ToricVarieties

A package to handle toric varieties

2021.01.12

12 January 2021

Sebastian Gutsche

Martin Bies

Sebastian Gutsche
Email: gutsche@mathematik.uni-siegen.de
Homepage: https://sebasguts.github.io
Address: Department Mathematik
Universität Siegen
Walter-Flex-Straße 3
57072 Siegen
Germany

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 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 the following command, before its functions become avaiable: *LoadPackage("ToricVarieties");*
- Please, send me an e-mail if you have any questions, remarks, suggestions, etc. concerning this package. Also, I would be please 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 Varieties: Examples

3.1.1 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_]/( x1*x1_-1, x2*x2_-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> divi := DivisorOfCharacter( [ 1,2 ],H5 );
<A principal divisor of a toric variety with coordinates ( 9, -2, 2, 1 )>
gap> BasisOfGlobalSections( divi );
[ |[ x1_*x2_^2 ]| ]
```

3.1.2 A smooth, complete toric variety which is not projective

Example

```
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>
```

3.1.3 Convenient construction of toric varieties

Example

```
gap> rays := [ [1,0],[-1,0],[0,1],[0,-1] ];
[ [ 1, 0 ], [ -1, 0 ], [ 0, 1 ], [ 0, -1 ] ]
gap> cones := [ [1,3],[1,4],[2,3],[2,4] ];
[ [1,3],[1,4],[2,3],[2,4] ]
gap> weights := [ [1,0],[1,0],[0,1],[0,1] ];
[ [1,0],[1,0],[0,1],[0,1] ]
gap> weights2 := [ [1,1],[1,1],[1,2],[1,2] ];
[ [1,1],[1,1],[1,2],[1,2] ]
gap> tor1 := ToricVariety( rays, cones, weights, "x1,x2,y1,y2" );
<A toric variety of dimension 2>
gap> CoxRing( tor1 );
Q[x2,y2,y1,x1]
(weights: [ ( 1, 0 ), ( 0, 1 ), ( 0, 1 ), ( 1, 0 ) ])
gap> tor2:= ToricVariety( rays, cones, weights, "q" );
<A toric variety of dimension 2>
```

```
gap> CoxRing( tor2 );
Q[q_2,q_4,q_3,q_1]
(weights: [ ( 1, 0 ), ( 0, 1 ), ( 0, 1 ), ( 1, 0 ) ])
gap> tor3:= ToricVariety( rays, cones, weights );
<A toric variety of dimension 2>
gap> CoxRing( tor3 );
Q[x_2,x_4,x_3,x_1]
(weights: [ ( 1, 0 ), ( 0, 1 ), ( 0, 1 ), ( 1, 0 ) ])
gap> tor4:= ToricVariety( rays, cones, weights2, "x1,x2,z1,z2" );
<A toric variety of dimension 2>
gap> CoxRing( tor4 );
Q[x2,z2,z1,x1]
(weights: [ ( 1, 1 ), ( 1, 2 ), ( 1, 2 ), ( 1, 1 ) ])
```

3.1.4 Toric varieties from gradings

The following example shows how to create the projective space $\mathbb{P}^2$ from the grading of its Cox ring. Note that this functionality requires the package TopcomInterface.

Example
```
gap> g := [[1,1,1]];
[ [ 1,1,1 ] ]
gap> v1 := ToricVarietyFromGrading( g );
<A toric variety of dimension 2>
gap> CoxRing( v1 );
Q[x_1,x_2,x_3]
(weights: [ 1, 1, 1 ])
```

The following example shows how to create the resolved conifold(s) from the grading of its Cox ring.

Example
```
gap> g2 := [[1,1,-1,-1]];
[ [ 1,1,-1,-1 ] ]
gap> v2 := ToricVarietiesFromGrading( g2 );
[ <A toric variety of dimension 3>, <A toric variety of dimension 3> ]
gap> CoxRing( v2[ 1 ] );
Q[x_1,x_2,x_3,x_4]
(weights: [ 1, -1, -1, 1 ])
gap> Display( SRIdeal( v2[ 1 ] ) );
x_2*x_3

A (left) ideal generated by the entry of the above matrix

(graded, degree of generator: -2)
gap> Display( SRIdeal( v2[ 2 ] ) );
x_1*x_4

A (left) ideal generated by the entry of the above matrix

(graded, degree of generator: 2)
```

3.1.5 Blowups of toric varieties by star subdivisions of fans

The following code exemplifies blowups of the 3-dimensional affine space.

Example

```
gap> rays := [ [1,0,0], [0,1,0], [0,0,1] ];
[ [1,0,0], [0,1,0], [0,0,1] ]
gap> max_cones := [ [1,2,3] ];
[ [1,2,3] ]
gap> fan := Fan( rays, max_cones );
<A fan in |R^3>
gap> C3 := ToricVariety( rays, max_cones, [[0],[0],[0]], "x1,x2,x3" );
<A toric variety of dimension 3>
gap> B1C3 := BlowupOfToricVariety( C3, "x1,x2,x3", "u0" );
<A toric variety of dimension 3>
gap> [ IsComplete( B1C3 ), IsOrbifold( B1C3 ), IsSmooth( B1C3 ) ];
[ false, true, true ]
gap> B2C3 := BlowupOfToricVariety( B1C3, "x1,u0", "u1" );
<A toric variety of dimension 3>
gap> Rank( ClassGroup( B2C3 ) );
3
gap> B3C3 := BlowupOfToricVariety( B2C3, "x1,u1", "u2" );
<A toric variety of dimension 3>
gap> CoxRing( B3C3 );
Q[x3,x2,x1,u0,u1,u2]
(weights: [ ( 0, 1, 0, 0 ), ( 0, 1, 0, 0 ), ( 0, 1, 1, 1 ),
( 0, -1, 1, 0 ), ( 0, 0, -1, 1 ), ( 0, 0, 0, -1 ) ])
```

Likewise, we can also perform blowups of the 3-dimensional projective space.

Example

