SheafCohomologyOnToricVarieties

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

Tools for FPGradedModules

2.1 Minimal free resolutions

2.1.1 LeftIdealForCAP (for IsList, IsHomalgGradedRing)

▷ `LeftIdealForCAP(L, R)` (operation)

Returns: a f.p. module presentation

The argument is a list L of generators of an ideal and a homalg graded ring R. This method then construct the left ideal in this ring generated by these generators.

2.1.2 RightIdealForCAP (for IsList, IsHomalgGradedRing)

▷ `RightIdealForCAP(L, R)` (operation)

Returns: a f.p. module presentation

The argument is a list L of generators of an ideal and a homalg graded ring R. This method then construct the right ideal in this ring generated by these generators.

2.1.3 MinimalFreeResolutionForCAP (for IsFpGradedLeftOrRightModulesObject)

▷ `MinimalFreeResolutionForCAP(M)` (attribute)

Returns: a complex of projective graded module morphisms

The argument is a graded left or right module presentation `M`. We then compute a minimal free resolution of `M`.

2.2 Betti tables

2.2.1 BettiTableForCAP (for IsFpGradedLeftOrRightModulesObject)

▷ `BettiTableForCAP(M)` (attribute)

Returns: a list of lists

The argument is a graded left or right module presentation `M`. We then compute the Betti table of `M`.

2.3 Example: Minimal free resolution and Betti table

Example

```
gap> P2 := ProjectiveSpace( 2 );
<A projective toric variety of dimension 2>
gap> IR := IrrelevantLeftIdealForCAP( P2 );;
gap> IsWellDefined( IR );
true
gap> resolution := MinimalFreeResolutionForCAP( IR );
<An object in Complex category of Category of graded
rows over Q[x_1,x_2,x_3] (with weights [ 1, 1, 1 ])>
gap> differential_function :=
>                     UnderlyingZFunctorCell( resolution )!.differential_func;
function( i ) ... end
gap> IsWellDefined( differential_function( -1 ) );
true
gap> IsWellDefined( differential_function( -2 ) );
true
gap> IsWellDefined( differential_function( -3 ) );
true
gap> BT := BettiTableForCAP( IR );
[ [ -1, -1, -1 ], [ -2, -2, -2 ], [ -3 ] ]
```

Chapter 3

Additional methods and properties for toric varieties

3.1 Input check for cohomology computations

3.1.1 IsValidInputForCohomologyComputations (for IsToricVariety)

▷ `IsValidInputForCohomologyComputations(`*V*`)` (property)

Returns: `true` or `false`

Returns if the given variety V is a valid input for cohomology computations. If the variable SHEAF_COHOMOLOGY_ON_TORIC_VARIETIES_INTERNAL_LAZY is set to false (default), then we just check if the variety is smooth, complete. In case of success we return true and false otherwise. If however SHEAF_COHOMOLOGY_ON_TORIC_VARIETIES_INTERNAL_LAZY is set to true, then we will check if the variety is smooth, complete or simplicial, projective. In case of success we return true and false other.

3.2 Stanley-Reisner and irrelevant ideal via FpGradedModules

3.2.1 GeneratorsOfIrrelevantIdeal (for IsToricVariety)

▷ `GeneratorsOfIrrelevantIdeal(`*vari*`)` (attribute)

Returns: a list

Returns the lift of generators of the irrelevant ideal of the variety *vari*.

3.2.2 IrrelevantLeftIdealForCAP (for IsToricVariety)

▷ `IrrelevantLeftIdealForCAP(`*vari*`)` (attribute)

Returns: a graded left ideal for CAP

Returns the irrelevant left ideal of the Cox ring of the variety *vari*, using the language of CAP.

3.2.3 IrrelevantRightIdealForCAP (for IsToricVariety)

▷ `IrrelevantRightIdealForCAP(`*vari*`)` (attribute)

Returns: a graded right ideal for CAP

Returns the irrelevant right ideal of the Cox ring of the variety *vari*, using the language of CAP.

3.2.4 GeneratorsOfSRIdeal (for IsToricVariety)

▷ `GeneratorsOfSRIdeal(vari)` (attribute)

Returns: a list

Returns the lift of generators of the Stanley-Reisner-ideal of the variety *vari*.

3.2.5 SRLeftIdealForCAP (for IsToricVariety)

▷ `SRLeftIdealForCAP(vari)` (attribute)

Returns: a graded left ideal for CAP

Returns the Stanley-Reißner left ideal of the Cox ring of the variety *vari*, using the langauge of CAP.

3.2.6 SRRightIdealForCAP (for IsToricVariety)

▷ `SRRightIdealForCAP(vari)` (attribute)

Returns: a graded right ideal for CAP

Returns the Stanley-Reißner right ideal of the Cox ring of the variety *vari*, using the langauge of CAP.

3.2.7 FpGradedLeftModules (for IsToricVariety)

▷ `FpGradedLeftModules(variety)` (attribute)

Returns: a CapCategory

Given a toric variety *variety* one can consider the Cox ring S of this variety, which is graded over the class group of *variety*. Subsequently one can consider the category of f.p. graded left S-modules. This attribute captures the corresponding CapCategory.

3.2.8 FpGradedRightModules (for IsToricVariety)

▷ `FpGradedRightModules(variety)` (attribute)

Returns: a CapCategory

Given a toric variety *variety* one can consider the Cox ring S of this variety, which is graded over the class group of *variety*. Subsequently one can consider the category of f.p. graded right S-modules. This attribute captures the corresponding CapCategory.

3.3 Monoms of given degree in the Cox ring

3.3.1 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.

3.3.2 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.

3.3.3 MonomsOfCoxRingOfDegreeByNormalizAsColumnMatrices (for IsToricVariety, IsList, IsPosInt, IsPosInt)

▷ `MonomsOfCoxRingOfDegreeByNormalizAsColumnMatrices(`*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'.

3.4 Example: Stanley-Reisner ideal for CAP

Example

```
gap> P2 := ProjectiveSpace( 2 );
<A projective toric variety of dimension 2>
gap> SR1 := SRLeftIdealForCAP( P2 );;
gap> IsWellDefined( SR1 );
true
gap> SR2 := SRRightIdealForCAP( P2 );;
gap> IsWellDefined( SR2 );
true
```

3.5 Example: Irrelevant ideal for CAP

Example

```
gap> P2 := ProjectiveSpace( 2 );
<A projective toric variety of dimension 2>
gap> IR1 := IrrelevantLeftIdealForCAP( P2 );;
gap> IsWellDefined( IR1 );
true
gap> IR2 := IrrelevantRightIdealForCAP( P2 );;
gap> IsWellDefined( IR2 );
true
```

3.6 Example: Monomials of given degree

Example

```
gap> P1 := ProjectiveSpace( 1 );
<A projective toric variety of dimension 1>
gap> var := P1*P1;
<A projective toric variety of dimension 2
which is a product of 2 toric varieties>
```

```
gap> Exponents( var, [ 1,1 ] );
[ [ 1, 1, 0, 0 ], [ 1, 0, 1, 0 ],
[ 0, 1, 0, 1 ], [ 0, 0, 1, 1 ] ]
gap> MonomsOfCoxRingOfDegreeByNormaliz( var, [1,2] );
[ x_1^2*x_2, x_1^2*x_3, x_1*x_2*x_4,
x_1*x_3*x_4, x_2*x_4^2, x_3*x_4^2 ]
gap> MonomsOfCoxRingOfDegreeByNormaliz( var, [-1,-1] );
[]
gap> l := MonomsOfCoxRingOfDegreeByNormalizAsColumnMatrices
>      ( var, [1,2], 2, 3 );;
gap> Display( l[ 1 ] );
0,
x_1^2*x_2,
0
(over a graded ring)
```

Chapter 4

Cohomology of coherent sheaves from resolution

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

4.1.1 CohomologiesList (for IsToricVariety, IsFpGradedLeftOrRightModulesObject)

▷ `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.

4.1.2 DeductionOfSheafCohomologyFromResolution (for IsToricVariety, IsFpGradedLeftOrRightModulesObject, 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 $\tilde{M}$.

4.2 Example: Pullback line bundle

Example

```
gap> var := ProjectiveSpace( 2 );
<A projective toric variety of dimension 2>
gap> cox_ring := CoxRing( var );
Q[x_1,x_2,x_3]
(weights: [ 1, 1, 1 ])
gap> vars := IndeterminatesOfPolynomialRing( cox_ring );
[ x_1, x_2, x_3 ]
gap> range := GradedRow( [[2],1]], cox_ring );
<A graded row of rank 1>
```

```
gap> source := GradedRow( [[[1],1]], cox_ring );
<A graded row of rank 1>
gap> matrix := HomalgMatrix( [[vars[1]] ], cox_ring );
<A 1 x 1 matrix over a graded ring>
gap> mor := GradedRowOrColumnMorphism( source, matrix, range );
<A morphism in Category of graded rows over
Q[x_1,x_2,x_3]
(with weights [ 1,1,1 ])>
gap> IsWellDefined( mor );
true
gap> pullback_line_bundle := FreydCategoryObject( mor );
<An object in Category of f.p. graded left modules over
Q[x_1,x_2,x_3] (with weights
[ 1, 1, 1 ])>
gap> coh := DeductionOfSheafCohomologyFromResolution( var, pullback_line_bundle );
[ 3, 0, 0 ]
```

Example

```
gap> P1 := ProjectiveSpace( 1 );
<A projective toric variety of dimension 1>
gap> P2 := ProjectiveSpace( 2 );
<A projective toric variety of dimension 2>
gap> var2 := P1 * P1 * P2;
<A projective toric variety of dimension 4
which is a product of 3 toric varieties>
gap> cox_ring2 := CoxRing( var2 );
Q[x_1,x_2,x_3,x_4,x_5,x_6,x_7]
(weights: [ ( 0, 0, 1 ), ( 0, 1, 0 ), ( 1, 0, 0 ),
( 1, 0, 0 ), ( 1, 0, 0 ), ( 0, 1, 0 ), ( 0, 0, 1 ) ])
gap> vars2 := IndeterminatesOfPolynomialRing( cox_ring2 );
[ x_1, x_2, x_3, x_4, x_5, x_6, x_7 ]
gap> range2 := GradedRow( [[[1,1,2],1]], cox_ring2 );
<A graded row of rank 1>
gap> source2 := GradedRow( [[[0,1,2],2]], cox_ring2 );
<A graded row of rank 2>
gap> matrix2 := HomalgMatrix( [[vars2[3]],[vars2[4]]], cox_ring2 );
<A 2 x 1 matrix over a graded ring>
gap> mor2 := GradedRowOrColumnMorphism( source2, matrix2, range2 );
<A morphism in Category of graded rows over
Q[x_1,x_2,x_3,x_4,x_5,x_6,x_7]
(with weights [ [ 0, 0, 1 ], [ 0, 1, 0 ], [ 1, 0, 0 ],
[ 1, 0, 0 ], [ 1, 0, 0 ], [ 0, 1, 0 ], [ 0, 0, 1 ] ])>
gap> IsWellDefined( mor2 );
true
gap> pullback_line_bundle2 := FreydCategoryObject( mor2 );
<An object in Category of f.p. graded left modules over
Q[x_1,x_2,x_3,x_4,x_5,x_6,x_7] (with weights
[ [ 0, 0, 1 ], [ 0, 1, 0 ], [ 1, 0, 0 ], [ 1, 0, 0 ],
[ 1, 0, 0 ], [ 0, 1, 0 ], [ 0, 0, 1 ] ])>
gap> coh2 := DeductionOfSheafCohomologyFromResolution( var2, pullback_line_bundle2 );
[ 6, 0, 0, 0, 0 ]
```

Example

```
gap> P2 := ProjectiveSpace( 2 );
```

```
<A projective toric variety of dimension 2>
gap> var3 := P2 * P2;
<A projective toric variety of dimension 4
which is a product of 2 toric varieties>
gap> cox_ring3 := CoxRing( var3 );
Q[x_1,x_2,x_3,x_4,x_5,x_6]
(weights: [ ( 0, 1 ), ( 1, 0 ), ( 1, 0 ),
( 1, 0 ), ( 0, 1 ), ( 0, 1 ) ])
gap> range3 := GradedRow( [[[1,1],4]], cox_ring3 );
<A graded row of rank 4>
gap> source3 := ZeroObject( CapCategory( range3 ) );
<A graded row of rank 0>
gap> matrix3 := HomalgZeroMatrix( 0, 4, cox_ring3 );
<An unevaluated 0 x 4 zero matrix over a graded ring>
gap> mor3 := GradedRowOrColumnMorphism( source3, matrix3, range3 );
<A morphism in Category of graded rows over
Q[x_1,x_2,x_3,x_4,x_5,x_6] (with weights
[ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ], [ 1, 0 ], [ 0, 1 ], [ 0, 1 ] ])>
gap> line_bundle3 := FreydCategoryObject( mor3 );
<An object in Category of f.p. graded left modules over
Q[x_1,x_2,x_3,x_4,x_5,x_6] (with weights
[ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ], [ 1, 0 ], [0, 1 ], [ 0, 1 ] ])>
gap> IsWellDefined( line_bundle3 );
true
gap> coh3 := DeductionOfSheafCohomologyFromResolution( var3, line_bundle3 );
[ 36, 0, 0, 0, 0 ]
```

Chapter 5

Wrapper for generators of semigroups and hyperplane constraints of cones

5.1 GAP Categories

5.1.1 IsSemigroupForPresentationsByProjectiveGradedModules (for IsObject)

▷ `IsSemigroupForPresentationsByProjectiveGradedModules(`*object*`)` (filter)

Returns: `true` or `false`

The GAP category of lists of integer-valued lists, which encode the generators of subsemigroups of Z^n.

