P1xP1, presentation_morphismL, [ 0,0 ], true );
Starting the matrix computation now...

NrRows: 2
NrColumns: 1
Have to go until i = 1
100% done...
matrix created...
Starting the matrix computation now...

NrRows: 4
NrColumns: 2
Have to go until i = 2
50% done...
100% done...
matrix created...
Starting the matrix computation now...

NrRows: 4
NrColumns: 1
Have to go until i = 1
100% done...
matrix created...
<A vector space presentation morphism of vector spaces embedded into (
a suitable power of) Q[x_1,x_2,x_3,x_4] (with weights
[ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ], [ 0, 1 ] ]) and given as cokernels>
gap> vec3L := UnderlyingVectorSpacePresentationMorphism( deg3L );
<A morphism of the presentation category over the Category of matrices
over Q>
gap> IsMonomorphism( vec3L );
true
gap> IsEpimorphism( vec3L );
false
gap> ckL := CokernelObject( vec3L );
<An object of the presentation category over the Category of matrices over Q>
gap> CokernelObject( UnderlyingMorphism( ckL ) );
<A vector space object over Q of dimension 2>
gap> obj1R := CAPPresentationCategoryObject( mappingR );
<A graded right module presentation over the ring Q[x_1,x_2,x_3,x_4]
(with weights [ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ], [ 0, 1 ] ])>
```

```
gap> deg1R := DegreeXLayerOfGradedLeftOrRightModulePresentation( P1xP1, obj1R, [ 0,0 ], true );
Starting the matrix computation now...

NrRows: 2
NrColumns: 1
Have to go until i = 1
100% done...
matrix created...
<A vector space embedded into (a suitable power of) Q[x_1,x_2,x_3,x_4]
(with weights [ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ], [ 0, 1 ] ]) given as the
cokernel of a vector space morphism>
gap> IsEqualForMorphisms( UnderlyingVectorSpaceMorphism( deg1R ), mor1R );
true
gap> UnderlyingVectorSpaceObject( deg1R );
<A vector space object over Q of dimension 1>
gap> obj2R := CAPPresentationCategoryObject( mappingR2 );
<A graded right module presentation over the ring Q[x_1,x_2,x_3,x_4]
(with weights [ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ], [ 0, 1 ] ])>
gap> deg2R := DegreeXLayerOfGradedLeftOrRightModulePresentation( P1xP1, obj2R, [ 3,4 ], true );
Starting the matrix computation now...

NrRows: 50
NrColumns: 20
Have to go until i = 20
0% done...
10% done...
20% done...
30% done...
40% done...
50% done...
60% done...
70% done...
80% done...
90% done...
100% done...
matrix created...
<A vector space embedded into (a suitable power of) Q[x_1,x_2,x_3,x_4]
(with weights [ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ], [ 0, 1 ] ]) given as the
cokernel of a vector space morphism>
gap> IsMonomorphism( UnderlyingVectorSpaceMorphism( deg2R ) );
true
gap> IsEpimorphism( UnderlyingVectorSpaceMorphism( deg2R ) );
false
gap> UnderlyingVectorSpaceObject( deg2R );
<A vector space object over Q of dimension 30>
gap> mappingR3 := CAPCategoryOfProjectiveGradedLeftOrRightModulesMorphism(
>               rangeR, HomalgMatrix( [ [ 1 ], [ 1 ]], S ), rangeR2 );
<A morphism in the category of projective graded right modules over
Q[x_1,x_2,x_3,x_4] (with weights [ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ], [ 0, 1 ] ])>
gap> mappingR4 := CAPCategoryOfProjectiveGradedLeftOrRightModulesMorphism(
>               sourceR, HomalgMatrix( [ [ "x_2" ], [ "x_2" ]], S ), rangeR2 );
<A morphism in the category of projective graded right modules over
```

```
Q[x_1,x_2,x_3,x_4] (with weights [ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ], [ 0, 1 ] ])>
gap> presentation_morphismR := CAPPresentationCategoryMorphism(
>                              CAPPresentationCategoryObject( mappingR ), mappingR3,
>                              CAPPresentationCategoryObject( mappingR4 ) );
<A morphism of graded right module presentations over Q[x_1,x_2,x_3,x_4]
(with weights [ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ], [ 0, 1 ] ])>
gap> deg3R := DegreeXLayerOfGradedLeftOrRightModulePresentationMorphism(
>           P1xP1, presentation_morphismR, [ 0,0 ], true );
Starting the matrix computation now...

NrRows: 2
NrColumns: 1
Have to go until i = 1
100% done...
matrix created...
Starting the matrix computation now...

NrRows: 4
NrColumns: 2
Have to go until i = 2
50% done...
100% done...
matrix created...
Starting the matrix computation now...