```
gap> rays := [ [1,0,0], [0,1,0], [0,0,1], [-1,-1,-1] ];
[ [1,0,0], [0,1,0], [0,0,1], [-1,-1,-1] ]
gap> max_cones := [ [1,2,3], [1,2,4], [1,3,4], [2,3,4] ];
[ [1,2,3], [1,2,4], [1,3,4], [2,3,4] ]
gap> fan := Fan( rays, max_cones );
<A fan in |R^3>
gap> P3 := ToricVariety( rays, max_cones, [[1],[1],[1],[1]], "x1,x2,x3,x4" );
<A toric variety of dimension 3>
gap> B1P3 := BlowupOfToricVariety( P3, "x1,x2,x3", "u0" );
<A toric variety of dimension 3>
gap> [ IsComplete( B1P3 ), IsOrbifold( B1P3 ), IsSmooth( B1P3 ) ];
[ true, true, true ]
gap> B2P3 := BlowupOfToricVariety( B1P3, "x1,u0", "u1" );
<A toric variety of dimension 3>
gap> Rank( ClassGroup( B2C3 ) );
3
gap> B3P3 := BlowupOfToricVariety( B2P3, "x1,u1", "u2" );
<A toric variety of dimension 3>
gap> CoxRing( B3P3 );
Q[x4,x3,x2,x1,u0,u1,u2]
(weights: [ ( 1, 0, 0, 0 ), ( 1, 1, 0, 0 ), ( 1, 1, 0, 0 ),
( 1, 1, 1, 1 ), ( 0, -1, 1, 0 ), ( 0, 0, -1, 1 ), ( 0, 0, 0, -1 ) ])
```

Also, we can perform blowups of a generalized Hirzebruch 3-fold.

Example

```
gap> vars := "u,s,v,t,r";
"u,s,v,t,r"
```

```
gap> rays := [ [0,0,-1],[1,0,0],[0,1,0],[-1,-1,-17],[0,0,1] ];
[ [0,0,-1],[1,0,0],[0,1,0],[-1,-1,-17],[0,0,1] ]
gap> cones := [ [1,2,3], [1,2,4], [1,3,4], [2,3,5], [2,4,5], [3,4,5] ];
[ [1,2,3], [1,2,4], [1,3,4], [2,3,5], [2,4,5], [3,4,5] ]
gap> weights := [ [1,-17], [0,1], [0,1], [0,1], [1,0] ];
[ [1,-17], [0,1], [0,1], [0,1], [1,0] ]
gap> H3fold := ToricVariety( rays, cones, weights, vars );
<A toric variety of dimension 3>
gap> B1H3fold := BlowupOfToricVariety( H3fold, "u,s", "u1" );
<A toric variety of dimension 3>
gap> CoxRing( B1H3fold );
Q[t,u,r,v,u1,s]
(weights: [ ( 0, 1, 0 ), ( 1, -17, 1 ), ( 1, 0, 0 ),
( 0, 1, 0 ), ( 0, 0, -1 ), ( 0, 1, 1 ) ])
```

This example easily extends to an entire sequence of blowups.

Example

```
gap> vars := "u,s,v,t,r,x,y,w";
"u,s,v,t,r,x,y,w"
gap> rays := [ [0,0,-1,-2,-3], [1,0,0,-2,-3], [0,1,0,-2,-3], [-1,-1,-17,-2,-3],
>            [0,0,1,-2,-3], [0, 0, 0, 1, 0],
> [0, 0, 0, 0, 1], [0, 0, 0, -2, -3] ];
[ [0,0,-1,-2,-3], [1,0,0,-2,-3], [0,1,0,-2,-3], [-1,-1,-17,-2,-3],
[0,0,1,-2,-3], [0, 0, 0, 1, 0], [0, 0, 0, 0, 1], [0, 0, 0, -2, -3] ]
gap> cones := [ [1,2,3,6,7], [1,2,3,6,8], [1,2,3,7,8], [1,2,4,6,7], [1,2,4,6,8],
>             [1,2,4,7,8], [1,3,4,6,7], [1,3,4,6,8], [1,3,4,7,8], [2,3,5,6,7],
>             [2,3,5,6,8], [2,3,5,7,8], [2,4,5,6,7], [2,4,5,6,8], [2,4,5,7,8],
>             [3,4,5,6,7], [3,4,5,6,8], [3,4,5,7,8] ];
[ [ 1, 2, 3, 6, 7 ], [ 1, 2, 3, 6, 8 ], [ 1, 2, 3, 7, 8 ],
  [ 1, 2, 4, 6, 7 ], [ 1, 2, 4, 6, 8 ], [ 1, 2, 4, 7, 8 ],
  [ 1, 3, 4, 6, 7 ], [ 1, 3, 4, 6, 8 ], [ 1, 3, 4, 7, 8 ],
  [ 2, 3, 5, 6, 7 ], [ 2, 3, 5, 6, 8 ], [ 2, 3, 5, 7, 8 ],
  [ 2, 4, 5, 6, 7 ], [ 2, 4, 5, 6, 8 ], [ 2, 4, 5, 7, 8 ],
  [ 3, 4, 5, 6, 7 ], [ 3, 4, 5, 6, 8 ], [ 3, 4, 5, 7, 8 ] ]
gap> w := [ [1,-17,0], [0,1,0], [0,1,0], [0,1,0], [1,0,0], [0,0,2], [0,0,3],
>         [-2,14,1] ];
[ [1,-17,0], [0,1,0], [0,1,0], [0,1,0], [1,0,0], [0,0,2], [0,0,3], [-2,14,1] ]
gap> base := ToricVariety( rays, cones, w, vars );
<A toric variety of dimension 5>
gap> b1 := BlowupOfToricVariety( base, "x,y,u", "u1" );
<A toric variety of dimension 5>
gap> b2 := BlowupOfToricVariety( b1, "x,y,u1", "u2" );
<A toric variety of dimension 5>
gap> b3 := BlowupOfToricVariety( b2, "y,u1", "u3" );
<A toric variety of dimension 5>
gap> b4 := BlowupOfToricVariety( b3, "y,u2", "u4" );
<A toric variety of dimension 5>
gap> b5 := BlowupOfToricVariety( b4, "u2,u3", "u5" );
<A toric variety of dimension 5>
gap> b6 := BlowupOfToricVariety( b5, "u1,u3", "u6" );
<A toric variety of dimension 5>
gap> b7 := BlowupOfToricVariety( b6, "u2,u4", "u7" );
<A toric variety of dimension 5>
```