5.1.2 IsAffineSemigroupForPresentationsByProjectiveGradedModules (for IsObject)

▷ `IsAffineSemigroupForPresentationsByProjectiveGradedModules(`*object*`)` (filter)

Returns: `true` or `false`

The GAP category of affine semigroups H in $\mathbb{Z}^n$. That means that there is a semigroup $G \subseteq \mathbb{Z}^n$ and $p \in \mathbb{Z}^n$ such that $H = p + G$.

5.2 Constructors

5.2.1 SemigroupForPresentationsByProjectiveGradedModules (for IsList, IsInt)

▷ `SemigroupForPresentationsByProjectiveGradedModules(`*L*`)` (operation)

Returns: a SemigroupGeneratorList

The argument is a list L and a non-negative integer d. We then check if this list could be the list of generators of a subsemigroup of Z^d. If so, we create the corresponding SemigroupGeneratorList.

5.2.2 SemigroupForPresentationsByProjectiveGradedModules (for IsList)

▷ `SemigroupForPresentationsByProjectiveGradedModules(`*arg*`)` (operation)

5.2.3 AffineSemigroupForPresentationsByProjectiveGradedModules (for IsSemigroupForPresentationsByProjectiveGradedModules, IsList)

▷ `AffineSemigroupForPresentationsByProjectiveGradedModules(L, p)` (operation)

Returns: an AffineSemigroup

The argument is a SemigroupForPresentationsByProjectiveGradedModules S and a point $p \in \mathbb{Z}^n$ encoded as list of integers. We then compute the affine semigroup $p+S$. Alternatively to S we allow the use of either a list of generators (or a list of generators together with the embedding dimension).

5.2.4 AffineSemigroupForPresentationsByProjectiveGradedModules (for IsList, IsList)

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

5.2.5 AffineSemigroupForPresentationsByProjectiveGradedModules (for IsList, IsInt, IsList)

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

5.3 Attributes

5.3.1 GeneratorList (for IsSemigroupForPresentationsByProjectiveGradedModules)

▷ `GeneratorList(L)` (attribute)

Returns: a list

The argument is a SemigroupForPresentationsByProjectiveGradedModules L. We then return the list of its generators.

5.3.2 EmbeddingDimension (for IsSemigroupForPresentationsByProjectiveGradedModules)

▷ `EmbeddingDimension(L)` (attribute)

Returns: a non-negative integer

The argument is a SemigroupForPresentationsByProjectiveGradedModules L. We then return the embedding dimension of this semigroup.

5.3.3 ConeHPresentationList (for IsSemigroupForPresentationsByProjectiveGradedModules)

▷ `ConeHPresentationList(L)` (attribute)

Returns: a list or fail

The argument is a SemigroupForPresentationsByProjectiveGradedModules L. If the associated semigroup is a cone semigroup, then (during construction) an H-presentation of that cone was computed. We return the list of the corresponding H-constraints. This conversion uses Normaliz and can

fail if the cone if not full-dimensional. In case that such a conversion error occured, the attribute is set to the value 'fail'.

5.3.4 Offset (for IsAffineSemigroupForPresentationsByProjectiveGradedModules)

▷ `Offset(S)` (attribute)

Returns: a list of integers

The argument is an AffineSemigroupForPresentationsByProjectiveGradedModules S. This one is given as $S = p + H$ for a point $p \in \mathbb{Z}^n$ and a semigroup $H \subseteq \mathbb{Z}^n$. We then return the offset p.

5.3.5 UnderlyingSemigroup (for IsAffineSemigroupForPresentationsByProjectiveGradedModules)

▷ `UnderlyingSemigroup(S)` (attribute)

Returns: a SemigroupGeneratorList

The argument is an IsAffineSemigroupForPresentationsByProjectiveGradedModules S. This one is given as $S = p + H$ for a point $p \in \mathbb{Z}^n$ and a semigroup $H \subseteq \mathbb{Z}^n$. We then return the SemigroupGeneratorList of H.

5.3.6 EmbeddingDimension (for IsAffineSemigroupForPresentationsByProjectiveGradedModules)

▷ `EmbeddingDimension(S)` (attribute)

Returns: a non-negative integer

The argument is an IsAffineSemigroupForPresentationsByProjectiveGradedModules S. We then return the embedding dimension of this affine semigroup.

5.4 Property

5.4.1 IsTrivial (for IsSemigroupForPresentationsByProjectiveGradedModules)

▷ `IsTrivial(L)` (property)

Returns: true or false

The argument is a SemigroupForPresentationsByProjectiveGradedModules L. This property returns 'true' if this semigroup is trivial and 'false' otherwise.

5.4.2 IsSemigroupOfCone (for IsSemigroupForPresentationsByProjectiveGradedModules)

▷ `IsSemigroupOfCone(L)` (property)

Returns: true, false

The argument is a SemigroupForPresentationsByProjectiveGradedModules L. We return if this is the semigroup of a cone.

5.4.3 IsTrivial (for IsAffineSemigroupForPresentationsByProjectiveGradedModules)

▷ `IsTrivial(L)` (property)

Returns: true or false

The argument is an AffineSemigroupForPresentationsByProjectiveGradedModules. This property returns 'true' if the underlying semigroup is trivial and otherwise 'false'.

5.4.4 IsAffineSemigroupOfCone (for IsAffineSemigroupForPresentationsByProjectiveGradedModules)

▷ `IsAffineSemigroupOfCone(H)` (property)

Returns: true, false or fail

The argument is an IsAffineSemigroupForPresentationsByProjectiveGradedModules H. We return if this is an AffineConeSemigroup. If Normaliz cannot decide this 'fail' is returned.

5.5 Operations

5.5.1 DecideIfIsConeSemigroupGeneratorList (for IsList)

▷ `DecideIfIsConeSemigroupGeneratorList(L)` (operation)

Returns: true, false or fail

The argument is a list L of generators of a semigroup in $\mathbb{Z}^n$. We then check if this is the semigroup of a cone. In this case we return 'true', otherwise 'false'. If the operation fails due to shortcommings in Normaliz we return 'fail'.

5.6 Check if point is contained in (affine) cone or (affine) semigroup

5.6.1 PointContainedInSemigroup (for IsSemigroupForPresentationsByProjectiveGradedModules, IsList)

▷ `PointContainedInSemigroup(S, p)` (operation)

Returns: true or false

The argument is a SemigroupForPresentationsByProjectiveGradedModules S of $\mathbb{Z}^n$, and an integral point p in this lattice. This operation then verifies if the point p is contained in S or not.

5.6.2 PointContainedInAffineSemigroup (for IsAffineSemigroupForPresentationsByProjectiveGradedModules, IsList)

▷ `PointContainedInAffineSemigroup(H, p)` (operation)

Returns: true or false

The argument is an IsAffineSemigroupForPresentationsByProjectiveGradedModules H and a point p. The second argument This method then checks if p lies in H.

5.7 Examples

The following commands are used to handle generators of semigroups in $\mathbb{Z}^n$, generators of cones in $\mathbb{Z}^n$ as well as hyperplane constraints that define cones in $\mathbb{Z}^n$. Here are some examples:

Example

```
gap> semigroup1 := SemigroupForPresentationsByProjectiveGradedModules(
>                [[ 1,0 ], [ 1,1 ]] );
<A cone-semigroup in Z^2 formed as the span of 2 generators>
gap> IsSemigroupForPresentationsByProjectiveGradedModules( semigroup1 );
true
gap> GeneratorList( semigroup1 );
[ [ 1, 0 ], [ 1, 1 ] ]
gap> semigroup2 := SemigroupForPresentationsByProjectiveGradedModules(
>                [[ 2,0 ], [ 1,1 ]] );
<A non-cone semigroup in Z^2 formed as the span of 2 generators>
gap> IsSemigroupForPresentationsByProjectiveGradedModules( semigroup2 );
true
gap> GeneratorList( semigroup2 );
[ [ 2, 0 ], [ 1, 1 ] ]
```

We can check if a semigroup in $\mathbb{Z}^n$ is the semigroup of a cone. In case we can look at an H-presentation of this cone.

Example

```
gap> IsSemigroupOfCone( semigroup1 );
true
gap> ConeHPresentationList( semigroup1 );
[ [ 0, 1 ], [ 1, -1 ] ]
gap> Display( ConeHPresentationList( semigroup1 ) );
[ [   0,  1 ],
  [   1, -1 ] ]
gap> IsSemigroupOfCone( semigroup2 );
false
gap> HasConeHPresentationList( semigroup2 );
false
```

We can check membership of points in semigroups.

Example

```
gap> PointContainedInSemigroup( semigroup2, [ 1,0 ] );
false
gap> PointContainedInSemigroup( semigroup2, [ 2,0 ] );
true
```

Given a semigroup $S \subseteq \mathbb{Z}^n$ and a point $p \in \mathbb{Z}^n$ we can consider

$$H := p + S = \{p + x\,,\, x \in S\}\,.$$

We term this an affine semigroup. Given that $S = C \cap \mathbb{Z}^n$ for a cone $C \subseteq \mathbb{Z}^n$, we use the term affine cone_semigroup. The constructors are as follows:

Example

```
gap> affine_semigroup1 := AffineSemigroupForPresentationsByProjectiveGradedModules(
>                     semigroup1, [ -1, -1 ] );
<A non-trivial affine cone-semigroup in Z^2>
gap> affine_semigroup2 := AffineSemigroupForPresentationsByProjectiveGradedModules(
>                     semigroup2, [ 2, 2 ] );
<A non-trivial affine non-cone semigroup in Z^2>
```

We can access the properties of these affine semigroups as follows.

Example

```
gap> IsAffineSemigroupOfCone( affine_semigroup2 );
false
gap> UnderlyingSemigroup( affine_semigroup2 );
<A non-cone semigroup in Z^2 formed as the span of 2 generators>
gap> Display( UnderlyingSemigroup( affine_semigroup2 ) );
A non-cone semigroup in Z^2 formed as the span of 2 generators -
generators are as follows:
[ [  2,  0 ],
  [  1,  1 ] ]
gap> IsAffineSemigroupOfCone( affine_semigroup1 );
true
gap> Offset( affine_semigroup2 );
[ 2, 2 ]
gap> ConeHPresentationList( UnderlyingSemigroup( affine_semigroup1 ) );
[ [ 0, 1 ], [ 1, -1 ] ]
```

Of course we can also decide membership in affine (cone_)semigroups.

