ToricVarieties

A GAP package for handling toric varieties.

Version 2012.12.22

October 2012

Sebastian Gutsche

This manual is best viewed as an HTML document. An OFFLINE version should be included in the documentation subfolder of the package.

Sebastian Gutsche

Email: sebastian.gutsche@rwth-aachen.de

Homepage: http://wwwb.math.rwth-aachen.de/~gutsche

Address: Lehrstuhl B für Mathematik, RWTH Aachen, Templergraben 64, 52056 Aachen, Germany

Copyright

Acknowledgements

Contents

Chapter 1

Introduction

1.1 What is the goal of the ToricVarieties package?

ToricVarieties provides data structures to handle toric varieties by their commutative algebra structure and by their combinatorics. For combinatorics, it uses the Convex package. Its goal is to provide a suitable framework to work with toric varieties. All combinatorial structures mentioned in this manual are the ones from Convex.

Chapter 2

Installation of the ToricVarieties Package

To install this package just extract the package's archive file to the GAP pkg directory.

By default the ToricVarieties package is not automatically loaded by GAP when it is installed. You must load the package with

```
LoadPackage( "ToricVarieties" );
```

before its functions become available.

Please, send me an e-mail if you have any questions, remarks, suggestions, etc. concerning this package. Also, I would be pleased to hear about applications of this package and about any suggestions for new methods to add to the package.

Sebastian Gutsche

Chapter 3

Toric varieties

3.1 Toric variety: Category and Representations

3.1.1 IsToricVariety

▷ `IsToricVariety(`*M*`)` (Category)

Returns: `true` or `false`

The GAP category of a toric variety.

3.2 Toric varieties: Properties

3.2.1 IsNormalVariety

▷ `IsNormalVariety(`*vari*`)` (property)

Returns: `true` or `false`

Checks if the toric variety *vari* is a normal variety.

3.2.2 IsAffine

▷ `IsAffine(`*vari*`)` (property)

Returns: `true` or `false`

Checks if the toric variety *vari* is an affine variety.

3.2.3 IsProjective

▷ `IsProjective(`*vari*`)` (property)

Returns: `true` or `false`

Checks if the toric variety *vari* is a projective variety.

3.2.4 IsComplete

▷ `IsComplete(`*vari*`)` (property)

Returns: `true` or `false`

Checks if the toric variety *vari* is a complete variety.

3.2.5 IsSmooth

▷ `IsSmooth(vari)` (property)

Returns: `true` or `false`

Checks if the toric variety *vari* is a smooth variety.

3.2.6 HasTorusfactor

▷ `HasTorusfactor(vari)` (property)

Returns: `true` or `false`

Checks if the toric variety *vari* has a torus factor.

3.2.7 HasNoTorusfactor

▷ `HasNoTorusfactor(vari)` (property)

Returns: `true` or `false`

Checks if the toric variety *vari* has no torus factor.

3.2.8 IsOrbifold

▷ `IsOrbifold(vari)` (property)

Returns: `true` or `false`

Checks if the toric variety *vari* has an orbifold, which is, in the toric case, equivalent to the simpliciality of the fan.

3.3 Toric varieties: Attributes

3.3.1 AffineOpenCovering

▷ `AffineOpenCovering(vari)` (attribute)

Returns: a list

Returns a torus invariant affine open covering of the variety *vari*. The affine open cover is computed out of the cones of the fan.

3.3.2 CoxRing

▷ `CoxRing(vari)` (attribute)

Returns: a ring

Returns the Cox ring of the variety *vari*. The actual method requires a string with a name for the variables. A method for computing the Cox ring without a variable given is not implemented. You will get an error.

3.3.3 ListOfVariablesOfCoxRing

▷ `ListOfVariablesOfCoxRing(vari)` (attribute)

Returns: a list

Returns a list of the variables of the cox ring of the variety *vari*.

3.3.4 ClassGroup

▷ `ClassGroup(vari)` (attribute)

Returns: a module

Returns the class group of the variety *vari* as factor of a free module.

3.3.5 PicardGroup

▷ `PicardGroup(vari)` (attribute)

Returns: a module

Returns the Picard group of the variety *vari* as factor of a free module.

3.3.6 TorusInvariantDivisorGroup

▷ `TorusInvariantDivisorGroup(vari)` (attribute)

Returns: a module

Returns the subgroup of the Weil divisor group of the variety *vari* generated by the torus invariant prime divisors. This is always a finitely generated free module over the integers.