```
gap> b8 := BlowupOfToricVariety( b7, "u3,u4", "u8" );
<A toric variety of dimension 5>
gap> b9 := BlowupOfToricVariety( b8, "u4,u5", "u9" );
<A toric variety of dimension 5>
gap> b10 := BlowupOfToricVariety( b9, "u5,u8", "u10" );
<A toric variety of dimension 5>
gap> b11 := BlowupOfToricVariety( b10, "u4,u8", "u11" );
<A toric variety of dimension 5>
gap> b12 := BlowupOfToricVariety( b11, "u4,u9", "u12" );
<A toric variety of dimension 5>
gap> b13 := BlowupOfToricVariety( b12, "u8,u9", "u13" );
<A toric variety of dimension 5>
gap> b14 := BlowupOfToricVariety( b13, "u9,u11", "u14" );
<A toric variety of dimension 5>
gap> b15 := BlowupOfToricVariety( b14, "u4,v", "d" );
<A toric variety of dimension 5>
gap> final_space := BlowupOfToricVariety( b15, "u3,u5", "u15" );
<A toric variety of dimension 5>
```

This sequence of blowups can also be performed with a single command.

Example

```
gap> final_space2 := SequenceOfBlowupsOfToricVariety( base,
>                      [ ["x,y,u","u1"],
>                      ["x,y,u1","u2"],
>                      ["y,u1","u3"],
>                      ["y,u2","u4"],
>                      ["u2,u3","u5"],
>                      ["u1,u3","u6"],
>                      ["u2,u4","u7"],
>                      ["u3,u4","u8"],
>                      ["u4,u5","u9"],
>                      ["u5,u8","u10"],
>                      ["u4,u8","u11"],
>                      ["u4,u9","u12"],
>                      ["u8,u9","u13"],
>                      ["u9,u11","u14"],
>                      ["u4,v","d"],
>                      ["u3,u5","u15"] ] );
<A toric variety of dimension 5>
gap> [ IsComplete( final_space2 ), IsOrbifold( final_space2 ),
>   IsSmooth( final_space2 ) ];
[ true, true, false ]
```

3.2 Toric variety: Category and Representations

3.2.1 IsToricVariety (for IsObject)

▷ `IsToricVariety(M)` (filter)

Returns: true or false

Checks if an object is a toric variety.

3.2.2 IsCategoryOfToricVarieties (for IsHomalgCategory)

▷ `IsCategoryOfToricVarieties(`*object*`)` (filter)

Returns: `true` or `false`

The *GAP* category of toric varieties.

3.2.3 twitter (for IsToricVariety)

▷ `twitter(`*vari*`)` (attribute)

Returns: a ring

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

3.3 Properties

3.3.1 IsNormalVariety (for IsToricVariety)

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

Returns: true or false

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

3.3.2 IsAffine (for IsToricVariety)

▷ `IsAffine(`*vari*`)` (property)

Returns: true or false

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

3.3.3 IsProjective (for IsToricVariety)

▷ `IsProjective(`*vari*`)` (property)

Returns: true or false

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

3.3.4 IsSmooth (for IsToricVariety)

▷ `IsSmooth(`*vari*`)` (property)

Returns: true or false

Checks if the toric variety *vari* is smooth.

3.3.5 IsComplete (for IsToricVariety)

▷ `IsComplete(`*vari*`)` (property)

Returns: true or false

Checks if the toric variety *vari* is complete.

3.3.6 HasTorusfactor (for IsToricVariety)

▷ `HasTorusfactor(`*vari*`)` (property)

Returns: true or false

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

3.3.7 HasNoTorusfactor (for IsToricVariety)

▷ `HasNoTorusfactor(vari)` (property)

Returns: true or false

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

3.3.8 IsOrbifold (for IsToricVariety)

▷ `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.

3.3.9 IsSimplicial (for IsToricVariety)

▷ `IsSimplicial(vari)` (property)

Returns: true or false

Checks if the toric variety *vari* is simplicial. This is a convenience method equivalent to IsOrbifold.

3.4 Attributes

3.4.1 AffineOpenCovering (for IsToricVariety)

▷ `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.

3.4.2 CoxRing (for IsToricVariety)

▷ `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.

3.4.3 ListOfVariablesOfCoxRing (for IsToricVariety)

▷ `ListOfVariablesOfCoxRing(vari)` (attribute)

Returns: a list

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

3.4.4 ClassGroup (for IsToricVariety)

▷ `ClassGroup(vari)` (attribute)

Returns: a module

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

3.4.5 TorusInvariantDivisorGroup (for IsToricVariety)

▷ `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.

3.4.6 MapFromCharacterToPrincipalDivisor (for IsToricVariety)

▷ `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 be viewed as a help method to compute divisor classes.

3.4.7 MapFromWeilDivisorsToClassGroup (for IsToricVariety)

▷ `MapFromWeilDivisorsToClassGroup(vari)` (attribute)

Returns: a morphism

Returns a map which maps a Weil divisor into the class group.