NrRows: 4
NrColumns: 1
Have to go until i = 1
100% done...
matrix created...
<A vector space presentation morphism of vector spaces embedded into (
a suitable power of) Q[x_1,x_2,x_3,x_4] (with weights
[ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ], [ 0, 1 ] ]) and given as cokernels>
gap> vec3R := UnderlyingVectorSpacePresentationMorphism( deg3R );
<A morphism of the presentation category over the Category of matrices
 over Q>
gap> IsMonomorphism( vec3R );
true
gap> IsEpimorphism( vec3R );
false
gap> ckR := CokernelObject( vec3R );
<An object of the presentation category over the Category of matrices over Q>
gap> CokernelObject( UnderlyingMorphism( ckR ) );
<A vector space object over Q of dimension 2>
```

# Chapter 4

# Nef and Mori Cone

## 4.1 New Properties For Toric Divisors

### 4.1.1 IsNef (for IsToricDivisor)

▷ `IsNef(divi)` (property)

**Returns:** true or false

Checks if the torus invariant Weil divisor `divi` is nef.

### 4.1.2 IsAmpleViaNefCone (for IsToricDivisor)

▷ `IsAmpleViaNefCone(divi)` (property)

**Returns:** true or false

Checks if the class of the torus invariant Weil divisor `divi` lies in the interior of the nef cone. Given that the ambient toric variety is projective this implies that `divi` is ample.

## 4.2 Attributes

### 4.2.1 CartierDataGroup (for IsToricVariety)

▷ `CartierDataGroup(vari)` (attribute)

**Returns:** a list of lists

Given a toric variety `vari`, we compute the integral vectors in $\mathbb{Z}^{n|\Sigma_{max}|}$, $n$ is the rank of the character lattice which encodes a toric Cartier divisor according to theorem 4.2.8. in Cox-Schenk-Little. We return a list of lists. When interpreting this list of lists as a matrix, then the kernel of this matrix is the set of these vectors.

### 4.2.2 NefConeInCartierDataGroup (for IsToricVariety)

▷ `NefConeInCartierDataGroup(vari)` (attribute)

**Returns:** a list of lists

Given a smooth and complete toric variety `vari`, we compute the nef cone within the proper Cartier data in $\mathbb{Z}^{n|\Sigma_{max}|}$, $n$ is the rank of the character lattice. We return a list of ray generators of this cone.

### 4.2.3 NefConeInTorusInvariantWeilDivisorGroup (for IsToricVariety)

▷ `NefConeInTorusInvariantWeilDivisorGroup(`*vari*`)` (attribute)

**Returns:** a list of lists

Given a smooth and complete toric variety, we compute the nef cone within $Div_T(X_\Sigma)$. We return a list of ray generators of this cone.

### 4.2.4 NefConeInClassGroup (for IsToricVariety)

▷ `NefConeInClassGroup(`*vari*`)` (attribute)

**Returns:** a list of lists

Given a smooth and complete toric variety, we compute the nef cone within $Cl(X_\Sigma)$. We return a list of ray generators of this cone.

### 4.2.5 NefCone (for IsToricVariety)

▷ `NefCone(`*arg*`)` (attribute)

### 4.2.6 ClassesOfSmallestAmpleDivisors (for IsToricVariety)

▷ `ClassesOfSmallestAmpleDivisors(`*vari*`)` (attribute)

**Returns:** a list of integers

Given a smooth and complete toric variety, we compute the smallest divisor class, such that the associated divisor is ample. This is based on theorem 6.3.22 in Cox-Schenk-Little, which implies that this task is equivalent to finding the smallest lattice point within the nef cone (in $Cl(X_\Sigma)$). We return a list of integers encoding this lattice point.

### 4.2.7 GroupOfProper1Cycles (for IsToricVariety)

▷ `GroupOfProper1Cycles(`*vari*`)` (attribute)

**Returns:** a kernel submodule

Given a simplicial and complete toric variety, we use proposition 6.4.1 of Cox-Schenk-Litte to compute the group of proper 1-cycles. We return the corresponding kernel submodule.

### 4.2.8 MoriCone (for IsToricVariety)

▷ `MoriCone(`*vari*`)` (attribute)

**Returns:** an NmzCone6

Given a smooth and complete toric variety, we can compute both the intersection form and the Nef cone in the class group. Then the Mori cone is the dual cone of the Nef cone with respect to the intersection product. We compute an H-presentation of this dual cone and use those to produce a cone with the normaliz interface.

## 4.3 Nef and Mori Cone: Examples

### 4.3.1 Projective Space

Example

