AdditionsForToricVarieties

A package to provide additional structures for toric varieties

2021.11.17

17 November 2021

Martin Bies

Martin Bies
Email: martin.bies@alumni.uni-heidelberg.de
Homepage: https://martinbies.github.io/
Address: Department of Mathematics
University of Pennsylvania
David Rittenhouse Laboratory
209 S 33rd St
Philadelphia
PA 19104

Copyright

Contents

Chapter 1

Introduction

1.1 What is the goal of the AdditionsForToricVarieties package?

AdditionsForToricVarieties provides additional structures which are not provided by the ToricVarieties package in the homalg_project yet required for the computations of sheaf cohomologies on toric varieties.

Chapter 2

Central functions and constants

2.1 Input check for valid input

2.1.1 IsValidInputForAdditionsForToricVarieties (for IsToricVariety)

▷ `IsValidInputForAdditionsForToricVarieties(`*V*`)` (property)

Returns: `true` or `false`

Returns if the given variety V is a valid input for the additional operations. This is the case if it is smooth and complete. If the boolean ADDITIONS_TORIC_VARIETIES_LAZY_INPUT is set to true, then this will also be satisfied if the variety is simplicial and projective.

Chapter 3

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

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

3.1.1 IsICTCurve (for IsObject)

▷ `IsICTCurve(object)` (filter)

Returns: true or false

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

3.2 Constructors for ICT Curves

3.2.1 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.

3.3 Attributes for ICT curves

3.3.1 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.

3.3.2 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.

3.3.3 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

3.3.4 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.

3.3.5 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.

3.3.6 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.

3.3.7 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.

3.3.8 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 $e_i = (0, \ldots, 0, 1, 0, \ldots, 0) \in Cl(X_\Sigma)$.

3.4 Operations with ICTCurves

3.4.1 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.

3.4.2 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.

3.4.3 IntersectionProduct (for IsToricDivisor, IsICTCurve)

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

3.5 GAP category for proper 1-cycles

3.5.1 IsProper1Cycle (for IsObject)

▷ `IsProper1Cycle(object)` (filter)

Returns: true or false

The GAP category for proper 1-cycles

3.6 Constructor For Proper 1-Cycles

3.6.1 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.

3.6.2 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.

3.7 Attributes for proper 1-cycles

3.7.1 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.

3.7.2 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}).

3.8 Operations with proper 1-cycles

3.8.1 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.

3.8.2 IntersectionProduct (for IsToricDivisor, IsProper1Cycle)

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

3.8.3 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.

3.9 Examples in projective space

Example

```
gap> HOMALG_IO.show_banners := false;;
gap> HOMALG_IO.suppress_PID := true;;
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 4

Monoms of Coxring of given degree

4.1 Monoms of given degree in the Cox ring

4.1.1 Exponents (for IsToricVariety, IsList)

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

Returns: a list of lists of integers

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

4.1.2 MonomsOfCoxRingOfDegreeByNormaliz (for IsToricVariety, IsList)

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

Returns: a list

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

4.1.3 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'.

4.2 Example: monoms of Cox ring of 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 5

Nef and Mori Cone

5.1 New Properties For Toric Divisors

5.1.1 IsNef (for IsToricDivisor)

▷ `IsNef(divi)` (property)

Returns: true or false

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

5.1.2 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.

5.2 Attributes

5.2.1 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.

5.2.2 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.

5.2.3 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.

5.2.4 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.

5.2.5 NefCone (for IsToricVariety)

▷ `NefCone(arg)` (attribute)

5.2.6 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.

5.2.7 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.

5.2.8 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.

5.3 Nef and Mori Cone: Examples

5.3.1 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 6

Stanley Reisner and irrelevant ideal as FPGradedModules

6.1 Irrelevant ideal via FpGradedModules

6.1.1 GeneratorsOfIrrelevantIdeal (for IsToricVariety)

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

Returns: a list

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

6.1.2 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.

6.1.3 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.

6.2 Stanley-Reisner ideal via FPGradedModules

6.2.1 GeneratorsOfSRIdeal (for IsToricVariety)

▷ `GeneratorsOfSRIdeal(`*`vari`*`)` (attribute)

Returns: a list

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

6.2.2 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.

6.2.3 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.

6.3 FPGraded left and right modules over the Cox ring

6.3.1 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.

6.3.2 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.

6.4 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
```

6.5 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
```

6.6 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 7

Wrapper for generators of semigroups and hyperplane constraints of cones

7.1 GAP Categories

7.1.1 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.

7.1.2 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 $H = p + G$.

7.2 Constructors

7.2.1 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.

7.2.2 SemigroupForPresentationsByProjectiveGradedModules (for IsList)

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

7.2.3 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).

7.2.4 AffineSemigroupForPresentationsByProjectiveGradedModules (for IsList, IsList)

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

7.2.5 AffineSemigroupForPresentationsByProjectiveGradedModules (for IsList, IsInt, IsList)

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

7.3 Attributes

7.3.1 GeneratorList (for IsSemigroupForPresentationsByProjectiveGradedModules)

▷ `GeneratorList(`*L*`)` (attribute)

Returns: a list

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

7.3.2 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.

7.3.3 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'.

7.3.4 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.

7.3.5 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.

7.3.6 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.

7.4 Property

7.4.1 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.

7.4.2 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.

7.4.3 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'.

7.4.4 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.

7.5 Operations

7.5.1 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'.

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

7.6.1 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.

7.6.2 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.

7.7 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 8

Vanishing sets on toric varieties

8.1 GAP category for vanishing sets

8.1.1 IsVanishingSet (for IsObject)

▷ `IsVanishingSet(`*arg*`)` (filter)

Returns: `true` or `false`

The GAP category of vanishing sets formed from affine semigroups.

8.2 Constructors

8.2.1 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'.

8.2.2 VanishingSet (for IsToricVariety, IsList, IsString)

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

8.3 Attributes

8.3.1 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.

8.3.2 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.

8.3.3 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 $H^i(X_\Sigma, \mathcal{O}(D)) = 0$. We return the integer i.

8.3.4 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 $H^i(X_\Sigma, \mathcal{O}(D)) = 0$ for all $i > 0$. If such a specification is known, it will be returned by this method.

8.3.5 AmbientToricVariety (for IsVanishingSet)

▷ `AmbientToricVariety(V)` (attribute)

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

8.4 Property

8.4.1 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.

8.5 Improved vanishing sets via cohomCalg

8.5.1 TurnDenominatorIntoShiftedSemigroup (for IsToricVariety, IsString)

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

8.5.2 VanishingSets (for IsToricVariety)

▷ `VanishingSets(arg)` (attribute)

8.5.3 ComputeVanishingSets (for IsToricVariety, IsBool)

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

8.5.4 PointContainedInVanishingSet (for IsVanishingSet, IsList)

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

8.6 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 ] ]
```

Index