3.4.8 Dimension (for IsToricVariety)

▷ `Dimension(vari)` (attribute)

Returns: an integer

Returns the dimension of the variety *vari*.

3.4.9 DimensionOfTorusfactor (for IsToricVariety)

▷ `DimensionOfTorusfactor(vari)` (attribute)

Returns: an integer

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

3.4.10 CoordinateRingOfTorus (for IsToricVariety)

▷ `CoordinateRingOfTorus(vari)` (attribute)

Returns: a ring

Returns the coordinate ring of the torus of the variety *vari*. This is by default done with the variables *x1* to *xn* where *n* is the dimension of the variety. To use a different set of variables, a convenience method is provided and described in the *methods* section.

3.4.11 ListOfVariablesOfCoordinateRingOfTorus (for IsToricVariety)

▷ `ListOfVariablesOfCoordinateRingOfTorus(vari)` (attribute)

Returns: a list

Returns the list of variables in the coordinate ring of the torus of the variety *vari*.

3.4.12 IsProductOf (for IsToricVariety)

▷ `IsProductOf(vari)` (attribute)

Returns: a list

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

3.4.13 CharacterLattice (for IsToricVariety)

▷ `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.

3.4.14 TorusInvariantPrimeDivisors (for IsToricVariety)

▷ `TorusInvariantPrimeDivisors(vari)` (attribute)

Returns: a list

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

3.4.15 IrrelevantIdeal (for IsToricVariety)

▷ `IrrelevantIdeal(vari)` (attribute)

Returns: an ideal

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

3.4.16 SRIdeal (for IsToricVariety)

▷ `SRIdeal(vari)` (attribute)

Returns: an ideal

Returns the Stanley-Reißner ideal of the Cox ring of the variety *vari*.

3.4.17 MorphismFromCoxVariety (for IsToricVariety)

▷ `MorphismFromCoxVariety(vari)` (attribute)

Returns: a morphism

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

3.4.18 CoxVariety (for IsToricVariety)

▷ `CoxVariety(vari)` (attribute)

Returns: a variety

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

3.4.19 FanOfVariety (for IsToricVariety)

▷ `FanOfVariety(vari)` (attribute)

Returns: a fan

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

3.4.20 CartierTorusInvariantDivisorGroup (for IsToricVariety)

▷ `CartierTorusInvariantDivisorGroup(vari)` (attribute)

Returns: a module

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

3.4.21 PicardGroup (for IsToricVariety)

▷ `PicardGroup(vari)` (attribute)

Returns: a module

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

3.4.22 NameOfVariety (for IsToricVariety)

▷ `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.

3.4.23 ZariskiCotangentSheaf (for IsToricVariety)

▷ `ZariskiCotangentSheaf(vari)` (attribute)

Returns: a f.p. graded S-module

This method returns a f. p. graded S-module (S being the Cox ring of the variety), such that the sheafification of this module is the Zariski cotangent sheaf of *vari*.

3.4.24 CotangentSheaf (for IsToricVariety)

▷ `CotangentSheaf(vari)` (attribute)

Returns: a f.p. graded S-module

This method returns a f. p. graded S-module (S being the Cox ring of the variety), such that the sheafification of this module is the cotangent sheaf of *vari*.

3.4.25 EulerCharacteristic (for IsToricVariety)

▷ `EulerCharacteristic(vari)` (attribute)

Returns: a non-negative integer

This method computes the Euler characteristic of *vari*.

3.5 Methods

3.5.1 UnderlyingSheaf (for IsToricVariety)

▷ `UnderlyingSheaf(vari)` (operation)

Returns: a sheaf

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

3.5.2 CoordinateRingOfTorus (for IsToricVariety, IsList)

▷ `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.

3.5.3 * (for IsToricVariety, IsToricVariety)

▷ `\*(vari1, vari2)` (operation)

Returns: a variety

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

3.5.4 CharacterToRationalFunction (for IsHomalgElement, IsToricVariety)

▷ `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*. This computation needs to know the coordinate ring of the torus of the variety *vari*. By default this ring is introduced with variables *x1* to *xn* where *n* is the dimension of the variety. If different variables should be used, then *CoordinateRingOfTorus* has to be set accordingly before calling this method.

3.5.5 CoxRing (for IsToricVariety, IsList)

▷ `CoxRing(vari, vars)` (operation)

Returns: a ring

Computes the Cox ring of the variety *vari*. *vars* needs to be a string. We allow for two different formats. Either, it is a string which does not contain ",". Then this string will be index and the resulting strings are then used as names for the variables of the Cox ring. Alternatively, one can also use a string containing ",". In this case, a "," is considered as separator and one can provide individual names for all variables of the Cox ring.