3.3.7 MapFromCharacterToPrincipalDivisor

▷ `MapFromCharacterToPrincipalDivisor(vari)` (attribute)

Returns: a morphism

Returns a map which maps an element of the character group into the torus invariant Weil group of the variety *vari*. This has to viewn as an help method to compute divisor classes.

3.3.8 Dimension

▷ `Dimension(vari)` (attribute)

Returns: an integer

Returns the dimension of the variety *vari*.

3.3.9 DimensionOfTorusfactor

▷ `DimensionOfTorusfactor(vari)` (attribute)

Returns: an integer

Returns the dimension of the torus factor of the variety *vari*.

3.3.10 CoordinateRingOfTorus

▷ `CoordinateRingOfTorus(vari)` (attribute)

Returns: a ring

Returns the coordinate ring of the torus of the variety *vari*. This method is not implemented, you need to call it with a second argument, which is a list of strings for the variables of the ring.

3.3.11 IsProductOf

▷ `IsProductOf(vari)` (attribute)

Returns: a list

If the variety *vari* is a product of 2 or more varieties, the list contain those varieties. If it is not a product or at least not generated as a product, the list only contains the variety itself.

3.3.12 CharacterLattice

▷ `CharacterLattice(vari)` (attribute)

Returns: a module

The method returns the character lattice of the variety *vari*, computed as the containing grid of the underlying convex object, if it exists.

3.3.13 TorusInvariantPrimeDivisors

▷ `TorusInvariantPrimeDivisors(vari)` (attribute)

Returns: a list

The method returns a list of the torus invariant prime divisors of the variety *vari*.

3.3.14 IrrelevantIdeal

▷ `IrrelevantIdeal(vari)` (attribute)

Returns: an ideal

Returns the irrelevant ideal of the cox ring of the variety *vari*.

3.3.15 MorphismFromCoxVariety

▷ `MorphismFromCoxVariety(vari)` (attribute)

Returns: a morphism

The method returns the quotient morphism from the variety of the Cox ring to the variety *vari*.

3.3.16 CoxVariety

▷ `CoxVariety(vari)` (attribute)

Returns: a variety

The method returns the Cox variety of the variety *vari*.

3.3.17 FanOfVariety

▷ `FanOfVariety(vari)` (attribute)

Returns: a fan

Returns the fan of the variety *vari*. This is set by default.

3.3.18 CartierTorusInvariantDivisorGroup

▷ `CartierTorusInvariantDivisorGroup(vari)` (attribute)

Returns: a module

Returns the the group of Cartier divisors of the variety *vari* as a subgroup of the divisor group.

3.3.19 NameOfVariety

▷ `NameOfVariety(vari)` (attribute)

Returns: a string

Returns the name of the variety *vari* if it has one and it is known or can be computed.

3.3.20 twitter

▷ `twitter(vari)` (attribute)

Returns: a ring

This is a dummy to get immediate methods triggered at some times. It never has a value.

3.4 Toric varieties: Methods

3.4.1 UnderlyingSheaf

▷ `UnderlyingSheaf(vari)` (operation)

Returns: a sheaf

The method returns the underlying sheaf of the variety *vari*.

3.4.2 CoordinateRingOfTorus (for a variety and a list of variables)

▷ `CoordinateRingOfTorus(vari, vars)` (operation)

Returns: a ring

Computes the coordinate ring of the torus of the variety *vari* with the variables *vars*. The argument *vars* need to be a list of strings with length dimension or two times dimension.

3.4.3 *

▷ `\*(vari1, vari2)` (operation)

Returns: a variety

Computes the categorial product of the varieties *vari1* and *vari2*.

3.4.4 CharacterToRationalFunction

▷ `CharacterToRationalFunction(elem, vari)` (operation)

Returns: a homalg element

Computes the rational function corresponding to the character grid element *elem* or to the list of integers *elem*. To compute rational functions you first need to compute to coordinate ring of the torus of the variety *vari*.

3.4.5 CoxRing (for a variety and a string of variables)

▷ `CoxRing(vari, vars)` (operation)

Returns: a ring

Computes the Cox ring of the variety *vari*. *vars* needs to be a string containing one variable, which will be numbered by the method.

3.4.6 WeilDivisorsOfVariety

▷ `WeilDivisorsOfVariety(vari)` (operation)

Returns: a list

Returns a list of the currently defined Divisors of the toric variety.

3.4.7 Fan

▷ `Fan(vari)` (operation)

Returns: a fan

Returns the fan of the variety *vari*. This is a rename for FanOfVariety.

3.5 Toric varieties: Constructors

3.5.1 ToricVariety

▷ `ToricVariety(conv)` (operation)

Returns: a ring

Creates a toric variety out of the convex object *conv*.