Example

```
gap> Display( affine_semigroup1 );
A non-trivial affine cone-semigroup in Z^2
Offset: [ -1, -1 ]
Hilbert basis: [ [ 1, 0 ], [ 1, 1 ] ]
gap> PointContainedInAffineSemigroup( affine_semigroup1, [ -2,-2 ] );
false
gap> PointContainedInAffineSemigroup( affine_semigroup1, [ 3,1 ] );
true
gap> Display( affine_semigroup2 );
A non-trivial affine non-cone semigroup in Z^2
Offset: [ 2, 2 ]
Semigroup generators: [ [ 2, 0 ], [ 1, 1 ] ]
gap> PointContainedInAffineSemigroup( affine_semigroup2, [ 3,2 ] );
false
gap> PointContainedInAffineSemigroup( affine_semigroup2, [ 3,3 ] );
true
```

Chapter 6

Vanishing sets on toric varieties

6.1 GAP category for vanishing sets

6.1.1 IsVanishingSet (for IsObject)

▷ `IsVanishingSet(arg)` (filter)

Returns: `true` or `false`

The GAP category of vanishing sets formed from affine semigroups.

6.2 Constructors

6.2.1 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'.

6.2.2 VanishingSet (for IsToricVariety, IsList, IsString)

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

6.3 Attributes

6.3.1 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.

6.3.2 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.

6.3.3 CohomologicalIndex (for IsVanishingSet)

▷ `CohomologicalIndex(V)` (attribute)

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

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

6.3.4 CohomologicalSpecification (for IsVanishingSet)

▷ `CohomologicalSpecification(V)` (attribute)

Returns: a string

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

6.3.5 AmbientToricVariety (for IsVanishingSet)

▷ `AmbientToricVariety(V)` (attribute)

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

6.4 Property

6.4.1 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.

6.5 Improved vanishing sets via cohomCalg

6.5.1 TurnDenominatorIntoShiftedSemigroup (for IsToricVariety, IsString)

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

6.5.2 VanishingSets (for IsToricVariety)

▷ `VanishingSets(arg)` (attribute)

6.5.3 ComputeVanishingSets (for IsToricVariety, IsBool)

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

6.5.4 PointContainedInVanishingSet (for IsVanishingSet, IsList)

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

6.6 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 projective 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 projective 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 7

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

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

7.1.1 IsICTCurve (for IsObject)

▷ `IsICTCurve(object)` (filter)

Returns: true or false

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

7.2 Constructors for ICT Curves

7.2.1 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_Σ and two non-negative and distinct integers i, j. We then consider the i-th and j-th maximal cones σ_i and σ_j. ! If $\tau := \sigma_i \cap \sigma_j$ satisfies $dim(\tau) = dim(\sigma_1) - 1$, then τ corresponds to an ICT-curve. We then construct this very ICT-curve.

7.3 Attributes for ICT curves

7.3.1 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.

7.3.2 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 τ associated to this curve C.

7.3.3 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

7.3.4 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.

7.3.5 LeftStructureSheaf (for IsICTCurve)

▷ `LeftStructureSheaf(C)` (attribute)

Returns: a f.p. graded left S-module

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

7.3.6 RightStructureSheaf (for IsICTCurve)

▷ `RightStructureSheaf(C)` (attribute)

Returns: a f.p. graded right S-module

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

7.3.7 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.

7.3.8 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 $(e_1,\ldots,e_k)$ with $e_i = (0,\ldots,0,1,0,\ldots,0) \in Cl\,(X_\Sigma)$.

7.4 Operations with ICTCurves

7.4.1 ICTCurves (for IsToricVariety)

▷ `ICTCurves(X_Sigma)` (attribute)

Returns: a list of ICT-curves.

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

7.4.2 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_Σ, this method computes their intersection product.

7.4.3 IntersectionProduct (for IsToricDivisor, IsICTCurve)

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

7.5 GAP category for proper 1-cycles

7.5.1 IsProper1Cycle (for IsObject)

▷ `IsProper1Cycle(object)` (filter)

Returns: true or false

The GAP category for proper 1-cycles

7.6 Constructor For Proper 1-Cycles

7.6.1 GeneratorsOfProper1Cycles (for IsToricVariety)

▷ `GeneratorsOfProper1Cycles(X_Sigma)` (attribute)

Returns: a list of ICT-curves.

For a smooth and complete toric variety X_Σ, 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_Σ in the constructor below, when computing the intersection form and the Nef-cone.

7.6.2 Proper1Cycle (for IsToricVariety, IsList)

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

Returns: a proper 1-cycle

The arguments are a smooth and complete toric variety X_Σ 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.

7.7 Attributes for proper 1-cycles

7.7.1 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.

7.7.2 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}).

7.8 Operations with proper 1-cycles

7.8.1 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_Σ, this method computes their intersection product.

7.8.2 IntersectionProduct (for IsToricDivisor, IsProper1Cycle)

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

7.8.3 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.

7.9 Examples in 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
```

Example

```
gap> P3 := ProjectiveSpace( 3 );
<A projective toric variety of dimension 3>
gap> C1 := ICTCurves( P3 )[ 1 ];
<An irreducible, complete, torus-invariant curve in a toric variety
 given as V( [ x_3, x_4 ] )>
gap> vars := DefiningVariables( C1 );
[ x_3, x_4 ]
gap> structureSheaf1 := LeftStructureSheaf( C1 );;
gap> IsWellDefined( structureSheaf1 );
true
gap> structureSheaf2 := RightStructureSheaf( C1 );;
gap> IsWellDefined( structureSheaf2 );
true
```

Chapter 8

Nef and Mori Cone

8.1 New Properties For Toric Divisors

8.1.1 IsNef (for IsToricDivisor)

▷ `IsNef(divi)` (property)

Returns: true or false

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

8.1.2 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.

8.2 Attributes

8.2.1 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.

8.2.2 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.

8.2.3 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.

8.2.4 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.

8.2.5 NefCone (for IsToricVariety)

▷ `NefCone(arg)` (attribute)

8.2.6 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.

8.2.7 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.

8.2.8 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.

8.3 Nef and Mori Cone: Examples

8.3.1 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 9

DegreeXLayerVectorSpaces and morphisms

9.1 GAP category of DegreeXLayerVectorSpaces

9.1.1 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

9.1.2 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

9.1.3 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

9.1.4 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

9.2 Constructors for DegreeXLayerVectorSpaces

9.2.1 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.

9.2.2 DegreeXLayerVectorSpaceMorphism (for IsDegreeXLayerVectorSpace, IsVectorSpaceMorphism, IsDegreeXLayerVectorSpace)

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

Returns: a DegreeXLayerVectorSpaceMorphism

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

9.2.3 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.

9.2.4 DegreeXLayerVectorSpacePresentationMorphism (for IsDegreeXLayerVectorSpacePresentation, IsVectorSpaceMorphism, IsDegreeXLayerVectorSpacePresentation)

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

Returns: a DegreeXLayerVectorSpacePresentationMorphism

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

9.3 Attributes for DegreeXLayerVectorSpaces

9.3.1 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).

9.3.2 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.

9.3.3 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').

9.3.4 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.

9.4 Attributes for DegreeXLayerVectorSpaceMorphisms

9.4.1 Source (for IsDegreeXLayerVectorSpaceMorphism)

▷ `Source(a)` (attribute)

Returns: a DegreeXLayerVectorSpace

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

9.4.2 Range (for IsDegreeXLayerVectorSpaceMorphism)

▷ `Range(a)` (attribute)

Returns: a DegreeXLayerVectorSpace

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

9.4.3 UnderlyingVectorSpaceMorphism (for IsDegreeXLayerVectorSpaceMorphism)

▷ `UnderlyingVectorSpaceMorphism(a)` (attribute)

Returns: a DegreeXLayerVectorSpace

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

9.4.4 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.

9.5 Attributes for DegreeXLayerVectorSpacePresentations

9.5.1 UnderlyingDegreeXLayerVectorSpaceMorphism (for IsDegreeXLayerVectorSpacePresentation)

▷ `UnderlyingDegreeXLayerVectorSpaceMorphism(a)` (attribute)

Returns: a DegreeXLayerVectorSpaceMorphism

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

9.5.2 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.

9.5.3 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.

9.5.4 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.

9.5.5 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.

9.6 Attributes for DegreeXLayerVectorSpacePresentationMorphisms

9.6.1 Source (for IsDegreeXLayerVectorSpacePresentationMorphism)

▷ `Source(a)` (attribute)

Returns: a DegreeXLayerVectorSpacePresentation

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

9.6.2 Range (for IsDegreeXLayerVectorSpacePresentationMorphism)

▷ `Range(a)` (attribute)

Returns: a DegreeXLayerVectorSpacePresentation

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

9.6.3 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.

9.6.4 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.

9.7 Convenience methods

9.7.1 FullInformation (for IsDegreeXLayerVectorSpacePresentation)

▷ `FullInformation(p)` (operation)

Returns: detailed information about p

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

9.7.2 FullInformation (for IsDegreeXLayerVectorSpacePresentationMorphism)

▷ `FullInformation(p)` (operation)

Returns: detailed information about p

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

9.8 Examples

9.8.1 DegreeXLayerVectorSpaces

Example

```
gap> mQ := HomalgFieldOfRationals();
Q
gap> P1 := ProjectiveSpace( 1 );
<A projective toric variety of dimension 1>
gap> cox_ring := CoxRing( P1 );
Q[x_1,x_2]
(weights: [ 1, 1 ])
gap> mons := MonomsOfCoxRingOfDegreeByNormalizAsColumnMatrices
>         ( P1, [1], 1, 1 );;
gap> vector_space := VectorSpaceObject( Length( mons ), mQ );
<A vector space object over Q of dimension 2>
gap> DXVS := DegreeXLayerVectorSpace( mons, cox_ring, vector_space, 1 );
<A vector space embedded into (Q[x_1,x_2] (with weights [ 1, 1 ]))^1>
gap> EmbeddingDimension( DXVS );
1
gap> Generators( DXVS );
[ <A 1 x 1 matrix over a graded ring>, <A 1 x 1 matrix over a graded ring> ]
```

9.8.2 Morphisms of DegreeXLayerVectorSpaces

Example

```
gap> mons2 := Concatenation(
>          MonomsOfCoxRingOfDegreeByNormalizAsColumnMatrices
>          ( P1, [1], 1, 2 ),
>          MonomsOfCoxRingOfDegreeByNormalizAsColumnMatrices
>          ( P1, [1], 2, 2 ) );;
gap> vector_space2 := VectorSpaceObject( Length( mons2 ), mQ );
<A vector space object over Q of dimension 4>
gap> DXVS2 := DegreeXLayerVectorSpace( mons2, cox_ring, vector_space2, 2 );
<A vector space embedded into (Q[x_1,x_2] (with weights [ 1, 1 ]))^2>
gap> matrix := HomalgMatrix( [ [ 1, 0, 0, 0 ],
>                              [ 0, 1, 0, 0 ] ], mQ );
<A matrix over an internal ring>
gap> vector_space_morphism := VectorSpaceMorphism( vector_space,
>                                                  matrix,
>                                                  vector_space2 );;
gap> IsWellDefined( vector_space_morphism );
true
gap> morDXVS := DegreeXLayerVectorSpaceMorphism(
>            DXVS, vector_space_morphism, DXVS2 );
<A morphism of two vector spaces embedded into
(suitable powers of) Q[x_1,x_2] (with weights [ 1, 1 ])>
gap> UnderlyingVectorSpaceMorphism( morDXVS );
<A morphism in Category of matrices over Q>
gap> UnderlyingHomalgGradedRing( morDXVS );
Q[x_1,x_2]
(weights: [ 1, 1 ])
```

9.8.3 DegreeXLayerVectorSpacePresentations

Example

```
gap> DXVSPresentation := DegreeXLayerVectorSpacePresentation( morDXVS );
<A vector space embedded into (a suitable power of)
Q[x_1,x_2] (with weights [ 1, 1 ]) given as the
cokernel of a vector space morphism>
gap> UnderlyingVectorSpaceObject( DXVSPresentation );
<A vector space object over Q of dimension 2>
gap> relation := RelationMorphism(
>             UnderlyingVectorSpacePresentation( DXVSPresentation ) );
<A morphism in Category of matrices over Q>
gap> m := UnderlyingMatrix( relation );
<A 2 x 4 matrix over an internal ring>
gap> m = matrix;
true
```

9.8.4 Morphisms of DegreeXLayerVectorSpacePresentations

Example