```
gap> P2 := ProjectiveSpace( 2 );
<A projective toric variety of dimension 2>
gap> P2xP2 := P2*P2;
<A projective toric variety of dimension 4 which is a product of 2 toric varieties>
gap> NefCone( P2 );
[ [ 1 ] ]
gap> NefCone( P2xP2 );
[ [ 0, 1 ], [ 1, 0 ] ]
gap> MoriCone( P2 );
[ [ 1 ] ]
gap> MoriCone( P2xP2 );
[ [ 0, 1 ], [ 1, 0 ] ]
gap> D1 := DivisorOfGivenClass( P2, [ -1 ] );
<A Cartier divisor of a toric variety with coordinates ( -1, 0, 0 )>
gap> IsAmpleViaNefCone( D1 );
false
gap> D2 := DivisorOfGivenClass( P2, [ 1 ] );
<A Cartier divisor of a toric variety with coordinates ( 1, 0, 0 )>
gap> IsAmpleViaNefCone( D2 );
true
gap> ClassesOfSmallestAmpleDivisors( P2 );
[ [ 1 ] ]
gap> ClassesOfSmallestAmpleDivisors( P2xP2 );
[ [ 1, 1 ] ]
```

# Chapter 5

# Irreducible, complete, torus-invariant curves and proper 1-cycles in a toric variety

## 5.1 GAP category of irreducible, complete, torus-invariant curves (= ICT curves)

### 5.1.1 IsICTCurve (for IsObject)

▷ `IsICTCurve(object)` (filter)

**Returns:** true or false

The GAP category for irreducible, complete, torus-invariant curves

## 5.2 Constructors for ICT Curves

### 5.2.1 ICTCurve (for IsToricVariety, IsInt, IsInt)

▷ `ICTCurve(X_Sigma, i, j)` (operation)

**Returns:** an ICT curve

The arguments are a smooth and complete toric variety $X_\Sigma$ and two non-negative and distinct integers $i, j$. We then consider the i-th and j-th maximal cones $\sigma_i$ and $\sigma_j$. ! If $\tau := \sigma_i \cap \sigma_j$ satisfies $dim(\tau) = dim(\sigma_1) - 1$, then $\tau$ corresponds to an ICT-curve. We then construct this very ICT-curve.

## 5.3 Attributes for ICT curves

### 5.3.1 AmbientToricVariety (for IsICTCurve)

▷ `AmbientToricVariety(C)` (attribute)

**Returns:** a toric variety

The argument is an ICT curve $C$. The output is the toric variety, in which this curve $C$ lies.

### 5.3.2 IntersectedMaximalCones (for IsICTCurve)

▷ `IntersectedMaximalCones(C)` (attribute)

**Returns:** a list of two positive and distinct integers

The argument is an ICT curve $C$. The output are two integers, which indicate which maximal rays were intersected to form the cone $\tau$ associated to this curve $C$.

### 5.3.3 RayGenerators (for IsICTCurve)

▷ `RayGenerators(C)` (attribute)

**Returns:** a list of lists of integers

The argument is an ICT curve $C$. The output is the list of ray-generators for the cone tau

### 5.3.4 DefiningVariables (for IsICTCurve)

▷ `DefiningVariables(C)` (attribute)

**Returns:** a list

The argument is an ICT curve $C$. The output is the list of variables whose simultaneous vanishing locus cuts out this curve.

### 5.3.5 StructureSheaf (for IsICTCurve)

▷ `StructureSheaf(C)` (attribute)

**Returns:** a f.p. graded module

The argument is an ICT curve $C$. The output is the f.p. graded S-module which sheafifes to the structure sheaf of this curve $C$.

### 5.3.6 IntersectionU (for IsICTCurve)

▷ `IntersectionU(C)` (attribute)

**Returns:** a list of integers

The argument is an ICT curve $C$. The output is the integral vector $u$ used to compute intersection products with Cartier divisors.

### 5.3.7 IntersectionList (for IsICTCurve)

▷ `IntersectionList(C)` (attribute)

**Returns:** a list of integers

The argument is an ICT curve $C$. The output is a list with the intersection numbers of a canonical base of the class group. This basis is to take $(e_1, \dots, e_k)$ with $e_i = (0, \dots, 0, 1, 0, \dots, 0) \in Cl(X_\Sigma)$.

## 5.4 Operations with ICTCurves

### 5.4.1 ICTCurves (for IsToricVariety)

▷ `ICTCurves(X_Sigma)` (attribute)

**Returns:** a list of ICT-curves.

For a smooth and complete toric variety $X_\Sigma$, this method computes a list of all ICT-curves in $X_\Sigma$. Note that those curves can be numerically equivalent.

### 5.4.2 IntersectionProduct (for IsICTCurve, IsToricDivisor)

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

**Returns:** an integer

Given an ICT-curve $C$ and a divisor $D$ in a smooth and complete toric variety $X_\Sigma$, this method computes their intersection product.

### 5.4.3 IntersectionProduct (for IsToricDivisor, IsICTCurve)

▷ `IntersectionProduct(arg1, arg2)` (operation)

## 5.5 GAP category for proper 1-cycles

### 5.5.1 IsProper1Cycle (for IsObject)

▷ `IsProper1Cycle(object)` (filter)

**Returns:** true or false

The GAP category for proper 1-cycles

## 5.6 Constructor For Proper 1-Cycles

### 5.6.1 GeneratorsOfProper1Cycles (for IsToricVariety)

▷ `GeneratorsOfProper1Cycles(X_Sigma)` (attribute)

**Returns:** a list of ICT-curves.

For a smooth and complete toric variety $X_\Sigma$, this method computes a list of all ICT-curves which are not numerically equivalent. We use this list of ICT-curves as a basis of proper 1-cycles on $X_\Sigma$ in the constructor below, when computing the intersection form and the Nef-cone.

### 5.6.2 Proper1Cycle (for IsToricVariety, IsList)

▷ `Proper1Cycle(X_Sigma, list)` (operation)

**Returns:** a proper 1-cycle

The arguments are a smooth and complete toric variety $X_\Sigma$ and a list of integers. We then use the integers in this list as 'coordinates' of the proper 1-cycle with respect to the list of proper 1-cycles produced by the previous method. We then return the corresponding proper 1-cycle.