3.5.6 WeilDivisorsOfVariety (for IsToricVariety)

▷ `WeilDivisorsOfVariety(vari)` (operation)

Returns: a list

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

3.5.7 Fan (for IsToricVariety)

▷ `Fan(vari)` (operation)

Returns: a fan

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

3.5.8 Factors (for IsToricVariety)

▷ `Factors(vari)` (operation)

3.5.9 BlowUpOnIthMinimalTorusOrbit (for IsToricVariety, IsInt)

▷ `BlowUpOnIthMinimalTorusOrbit(`*vari, p*`)` (operation)

3.5.10 ZariskiCotangentSheafViaEulerSequence

▷ `ZariskiCotangentSheafViaEulerSequence(`*arg*`)` (function)

3.5.11 ZariskiCotangentSheafViaPoincareResidueMap

▷ `ZariskiCotangentSheafViaPoincareResidueMap(`*arg*`)` (function)

3.5.12 ithBettiNumber (for IsToricVariety, IsInt)

▷ `ithBettiNumber(`*vari, p*`)` (operation)

3.5.13 NrOfqRationalPoints (for IsToricVariety, IsInt)

▷ `NrOfqRationalPoints(`*vari, p*`)` (operation)

3.6 Constructors

3.6.1 ToricVariety (for IsToricVariety)

▷ `ToricVariety(`*vari*`)` (operation)

3.6.2 ToricVariety (for IsList)

▷ `ToricVariety(`*vari*`)` (operation)

3.6.3 ToricVariety (for IsConvexObject)

▷ `ToricVariety(`*conv*`)` (operation)

Returns: a variety

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

3.6.4 ToricVariety (for IsList, IsList, IsList)

▷ `ToricVariety(`*rays, cones, degree_list*`)` (operation)

Returns: a variety

Creates a toric variety from a list *rays* of ray generators and cones *cones*. Beyond the functionality of the other methods, this constructor allows to assign specific gradings to the homogeneous variables of the Cox ring. With respect to the order in which the rays appear in the list *rays*, we assign gradings as provided by the third argument *degree_list* . The latter is a list of integers.

3.6.5 ToricVariety (for IsList, IsList, IsList, IsList)

▷ ToricVariety(*rays, cones, degree_list, var_list*) (operation)

Returns: a variety

Creates a toric variety from a list *rays* of ray generators and cones *cones*. Beyond the functionality of the other methods, this constructor allows to assign specific gradings and homogeneous variable names to the ray generators of this toric variety. With respect to the order in which the rays appear in the list *rays*, we assign gradings and variable names as provided by the third and fourth argument. These are the list of gradings *degree_list* and the list of variables names *var_list*. The former is a list of integers and the latter a list of strings.

3.6.6 ToricVarietiesFromGrading (for IsList)

▷ ToricVarietiesFromGrading(*a, list, of, lists, of, integers*) (operation)

Returns: a list of toric varieties

Given a $\mathbb{Z}^n$-grading of a polynomial ring, this method computes all toric varieties, which are normal and have no-torus factor and whose Cox ring obeys the given $\mathbb{Z}^n$-grading.

3.6.7 ToricVarietyFromGrading (for IsList)

▷ ToricVarietyFromGrading(*a, list, of, lists, of, integers*) (operation)

Returns: a toric variety

Given a $\mathbb{Z}^n$-grading of a polynomial ring, this method computes a toric variety, which is normal and has no-torus factor and whose Cox ring obeys the given $\mathbb{Z}^n$-grading.

Chapter 4

Toric subvarieties

4.1 The GAP category

4.1.1 IsToricSubvariety (for IsToricVariety)

▷ `IsToricSubvariety(M)` (filter)

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 Properties

4.2.1 IsClosedSubvariety (for IsToricSubvariety)

▷ `IsClosedSubvariety(vari)` (property)

Returns: true or false

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

4.2.2 IsOpen (for IsToricSubvariety)

▷ `IsOpen(vari)` (property)

Returns: true or false

Checks if a subvariety is a closed subset.

4.2.3 IsWholeVariety (for IsToricSubvariety)

▷ `IsWholeVariety(vari)` (property)

Returns: true or false

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

4.3 Attributes

4.3.1 UnderlyingToricVariety (for IsToricSubvariety)

▷ `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 (for IsToricSubvariety)

▷ `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 (for IsToricSubvariety)

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

Returns: a variety

Returns the ambient toric variety of the subvariety *vari*

4.4 Methods

4.4.1 ClosureOfTorusOrbitOfCone (for IsToricVariety, IsCone)

▷ `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 Constructors

4.5.1 ToricSubvariety (for IsToricVariety, IsToricVariety)

▷ `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: Examples

5.1.1 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_]/( x1*x1_-1, x2*x2_-1, x3*x3_-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"
```