```
gap> zero_space := ZeroObject( CapCategory( vector_space ) );;
gap> source := DegreeXLayerVectorSpace( [], cox_ring, zero_space, 1 );;
gap> vector_space_morphism := ZeroMorphism( zero_space, vector_space );;
gap> morDXVS2 := DegreeXLayerVectorSpaceMorphism(
>             source, vector_space_morphism, DXVS );;
gap> DXVSPresentation2 := DegreeXLayerVectorSpacePresentation( morDXVS2 );
<A vector space embedded into (a suitable power of)
Q[x_1,x_2] (with weights [ 1, 1 ]) given as the
cokernel of a vector space morphism>
gap> matrix := HomalgMatrix( [ [ 0, 0, 1, 0 ],
>                              [ 0, 0, 0, 1 ] ], mQ );
<A matrix over an internal ring>
gap> source := Range( UnderlyingVectorSpaceMorphism( DXVSPresentation2 ) );;
gap> range := Range( UnderlyingVectorSpaceMorphism( DXVSPresentation ) );;
gap> vector_space_morphism := VectorSpaceMorphism( source, matrix, range );;
gap> IsWellDefined( vector_space_morphism );
true
gap> DXVSPresentationMorphism := DegreeXLayerVectorSpacePresentationMorphism(
>                                       DXVSPresentation2,
>                                       vector_space_morphism,
>                                       DXVSPresentation );
<A vector space presentation morphism of vector spaces embedded into
(a suitable power of) Q[x_1,x_2] (with weights [ 1, 1 ]) and given as
cokernels>
gap> uVSMor := UnderlyingVectorSpacePresentationMorphism
>                                       ( DXVSPresentationMorphism );
<A morphism in Freyd( Category of matrices over Q )>
gap> IsWellDefined( uVSMor );
true
```

Chapter 10

Truncations of graded rows and columns

10.1 Truncations of graded rows and columns

10.1.1 TruncateGradedRowOrColumn (for IsToricVariety, IsGradedRowOrColumn, IsList, IsFieldForHomalg)

▷ `TruncateGradedRowOrColumn(V, M, degree_list, field)` (operation)

Returns: Vector space

The arguments are a toric variety V, a graded row or column M over 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 as a HomalgModuleElement. Based on this input, the method computes the truncation of M to the specified degree. We return this finite dimensional vector space. Optionally, we allow for a field F as fourth input. This field is then used to construct the vector space. Otherwise, we use the coefficient field of the Cox ring of V.

10.1.2 TruncateGradedRowOrColumn (for IsToricVariety, IsGradedRowOrColumn, IsHomalgModuleElement, IsFieldForHomalg)

▷ `TruncateGradedRowOrColumn(V, M, m, field)` (operation)

Returns: Vector space

As above, but with a HomalgModuleElement m specifying the degree.

10.1.3 TruncateGradedRowOrColumn (for IsToricVariety, IsGradedRowOrColumn, IsList)

▷ `TruncateGradedRowOrColumn(V, M, degree)` (operation)

Returns: Vector space

As above, but the coefficient ring of the Cox ring will be used as field

10.1.4 TruncateGradedRowOrColumn (for IsToricVariety, IsGradedRowOrColumn, IsHomalgModuleElement)

▷ `TruncateGradedRowOrColumn(V, M, m)` (operation)

Returns: Vector space

As above, but a HomalgModuleElement m specifies the degree and we use the coefficient ring of the Cox ring as field.

10.1.5 DegreeXLayerOfGradedRowOrColumn (for IsToricVariety, IsGradedRowOrColumn, IsList, IsFieldForHomalg)

▷ `DegreeXLayerOfGradedRowOrColumn(V, M, degree_list, field)` (operation)

Returns: DegreeXLayerVectorSpace

The arguments are a toric variety V, a graded row or column M over 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 as a HomalgModuleElement. Based on this input, the method computes the truncation of M to the specified degree. This is a finite dimensional vector space. We return the corresponding DegreeXLayerVectorSpace. Optionally, we allow for a field F as fourth input. This field is used to construct the DegreeXLayerVectorSpace. Namely, the wrapper DegreeXLayerVectorSpace contains a representation of the obtained vector space as F^n. In case F is specified, we use this particular field. Otherwise, HomalgFieldOfRationals() will be used.

10.1.6 DegreeXLayerOfGradedRowOrColumn (for IsToricVariety, IsGradedRowOrColumn, IsHomalgModuleElement, IsFieldForHomalg)

▷ `DegreeXLayerOfGradedRowOrColumn(V, M, m, field)` (operation)

Returns: DegreeXLayerVectorSpace

As above, but with a HomalgModuleElement m specifying the degree.

10.1.7 DegreeXLayerOfGradedRowOrColumn (for IsToricVariety, IsGradedRowOrColumn, IsList)

▷ `DegreeXLayerOfGradedRowOrColumn(V, M, degree)` (operation)

Returns: DegreeXLayerVectorSpace

As above, but the coefficient ring of the Cox ring will be used as field

10.1.8 DegreeXLayerOfGradedRowOrColumn (for IsToricVariety, IsGradedRowOrColumn, IsHomalgModuleElement)

▷ `DegreeXLayerOfGradedRowOrColumn(V, M, m)` (operation)

Returns: DegreeXLayerVectorSpace

As above, but a HomalgModuleElement m specifies the degree and we use the coefficient ring of the Cox ring as field.

10.2 Formats for generators of truncations of graded rows and columns

10.2.1 GeneratorsOfDegreeXLayerOfGradedRowOrColumnAsListOfColumnMatrices (for IsToricVariety, IsGradedRowOrColumn, IsList)

▷ `GeneratorsOfDegreeXLayerOfGradedRowOrColumnAsListOfColumnMatrices(V, M, l)` (operation)

Returns: a list

The arguments are a variety V, a graded row or column M and a list l, specifying a degree in the class group of the Cox ring of V. We then compute the truncation of M to the specified degree and return its generators as list of column matrices.

10.2.2 GeneratorsOfDegreeXLayerOfGradedRowOrColumnAsListOfColumnMatrices (for IsToricVariety, IsGradedRowOrColumn, IsHomalgModuleElement)

▷ `GeneratorsOfDegreeXLayerOfGradedRowOrColumnAsListOfColumnMatrices(V, M, m)` (operation)

Returns: a list

The arguments are a variety V, a graded row or column M and a HomalgModuleElement m, specifying a degree in the class group of the Cox ring of V. We then compute the truncation of M to the specified degree and return its generators as list of column matrices.

10.2.3 GeneratorsOfDegreeXLayerOfGradedRowOrColumnAsUnionOfColumnMatrices (for IsToricVariety, IsGradedRowOrColumn, IsList)

▷ `GeneratorsOfDegreeXLayerOfGradedRowOrColumnAsUnionOfColumnMatrices(V, M, m)` (operation)

Returns: a list

The arguments are a variety V, a graded row or column M and a list l, specifying a degree in the class group of the Cox ring of V. We then compute the truncation of M to the specified degree and its generators as column matrices. The matrix formed from the union of these column matrices is returned.

10.2.4 GeneratorsOfDegreeXLayerOfGradedRowOrColumnAsUnionOfColumnMatrices (for IsToricVariety, IsGradedRowOrColumn, IsHomalgModuleElement)

▷ `GeneratorsOfDegreeXLayerOfGradedRowOrColumnAsUnionOfColumnMatrices(V, M, m)` (operation)

Returns: a list

The arguments are a variety V, a graded row or column M and a HomalgModuleElement m, specifying a degree in the class group of the Cox ring of V. We then compute the truncation of M to the specified degree and its generators as column matrices. The matrix formed from the union of these column matrices is returned.

10.2.5 GeneratorsOfDegreeXLayerOfGradedRowOrColumnAsListsOfRecords (for IsToricVariety, IsGradedRowOrColumn, IsList)

▷ `GeneratorsOfDegreeXLayerOfGradedRowOrColumnAsListsOfRecords(V, M, l)` (operation)

Returns: a list

The arguments are a variety V, a graded row or column M and a list l, specifying a degree in the class group of the Cox ring of V. We then compute the truncation of M to the specified degree and return its generators as list [n, rec_list]. n specifies the number of generators. rec_list is a list of record. The i-th record contains the generators of the i-th direct summand of M.

The arguments are a variety V, a graded row or column M and a HomalgModuleElement m, specifying a degree in the class group of the Cox ring of V. We then compute the truncation of M to the

specified degree and return its generators as list [n, rec_list]. n specifies the number of generators. rec_list is a list of record. The i-th record contains the generators of the i-th direct summand of M.

10.2.6 GeneratorsOfDegreeXLayerOfGradedRowOrColumnAsListsOfRecords (for IsToricVariety, IsGradedRowOrColumn, IsHomalgModuleElement)

▷ `GeneratorsOfDegreeXLayerOfGradedRowOrColumnAsListsOfRecords(V, M, m)` (operation)
Returns: a list

10.2.7 GeneratorsOfDegreeXLayerOfGradedRowOrColumnAsListList (for IsToricVariety, IsGradedRowOrColumn, IsList)

▷ `GeneratorsOfDegreeXLayerOfGradedRowOrColumnAsListList(V, M, l)` (operation)
Returns: a list

The arguments are a variety V, a graded row or column M and a list l, specifying a degree in the class group of the Cox ring of V. We then compute the truncation of M to the specified degree and identify its generators. We format each generator as list [n, g], where g denotes a generator of the n-th direct summand of M. We return the list of all these lists [n, g].

10.2.8 GeneratorsOfDegreeXLayerOfGradedRowOrColumnAsListList (for IsToricVariety, IsGradedRowOrColumn, IsHomalgModuleElement)

▷ `GeneratorsOfDegreeXLayerOfGradedRowOrColumnAsListList(V, M, m)` (operation)
Returns: a list

The arguments are a variety V, a graded row or column M and a HomalgModuleElement m, specifying a degree in the class group of the Cox ring of V. We then compute the truncation of M to the specified degree and identify its generators. We format each generator as list [n, g], where g denotes a generator of the n-th direct summand of M. We return the list of all these lists [n, g].

10.3 Truncations of graded row and column morphisms

10.3.1 TruncateGradedRowOrColumnMorphism (for IsToricVariety, IsGradedRowOrColumnMorphism, IsList, IsBool, IsFieldForHomalg)

▷ `TruncateGradedRowOrColumnMorphism(V, a, d, B, F)` (operation)
Returns: a vector space morphism

The arguments are a toric variety V, a morphism a of graded rows or columns, a list d specifying a degree in the class group of V, a field F for homalg and a boolean B. We then truncate m to the specified degree d. We express this result as morphism of vector spaces over the field F. We return this vector space morphism. If the boolean B is true, we display additional output during the computation, otherwise this output is surpressed.

10.3.2 TruncateGradedRowOrColumnMorphism (for IsToricVariety, IsGradedRowOrColumnMorphism, IsHomalgModuleElement, IsBool, IsHomalgRing)

▷ `TruncateGradedRowOrColumnMorphism(V, a, m, B, F)` (operation)
Returns: a vector space morphism

The arguments are a toric variety V, a morphism a of graded rows or columns, and a HomalgModuleElement m specifying a degree in the class group of V, a field F for homalg and a boolean B. We then truncate m to the specified degree d. We express this result as morphism of vector spaces over the field F. We return this vector space morphism. If the boolean B is true, we display additional output during the computation, otherwise this output is surpressed.

10.3.3 TruncateGradedRowOrColumnMorphism (for IsToricVariety, IsGradedRowOrColumnMorphism, IsList, IsBool)

▷ `TruncateGradedRowOrColumnMorphism(V, a, d, B)` (operation)

Returns: a vector space morphism

This method operates just as 'TruncateGradedRowOrColumnMorphism' above. However, here the field F is taken as the field of coefficients of the Cox ring of the variety V.

10.3.4 TruncateGradedRowOrColumnMorphism (for IsToricVariety, IsGradedRowOrColumnMorphism, IsHomalgModuleElement, IsBool)

▷ `TruncateGradedRowOrColumnMorphism(V, a, m, B)` (operation)

Returns: a vector space morphism

This method operates just as 'TruncateGradedRowOrColumnMorphism' above. However, here the field F is taken as the field of coefficients of the Cox ring of the variety V.

10.3.5 TruncateGradedRowOrColumnMorphism (for IsToricVariety, IsGradedRowOrColumnMorphism, IsList)

▷ `TruncateGradedRowOrColumnMorphism(V, a, d)` (operation)

Returns: a vector space morphism

This method operates just as 'TruncateGradedRowOrColumnMorphism' above. However, here the field F is taken as the field of coefficients of the Cox ring of the variety V. Also, B is set to false, i.e. no additional information is being displayed.

10.3.6 TruncateGradedRowOrColumnMorphism (for IsToricVariety, IsGradedRowOrColumnMorphism, IsHomalgModuleElement)

▷ `TruncateGradedRowOrColumnMorphism(V, a, m)` (operation)

Returns: a vector space morphism

This method operates just as 'TruncateGradedRowOrColumnMorphism' above. However, here the field F is taken as the field of coefficients of the Cox ring of the variety V. Also, B is set to false, i.e. no additional information is being displayed.

10.3.7 DegreeXLayerOfGradedRowOrColumnMorphism (for IsToricVariety, IsGradedRowOrColumnMorphism, IsList, IsFieldForHomalg, IsBool)

▷ `DegreeXLayerOfGradedRowOrColumnMorphism(V, a, d, F, B)` (operation)

Returns: a DegreeXLayerVectorSpaceMorphism

The arguments are a toric variety V, a morphism a of graded rows or columns, a list d specifying a degree in the class group of V, a field F for homalg and a boolean B. We then truncate m to the specified degree d. We express this result as morphism of vector spaces over the field F. We return the

corresponding DegreeXLayerVectorSpaceMorphism. If the boolean B is true, we display additional output during the computation, otherwise this output is surpressed.

10.3.8 DegreeXLayerOfGradedRowOrColumnMorphism (for IsToricVariety, IsGradedRowOrColumnMorphism, IsHomalgModuleElement, IsHomalgRing, IsBool)

▷ `DegreeXLayerOfGradedRowOrColumnMorphism(V, a, m, F, B)` (operation)

Returns: a DegreeXLayerVectorSpaceMorphism

The arguments are a toric variety V, a morphism a of graded rows or columns, a HomalgModuleElement m specifying a degree in the class group of V, a field F for homalg and a boolean B. We then truncate m to the specified degree d. We express this result as morphism of vector spaces over the field F. We return the corresponding DegreeXLayerVectorSpaceMorphism. If the boolean B is true, we display additional output during the computation, otherwise this output is surpressed.

10.3.9 DegreeXLayerOfGradedRowOrColumnMorphism (for IsToricVariety, IsGradedRowOrColumnMorphism, IsList, IsBool)

▷ `DegreeXLayerOfGradedRowOrColumnMorphism(V, a, d, B)` (operation)

Returns: a vector space morphism

This method operates just as 'DegreeXLayerOfGradedRowOrColumnMorphism' above. However, here the field F is taken as the field of coefficients of the Cox ring of the variety V.

10.3.10 DegreeXLayerOfGradedRowOrColumnMorphism (for IsToricVariety, IsGradedRowOrColumnMorphism, IsHomalgModuleElement, IsBool)

▷ `DegreeXLayerOfGradedRowOrColumnMorphism(V, a, m, B)` (operation)

Returns: a vector space morphism

This method operates just as 'DegreeXLayerOfGradedRowOrColumnMorphism' above. However, here the field F is taken as the field of coefficients of the Cox ring of the variety V.

10.3.11 DegreeXLayerOfGradedRowOrColumnMorphism (for IsToricVariety, IsGradedRowOrColumnMorphism, IsList)

▷ `DegreeXLayerOfGradedRowOrColumnMorphism(V, a, d)` (operation)

Returns: a vector space morphism

This method operates just as 'DegreeXLayerOfGradedRowOrColumnMorphism' above. However, here the field F is taken as the field of coefficients of the Cox ring of the variety V. Also, B is set to false, i.e. no additional information is being displayed.

10.3.12 DegreeXLayerOfGradedRowOrColumnMorphism (for IsToricVariety, IsGradedRowOrColumnMorphism, IsHomalgModuleElement)

▷ `DegreeXLayerOfGradedRowOrColumnMorphism(V, a, m)` (operation)

Returns: a vector space morphism

This method operates just as 'DegreeXLayerOfGradedRowOrColumnMorphism' above. However, here the field F is taken as the field of coefficients of the Cox ring of the variety V. Also, B is set to false, i.e. no additional information is being displayed.

10.4 Truncations of morphisms of graded rows and columns in parallel

10.4.1 TruncateGradedRowOrColumnMorphismInParallel (for IsToricVariety, IsGradedRowOrColumnMorphism, IsList, IsPosInt, IsBool, IsFieldForHomalg)

▷ `TruncateGradedRowOrColumnMorphismInParallel(V, a, d, N, B, F)` (operation)

Returns: a vector space morphism

This method operates just as 'TruncateGradedRowOrColumnMorphism' above. However, as fourth argument an integer *N* is to be specified. The computation of the truncation will then be performed in parallel in *N* child processes.

10.4.2 TruncateGradedRowOrColumnMorphismInParallel (for IsToricVariety, IsGradedRowOrColumnMorphism, IsHomalgModuleElement, IsPosInt, IsBool, IsFieldForHomalg)

▷ `TruncateGradedRowOrColumnMorphismInParallel(V, a, m, N, B, F)` (operation)

Returns: a vector space morphism

This method operates just as 'TruncateGradedRowOrColumnMorphism' above. However, as fourth argument an integer *N* is to be specified. The computation of the truncation will then be performed in parallel in *N* child processes.

10.4.3 TruncateGradedRowOrColumnMorphismInParallel (for IsToricVariety, IsGradedRowOrColumnMorphism, IsList, IsPosInt, IsBool)

▷ `TruncateGradedRowOrColumnMorphismInParallel(V, a, d, N, B)` (operation)

Returns: a vector space morphism

This method operates just as 'TruncateGradedRowOrColumnMorphism' above. However, as fourth argument an integer *N* is to be specified. The computation of the truncation will then be performed in parallel in *N* child processes.

10.4.4 TruncateGradedRowOrColumnMorphismInParallel (for IsToricVariety, IsGradedRowOrColumnMorphism, IsHomalgModuleElement, IsPosInt, IsBool)

▷ `TruncateGradedRowOrColumnMorphismInParallel(V, a, m, N, B)` (operation)

Returns: a vector space morphism

This method operates just as 'TruncateGradedRowOrColumnMorphism' above. However, as fourth argument an integer *N* is to be specified. The computation of the truncation will then be performed in parallel in *N* child processes.

10.4.5 TruncateGradedRowOrColumnMorphismInParallel (for IsToricVariety, IsGradedRowOrColumnMorphism, IsList, IsPosInt)

▷ `TruncateGradedRowOrColumnMorphismInParallel(V, a, d, N)` (operation)

Returns: a vector space morphism

This method operates just as 'TruncateGradedRowOrColumnMorphism' above. However, as fourth argument an integer *N* is to be specified. The computation of the truncation will then be performed in parallel in *N* child processes.

10.4.6 TruncateGradedRowOrColumnMorphismInParallel (for IsToricVariety, IsGradedRowOrColumnMorphism, IsHomalgModuleElement, IsPosInt)

▷ `TruncateGradedRowOrColumnMorphismInParallel(V, a, m, N)` (operation)

Returns: a vector space morphism

This method operates just as 'TruncateGradedRowOrColumnMorphism' above. However, as fourth argument an integer N is to be specified. The computation of the truncation will then be performed in parallel in N child processes.

10.5 Examples

10.5.1 Truncations of graded rows and columns

Example