3.6 Toric varieties: Examples

3.6.1 The Hirzebruch surface of index 5

Example

```
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> IsComplete( H5 );
true
gap> IsAffine( H5 );
false
gap> IsOrbifold( H5 );
true
gap> IsProjective( H5 );
true
gap> TorusInvariantPrimeDivisors(H5);
[ <A prime divisor of a toric variety with coordinates [ 1, 0, 0, 0 ]>,
  <A prime divisor of a toric variety with coordinates [ 0, 1, 0, 0 ]>,
  <A prime divisor of a toric variety with coordinates [ 0, 0, 1, 0 ]>,
  <A prime divisor of a toric variety with coordinates [ 0, 0, 0, 1 ]> ]
gap> P := TorusInvariantPrimeDivisors(H5);
[ <A prime divisor of a toric variety with coordinates [ 1, 0, 0, 0 ]>,
  <A prime divisor of a toric variety with coordinates [ 0, 1, 0, 0 ]>,
  <A prime divisor of a toric variety with coordinates [ 0, 0, 1, 0 ]>,
  <A prime divisor of a toric variety with coordinates [ 0, 0, 0, 1 ]> ]
gap> A := P[ 1 ] - P[ 2 ] + 4*P[ 3 ];
<A divisor of a toric variety with coordinates [ 1, -1, 4, 0 ]>
gap> A;
<A divisor of a toric variety with coordinates [ 1, -1, 4, 0 ]>
```

```
gap> IsAmple(A);
false
gap> CoordinateRingOfTorus(H5,"x");;
Q[x1,x1_,x2,x2_]/( x2*x2_-1, x1*x1_-1 )
gap> D:=CreateDivisor([0,0,0,0],H5);
<A divisor of a toric variety with coordinates 0>
gap> BasisOfGlobalSections(D);
[ |[ 1 ]| ]
gap> D:=Sum(P);
<A divisor of a toric variety with coordinates [ 1, 1, 1, 1 ]>
gap> BasisOfGlobalSections(D);
[ |[ x1_ ]|, |[ x1_*x2 ]|, |[ 1 ]|, |[ x2 ]|,
  |[ x1 ]|, |[ x1*x2 ]|, |[ x1^2*x2 ]|,
  |[ x1^3*x2 ]|, |[ x1^4*x2 ]|, |[ x1^5*x2 ]|,
  |[ x1^6*x2 ]| ]
gap> DivisorOfCharacter([1,2],H5);
<A principal divisor of a toric variety with coordinates [ 9, 2, 1, -2 ]>
gap> BasisOfGlobalSections(last);
[ |[ x1_*x2_^2 ]| ]
```

Chapter 4

Toric subvarieties

4.1 Toric subvarieties: Category and Representations

4.1.1 IsToricSubvariety

▷ `IsToricSubvariety(`*M*`)` (Category)

Returns: `true` or `false`

The GAP category of a toric subvariety. Every toric subvariety is a toric variety, so every method applicable to toric varieties is also applicable to toric subvarieties.

4.2 Toric subvarieties: Properties

4.2.1 IsClosed

▷ `IsClosed(`*vari*`)` (property)

Returns: `true` or `false`

Checks if the subvariety *vari* is a closed subset of its ambient variety.

4.2.2 IsOpen

▷ `IsOpen(`*vari*`)` (property)

Returns: `true` or `false`

Checks if a subvariety is a closed subset.

4.2.3 IsWholeVariety

▷ `IsWholeVariety(`*vari*`)` (property)

Returns: `true` or `false`

Returns true if the subvariety *vari* is the whole variety.

4.3 Toric subvarieties: Attributes

4.3.1 UnderlyingToricVariety

▷ `UnderlyingToricVariety(`*vari*`)` (attribute)

Returns: a variety

Returns the toric variety which is represented by *vari*. This method implements the forgetful functor subvarieties -> varieties.

4.3.2 InclusionMorphism

▷ `InclusionMorphism(`*vari*`)` (attribute)

Returns: a morphism

If the variety *vari* is an open subvariety, this method returns the inclusion morphism in its ambient variety. If not, it will fail.

4.3.3 AmbientToricVariety

▷ `AmbientToricVariety(`*vari*`)` (attribute)

Returns: a variety

Returns the ambient toric variety of the subvariety *vari*

4.4 Toric subvarieties: Methods

4.4.1 ClosureOfTorusOrbitOfCone

▷ `ClosureOfTorusOrbitOfCone(`*vari, cone*`)` (operation)

Returns: a subvariety

The method returns the closure of the orbit of the torus contained in *vari* which corresponds to the cone *cone* as a closed subvariety of *vari*.