5.2 The GAP category

5.2.1 IsAffineToricVariety (for IsToricVariety)

▷ `IsAffineToricVariety(M)` (filter)

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.

5.3 Attributes

5.3.1 CoordinateRing (for IsAffineToricVariety)

▷ `CoordinateRing(vari)` (attribute)

Returns: a ring

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

5.3.2 ListOfVariablesOfCoordinateRing (for IsAffineToricVariety)

▷ `ListOfVariablesOfCoordinateRing(vari)` (attribute)

Returns: a list

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

5.3.3 MorphismFromCoordinateRingToCoordinateRingOfTorus (for IsToricVariety)

▷ `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 (for IsToricVariety)

▷ `ConeOfVariety(vari)` (attribute)

Returns: a cone

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

5.4 Methods

5.4.1 CoordinateRing (for IsToricVariety, IsList)

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

Returns: a ring

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

5.4.2 Cone (for IsToricVariety)

▷ `Cone(vari)` (operation)

Returns: a cone

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

5.5 Constructors

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

Chapter 6

Projective toric varieties

6.1 Projective toric varieties: Examples

6.1.1 P1xP1 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_]/( x1*x1_-1, x2*x2_-1 )
gap> IsVeryAmple( Polytope( P1P1 ) );
true
gap> ProjectiveEmbedding( P1P1 );
[ |[ x1_*x2_ ]|, |[ x1_ ]|, |[ x1_*x2 ]|, |[ x2_ ]|,
|[ 1 ]|, |[ x2 ]|, |[ x1*x2_ ]|, |[ x1 ]|, |[ x1*x2 ]| ]
gap> Length( ProjectiveEmbedding( P1P1 ) );
9
gap> CoxRing( P1P1 );
Q[x_1,x_2,x_3,x_4]
(weights: [ ( 0, 1 ), ( 1, 0 ), ( 1, 0 ), ( 0, 1 ) ])
gap> Display( SRIdeal( P1P1 ) );
x_1*x_4,
x_2*x_3

A (left) ideal generated by the 2 entries of the above matrix

(graded, degrees of generators: [ ( 0, 2 ), ( 2, 0 ) ])
gap> Display( IrrelevantIdeal( P1P1 ) );
x_1*x_2,
x_1*x_3,
x_2*x_4,
x_3*x_4
```

```
A (left) ideal generated by the 4 entries of the above matrix

(graded, degrees of generators: [ ( 1, 1 ), ( 1, 1 ), ( 1, 1 ), ( 1, 1 ) ])
```

6.1.2 P1xP1 from product of P1s

Example

```
gap> P1 := ProjectiveSpace( 1 );
<A projective toric variety of dimension 1>
gap> IsComplete( P1 );
true
gap> IsSmooth( P1 );
true
gap> Dimension( P1 );
1
gap> CoxRing( P1, "q" );
Q[q_1,q_2]
(weights: [ 1, 1 ])
gap> P1xP1 := P1*P1;
<A projective smooth toric variety of dimension 2 which is a product
of 2 toric varieties>
gap> ByASmallerPresentation( ClassGroup( P1xP1 ) );
<A free left module of rank 2 on free generators>
gap> CoxRing( P1xP1, "x1,y1,y2,x2" );
Q[x1,y1,y2,x2]
(weights: [ ( 0, 1 ), ( 1, 0 ), ( 1, 0 ), ( 0, 1 ) ])
gap> Display( SRIdeal( P1xP1 ) );
x1*x2,
y1*y2

A (left) ideal generated by the 2 entries of the above matrix

(graded, degrees of generators: [ ( 0, 2 ), ( 2, 0 ) ])
gap> Display( IrrelevantIdeal( P1xP1 ) );
x1*y1,
x1*y2,
y1*x2,
y2*x2