```
gap> P2 := ProjectiveSpace( 2 );
<A projective toric variety of dimension 2>
gap> cox_ring := CoxRing( P2 );
Q[x_1,x_2,x_3]
(weights: [ 1, 1, 1 ])
gap> row := GradedRow( [[[2],1]], cox_ring );
<A graded row of rank 1>
gap> trunc1 := DegreeXLayerOfGradedRowOrColumn( P2, row, [ -3 ] );
<A vector space embedded into (Q[x_1,x_2,x_3] (with weights [ 1, 1, 1 ]))^1>
gap> Length( Generators( trunc1 ) );
0
gap> trunc2 := DegreeXLayerOfGradedRowOrColumn( P2, row, [ -1 ] );
<A vector space embedded into (Q[x_1,x_2,x_3] (with weights [ 1, 1, 1 ]))^1>
gap> Length( Generators( trunc2 ) );
3
```

10.5.2 Formats for generators of truncations of graded rows and columns

Example

```
gap> row2 := GradedRow( [[[2],2]], cox_ring );
<A graded row of rank 2>
gap> gens1 := GeneratorsOfDegreeXLayerOfGradedRowOrColumnAsListOfColumnMatrices
>             (P2, row2, [ -1 ] );;
gap> Length( gens1 );
6
gap> gens1[ 1 ];
<A 2 x 1 matrix over a graded ring>
gap> Display( gens1[ 1 ] );
x_1,
0
(over a graded ring)
gap> Display( gens1[ 4 ] );
0,
x_1
(over a graded ring)
gap> gens2 := GeneratorsOfDegreeXLayerOfGradedRowOrColumnAsListsOfRecords
>             (P2, row2, [ -1 ] );
[ 6, [ rec( x_1 := 1, x_2 := 2, x_3 := 3 ),
```

```
        rec( x_1 := 4, x_2 := 5, x_3 := 6 ) ] ]
gap> gens3 := GeneratorsOfDegreeXLayerOfGradedRowOrColumnAsUnionOfColumnMatrices
>             (P2, row2, [ -1 ] );
<A 2 x 6 mutable matrix over a graded ring>
gap> Display( gens3 );
x_1,x_2,x_3,0,  0,  0,
0,  0,  0,  x_1,x_2,x_3
(over a graded ring)
gap> gens4 := GeneratorsOfDegreeXLayerOfGradedRowOrColumnAsListList
>             (P2, row2, [ -1 ] );
[ [ 1, x_1 ], [ 1, x_2 ], [ 1, x_3 ], [ 2, x_1 ], [ 2, x_2 ], [ 2, x_3 ] ]
```

10.5.3 Truncatons of morphisms of graded rows and columns

Example

```
gap> source := GradedRow( [[-1],1]], cox_ring );
<A graded row of rank 1>
gap> range := GradedRow( [[0],1]], cox_ring );
<A graded row of rank 1>
gap> trunc_generators := GeneratorsOfDegreeXLayerOfGradedRowOrColumnAsListsOfRecords
>                        (P2, range, [ 2 ] );
[ 6, [ rec( ("x_1*x_2") := 2, ("x_1*x_3") := 4, ("x_1^2") := 1,
            ("x_2*x_3") := 5, ("x_2^2") := 3, ("x_3^2") := 6 ) ] ]
gap> vars := IndeterminatesOfPolynomialRing( cox_ring );;
gap> matrix := HomalgMatrix( [[ vars[ 1 ] ]], cox_ring );
<A 1 x 1 matrix over a graded ring>
gap> mor := GradedRowOrColumnMorphism( source, matrix, range );
<A morphism in Category of graded rows over
Q[x_1,x_2,x_3] (with weights [ 1, 1, 1 ])>
gap> IsWellDefined( mor );
true
gap> trunc_mor := TruncateGradedRowOrColumnMorphism( P2, mor, [ 2 ] );
<A morphism in Category of matrices over Q (with weights [ 1 ])>
gap> Display( UnderlyingMatrix( trunc_mor ) );
1,0,0,0,0,0,
0,1,0,0,0,0,
0,0,0,1,0,0
(over a graded ring)
gap> matrix2 := HomalgMatrix( [[ 1/2*vars[ 1 ] ]], cox_ring );
<A 1 x 1 matrix over a graded ring>
gap> mor2 := GradedRowOrColumnMorphism( source, matrix2, range );
<A morphism in Category of graded rows over
Q[x_1,x_2,x_3] (with weights [ 1, 1, 1 ])>
gap> IsWellDefined( mor2 );
true
gap> trunc_mor2 := TruncateGradedRowOrColumnMorphism( P2, mor2, [ 2 ] );
<A morphism in Category of matrices over Q (with weights [ 1 ])>
gap> Display( UnderlyingMatrix( trunc_mor2 ) );
1/2,0,0,0,0,0,
0,1/2,0,0,0,0,
0,0,0,1/2,0,0
(over a graded ring)
```

10.5.4 Truncatons of morphisms of graded rows and columns in parallel

Example