## 5.7 Attributes for proper 1-cycles

### 5.7.1 AmbientToricVariety (for IsProper1Cycle)

▷ `AmbientToricVariety(C)` (attribute)

**Returns:** a toric variety

The argument is a proper 1-cycle $C$. The output is the toric variety, in which this cycle $C$ lies.

### 5.7.2 UnderlyingGroupElement (for IsProper1Cycle)

▷ `UnderlyingGroupElement(C)` (attribute)

**Returns:** a list

The argument is a proper 1-cycle. We then return the underlying group element (with respect to the generators computed from the method \emph{GeneratorsOfProper1Cycles}).

## 5.8 Operations with proper 1-cycles

### 5.8.1 IntersectionProduct (for IsProper1Cycle, IsToricDivisor)

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

**Returns:** an integer

Given a proper 1-cycle $C$ and a divisor $D$ in a smooth and complete toric variety $X_\Sigma$, this method computes their intersection product.

### 5.8.2 IntersectionProduct (for IsToricDivisor, IsProper1Cycle)

▷ `IntersectionProduct(arg1, arg2)` (operation)

### 5.8.3 IntersectionForm (for IsToricVariety)

▷ `IntersectionForm(vari)` (attribute)

**Returns:** a list of lists

Given a simplicial and complete toric variety, we can use proposition 6.4.1 of Cox-Schenk-Litte to compute the intersection form $N^1(X_\Sigma) \times N_1(X_\Sigma) \to \mathbb{R}$. We return a list of lists that encodes this mapping.

## 5.9 Irreducible, complete, torus-invariant curves and proper 1-cycles in a toric variety: Examples

### 5.9.1 Projective Space

Example
```
gap> P2 := ProjectiveSpace( 2 );
<A projective toric variety of dimension 2>
gap> ICTCurves( P2 );
[ <An irreducible, complete, torus-invariant curve in a toric variety given as V( [ x_3 ] )>,
  <An irreducible, complete, torus-invariant curve in a toric variety given as V( [ x_2 ] )>,
  <An irreducible, complete, torus-invariant curve in a toric variety given as V( [ x_1 ] )> ]
gap> C1 := ICTCurves( P2 )[ 1 ];
<An irreducible, complete, torus-invariant curve in a toric variety given as V( [ x_3 ] )>
gap> IntersectionForm( P2 );
[ [ 1 ] ]
gap> IntersectionProduct( C1, DivisorOfGivenClass( P2, [ 1 ] ) );
1
gap> IntersectionProduct( DivisorOfGivenClass( P2, [ 5 ] ), C1 );
5
```

# Chapter 6

# Sheaf cohomology computations on (direct products of) projective spaces

## 6.1 Specialised methods on CPn

### 6.1.1 H0OnProjectiveSpaceViaLinearRegularity (for IsToricVariety, IsGradedModuleOrGradedSubmoduleRep)

▷ `H0OnProjectiveSpaceViaLinearRegularity(vari, M)` (operation)

**Returns:** an integer

Given that variety is a complex projective space with Coxring $S$ and $M$ a f. p. graded S-module, this method computes the dimension of the vector space $H^0\left(X_\Sigma, \widetilde{M}\right)$ and returns this integer. To achieve this an integer $e$ is computed such that $H^0\left(\tilde{M}\right) \cong H^0_S\left(\mathfrak{m}^e, \tilde{M}\right)_0$. This integer $e$ in turn is computed by use of linear regularity of the module $M$.

### 6.1.2 H0OnProjectiveSpaceForAllPositiveTwistsViaLinearRegularity (for IsToricVariety, IsGradedModuleOrGradedSubmoduleRep)

▷ `H0OnProjectiveSpaceForAllPositiveTwistsViaLinearRegularity(vari, M)` (operation)

**Returns:** a function

Given that variety is a complex projective space with Coxring $S$ and $M$ a f. p. graded S-module, this method computes a function $f\colon \mathbb{N}_{\geq 0} \to \mathbb{N}_{\geq 0}$ such that $f(n) = dim\left(H^0\left(X_\Sigma, \widetilde{M}(n)\right)\right)$. This function is returned.

### 6.1.3 H0OnProjectiveSpaceInRangeViaLinearRegularity (for IsToricVariety, IsGradedModuleOrGradedSubmoduleRep, IsList)

▷ `H0OnProjectiveSpaceInRangeViaLinearRegularity(vari, M, range)` (operation)

**Returns:** a list of non-negative integers

Given that variety is a complex projective space with Coxring $S$ and $M$ a f. p. graded S-module, this method uses the function $f\colon \mathbb{N}_{\geq 0} \to \mathbb{N}_{\geq 0}$ computed by H0OnCPNForAllTwistsViaLinReg and evaluates it for all integers in range. The resulting list of non-negative integers is returned.

## 6.2 Experimental methods for computations on direct products on Pns implemented for the GradedModules Package

### 6.2.1 H0OnDirectProductsOfProjectiveSpaces (for IsToricVariety, IsGradedModuleOrGradedSubmoduleRep, IsBool)

▷ `H0OnDirectProductsOfProjectiveSpaces(vari, M, b)` (operation)

**Returns:** a vector space

Given that variety `vari` is a direct product of complex projective spaces with Coxring $S$ and that $M$ a f. p. graded S-module, this is an experimental method to compute sheaf cohomology. Note that all coherent sheaves on direct products of projective spaces have finite dimensional vector spaces as their cohomology groups. Thus this method returns such a vector space. A boolean $b$ can be specified as third argument. If $b = true$ we saturate the module before the core procedure is applied. The default value is 'false', i.e. do not saturate the module.

# Chapter 7

# Sheaf cohomology on (direct products of) projective spaces

## 7.1 Sheaf cohomology on direct product of projective spaces: Examples

### 7.1.1 Examples for graded modules on P1 and P1xP1

Example