4.5 Toric subvarieties: Constructors

4.5.1 ToricSubvariety

▷ `ToricSubvariety(`*vari, ambvari*`)` (operation)

Returns: a subvariety

The method returns the closure of the orbit of the torus contained in *vari* which corresponds to the cone *cone* as a closed subvariety of *vari*.

Chapter 5

Affine toric varieties

5.1 Affine toric varieties: Category and Representations

5.1.1 IsAffineToricVariety

▷ `IsAffineToricVariety(M)` (Category)

Returns: `true` or `false`

The GAP category of an affine toric variety. All affine toric varieties are toric varieties, so everything applicable to toric varieties is applicable to affine toric varieties.

5.2 Affine toric varieties: Properties

Affine toric varieties have no additional properties. Remember that affine toric varieties are toric varieties, so every property of a toric variety is a property of an affine toric variety.

5.3 Affine toric varieties: Attributes

5.3.1 CoordinateRing

▷ `CoordinateRing(vari)` (attribute)

Returns: a ring

Returns the coordinate ring of the affine toric variety *vari*. The computation is mainly done in ToricIdeals package.

5.3.2 ListOfVariablesOfCoordinateRing

▷ `ListOfVariablesOfCoordinateRing(vari)` (attribute)

Returns: a list

Returns a list containing the variables of the CoordinateRing of the variety *vari*.

5.3.3 MorphismFromCoordinateRingToCoordinateRingOfTorus

▷ `MorphismFromCoordinateRingToCoordinateRingOfTorus(vari)` (attribute)

Returns: a morphism

Returns the morphism between the coordinate ring of the variety *vari* and the coordinate ring of its torus. This defines the embedding of the torus in the variety.

5.3.4 ConeOfVariety

▷ `ConeOfVariety(vari)` (attribute)

Returns: a cone

Returns the cone ring of the affine toric variety *vari*.

5.4 Affine toric varieties: Methods

5.4.1 CoordinateRing (for affine Varieties)

▷ `CoordinateRing(vari, indet)` (operation)

Returns: a variety

Computes the coordinate ring of the affine toric variety *vari* with indeterminates *indet*.

5.4.2 Cone

▷ `Cone(vari)` (operation)

Returns: a cone

Returns the cone of the variety *vari*. Another name for ConeOfVariety for compatibility and shortness.

5.5 Affine toric varieties: Constructors

The constructors are the same as for toric varieties. Calling them with a cone will result in an affine variety.

5.6 Affine toric Varieties: Examples

5.6.1 Affine space

Example

```
gap> C:=Cone( [[1,0,0],[0,1,0],[0,0,1]] );
<A cone in |R^3>
gap> C3:=ToricVariety(C);
<An affine normal toric variety of dimension 3>
gap> Dimension(C3);
3
gap> IsOrbifold(C3);
true
gap> IsSmooth(C3);
true
gap> CoordinateRingOfTorus(C3,"x");
Q[x1,x1_,x2,x2_,x3,x3_]/( x3*x3_-1, x2*x2_-1, x1*x1_-1 )
gap> CoordinateRing(C3,"x");
Q[x_1,x_2,x_3]
gap> MorphismFromCoordinateRingToCoordinateRingOfTorus(C3);
```

```
<A monomorphism of rings>
gap> C3;
<An affine normal smooth toric variety of dimension 3>
gap> StructureDescription(C3);
"|A^3"
```

Chapter 6

Projective toric varieties

6.1 Projective toric varieties: Category and Representations

6.1.1 IsProjectiveToricVariety

▷ `IsProjectiveToricVariety(`*M*`)` (Category)

Returns: `true` or `false`

The GAP category of a projective toric variety.

6.2 Projective toric varieties: Properties

Projective toric varieties have no additional properties. Remember that projective toric varieties are toric varieties, so every property of a toric variety is a property of an projective toric variety.

6.3 Projective toric varieties: Attributes

6.3.1 AffineCone

▷ `AffineCone(`*vari*`)` (attribute)

Returns: a variety

Returns the affine cone of the projective toric variety *vari*.

6.3.2 PolytopeOfVariety

▷ `PolytopeOfVariety(`*vari*`)` (attribute)

Returns: a polytope

Returns the polytope corresponding to the projective toric variety *vari*, if it exists.

6.3.3 ProjectiveEmbedding

▷ `ProjectiveEmbedding(`*vari*`)` (attribute)

Returns: a list

Returns characters for a closed embedding in an projective space for the projective toric variety *vari*.