```
gap> trunc_mor_parallel := TruncateGradedRowOrColumnMorphismInParallel
>                                                ( P2, mor, [ 2 ], 2 );
<A morphism in Category of matrices over Q (with weights [ 1 ])>
gap> Display( UnderlyingMatrix( trunc_mor_parallel ) );
1,0,0,0,0,0,
0,1,0,0,0,0,
0,0,0,1,0,0
(over a graded ring)
gap> trunc_mor2_parallel := TruncateGradedRowOrColumnMorphismInParallel
>                                                ( P2, mor2, [ 2 ], 2 );
<A morphism in Category of matrices over Q (with weights [ 1 ])>
gap> Display( UnderlyingMatrix( trunc_mor2_parallel ) );
1/2,0,0,0,0,0,
0,1/2,0,0,0,0,
0,0,0,1/2,0,0
(over a graded ring)
gap> trunc_mor2_parallel2 := TruncateGradedRowOrColumnMorphismInParallel
>                                                ( P2, mor2, [ 10 ], 3 );;
gap> IsWellDefined( trunc_mor2_parallel2 );
true
gap> NrRows( UnderlyingMatrix( trunc_mor2_parallel2 ) );
55
gap> NrColumns( UnderlyingMatrix( trunc_mor2_parallel2 ) );
66
```

Chapter 11

Truncations of f.p. graded modules

11.1 Truncations of fp graded modules

11.1.1 TruncateFPGradedModule (for IsToricVariety, IsFpGradedLeftOrRightModulesObject, IsList, IsBool, IsFieldForHomalg)

▷ TruncateFPGradedModule(*V, M, d, B, F*) (operation)

Returns: a FreydCategoryObject

The arguments are a toric variety V, an f.p. graded module M, a list d (specifying a element of the class group of V) a boolean B and a field F. We then compute the truncation of M to the degree d and return the corresponding vector space presentation as a FreydCategoryObject. If B is true, we display additional information during the computation. The latter may be useful for longer computations.

Example

```
gap> P2 := ProjectiveSpace( 2 );
<A projective toric variety of dimension 2>
gap> cox_ring := CoxRing( P2 );
Q[x_1,x_2,x_3]
(weights: [ 1, 1, 1 ])
gap> source := GradedRow( [[[-1],1]], cox_ring );
<A graded row of rank 1>
gap> range := GradedRow( [[[0],1]], cox_ring );
<A graded row of rank 1>
gap> vars := IndeterminatesOfPolynomialRing( cox_ring );;
gap> matrix := HomalgMatrix( [[ vars[ 1 ] ]], cox_ring );
<A 1 x 1 matrix over a graded ring>
gap> obj1 := FreydCategoryObject(
>          GradedRowOrColumnMorphism( source, matrix, range ) );
<An object in Category of f.p. graded
left modules over Q[x_1,x_2,x_3]
(with weights [ 1, 1, 1 ])>
gap> IsWellDefined( obj1 );
true
gap> trunc_obj1 := TruncateFPGradedModule( P2, obj1, [ 2 ] );
<An object in Freyd( Category of matrices
over Q (with weights [ 1 ]) )>
gap> IsWellDefined( trunc_obj1 );
true
gap> Display( UnderlyingMatrix( RelationMorphism( trunc_obj1 ) ) );
```

```
1,0,0,0,0,0,
0,1,0,0,0,0,
0,0,0,1,0,0
(over a graded ring)
gap> trunc_obj2 := TruncateFPGradedModuleInParallel( P2, obj1, [ 2 ], 2 );
<An object in Freyd( Category of matrices
over Q (with weights [ 1 ]) )>
gap> IsWellDefined( trunc_obj2 );
true
gap> Display( UnderlyingMatrix( RelationMorphism( trunc_obj2 ) ) );
1,0,0,0,0,0,
0,1,0,0,0,0,
0,0,0,1,0,0
(over a graded ring)
```

11.2 Truncations of fp graded modules in parallel

11.2.1 TruncateFPGradedModuleInParallel (for IsToricVariety, IsFpGradedLeftOrRightModulesObject, IsList, IsPosInt, IsBool, IsFieldForHomalg)

▷ `TruncateFPGradedModuleInParallel(V, M, d, N, B., F)` (operation)

Returns: a FreydCategoryObject

The arguments are a toric variety V, an f.p. graded module M, a list d (specifying a element of the class group of V), an integer N, a boolean B and a field F. We then compute the truncation of M to the degree d and return the corresponding vector space presentation encoded as a FreydCategoryObject. This is performed in N child processes in parallel. If B is true, we display additional information during the computation. The latter may be useful for longer computations.

11.3 Truncations of fp graded modules morphisms

11.3.1 TruncateFPGradedModuleMorphism (for IsToricVariety, IsFpGradedLeftOrRightModulesMorphism, IsList, IsBool, IsFieldForHomalg)

▷ `TruncateFPGradedModuleMorphism(V, M, d, B, F)` (operation)

Returns: a FreydCategoryMorphism

The arguments are a toric variety V, an f.p. graded module morphism M, a list d (specifying a element of the class group of V), a boolean B and a field F. We then compute the truncation of M to the degree d and return the corresponding morphism of vector space presentations encoded as a FreydCategoryMorphism. If B is true, we display additional information during the computation. The latter may be useful for longer computations.

11.4 Truncations of fp graded modules morphisms in parallel

11.4.1 TruncateFPGradedModuleMorphismInParallel (for IsToricVariety, IsFpGradedLeftOrRightModulesMorphism, IsList, IsList, IsBool, IsFieldForHomalg)

▷ `TruncateFPGradedModuleMorphismInParallel(V, M, d[, N1, N2, N3], B, F)` (operation)

Returns: a FreydCategoryMorphism

The arguments are a toric variety V, an f.p. graded module morphism M, a list d (specifying a element of the class group of V), a list of 3 non-negative integers [N_1, N_2, N_3], a boolean B and a field F. We then compute the truncation of M to the degree d and return the corresponding morphism of vector space presentations encoded as a FreydCategoryMorphism. This is done in parallel: the truncation of the source is done by N_1 child processes in parallel, the truncation of the morphism datum is done by N_2 child processes and the truncation of the range of M by N_3 processes. If the boolean B is set to true, we display additional information during the computation. The latter may be useful for longer computations.

11.5 Truncations of f.p. graded module morphisms

Example

```
gap> source := GradedRow( [[-1],1]], cox_ring );
<A graded row of rank 1>
gap> range := GradedRow( [[1],2]], cox_ring );
<A graded row of rank 2>
gap> matrix := HomalgMatrix( [[ vars[ 1 ] * vars[ 2 ],
>                               vars[ 1 ] * vars[ 3 ] ]], cox_ring );
<A 1 x 2 matrix over a graded ring>
gap> obj2 := FreydCategoryObject(
>          GradedRowOrColumnMorphism( source, matrix, range ) );
<An object in Category of f.p. graded
left modules over Q[x_1,x_2,x_3]
(with weights [ 1, 1, 1 ])>
gap> source := GradedRow( [[0],1]], cox_ring );
<A graded row of rank 1>
gap> range := GradedRow( [[1],2]], cox_ring );
<A graded row of rank 2>
gap> matrix := HomalgMatrix( [[ vars[ 2 ], vars[ 3 ] ]], cox_ring );
<A 1 x 2 matrix over a graded ring>
gap> mor := GradedRowOrColumnMorphism( source, matrix, range );
<A morphism in Category of graded rows
over Q[x_1,x_2,x_3] (with weights [ 1, 1, 1 ])>
gap> pres_mor := FreydCategoryMorphism( obj1, mor, obj2 );
<A morphism in Category of f.p. graded
left modules over Q[x_1,x_2,x_3]
(with weights [ 1, 1, 1 ])>
gap> IsWellDefined( pres_mor );
true
gap> trunc_pres_mor1 := TruncateFPGradedModuleMorphism( P2, pres_mor, [ 2 ] );
<A morphism in Freyd( Category of
matrices over Q (with weights [ 1 ]) )>
gap> IsWellDefined( trunc_pres_mor1 );
true
```

```
gap> trunc_pres_mor2 := TruncateFPGradedModuleMorphismInParallel
>                            ( P2, pres_mor, [ 2 ], [ 2, 2, 2 ] );
<A morphism in Freyd( Category of
matrices over Q (with weights [ 1 ]))>
gap> IsWellDefined( trunc_pres_mor2 );
true
```

Chapter 12

Truncation functors for f.p. graded modules

12.1 Truncation functor for graded rows and columns

12.1.1 TruncationFunctorForGradedRows (for IsToricVariety, IsList)

▷ `TruncationFunctorForGradedRows(V, d)` (operation)

Returns: a functor

The arguments are a toric variety V and degree_list d 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, this method returns the functor for the truncation of graded rows over the Cox ring of V to degree d.

12.1.2 TruncationFunctorForGradedColumns (for IsToricVariety, IsList)

▷ `TruncationFunctorForGradedColumns(V, d)` (operation)

Returns: a functor

The arguments are a toric variety V and degree_list d 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, this method returns the functor for the truncation of graded columns over the Cox ring of V to degree d.

12.2 Truncation functor for f.p. graded modules

12.2.1 TruncationFunctorForFpGradedLeftModules (for IsToricVariety, IsList)

▷ `TruncationFunctorForFpGradedLeftModules(V, d)` (operation)

Returns: a functor

The arguments are a toric variety V and degree list d, which specifies an element of the degree group of the toric variety V. d can either be a list of integers or a HomalgModuleElement. Based on this input, this method returns the functor for the truncation of f.p. graded right modules to degree d.

12.2.2 TruncationFunctorForFpGradedRightModules (for IsToricVariety, IsList)

▷ `TruncationFunctorForFpGradedRightModules(V, d)` (operation)

Returns: a functor

The arguments are a toric variety V and degree list d, which specifies an element of the degree group of the toric variety V. d can either be a list of integers or a HomalgModuleElement. Based on this input, this method returns the functor for the truncation of f.p. graded right modules to degree d.

12.3 Examples

Example