```
gap> F1 := Fan( [[1],[-1]],[[1],[2]] );
<A fan in |R^1>
gap> P1 := ToricVariety( F1 );
<A toric variety of dimension 1>
gap> ByASmallerPresentation( ClassGroup( P1 ) );
<A free left module of rank 1 on a free generator>
gap> CoxRing( P1 );
Q[x_1,x_2]
(weights: [ 1, 1 ])
gap> C := GradedLeftSubmodule( [ "x_1", "x_2" ], CoxRing( P1 ) );
<A graded torsion-free (left) ideal given by 2 generators>
gap> P1xP1 := P1 * P1;
<A toric variety of dimension 2 which is a product of 2 toric varieties>
gap> ByASmallerPresentation( ClassGroup( P1xP1 ) );
<A free left module of rank 2 on free generators>
gap> C2 := GradedLeftSubmoduleForCAP( [[ "x_2*x_4" ], [ "x_2*x_3" ], [ "x_1 * x_4" ],
>       [ "x_1 * x_3" ]], CoxRing( P1xP1 ) );
<A graded left ideal of Q[x_1,x_2,x_3,x_4]
(with weights [ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ], [ 0, 1 ] ])>
gap> V := CAPCategoryOfProjectiveGradedLeftModulesObject(
>      [[[-1,-1],1],[[2,0],1]], CoxRing( P1xP1 ) );
<A projective graded left module of rank 2>
gap> V := ApplyFunctor( EmbeddingOfProjCategory( CapCategory( V ) ), V );
<A graded left module presentation over the ring Q[x_1,x_2,x_3,x_4]
(with weights [ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ], [ 0, 1 ] ])>
gap> VPrime := CAPCategoryOfProjectiveGradedLeftModulesObject( [[[-1,-1],1]], CoxRing( P1xP1 ) );
<A projective graded left module of rank 1>
gap> VPrime := ApplyFunctor( EmbeddingOfProjCategory( CapCategory( VPrime ) ), VPrime );
<A graded left module presentation over the ring Q[x_1,x_2,x_3,x_4]
(with weights [ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ], [ 0, 1 ] ])>
```

### 7.1.2 Examples for cohomology computations on P1

Note that these computations still rely on the GradedModules package and thus do not yet use CAP. This will hopefully change soon.

Example
```
gap> H0OnProjectiveSpaceViaLinearRegularity( P1, C );
1
gap> H0OnProjectiveSpaceInRangeViaLinearRegularity( P1, C, [ 0 .. 5 ] );
[ [ 0, 1 ], [ 1, 2 ], [ 2, 3 ], [ 3, 4 ], [ 4, 5 ], [ 5, 6 ] ]
```

# Chapter 8

# Sheaf cohomology via my cohomology theorem

## 8.1 Specialised InternalHomOnObjects methods

### 8.1.1 InternalHomDegreeZeroOnObjects (for IsToricVariety, IsGradedLeftOrRightModulePresentationForCAP,IsGradedLeftOrRightModulePresentationForCAP, IsBool)

▷ `InternalHomDegreeZeroOnObjects(`*arg1, arg2, arg3, arg4*`)` (operation)

### 8.1.2 ComputeInput (for IsToricVariety, IsCAPCategoryOfProjectiveGradedLeftOrRightModulesMorphism, IsList)

▷ `ComputeInput(`*arg1, arg2, arg3*`)` (operation)

### 8.1.3 InternalHomDegreeZeroOnObjectsParallel (for IsToricVariety, IsGradedLeftOrRightModulePresentationForCAP,IsGradedLeftOrRightModulePresentationForCAP, IsBool)

▷ `InternalHomDegreeZeroOnObjectsParallel(`*arg1, arg2, arg3, arg4*`)` (operation)

### 8.1.4 InternalHomEmbeddingDegreeZeroOnObjectsParallel (for IsToricVariety, IsGradedLeftOrRightModulePresentationForCAP,IsGradedLeftOrRightModulePresentationForCAP, IsBool, IsHomalgRing)

▷ `InternalHomEmbeddingDegreeZeroOnObjectsParallel(`*arg1, arg2, arg3, arg4, arg5*`)` (operation)

## 8.2 Specialised InternalHomOnMorphisms methods

### 8.2.1 InternalHomDegreeZeroOnMorphisms (for IsToricVariety, IsGradedLeftOrRightModulePresentationMorphismForCAP,IsGradedLeftOrRightModulePresentationMorphismForCAP, IsBool)

▷ `InternalHomDegreeZeroOnMorphisms(`*arg1, arg2, arg3, arg4*`)` (operation)

### 8.2.2 InternalHomDegreeZeroOnMorphismsParallel (for IsToricVariety, IsGradedLeftOrRightModulePresentationMorphismForCAP,IsGradedLeftOrRightModulePresentationMorphismForCAP, IsBool, IsHomalgRing)

▷ `InternalHomDegreeZeroOnMorphismsParallel(`*arg1, arg2, arg3, arg4, arg5*`)` (operation)

### 8.2.3 GradedExtDegreeZeroOnObjects (for IsInt, IsToricVariety, IsGradedLeftOrRightModulePresentationForCAP,IsGradedLeftOrRightModulePresentationForCAP, IsBool)

▷ `GradedExtDegreeZeroOnObjects(`*arg1, arg2, arg3, arg4, arg5*`)` (operation)

### 8.2.4 GradedExtDegreeZeroOnObjectsParallel (for IsInt, IsToricVariety, IsGradedLeftOrRightModulePresentationForCAP,IsGradedLeftOrRightModulePresentationForCAP, IsBool)

▷ `GradedExtDegreeZeroOnObjectsParallel(`*arg1, arg2, arg3, arg4, arg5*`)` (operation)

## 8.3 Computation of H0 by my theorem

### 8.3.1 H0 (for IsToricVariety, IsGradedLeftOrRightModulePresentationForCAP, IsBool, IsBool, IsBool)

▷ `H0(`*vari, M*`)` (operation)

**Returns:** a list consisting of an integer and a vector space

Computation of sheaf cohomology from my own theorem