A (left) ideal generated by the 4 entries of the above matrix

(graded, degrees of generators: [ ( 1, 1 ), ( 1, 1 ), ( 1, 1 ), ( 1, 1 ) ])
```

6.2 The GAP category

6.2.1 IsProjectiveToricVariety (for IsToricVariety)

▷ `IsProjectiveToricVariety(M)` (filter)

Returns: true or false

The *GAP* category of a projective toric variety.

6.3 Attribute

6.3.1 PolytopeOfVariety (for IsToricVariety)

▷ `PolytopeOfVariety(vari)` (attribute)

Returns: a polytope

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

6.3.2 AffineCone (for IsToricVariety)

▷ `AffineCone(vari)` (attribute)

Returns: a cone

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

6.3.3 ProjectiveEmbedding (for IsToricVariety)

▷ `ProjectiveEmbedding(vari)` (attribute)

Returns: a list

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

6.4 Properties

6.4.1 IsIsomorphicToProjectiveSpace (for IsToricVariety)

▷ `IsIsomorphicToProjectiveSpace(vari)` (property)

Returns: true or false

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

6.4.2 IsDirectProductOfPNs (for IsToricVariety)

▷ `IsDirectProductOfPNs(vari)` (property)

Returns: true or false

Checks if the given toric variety *vari* is a direct product of projective spaces.

6.5 Methods

6.5.1 Polytope (for IsToricVariety)

▷ `Polytope(vari)` (operation)

Returns: a polytope

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

6.5.2 AmpleDivisor (for IsToricVariety and HasPolytopeOfVariety)

▷ `AmpleDivisor(vari)` (operation)

Returns: an ample divisor

Given a projective toric variety `vari` constructed from a polytope, this method computes the toric divisor associated to this polytope. By general theory (see Cox-Schenk-Little) this divisor is known to be ample. Thus this method computes an ample divisor on the given toric variety.

6.6 Constructors

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

Chapter 7

Toric morphisms

7.1 Toric morphisms: Examples

7.1.1 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 free left module of rank 1 on a free generator>
gap> Display(ByASmallerPresentation(ClassGroup( P1 )));
Z^(1 x 1)
gap> ClassGroup( P2 );
<A free left module of rank 1 on a free generator>
gap> Display(ByASmallerPresentation(ClassGroup( P2 )));
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(ClassGroup(M));
[ [  0,  1 ],
  [  1,  0 ] ]

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

7.2 The GAP category

7.2.1 IsToricMorphism (for IsObject)

▷ `IsToricMorphism(`*M*`)` (filter)

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.

7.3 Properties

7.3.1 IsMorphism (for IsToricMorphism)

▷ `IsMorphism(`*morph*`)` (property)

Returns: true or false

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

7.3.2 IsProper (for IsToricMorphism)

▷ `IsProper(`*morph*`)` (property)

Returns: true or false

Checks if the defined morphism *morph* is proper.

7.4 Attributes

7.4.1 SourceObject (for IsToricMorphism)

▷ `SourceObject(`*morph*`)` (attribute)

Returns: a variety

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

7.4.2 UnderlyingGridMorphism (for IsToricMorphism)

▷ `UnderlyingGridMorphism(`*morph*`)` (attribute)

Returns: a map

Returns the grid map which defines *morph*.

7.4.3 ToricImageObject (for IsToricMorphism)

▷ `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.

7.4.4 RangeObject (for IsToricMorphism)

▷ `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.

7.4.5 MorphismOnWeilDivisorGroup (for IsToricMorphism)

▷ `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.

7.4.6 ClassGroup (for IsToricMorphism)

▷ `ClassGroup(morph)` (attribute)

Returns: a morphism

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

7.4.7 MorphismOnCartierDivisorGroup (for IsToricMorphism)

▷ `MorphismOnCartierDivisorGroup(morph)` (attribute)

Returns: a morphism

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

7.4.8 PicardGroup (for IsToricMorphism)

▷ `PicardGroup(morph)` (attribute)

Returns: a morphism

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

7.4.9 Source (for IsToricMorphism)

▷ `Source(morph)` (attribute)

Returns: a variety

Return the source of the toric morphism *morph*.

7.4.10 Range (for IsToricMorphism)

▷ `Range(morph)` (attribute)

Returns: a variety

Returns the range of the toric morphism *morph* if specified.

7.4.11 MorphismOnIthFactor (for IsToricMorphism)

▷ `MorphismOnIthFactor(morph)` (attribute)

7.5 Methods

7.5.1 UnderlyingListList (for IsToricMorphism)

▷ `UnderlyingListList(morph)` (operation)

Returns: a list

Returns a list of list which represents the grid homomorphism.

7.6 Constructors

7.6.1 ToricMorphism (for IsToricVariety, IsList)

▷ `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.

7.6.2 ToricMorphism (for IsToricVariety, IsList, IsToricVariety)

▷ `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*.

Chapter 8

Toric divisors

8.1 Toric divisors: Examples

8.1.1 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[1]+P[2];
<A divisor of a toric variety with coordinates ( 1, 1, 0, 0 )>
gap> IsBasepointFree(D);
true
gap> IsAmple(D);
true
gap> CoordinateRingOfTorus(H7,"x");
Q[x1,x1_,x2,x2_]/( x1*x1_-1, x2*x2_-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(RingMorphismOfDivisor(D));
Q[x_1,x_2,x_3,x_4]
(weights: [ ( 0, 1 ), ( 1, 0 ), ( 1, -7 ), ( 0, 1 ) ])
  ^
  |
```

```
[ x_1*x_2, x_1^8*x_3, x_2*x_4, x_1^7*x_3*x_4, x_1^6*x_3*x_4^2,
  x_1^5*x_3*x_4^3, x_1^4*x_3*x_4^4, x_1^3*x_3*x_4^5, x_1^2*x_3*x_4^6,
  x_1*x_3*x_4^7, x_3*x_4^8 ]
  |
  |
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> MonomsOfCoxRingOfDegree(D);
[ x_1*x_2, x_1^8*x_3, x_2*x_4, x_1^7*x_3*x_4, x_1^6*x_3*x_4^2,
  x_1^5*x_3*x_4^3, x_1^4*x_3*x_4^4, x_1^3*x_3*x_4^5, x_1^2*x_3*x_4^6,
  x_1*x_3*x_4^7, x_3*x_4^8 ]
gap> D2:=D-2*P[2];
<A divisor of a toric variety with coordinates ( 1, -1, 0, 0 )>
gap> IsBasepointFree(D2);
false
gap> IsAmple(D2);
false
```

8.1.2 Polytope of toric divisors

Example

```
gap> P1 := ProjectiveSpace( 1 );
<A projective toric variety of dimension 1>
gap> divisor := DivisorOfGivenClass( P1, [ -1 ] );
<A divisor of a toric variety with coordinates ( -1, 0 )>
gap> polytope := PolytopeOfDivisor( divisor );
<A polytope in |R^1>
```

8.2 The GAP category

8.2.1 IsToricDivisor (for IsObject)

▷ `IsToricDivisor(`*M*`)` (filter)

Returns: true or false

The *GAP* category of torus invariant Weil divisors.

8.2.2 twitter (for IsToricDivisor)

▷ `twitter(`*arg*`)` (attribute)

8.3 Properties

8.3.1 IsCartier (for IsToricDivisor)

▷ `IsCartier(`*divi*`)` (property)

Returns: true or false

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

8.3.2 IsPrincipal (for IsToricDivisor)

▷ `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.

8.3.3 IsPrimedivisor (for IsToricDivisor)

▷ `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.

8.3.4 IsBasepointFree (for IsToricDivisor)

▷ `IsBasepointFree(divi)` (property)

Returns: true or false

Checks if the divisor *divi* is basepoint free.

8.3.5 IsAmple (for IsToricDivisor)

▷ `IsAmple(divi)` (property)

Returns: true or false

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

8.3.6 IsVeryAmple (for IsToricDivisor)

▷ `IsVeryAmple(divi)` (property)

Returns: true or false

Checks if the divisor *divi* is very ample.

8.3.7 IsNumericallyEffective (for IsToricDivisor)

▷ `IsNumericallyEffective(divi)` (property)

Returns: true or false

Checks if the divisor *divi* is nef.

8.4 Attributes

8.4.1 CartierData (for IsToricDivisor)

▷ `CartierData(divi)` (attribute)

Returns: a list

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

8.4.2 CharacterOfPrincipalDivisor (for IsToricDivisor)

▷ `CharacterOfPrincipalDivisor(divi)` (attribute)

Returns: a homalg module element

Returns the character corresponding to the principal divisor *divi*.