```
gap> P2 := ProjectiveSpace( 2 );
<A projective toric variety of dimension 2>
gap> P1 := ProjectiveSpace( 1 );
<A projective toric variety of dimension 1>
gap> tor := P2 * P1;
<A projective toric variety of dimension 3
which is a product of 2 toric varieties>
gap> TruncationFunctorForGradedRows( tor, [ 2, 3 ] );
Trunction functor for Category of graded rows
over Q[x_1,x_2,x_3,x_4,x_5] (with weights
[ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ],
[ 0, 1 ], [ 0, 1 ] ] ) to the degree [ 2, 3 ]
gap> TruncationFunctorForFpGradedLeftModules( tor, [ 4, 5 ] );
Truncation functor for Category of f.p.
graded left modules over Q[x_1,x_2,x_3,x_4,x_5]
(with weights [ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ],
[ 0, 1 ], [ 0, 1 ] ]) to the degree [ 4, 5 ]
```

Chapter 13

Truncations of GradedExt for f.p. graded modules

13.1 Truncations of InternalHoms of FpGradedModules

13.1.1 TruncateInternalHom (for IsToricVariety, IsFpGradedLeftOrRightModulesObject, IsFpGradedLeftOrRightModulesObject, IsList, IsBool, IsFieldForHomalg)

▷ `TruncateInternalHom(`*arg1, arg2, arg3, arg4, arg5, arg6*`)` (operation)

13.1.2 TruncateInternalHomEmbedding (for IsToricVariety, IsFpGradedLeftOrRightModulesObject, IsFpGradedLeftOrRightModulesObject, IsList, IsBool, IsFieldForHomalg)

▷ `TruncateInternalHomEmbedding(`*arg1, arg2, arg3, arg4, arg5, arg6*`)` (operation)

13.1.3 TruncateInternalHom (for IsToricVariety, IsFpGradedLeftOrRightModulesMorphism, IsFpGradedLeftOrRightModulesMorphism, IsList, IsBool, IsFieldForHomalg)

▷ `TruncateInternalHom(`*arg1, arg2, arg3, arg4, arg5, arg6*`)` (operation)

13.2 Truncations of InternalHoms of FpGradedModules to degree zero

13.2.1 TruncateInternalHomToZero (for IsToricVariety, IsFpGradedLeftOrRightModulesObject, IsFpGradedLeftOrRightModulesObject, IsBool, IsFieldForHomalg)

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

13.2.2 TruncateInternalHomEmbeddingToZero (for IsToricVariety, IsFpGradedLeftOrRightModulesObject, IsFpGradedLeftOrRightModulesObject, IsBool, IsFieldForHomalg)

▷ TruncateInternalHomEmbeddingToZero(*arg1, arg2, arg3, arg4, arg5*) (operation)

13.2.3 TruncateInternalHomToZero (for IsToricVariety, IsFpGradedLeftOrRightModulesMorphism, IsFpGradedLeftOrRightModulesMorphism, IsBool, IsFieldForHomalg)

▷ TruncateInternalHomToZero(*arg1, arg2, arg3, arg4, arg5*) (operation)

13.3 Truncations of InternalHoms of FpGradedModules in parallel

13.3.1 TruncateInternalHomInParallel (for IsToricVariety, IsFpGradedLeftOrRightModulesObject, IsFpGradedLeftOrRightModulesObject, IsList, IsBool, IsFieldForHomalg)

▷ TruncateInternalHomInParallel(*arg1, arg2, arg3, arg4, arg5, arg6*) (operation)

13.3.2 TruncateInternalHomEmbeddingInParallel (for IsToricVariety, IsFpGradedLeftOrRightModulesObject, IsFpGradedLeftOrRightModulesObject, IsList, IsBool, IsFieldForHomalg)

▷ TruncateInternalHomEmbeddingInParallel(*arg1, arg2, arg3, arg4, arg5, arg6*) (operation)

13.3.3 TruncateInternalHomInParallel (for IsToricVariety, IsFpGradedLeftOrRightModulesMorphism, IsFpGradedLeftOrRightModulesMorphism, IsList, IsBool, IsFieldForHomalg)

▷ TruncateInternalHomInParallel(*arg1, arg2, arg3, arg4, arg5, arg6*) (operation)

13.4 Truncations of InternalHoms of FpGradedModules to degree zero in parallel

13.4.1 TruncateInternalHomToZeroInParallel (for IsToricVariety, IsFpGradedLeftOrRightModulesObject, IsFpGradedLeftOrRightModulesObject, IsBool, IsFieldForHomalg)

▷ TruncateInternalHomToZeroInParallel(*arg1, arg2, arg3, arg4, arg5*) (operation)

13.4.2 TruncateInternalHomEmbeddingToZeroInParallel (for IsToricVariety, IsFpGradedLeftOrRightModulesObject, IsFpGradedLeftOrRightModulesObject, IsBool, IsFieldForHomalg)

▷ TruncateInternalHomEmbeddingToZeroInParallel(*arg1, arg2, arg3, arg4, arg5*) (operation)

13.4.3 TruncateInternalHomToZeroInParallel (for IsToricVariety, IsFpGradedLeftOrRightModulesMorphism, IsFpGradedLeftOrRightModulesMorphism, IsBool, IsFieldForHomalg)

▷ TruncateInternalHomToZeroInParallel(*arg1, arg2, arg3, arg4, arg5*) (operation)

13.4.4 TruncateGradedExt (for IsInt, IsToricVariety, IsFpGradedLeftOrRightModulesObject, IsFpGradedLeftOrRightModulesObject, IsList, IsList)

▷ TruncateGradedExt(*arg1, arg2, arg3, arg4, arg5, arg6*) (operation)

13.4.5 TruncateGradedExtToZero (for IsInt, IsToricVariety, IsFpGradedLeftOrRightModulesObject, IsFpGradedLeftOrRightModulesObject, IsBool, IsFieldForHomalg)

▷ TruncateGradedExtToZero(*arg1, arg2, arg3, arg4, arg5, arg6*) (operation)

13.4.6 TruncateGradedExtInParallel (for IsInt, IsToricVariety, IsFpGradedLeftOrRightModulesObject, IsFpGradedLeftOrRightModulesObject, IsList, IsList)

▷ TruncateGradedExtInParallel(*arg1, arg2, arg3, arg4, arg5, arg6*) (operation)

13.4.7 TruncateGradedExtToZeroInParallel (for IsInt, IsToricVariety, IsFpGradedLeftOrRightModulesObject, IsFpGradedLeftOrRightModulesObject, IsBool, IsFieldForHomalg)

▷ TruncateGradedExtToZeroInParallel(*arg1, arg2, arg3, arg4, arg5, arg6*) (operation)

13.5 Examples

13.5.1 Truncation of IntHom

Example

```
gap> P2 := ProjectiveSpace( 2 );
<A projective toric variety of dimension 2>
gap> cox_ring := CoxRing( P2 );
```

```
Q[x_1,x_2,x_3]
(weights: [ 1, 1, 1 ])
gap> source := GradedRow( [[[-1],1]], cox_ring );
<A graded row of rank 1>
gap> range := GradedRow( [[[0],1]], cox_ring );
<A graded row of rank 1>
gap> vars := IndeterminatesOfPolynomialRing( cox_ring );;
gap> matrix := HomalgMatrix( [[ vars[ 1 ] ]], cox_ring );
<A 1 x 1 matrix over a graded ring>
gap> obj1 := FreydCategoryObject(
>          GradedRowOrColumnMorphism( source, matrix, range ) );
<An object in Category of f.p. graded
left modules over Q[x_1,x_2,x_3]
(with weights [ 1, 1, 1 ])>
gap> IsWellDefined( obj1 );
true
gap> source := GradedRow( [[[-1],1]], cox_ring );
<A graded row of rank 1>
gap> range := GradedRow( [[[1],2]], cox_ring );
<A graded row of rank 2>
gap> matrix := HomalgMatrix( [[ vars[ 1 ] * vars[ 2 ],
>                               vars[ 1 ] * vars[ 3 ] ]], cox_ring );
<A 1 x 2 matrix over a graded ring>
gap> obj2 := FreydCategoryObject(
>          GradedRowOrColumnMorphism( source, matrix, range ) );
<An object in Category of f.p. graded
left modules over Q[x_1,x_2,x_3]
(with weights [ 1, 1, 1 ])>
gap> IsWellDefined( obj2 );
true
gap> source := GradedRow( [[[0],1]], cox_ring );
<A graded row of rank 1>
gap> range := GradedRow( [[[1],2]], cox_ring );
<A graded row of rank 2>
gap> matrix := HomalgMatrix( [[ vars[ 2 ], vars[ 3 ] ]], cox_ring );
<A 1 x 2 matrix over a graded ring>
gap> mor := GradedRowOrColumnMorphism( source, matrix, range );
<A morphism in Category of graded rows
over Q[x_1,x_2,x_3] (with weights [ 1, 1, 1 ])>
gap> pres_mor := FreydCategoryMorphism( obj1, mor, obj2 );
<A morphism in Category of f.p. graded
left modules over Q[x_1,x_2,x_3]
(with weights [ 1, 1, 1 ])>
gap> IsWellDefined( pres_mor );
true
gap> Q := HomalgFieldOfRationalsInSingular();
Q
gap> m1 := TruncateInternalHom( P2, obj1, obj2, [ 4 ], false, Q );
<An object in Freyd( Category of matrices over Q )>
gap> IsWellDefined( m1 );
true
gap> m2 := TruncateInternalHomEmbedding( P2, obj1, obj2, [ 4 ], false, Q );
```

```
<A monomorphism in Freyd( Category of matrices over Q )>
gap> IsWellDefined( m2 );
true
gap> m3 := TruncateInternalHom( P2, pres_mor, IdentityMorphism( obj2 ), [ 4 ], false, Q );
<A morphism in Freyd( Category of matrices over Q )>
gap> IsWellDefined( m3 );
true
```

13.5.2 Truncation of IntHom to degree zero

Example

```
gap> m4 := TruncateInternalHomToZero( P2, obj1, obj2, false, Q );
<An object in Freyd( Category of matrices over Q )>
gap> IsWellDefined( m4 );
true
gap> m5 := TruncateInternalHomEmbeddingToZero( P2, obj1, obj2, false, Q );
<A monomorphism in Freyd( Category of matrices over Q )>
gap> IsWellDefined( m5 );
true
gap> m6 := TruncateInternalHomToZero( P2, pres_mor, IdentityMorphism( obj2 ), false, Q );
<A morphism in Freyd( Category of matrices over Q )>
gap> IsWellDefined( m6 );
true
```

13.5.3 Truncation of IntHom in parallel

Example

```
gap> m7 := TruncateInternalHomInParallel( P2, obj1, obj2, [ 4 ], false, Q );
<An object in Freyd( Category of matrices over Q )>
gap> m1 = m7;
true
gap> m8 := TruncateInternalHomEmbeddingInParallel( P2, obj1, obj2, [ 4 ], false, Q );
<A monomorphism in Freyd( Category of matrices over Q )>
gap> m8 = m2;
true
gap> m9 := TruncateInternalHomInParallel( P2, pres_mor, IdentityMorphism( obj2 ), [ 4 ], false, Q
<A morphism in Freyd( Category of matrices over Q )>
gap> m9 = m3;
true
```

13.5.4 Truncation of IntHom to degree zero in parallel

Example

```
gap> m10 := TruncateInternalHomToZeroInParallel( P2, obj1, obj2, false, Q );
<An object in Freyd( Category of matrices over Q )>
gap> m10 = m4;
true
gap> m11 := TruncateInternalHomEmbeddingToZeroInParallel( P2, obj1, obj2, false, Q );
<A monomorphism in Freyd( Category of matrices over Q )>
gap> m11 = m5;
true
gap> m12 := TruncateInternalHomToZeroInParallel( P2, pres_mor, IdentityMorphism( obj2 ), false, Q
<A morphism in Freyd( Category of matrices over Q )>
```