### 8.3.2 H0 (for IsToricVariety, IsGradedLeftOrRightModulePresentationForCAP, IsBool, IsBool)

▷ `H0(`*arg1, arg2, arg3, arg4*`)` (operation)

### 8.3.3 H0 (for IsToricVariety, IsGradedLeftOrRightModulePresentationForCAP, IsBool)

▷ `H0(arg1, arg2, arg3)` (operation)

### 8.3.4 H0 (for IsToricVariety, IsGradedLeftOrRightModulePresentationForCAP)

▷ `H0(arg1, arg2)` (operation)

### 8.3.5 H0Parallel (for IsToricVariety, IsGradedLeftOrRightModulePresentationForCAP, IsBool, IsBool, IsBool)

▷ `H0Parallel(vari, M)` (operation)

**Returns:** a list consisting of an integer and a vector space

Computation of sheaf cohomology from my own theorem, but in parallel (by forking).

### 8.3.6 H0Parallel (for IsToricVariety, IsGradedLeftOrRightModulePresentationForCAP, IsBool, IsBool)

▷ `H0Parallel(arg1, arg2, arg3, arg4)` (operation)

### 8.3.7 H0Parallel (for IsToricVariety, IsGradedLeftOrRightModulePresentationForCAP, IsBool)

▷ `H0Parallel(arg1, arg2, arg3)` (operation)

### 8.3.8 H0Parallel (for IsToricVariety, IsGradedLeftOrRightModulePresentationForCAP)

▷ `H0Parallel(arg1, arg2)` (operation)

## 8.4 Computation of Hi by my theorem

### 8.4.1 Hi (for IsToricVariety, IsGradedLeftOrRightModulePresentationForCAP, IsInt, IsBool, IsBool, IsBool)

▷ `Hi(vari, M, i)` (operation)

**Returns:** a list consisting of an integer and a vector space

Computation of sheaf cohomology from my own theorem

### 8.4.2 Hi (for IsToricVariety, IsGradedLeftOrRightModulePresentationForCAP, IsInt, IsBool, IsBool)

▷ `Hi(arg1, arg2, arg3, arg4, arg5)` (operation)

### 8.4.3 Hi (for IsToricVariety, IsGradedLeftOrRightModulePresentationForCAP, IsInt, IsBool)

▷ `Hi(arg1, arg2, arg3, arg4)` (operation)

### 8.4.4 Hi (for IsToricVariety, IsGradedLeftOrRightModulePresentationForCAP, IsInt)

▷ `Hi(arg1, arg2, arg3)` (operation)

### 8.4.5 HiParallel (for IsToricVariety, IsGradedLeftOrRightModulePresentationForCAP, IsInt, IsBool, IsBool, IsBool)

▷ `HiParallel(vari, M)` (operation)

**Returns:** a list consisting of an integer and a vector space

Computation of sheaf cohomology from my own theorem, but in parallel (by forking).

### 8.4.6 HiParallel (for IsToricVariety, IsGradedLeftOrRightModulePresentationForCAP, IsInt, IsBool, IsBool)

▷ `HiParallel(arg1, arg2, arg3, arg4, arg5)` (operation)

### 8.4.7 HiParallel (for IsToricVariety, IsGradedLeftOrRightModulePresentationForCAP, IsInt, IsBool)

▷ `HiParallel(arg1, arg2, arg3, arg4)` (operation)

### 8.4.8 HiParallel (for IsToricVariety, IsGradedLeftOrRightModulePresentationForCAP, IsInt)

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

## 8.5 Computation of all sheaf cohomologies from my own theorem

### 8.5.1 AllHi (for IsToricVariety, IsGradedLeftOrRightModulePresentationForCAP, IsBool, IsBool)

▷ `AllHi(vari, M)` (operation)

**Returns:** a list of lists, which in turn consist each of an integer and a vector space

Computation of all sheaf cohomology classes from my theorem

### 8.5.2 AllHi (for IsToricVariety, IsGradedLeftOrRightModulePresentationForCAP, IsBool)

▷ `AllHi(`*`arg1, arg2, arg3`*`)` (operation)

### 8.5.3 AllHi (for IsToricVariety, IsGradedLeftOrRightModulePresentationForCAP)

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

### 8.5.4 AllHiParallel (for IsToricVariety, IsGradedLeftOrRightModulePresentationForCAP, IsBool, IsBool)

▷ `AllHiParallel(`*`vari, M`*`)` (operation)

**Returns:** a list of lists, which in turn consist each of an integer and a vector space

Computation of all sheaf cohomology classes from my theorem, but by use of parallelisation.

### 8.5.5 AllHiParallel (for IsToricVariety, IsGradedLeftOrRightModulePresentationForCAP, IsBool)

▷ `AllHiParallel(`*`arg1, arg2, arg3`*`)` (operation)

### 8.5.6 AllHiParallel (for IsToricVariety, IsGradedLeftOrRightModulePresentationForCAP)

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

## 8.6 Write matrices to be used for H0 computation via GS to files

### 8.6.1 H0ByGSWritingMatricesUsedByFastInternalHomToFilesForCAP (for IsToricVariety, IsGradedLeftOrRightModulePresentationForCAP, IsBool)

▷ `H0ByGSWritingMatricesUsedByFastInternalHomToFilesForCAP(`*`arg1, arg2, arg3`*`)` (operation)

# Chapter 9

# Sheaf cohomology via my theorem

## 9.1 Sheaf cohomology via my theorem: Examples

### 9.1.1 Examples for PresentationsByProjectiveGradedModules on P1 and P1xP1

Example