8.4.3 ClassOfDivisor (for IsToricDivisor)

▷ `ClassOfDivisor(divi)` (attribute)

Returns: a homalg module element

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

8.4.4 PolytopeOfDivisor (for IsToricDivisor)

▷ `PolytopeOfDivisor(divi)` (attribute)

Returns: a polytope

Returns the polytope corresponding to the divisor *divi*.

8.4.5 BasisOfGlobalSections (for IsToricDivisor)

▷ `BasisOfGlobalSections(divi)` (attribute)

Returns: a list

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

8.4.6 IntegerForWhichIsSureVeryAmple (for IsToricDivisor)

▷ `IntegerForWhichIsSureVeryAmple(divi)` (attribute)

Returns: an integer

Returns an integer `n` such that n·divi is very ample.

8.4.7 AmbientToricVariety (for IsToricDivisor)

▷ `AmbientToricVariety(divi)` (attribute)

Returns: a variety

Returns the toric variety which contains the prime divisors of the divisor *divi*.

8.4.8 UnderlyingGroupElement (for IsToricDivisor)

▷ `UnderlyingGroupElement(divi)` (attribute)

Returns: a homalg module element

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

8.4.9 UnderlyingToricVariety (for IsToricDivisor)

▷ `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.

8.4.10 DegreeOfDivisor (for IsToricDivisor)

▷ `DegreeOfDivisor(divi)` (attribute)

Returns: an integer

Returns the degree of the divisor *divi*. This is not to be confused with the (divisor) class of *divi*!

8.4.11 VarietyOfDivisorpolytope (for IsToricDivisor)

▷ `VarietyOfDivisorpolytope(divi)` (attribute)

Returns: a variety

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

8.4.12 MonomsOfCoxRingOfDegree (for IsToricDivisor)

▷ `MonomsOfCoxRingOfDegree(divi)` (attribute)

Returns: a list

Returns the monoms in the Cox ring of degree equal to the (divisor) class of the divisor *divi*.

8.4.13 CoxRingOfTargetOfDivisorMorphism (for IsToricDivisor)

▷ `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.

8.4.14 RingMorphismOfDivisor (for IsToricDivisor)

▷ `RingMorphismOfDivisor(divi)` (attribute)

Returns: a ring map

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*.

8.5 Methods

8.5.1 VeryAmpleMultiple (for IsToricDivisor)

▷ `VeryAmpleMultiple(divi)` (operation)

Returns: a divisor

Returns a very ample multiple of the ample divisor *divi*. The method will fail if divisor is not ample.

8.5.2 CharactersForClosedEmbedding (for IsToricDivisor)

▷ `CharactersForClosedEmbedding(divi)` (operation)

Returns: a list

Returns characters for closed embedding defined via the ample divisor `divi`. The method fails if the divisor `divi` is not ample.

8.5.3 \+ (for IsToricDivisor, IsToricDivisor)

▷ `\+(divi1, divi2)` (operation)

Returns: a divisor

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

8.5.4 \- (for IsToricDivisor, IsToricDivisor)

▷ `\-(divi1, divi2)` (operation)

Returns: a divisor

Returns the divisor `divi1` minus `divi2`.

8.5.5 * (for IsInt, IsToricDivisor)

▷ `\*(k, divi)` (operation)

Returns: a divisor

Returns `k` times the divisor `divi`.

8.5.6 MonomsOfCoxRingOfDegree (for IsToricVariety, IsHomalgElement)

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

Returns: a list

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

8.5.7 DivisorOfGivenClass (for IsToricVariety, IsHomalgElement)

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

Returns: a divisor

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.

8.5.8 AddDivisorToItsAmbientVariety (for IsToricDivisor)

▷ `AddDivisorToItsAmbientVariety(divi)` (operation)

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

8.5.9 Polytope (for IsToricDivisor)

▷ `Polytope(divi)` (operation)

Returns: a polytope

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

8.5.10 CoxRingOfTargetOfDivisorMorphism (for IsToricDivisor, IsString)

▷ `CoxRingOfTargetOfDivisorMorphism(divi, string)` (operation)

Returns: a ring

Given a toric divisor `divi`, it induces a toric morphism. The target of this morphism is a toric variety. This method returns the Cox ring of this target. The variables are named according to `string`.

8.6 Constructors

8.6.1 DivisorOfCharacter (for IsHomalgElement, IsToricVariety)

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

Returns: a divisor

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

8.6.2 DivisorOfCharacter (for IsList, IsToricVariety)

▷ `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`.

8.6.3 CreateDivisor (for IsHomalgElement, IsToricVariety)

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

Returns: a divisor

Returns the divisor of the toric variety `vari` which corresponds to the Weil group element `elem`. by the list `lis`.

8.6.4 CreateDivisor (for IsList, IsToricVariety)

▷ `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`.

Chapter 9

Blowups of toric varieties

9.1 Constructors

9.1.1 BlowupOfToricVariety (for IsToricVariety, IsList, IsString)

▷ `BlowupOfToricVariety(a, toric, variety, a, list, and, a, string)` (operation)

Returns: a variety

The arguments are a toric variety vari, a string s which specifies the locus to be blown up and a string which specifies how to name the new blowup coordinate. Based on this, this method creates the blowup of a toric variety. This process rests on a star sub-division of the fan (c.f. 3.3.17 in Cox-Little-Schenk)

9.1.2 SequenceOfBlowupsOfToricVariety (for IsToricVariety, IsList)

▷ `SequenceOfBlowupsOfToricVariety(a, toric, variety, and, a, list)` (operation)

Returns: a variety

The arguments are a toric variety vari and a list of lists. Each entry of this list must contain the information for one blowup, i.e. be made up of the two lists used as input for the method BlowupOfToricVariety. This method then performs this sequence of blowups and returns the corresponding toric variety.

Index