6.4 Projective toric varieties: Methods

6.4.1 Polytope

▷ `Polytope(vari)` (operation)

Returns: a polytope

Returns the polytope of the variety *vari*. Another name for PolytopeOfVariety for compatibility and shortness.

6.5 Projective toric varieties: Constructors

The constructors are the same as for toric varieties. Calling them with a polytope will result in an projective variety.

6.6 Projective toric varieties: Examples

6.6.1 PxP1 created by a polytope

Example

```
gap> P1P1 := Polytope( [[1,1],[1,-1],[-1,-1],[-1,1]] );
<A polytope in |R^2>
gap> P1P1 := ToricVariety( P1P1 );
<A projective toric variety of dimension 2>
gap> IsProjective( P1P1 );
true
gap> IsComplete( P1P1 );
true
gap> CoordinateRingOfTorus( P1P1, "x" );
Q[x1,x1_,x2,x2_]/( x2*x2_-1, x1*x1_-1 )
gap> IsVeryAmple( Polytope( P1P1 ) );
true
gap> ProjectiveEmbedding( P1P1 );
[ |[ x1_*x2_ ]|, |[ x1_ ]|, |[ x1_*x2 ]|, |[ x2_ ]|,
|[ 1 ]|, |[ x2 ]|, |[ x1*x2_ ]|, |[ x1 ]|, |[ x1*x2 ]| ]
gap> Length( last );
9
```

Chapter 7

Toric morphisms

7.1 Toric morphisms: Category and Representations

7.1.1 IsToricMorphism

▷ `IsToricMorphism(M)` (Category)

Returns: `true` or `false`

The GAP category of toric morphisms. A toric morphism is defined by a grid homomorphism, which is compatible with the fan structure of the two varieties.

7.2 Toric morphisms: Properties

7.2.1 IsMorphism

▷ `IsMorphism(morph)` (property)

Returns: `true` or `false`

Checks if the grid morphism *morph* respects the fan structure.

7.2.2 IsProper

▷ `IsProper(morph)` (property)

Returns: `true` or `false`

Checks if the defined morphism *morph* is proper.

7.3 Toric morphisms: Attributes

7.3.1 SourceObject

▷ `SourceObject(morph)` (attribute)

Returns: a variety

Returns the source object of the morphism *morph*. This attribute is a must have.

7.3.2 UnderlyingGridMorphism

▷ `UnderlyingGridMorphism(morph)` (attribute)

Returns: a map

Returns the grid map which defines *morph*.

7.3.3 ToricImageObject

▷ `ToricImageObject(morph)` (attribute)

Returns: a variety

Returns the variety which is created by the fan which is the image of the fan of the source of *morph*. This is not an image in the usual sense, but a toric image.

7.3.4 RangeObject

▷ `RangeObject(morph)` (attribute)

Returns: a variety

Returns the range of the morphism *morph*. If no range is given (yes, this is possible), the method returns the image.

7.3.5 MorphismOnWeilDivisorGroup

▷ `MorphismOnWeilDivisorGroup(morph)` (attribute)

Returns: a morphism

Returns the associated morphism between the divisor group of the range of *morph* and the divisor group of the source.

7.3.6 ClassGroup (for toric morphisms)

▷ `ClassGroup(morph)` (attribute)

Returns: a morphism

Returns the associated morphism between the class groups of source and range of the morphism *morph*

7.3.7 MorphismOnCartierDivisorGroup

▷ `MorphismOnCartierDivisorGroup(morph)` (attribute)

Returns: a morphism

Returns the associated morphism between the Cartier divisor groups of source and range of the morphism *morph*

7.3.8 PicardGroup (for toric morphisms)

▷ `PicardGroup(morph)` (attribute)

Returns: a morphism

Returns the associated morphism between the class groups of source and range of the morphism *morph*

7.4 Toric morphisms: Methods

7.4.1 UnderlyingListList

▷ `UnderlyingListList(morph)` (attribute)

Returns: a list

Returns a list of list which represents the grid homomorphism.

7.5 Toric morphisms: Constructors

7.5.1 ToricMorphism (for a source and a matrix)

▷ `ToricMorphism(vari, lis)` (operation)

Returns: a morphism

Returns the toric morphism with source *vari* which is represented by the matrix *lis*. The range is set to the image.

7.5.2 ToricMorphism (for a source, matrix and target)

▷ `ToricMorphism(vari, lis, vari2)` (operation)

Returns: a morphism

Returns the toric morphism with source *vari* and range *vari2* which is represented by the matrix *lis*.