```
gap> m12 = m6;
true
```

13.5.5 Truncation of GradedExt

Example

```
gap> v1 := TruncateGradedExt( 1, P2, obj1, obj2, [ 4 ], [ false, Q ] );
<An object in Freyd( Category of matrices over Q )>
gap> IsWellDefined( v1 );
true
gap> v2 := TruncateGradedExt( 1, P2, obj1, obj2, [ 0 ], [ false, Q ] );
<An object in Freyd( Category of matrices over Q )>
gap> IsWellDefined( v2 );
true
gap> v3 := TruncateGradedExtToZero( 1, P2, obj1, obj2, false, Q );
<An object in Freyd( Category of matrices over Q )>
gap> v3 = v2;
true
gap> v4 := TruncateGradedExtInParallel( 1, P2, obj1, obj2, [ 4 ], [ false, Q ] );
<An object in Freyd( Category of matrices over Q )>
gap> IsWellDefined( v4 );
true
gap> v5 := TruncateGradedExtInParallel( 1, P2, obj1, obj2, [ 0 ], [ false, Q ] );
<An object in Freyd( Category of matrices over Q )>
gap> IsWellDefined( v5 );
true
gap> v6 := TruncateGradedExtToZeroInParallel( 1, P2, obj1, obj2, false, Q );
<An object in Freyd( Category of matrices over Q )>
gap> v6 = v5;
true
```

Chapter 14

Sheaf cohomology by use of https://arxiv.org/abs/1802.08860

14.1 Preliminaries

14.1.1 ParameterCheck (for IsToricVariety, IsFpGradedLeftOrRightModulesObject, IsFpGradedLeftOrRightModulesObject, IsInt)

▷ `ParameterCheck(V, M1, M2, i)` (operation)

Returns: true or false

Given a toric variety V, we eventually wish to compute the i-th sheaf cohomology of the sheafification of the f.p. graded S-module M_2 (S being the Cox ring of vari). To this end we use modules M_1 which sheafify to the structure sheaf of vari. This method tests if the truncation to degree zero of $Ext^i_S(M_1,M_2)$ is isomorphic to $H^i(V,\widetilde{M_2})$.

14.1.2 FindIdeal (for IsToricVariety, IsFpGradedLeftOrRightModulesObject, IsInt)

▷ `FindIdeal(V, M, i)` (operation)

Returns: a list

Given a toric variety V and an f.p. graded S-module M (S being the Cox ring of vari), we wish to compute the i-th sheaf cohomology of $\tilde{M}$. To this end, this method identifies an ideal I of S such that $\tilde{I}$ is the structure sheaf of V and such that $Ext^i_S(I,M)$ is isomorphic to $H^i(V,\tilde{M})$. We identify I by determining an ample degree $d \in \mathrm{Cl}(V)$. Then, for a suitable non-negative integer e, the generators of I are the e-th power of all monomials of degree d in the Cox ring of S. We return the list [e, d, I].

14.2 Computation of global sections

14.2.1 H0 (for IsToricVariety, IsFpGradedLeftOrRightModulesObject)

▷ `H0(V, M)` (operation)

Returns: a vector space

Given a variety V and an f.p. graded S-module M (S being the Cox ring of V), this method computes $H^0(V,\tilde{M})$.

14.2.2 H0Parallel (for IsToricVariety, IsFpGradedLeftOrRightModulesObject)

▷ `H0Parallel(V, M)` (operation)

Returns: a vector space

Given a variety V and an f.p. graded S-module M (S being the Cox ring of V), this method computes $H^0(V,\tilde{M})$. This method is parallelized and is thus best suited for long and complicated computations.

14.3 Computation of the i-th sheaf cohomologies

14.3.1 Hi (for IsToricVariety, IsFpGradedLeftOrRightModulesObject, IsInt)

▷ `Hi(V, M, i)` (operation)

Returns: a vector space

Given a variety V and an f.p. graded S-module M (S being the Cox ring of V), this method computes $H^i(V,\tilde{M})$.

14.3.2 HiParallel (for IsToricVariety, IsFpGradedLeftOrRightModulesObject, IsInt)

▷ `HiParallel(V, M, i)` (operation)

Returns: a vector space

Given a variety V and an f.p. graded S-module M (S being the Cox ring of V), this method computes $H^i(V,\tilde{M})$. This method is parallelized and is thus best suited for long and complicated computations.

14.4 Computation of all sheaf cohomologies

14.4.1 AllHi (for IsToricVariety, IsFpGradedLeftOrRightModulesObject)

▷ `AllHi(V, M)` (operation)

Returns: a list of vector spaces

Given a variety V and an f.p. graded S-module M (S being the Cox ring of V), this method computes all sheaf cohomologies $H^*(V,\tilde{M})$.

14.4.2 AllHiParallel (for IsToricVariety, IsFpGradedLeftOrRightModulesObject)

▷ `AllHiParallel(V, M)` (operation)

Returns: a list of vector spaces

Given a variety V and an f.p. graded S-module M (S being the Cox ring of V), this method computes all sheaf cohomologies $H^*(V,\tilde{M})$. This method is parallelized and is thus best suited for long and complicated computations.

14.5 Examples

14.5.1 Sheaf cohomology of toric vector bundles

Example

```
gap> F1 := Fan( [[1],[-1]],[[1],[2]] );
<A fan in |R^1>
```

```
gap> P1 := ToricVariety( F1 );
<A toric variety of dimension 1>
gap> P1xP1 := P1 * P1;
<A toric variety of dimension 2 which is a product of 2 toric varieties>
gap> VForCAP := AsFreydCategoryObject( GradedRow( [[[1,1],1],[[-2,0],1]],
>                                                  CoxRing( P1xP1 ) ) );
<A projective object in Category of f.p. graded
left modules over Q[x_1,x_2,x_3,x_4] (with weights
[ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ], [ 0, 1 ] ])>
gap> V2ForCAP := AsFreydCategoryObject( GradedRow( [[[-2,0],1]],
>                                                  CoxRing( P1xP1 ) ) );
<A projective object in Category of f.p. graded
left modules over Q[x_1,x_2,x_3,x_4] (with weights
[ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ], [ 0, 1 ] ])>
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> AllHiParallel( 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, V2ForCAP, false, false );
Computing h^0
-----------------------------------------------

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

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

[ [ 0, <A vector space object over Q of dimension 0> ],
  [ 1, <A vector space object over Q of dimension 1> ],
```

```
  [ 1, <A vector space object over Q of dimension 0> ] ]
gap> AllHiParallel( P1xP1, V2ForCAP, false, false );
Computing h^0
-----------------------------------------------

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

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

[ [ 0, <A vector space object over Q of dimension 0> ],
  [ 1, <A vector space object over Q of dimension 1> ],
  [ 1, <A vector space object over Q of dimension 0> ] ]
```

14.5.2 Sheaf cohomologies of the irrelevant ideal of P1xP1

Example

```
gap> irP1xP1 := IrrelevantLeftIdealForCAP( P1xP1 );
<An object in Category of f.p. graded left
modules over Q[x_1,x_2,x_3,x_4] (with weights
[ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ], [ 0, 1 ] ])>
gap> AllHi( P1xP1, irP1xP1, 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> AllHiParallel( P1xP1, irP1xP1, 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> ] ]
```

Chapter 15

Tools for cohomology computations

15.1 Turn CAP Graded Modules into old graded modules and vice versa

15.1.1 TurnIntoOldGradedModule (for IsFpGradedLeftOrRightModulesObject)

▷ `TurnIntoOldGradedModule(`*M*`)` (operation)

Returns: the corresponding graded modules in terms of the 'old' packages GradedModules

The argument is a graded left or right module presentation M for CAP

15.2 Save CAP f.p. graded module to file

15.2.1 SaveToFileAsOldGradedModule (for IsString, IsFpGradedLeftOrRightModulesObject)

▷ `SaveToFileAsOldGradedModule(`*M*`)` (operation)

Returns: true (in case of success) or raises error in case the file could not be written

The argument is a graded left or right module presentation M for CAP and saves this module to file as 'old' graded module presentation. By default, the files are saved in the main directory of the package 'SheafCohomologyOnToricVarieties'.

15.2.2 SaveToFileAsCAPGradedModule (for IsString, IsFpGradedLeftOrRightModulesObject)

▷ `SaveToFileAsCAPGradedModule(`*M*`)` (operation)

Returns: true (in case of success) or raises error in case the file could not be written

The argument is a graded left or right module presentation M for CAP and saves this module to file as CAP graded module presentation. By default, the files are saved in the main directory of the package 'SheafCohomologyOnToricVarieties'.

15.3 Approximation Of Sheaf Cohomologies

15.3.1 BPowerLeft (for IsToricVariety, IsInt)

▷ `BPowerLeft(V, e)` (operation)

Returns: a CAP graded left module

The argument is a toric variety V and a non-negative integer e. The method computes the e-th Frobenius power of the irrelevant left ideal of V.

15.3.2 BPowerRight (for IsToricVariety, IsInt)

▷ `BPowerRight(V, e)` (operation)

Returns: a CAP graded right module

The argument is a toric variety V and a non-negative integer e. The method computes the e-th Frobenius power of the irrelevant right ideal of V.

15.3.3 ApproxH0 (for IsToricVariety, IsInt, IsFpGradedLeftOrRightModulesObject)

▷ `ApproxH0(V, e, M)` (operation)

Returns: a non-negative integer

The argument is a toric variety V, a non-negative integer e and a graded CAP module M. The method computes the degree zero layer of Hom(B(e), M) and returns its vector space dimension.

15.3.4 ApproxH0Parallel (for IsToricVariety, IsInt, IsFpGradedLeftOrRightModulesObject)

▷ `ApproxH0Parallel(V, e, M)` (operation)

Returns: a non-negative integer

The argument is a toric variety V, a non-negative integer e and a graded CAP module M. The method computes the degree zero layer of Hom(B(e), M) by use of parallelisation and returns its vector space dimension.

15.3.5 ApproxHi (for IsToricVariety, IsInt, IsInt, IsFpGradedLeftOrRightModulesObject)

▷ `ApproxHi(V, i, e, M)` (operation)

Returns: a non-negative integer

The argument is a toric variety V, non-negative integers i, e and a graded CAP module M. The method computes the degree zero layer of Ext^i(B(e), M) and returns its vector space dimension.

15.3.6 ApproxHiParallel (for IsToricVariety, IsInt, IsInt, IsFpGradedLeftOrRightModulesObject)

▷ `ApproxHiParallel(V, i, e, M)` (operation)

Returns: a non-negative integer

The argument is a toric variety V, non-negative integer i, e and a graded CAP module M. The method computes the degree zero layer of Ext^i(B(e), M) by use of parallelisation and returns its vector space dimension.

15.4 Examples

15.4.1 Conversion of modules

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> irP1xP1 := IrrelevantLeftIdealForCAP( P1xP1 );
<An object in Category of f.p. graded left
modules over Q[x_1,x_2,x_3,x_4] (with weights
[ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ], [ 0, 1 ] ])>
gap> module2 := TurnIntoOldGradedModule( irP1xP1 );
<A graded left module presented by 4 relations for 4 generators>
gap> module3 := TurnIntoCAPGradedModule( module2 );
<An object in Category of f.p. graded left
modules over Q[x_1,x_2,x_3,x_4] (with weights
[ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ], [ 0, 1 ] ])>
gap> module3 = irP1xP1;
true
```

15.4.2 Approximation of 0-th sheaf cohomology

Example

```
gap> ApproxH0( P1xP1, 0, irP1xP1 );
<A vector space object over Q of dimension 0>
gap> ApproxH0( P1xP1, 1, irP1xP1 );
<A vector space object over Q of dimension 1>
gap> ApproxH0( P1xP1, 2, irP1xP1 );
<A vector space object over Q of dimension 1>
gap> ApproxH0Parallel( P1xP1, 0, irP1xP1 );
<A vector space object over Q of dimension 0>
gap> ApproxH0Parallel( P1xP1, 1, irP1xP1 );
<A vector space object over Q of dimension 1>
gap> ApproxH0Parallel( P1xP1, 2, irP1xP1 );
<A vector space object over Q of dimension 1>
```

15.4.3 Approximation of 1-st sheaf cohomology

Example

```
gap> F1 := Fan( [[1],[-1]],[[1],[2]] );
<A fan in |R^1>
gap> P1 := ToricVariety( F1 );
<A toric variety of dimension 1>
gap> P1xP1 := P1 * P1;
<A toric variety of dimension 2 which is a product of 2 toric varieties>
gap> VForCAP := AsFreydCategoryObject( GradedRow( [[[1,1],1],[[-2,0],1]],
>                                                 CoxRing( P1xP1 ) ) );
<A projective object in Category of f.p. graded
left modules over Q[x_1,x_2,x_3,x_4] (with weights
[ [ 0, 1 ], [ 1, 0 ], [ 1, 0 ], [ 0, 1 ] ])>
gap> ApproxHi( P1xP1, 1, 0, VForCAP );
```

```
<A vector space object over Q of dimension 0>
gap> ApproxHi( P1xP1, 1, 1, VForCAP );
<A vector space object over Q of dimension 1>
gap> ApproxHi( P1xP1, 1, 2, VForCAP );
<A vector space object over Q of dimension 1>
gap> ApproxHiParallel( P1xP1, 1, 0, VForCAP );
<A vector space object over Q of dimension 0>
gap> ApproxHiParallel( P1xP1, 1, 1, VForCAP );
<A vector space object over Q of dimension 1>
gap> ApproxHiParallel( P1xP1, 1, 2, VForCAP );
<A vector space object over Q of dimension 1>
```

Index