```
gap> F1 := Fan( [[1],[-1]],[[1],[2]] );
<A fan in |R^1>
gap> P1 := ToricVariety( F1 );
<A toric variety of dimension 1>
gap> ByASmallerPresentation( ClassGroup( P1 ) );
<A free left module of rank 1 on a free generator>
gap> CoxRing( P1 );
Q[x_1,x_2]
(weights: [ 1, 1 ])
gap> CForCAP := GradedLeftSubmoduleForCAP( [[ "x_1" ], ["x_2" ]], CoxRing( P1 ) );
<A graded left ideal of Q[x_1,x_2] (with weights [ 1, 1 ])>
gap> P1xP1 := P1 * P1;
<A toric variety of dimension 2 which is a product of 2 toric varieties>
gap> ByASmallerPresentation( ClassGroup( P1xP1 ) );
<A free left module of rank 2 on free generators>
gap> C2ForCAP := GradedLeftSubmoduleForCAP( [[ "x_2*x_4" ], [ "x_2*x_1" ], [ "x_3 * x_4" ],
>         [ "x_1 * x_3" ]], CoxRing( P1xP1 ) );
<A graded left ideal of Q[x_1,x_2,x_3,x_4]
(with weights [ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ], [ 0, 1 ] ])>
gap> VForCAP := CAPCategoryOfProjectiveGradedLeftModulesObject(
>       [[[-1,-1],1],[[2,0],1]], CoxRing( P1xP1 ) );
<A projective graded left module of rank 2>
gap> VForCAP := ApplyFunctor( EmbeddingOfProjCategory( CapCategory( VForCAP ) ), VForCAP );
<A graded left module presentation over the ring Q[x_1,x_2,x_3,x_4]
(with weights [ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ], [ 0, 1 ] ])>
gap> VPrimeForCAP := CAPCategoryOfProjectiveGradedLeftModulesObject( [[[-1,-1],1]], CoxRing( P1xP
<A projective graded left module of rank 1>
gap> VPrimeForCAP := ApplyFunctor( EmbeddingOfProjCategory( CapCategory( VPrimeForCAP ) ), VPrime
<A graded left module presentation over the ring Q[x_1,x_2,x_3,x_4]
(with weights [ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ], [ 0, 1 ] ])>
gap> Hi( P1, CForCAP, 0, false, false, false );
[ 1, <A vector space object over Q of dimension 1> ]
gap> Hi( P1xP1, C2ForCAP, 0, false, false, false );
```