7.6 Toric morphisms: Examples

7.6.1 Morphism between toric varieties and their class groups

Example

```
gap> P1 := Polytope([[0],[1]]);
<A polytope in |R^1>
gap> P2 := Polytope([[0,0],[0,1],[1,0]]);
<A polytope in |R^2>
gap> P1 := ToricVariety( P1 );
<A projective toric variety of dimension 1>
gap> P2 := ToricVariety( P2 );
<A projective toric variety of dimension 2>
gap> P1P2 := P1*P2;
<A projective toric variety of dimension 3
 which is a product of 2 toric varieties>
gap> ClassGroup( P1 );
<A non-torsion left module presented by 1 relation for 2 generators>
gap> Display(ByASmallerPresentation(last));
Z^(1 x 1)
gap> ClassGroup( P2 );
<A non-torsion left module presented by 2 relations for 3 generators>
gap> Display(ByASmallerPresentation(last));
Z^(1 x 1)
gap> ClassGroup( P1P2 );
<A free left module of rank 2 on free generators>
gap> Display( last );
```

```
Z^(1 x 2)
gap> PicardGroup( P1P2 );
<A free left module of rank 2 on free generators>
gap> P1P2;
<A projective smooth toric variety of dimension 3
 which is a product of 2 toric varieties>
gap> P2P1:=P2*P1;
<A projective toric variety of dimension 3
 which is a product of 2 toric varieties>
gap> M := [[0,0,1],[1,0,0],[0,1,0]];
[ [ 0, 0, 1 ], [ 1, 0, 0 ], [ 0, 1, 0 ] ]
gap> M := ToricMorphism(P1P2,M,P2P1);
<A "homomorphism" of right objects>
gap> IsMorphism(M);
true
gap> ClassGroup(M);
<A homomorphism of left modules>
gap> Display(last);
[ [  0,  1 ],
  [  1,  0 ] ]

the map is currently represented by the above 2 x 2 matrix
gap> ByASmallerPresentation(ClassGroup(M));
<A non-zero homomorphism of left modules>
gap> Display(last);
[ [  0,  1 ],
  [  1,  0 ] ]

the map is currently represented by the above 2 x 2 matrix
```

Chapter 8

Toric divisors

8.1 Toric divisors: Category and Representations

8.1.1 IsToricDivisor

▷ `IsToricDivisor(M)` (Category)

Returns: `true` or `false`

The GAP category of torus invariant Weil divisors.

8.2 Toric divisors: Properties

8.2.1 IsCartier

▷ `IsCartier(divi)` (property)

Returns: `true` or `false`

Checks if the torus invariant Weil divisor *divi* is Cartier i.e. if it is locally principal.

8.2.2 IsPrincipal

▷ `IsPrincipal(divi)` (property)

Returns: `true` or `false`

Checks if the torus invariant Weil divisor *divi* is principal which in the toric invariant case means that it is the divisor of a character.

8.2.3 IsPrimedivisor

▷ `IsPrimedivisor(divi)` (property)

Returns: `true` or `false`

Checks if the Weil divisor *divi* represents a prime divisor, i.e. if it is a standard generator of the divisor group.

8.2.4 IsBasepointFree

▷ `IsBasepointFree(divi)` (property)

Returns: `true` or `false`

Checks if the divisor *divi* is basepoint free. What else?

8.2.5 IsAmple

▷ `IsAmple(divi)` (property)

Returns: `true` or `false`

Checks if the divisor *divi* is ample, i.e. if it is colored red, yellow and green.

8.2.6 IsVeryAmple

▷ `IsVeryAmple(divi)` (property)

Returns: `true` or `false`

Checks if the divisor *divi* is very ample.

8.3 Toric divisors: Attributes

8.3.1 CartierData

▷ `CartierData(divi)` (attribute)

Returns: a list

Returns the Cartier data of the divisor *divi*, if it is Cartier, and fails otherwise.

8.3.2 CharacterOfPrincipalDivisor

▷ `CharacterOfPrincipalDivisor(divi)` (attribute)

Returns: an element

Returns the character corresponding to principal divisor *divi*.

8.3.3 ToricVarietyOfDivisor

▷ `ToricVarietyOfDivisor(divi)` (attribute)

Returns: a variety

Returns the closure of the torus orbit corresponding to the prime divisor *divi*. Not implemented for other divisors. Maybe we should add the support here. Is this even a toric variety? Exercise left to the reader.

8.3.4 ClassOfDivisor

▷ `ClassOfDivisor(divi)` (attribute)

Returns: an element

Returns the class group element corresponding to the divisor *divi*.

8.3.5 PolytopeOfDivisor

▷ `PolytopeOfDivisor(divi)` (attribute)

Returns: a polytope

Returns the polytope corresponding to the divisor *divi*.

8.3.6 BasisOfGlobalSections

▷ `BasisOfGlobalSections(divi)` (attribute)

Returns: a list

Returns a basis of the global section module of the quasi-coherent sheaf of the divisor `divi`.

8.3.7 IntegerForWhichIsSureVeryAmple

▷ `IntegerForWhichIsSureVeryAmple(divi)` (attribute)

Returns: an integer

Returns an integer which, to be multiplied with the ample divisor `divi`, someone gets a very ample divisor.

8.3.8 AmbientToricVariety (for toric divisors)

▷ `AmbientToricVariety(divi)` (attribute)

Returns: a variety

Returns the containing variety of the prime divisors of the divisor `divi`.

8.3.9 UnderlyingGroupElement

▷ `UnderlyingGroupElement(divi)` (attribute)

Returns: an element

Returns an element which represents the divisor `divi` in the Weil group.

8.3.10 UnderlyingToricVariety (for prime divisors)

▷ `UnderlyingToricVariety(divi)` (attribute)

Returns: a variety

Returns the closure of the torus orbit corresponding to the prime divisor `divi`. Not implemented for other divisors. Maybe we should add the support here. Is this even a toric variety? Exercise left to the reader.

8.3.11 DegreeOfDivisor

▷ `DegreeOfDivisor(divi)` (attribute)

Returns: an integer

Returns the degree of the divisor `divi`.

8.3.12 MonomsOfCoxRingOfDegree

▷ `MonomsOfCoxRingOfDegree(divi)` (attribute)

Returns: a list

Returns the variety corresponding to the polytope of the divisor `divi`.

8.3.13 CoxRingOfTargetOfDivisorMorphism

▷ `CoxRingOfTargetOfDivisorMorphism(divi)` (attribute)

Returns: a ring

A basepoint free divisor *divi* defines a map from its ambient variety in a projective space. This method returns the cox ring of such a projective space.

8.3.14 RingMorphismOfDivisor

▷ `RingMorphismOfDivisor(divi)` (attribute)

Returns: a ring

A basepoint free divisor *divi* defines a map from its ambient variety in a projective space. This method returns the morphism between the cox ring of this projective space to the cox ring of the ambient variety of *divi*.

8.4 Toric divisors: Methods

8.4.1 VeryAmpleMultiple

▷ `VeryAmpleMultiple(divi)` (operation)

Returns: a divisor

Returns a very ample multiple of the ample divisor *divi*. Will fail if divisor is not ample.

8.4.2 CharactersForClosedEmbedding

▷ `CharactersForClosedEmbedding(divi)` (operation)

Returns: a list

Returns characters for closed embedding defined via the ample divisor *divi*. Fails if divisor is not ample.

8.4.3 MonomsOfCoxRingOfDegree (for an homalg element)

▷ `MonomsOfCoxRingOfDegree(vari, elem)` (operation)

Returns: a list

Returns the monoms of the Cox ring of the variety *vari* with degree to the class group element *elem*. The variable *elem* can also be a list.

8.4.4 DivisorOfGivenClass

▷ `DivisorOfGivenClass(vari, elem)` (operation)

Returns: a list

Computes a divisor of the variety *divi* which is member of the divisor class presented by *elem*. The variable *elem* can be a homalg element or a list presenting an element.

8.4.5 AddDivisorToItsAmbientVariety

▷ `AddDivisorToItsAmbientVariety(divi)` (operation)

Returns:

Adds the divisor *divi* to the Weil divisor list of its ambient variety.

8.4.6 Polytope (for toric divisors)

▷ `Polytope(divi)` (operation)

Returns: a polytope

Returns the polytope of the divisor *divi*. Another name for PolytopeOfDivisor for compatibility and shortness.

8.4.7 +

▷ `+(divi1, divi2)` (operation)

Returns: a divisor

Returns the sum of the divisors *divi1* and *divi2*.

8.4.8 -

▷ `-(divi1, divi2)` (operation)

Returns: a divisor

Returns the divisor *divi1* minus *divi2*.

8.4.9 * (for toric divisors)

▷ `*(k, divi)` (operation)

Returns: a divisor

Returns *k* times the divisor *divi*.

8.5 Toric divisors: Constructors

8.5.1 DivisorOfCharacter

▷ `DivisorOfCharacter(elem, vari)` (operation)

Returns: a divisor

Returns the divisor of the toric variety *vari* which corresponds to the character *elem*.