```
[ 1, <A vector space object over Q of dimension 1> ]
gap> Hi( P1xP1, VForCAP, 0, false, false, false );
[ 0, <A vector space object over Q of dimension 4> ]
gap> Hi( P1xP1, VPrimeForCAP, 0, false, false, false );
[ 0, <A vector space object over Q of dimension 4> ]
gap> AllHi( P1, CForCAP, false, false );
Computing h^0
----------------------------------------------

Computing h^1
----------------------------------------------

[ [ 1, <A vector space object over Q of dimension 1> ],
  [ 1, <A vector space object over Q of dimension 0> ] ]
gap> AllHi( P1xP1, C2ForCAP, false, false );
Computing h^0
----------------------------------------------

Computing h^1
----------------------------------------------

Computing h^2
----------------------------------------------

[ [ 1, <A vector space object over Q of dimension 1> ],
  [ 1, <A vector space object over Q of dimension 0> ],
  [ 0, <A vector space object over Q of dimension 0> ] ]
gap> AllHi( P1xP1, VForCAP, false, false );
Computing h^0
----------------------------------------------

Computing h^1
----------------------------------------------

Computing h^2
----------------------------------------------

[ [ 0, <A vector space object over Q of dimension 4> ],
  [ 1, <A vector space object over Q of dimension 1> ],
  [ 1, <A vector space object over Q of dimension 0> ] ]
gap> AllHi( P1xP1, VPrimeForCAP, false, false );
Computing h^0
----------------------------------------------

Computing h^1
----------------------------------------------

Computing h^2
----------------------------------------------

[ [ 0, <A vector space object over Q of dimension 4> ],
  [ 0, <A vector space object over Q of dimension 0> ],
    [ 0, <A vector space object over Q of dimension 0> ] ]
```

# Chapter 10

# Cohomology of vector bundles on (smooth and compact) toric varieties from cohomCalg

## 10.1 All sheaf cohomologies of vector bundles from cohomCalg

### 10.1.1 AllCohomologiesFromCohomCalg (for IsToricVariety, IsGradedLeftOrRightModulePresentationForCAP, IsBool)

▷ `AllCohomologiesFromCohomCalg(`*V, M, B*`)` (operation)

**Returns:** a list of vector spaces

The arguments are a toric variety *V* with Coxring *S* and a f.p. graded S-module *M*. Given that *M* is free, i.e. `\tilde{M}` is a vector bundle on *V*, then this method computes all cohomology classes of this vector bundle. A boolean *B* can be specified as third argument. If *B* equals true, then a list of vector spaces is produced, otherwise only the dimensions of these vector spaces are returned. The default value is false.

# Chapter 11

# Cohomology of coherent sheaves from resolution

## 11.1 Mapping between the cohomology classes computed by the theorem of GS

### 11.1.1 CohomologiesList (for IsToricVariety, IsGradedLeftOrRightModulePresentationForCAP)

▷ `CohomologiesList(vari, M)` (operation)

**Returns:** a list of lists of integers

Given a smooth and projective toric variety *vari* with Coxring $S$ and a f. p. graded S-modules $M$, this method computes a minimal free resolution of $M$ and then the dimension of the cohomology classes of the projective modules in this minimal free resolution.

## 11.2 Deductions On Sheaf Cohomology From Cohomology Of projective modules in a minimal free resolution

### 11.2.1 DeductionOfSheafCohomologyFromResolution (for IsToricVariety, IsGradedLeftOrRightModulePresentationForCAP, IsBool)

▷ `DeductionOfSheafCohomologyFromResolution(vari, M)` (operation)

**Returns:** a list

Given a smooth and projective toric variety *vari* with Coxring $S$ and a f. p. graded S-modules $M$, this method computes a minimal free resolution of $M$ and then the dimension of the cohomology classes of the projective modules in this minimal free resolution. From this information we draw conclusions on the sheaf cohomologies of the sheaf $\tilde{M}$.

# Chapter 12

# Maps In Resolution

## 12.1 Compute maps in minimal free resolution of a sheaf

### 12.1.1 MapsInResolution (for IsToricVariety, IsGradedLeftOrRightModulePresentationForCAP)

▷ `MapsInResolution(`*V, M*`)` (operation)

**Returns:** a list of vector space morphisms

The arguments are a toric variety $V$ and a f.p. graded module presentation $M$ over the Cox ring of this variety.

### 12.1.2 InducedCohomologyMaps (for IsToricVariety, IsCAPCategoryOfProjectiveGradedLeftModulesMorphism)

▷ `InducedCohomologyMaps(`*V, m*`)` (operation)

**Returns:** the list of induced vector space morphisms between the sheaf cohomologies of the associated vector bundles

The arguments are a toric variety $V$ and a map of projective graded module presentation $m$ over the Cox ring of this variety.

# Index