8.5.2 DivisorOfCharacter (for a list of integers)

▷ `DivisorOfCharacter(lis, vari)` (operation)

Returns: a divisor

Returns the divisor of the toric variety *vari* which corresponds to the character which is created by the list *lis*.

8.5.3 CreateDivisor (for a homalg element)

▷ `CreateDivisor(elem, vari)` (operation)

Returns: a divisor

Returns the divisor of the toric variety *vari* which corresponds to the Weil group element *elem*.

8.5.4 CreateDivisor (for a list of integers)

▷ CreateDivisor(*lis, vari*) (operation)

Returns: a divisor

Returns the divisor of the toric variety *vari* which corresponds to the Weil group element which is created by the list *lis*.

8.6 Toric divisors: Examples

8.6.1 Divisors on a toric variety

Example
```
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> P := TorusInvariantPrimeDivisors( H7 );
[ <A prime divisor of a toric variety with coordinates [ 1, 0, 0, 0 ]>,
  <A prime divisor of a toric variety with coordinates [ 0, 1, 0, 0 ]>,
  <A prime divisor of a toric variety with coordinates [ 0, 0, 1, 0 ]>,
  <A prime divisor of a toric variety with coordinates [ 0, 0, 0, 1 ]> ]
gap> D := P[3]+P[4];
<A divisor of a toric variety with coordinates [ 0, 0, 1, 1 ]>
gap> IsBasepointFree(D);
true
gap> IsAmple(D);
true
gap> CoordinateRingOfTorus(H7,"x");
Q[x1,x1_,x2,x2_]/( x2*x2_-1, x1*x1_-1 )
gap> Polytope(D);
<A polytope in |R^2>
gap> CharactersForClosedEmbedding(D);
[ |[ 1 ]|, |[ x2 ]|, |[ x1 ]|, |[ x1*x2 ]|, |[ x1^2*x2 ]|,
  |[ x1^3*x2 ]|, |[ x1^4*x2 ]|, |[ x1^5*x2 ]|,
  |[ x1^6*x2 ]|, |[ x1^7*x2 ]|, |[ x1^8*x2 ]| ]
gap> CoxRingOfTargetOfDivisorMorphism(D);
Q[x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_8,x_9,x_10,x_11]
(weights: [ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 ])
gap> RingMorphismOfDivisor(D);
<A "homomorphism" of rings>
gap> Display(last);
Q[x_1,x_2,x_3,x_4]
(weights: [ [ 0, 0, 1, -7 ], [ 0, 0, 0, 1 ], [ 0, 0, 1, 0 ], [ 0, 0, 0, 1 ] ])
  ^
  |
[ x_3*x_4, x_1*x_4^8, x_2*x_3, x_1*x_2*x_4^7, x_1*x_2^2*x_4^6,
  x_1*x_2^3*x_4^5, x_1*x_2^4*x_4^4, x_1*x_2^5*x_4^3,
  x_1*x_2^6*x_4^2, x_1*x_2^7*x_4, x_1*x_2^8 ]
  |
  |
Q[x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_8,x_9,x_10,x_11]
(weights: [ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 ])
gap> ByASmallerPresentation(ClassGroup(H7));
```

```
<A free left module of rank 2 on free generators>
gap> Display(RingMorphismOfDivisor(D));
Q[x_1,x_2,x_3,x_4]
(weights: [ [ 1, -7 ], [ 0, 1 ], [ 1, 0 ], [ 0, 1 ] ])
  ^
  |
[ x_3*x_4, x_1*x_4^8, x_2*x_3, x_1*x_2*x_4^7, x_1*x_2^2*x_4^6,
  x_1*x_2^3*x_4^5, x_1*x_2^4*x_4^4, x_1*x_2^5*x_4^3,
  x_1*x_2^6*x_4^2, x_1*x_2^7*x_4, x_1*x_2^8 ]
  |
  |
Q[x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_8,x_9,x_10,x_11]
(weights: [ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 ])
gap> MonomsOfCoxRingOfDegree(D);
[ x_3*x_4, x_1*x_4^8, x_2*x_3, x_1*x_2*x_4^7, x_1*x_2^2*x_4^6,
  x_1*x_2^3*x_4^5, x_1*x_2^4*x_4^4, x_1*x_2^5*x_4^3,
  x_1*x_2^6*x_4^2, x_1*x_2^7*x_4, x_1*x_2^8 ]
gap> D2:=D-2*P[2];
<A divisor of a toric variety with coordinates [ 0, -2, 1, 1 ]>
gap> IsBasepointFree(D2);
false
gap> IsAmple(D2);
false
```

Index