NConvex

A Gap package to perform polyhedral computations

2022.09-01

30 September 2022

Kamal Saleh

Sebastian Gutsche

Kamal Saleh
Email: kamal.saleh@uni-siegen.de
Homepage: https://github.com/kamalsaleh
Address: Department Mathematik
Universität Siegen
Walter-Flex-Straße 3
57068 Siegen
Germany

Sebastian Gutsche
Email: gutsche@mathematik.uni-siegen.de
Homepage: https://sebasguts.github.io/
Address: Department Mathematik
Universität Siegen
Walter-Flex-Straße 3
57068 Siegen
Germany

Contents

Chapter 1

Introduction

The **NConvex** package is a GAP package. Its aim is to carry out polyhedral constructions and computations, namely computing properties and attributes of cones, polyhedrons, polytopes and fans. Its has been written to provide the needed tools for the package \textbf{ToricVarieties}. All written as part of the homalg-project.

1.1 Installation

The package can easily be obtained by cloning the repository

`https://github.com/homalg-project/NConvex.git`

in the pkg directory of the Gap installation or your local directory for Gap packages.

1.2 Requirements

Here is a list of the required Gap packages:

- The Gap package **AutoDoc** is required to create the documentation and to perform tests. A fresh version can be installed from

  `https://github.com/gap-packages/AutoDoc.git`

- The Gap package **CddInterface** is required to convert between H-rep and V-rep of polyhedrons. It can be obtained at:

  `https://github.com/homalg-project/CddInterface.git`

- The Gap/homalg-project package **Modules**. You can install the package by cloning the **homalg_project** repository from

  `https://github.com/homalg-project/homalg_project.git`

- The Gap package **NormalizInterface**. You can install it from

  `https://github.com/gap-packages/NormalizInterface.git`

- In case **NormalizInterface** is not available, then you can use the Gap/homalg package **4ti2Interface**. It is already included in the **homalg_project**. Make sure to accordingly change the dependencies entry in PackageInfo.g

Chapter 2

Functionality

Chapter 3

Convex objects

3.1 Attributes

3.1.1 AmbientSpaceDimension (for IsConvexObject)

▷ `AmbientSpaceDimension(`*obj*`)` (attribute)

Returns: integer

Returns the dimension of the ambient space, i.e., the space that contains the convex object.

3.1.2 Dimension (for IsConvexObject)

▷ `Dimension(`*obj*`)` (attribute)

Returns: integer

Returns the dimension of the covex object.

3.1.3 IsFullDimensional (for IsConvexObject)

▷ `IsFullDimensional(`*obj*`)` (property)

Returns: boolian

Returns whether the convex object is full dimensional or not.

3.1.4 InteriorPoint (for IsConvexObject)

▷ `InteriorPoint(`*obj*`)` (attribute)

Returns: a point in the object

Returns an interior point of the covex object.

Chapter 4

Cones

4.1 Creating cones

4.1.1 ConeByInequalities (for IsList)

▷ `ConeByInequalities(`*`L`*`)` (operation)

Returns: a `Cone` Object

The function takes a list of lists $[L_1, L_2, ...]$ where each L_j represents an inequality and returns the cone defined by them. For example the j'th entry $L_j = [a_{j1}, a_{j2}, ..., a_{jn}]$ corresponds to the inequality $\sum_{i=1}^{n} a_{ji} x_i \geq 0$.

4.1.2 ConeByEqualitiesAndInequalities (for IsList, IsList)

▷ `ConeByEqualitiesAndInequalities(`*`Eq, Ineq`*`)` (operation)

Returns: a `Cone` Object

The function takes two lists. The first list is the equalities and the second is the inequalities and returns the cone defined by them.

4.1.3 Cone (for IsList)

▷ `Cone(`*`L`*`)` (operation)

Returns: a `Cone` Object

The function takes a list in which every entry represents a ray in the ambient vector space and returns the cone defined by them.

4.1.4 Cone (for IsCddPolyhedron)

▷ `Cone(`*`cdd_cone`*`)` (operation)

Returns: a `Cone` Object

This function takes a cone defined in *CddInterface* and converts it to a cone in *NConvex*

4.2 Attributes of Cones

4.2.1 DefiningInequalities (for IsCone)

▷ `DefiningInequalities(C)` (attribute)

Returns: a list

Returns the list of the defining inequalities of the cone `C`.

4.2.2 EqualitiesOfCone (for IsCone)

▷ `EqualitiesOfCone(C)` (attribute)

Returns: a list

Returns the list of the equalities in the defining inequalities of the cone `C`.

4.2.3 DualCone (for IsCone)

▷ `DualCone(C)` (attribute)

Returns: a cone

Returns the dual cone of the cone `C`.

4.2.4 FacesOfCone (for IsCone)

▷ `FacesOfCone(C)` (attribute)

Returns: a list of cones

Returns the list of all faces of the cone `C`.

4.2.5 Facets (for IsCone)

▷ `Facets(C)` (attribute)

Returns: a list of cones

Returns the list of all facets of the cone `C`.

4.2.6 FVector (for IsCone)

▷ `FVector(C)` (attribute)

Returns: a list

Returns a list whose i'th entry is the number of faces of dimension i.

4.2.7 RelativeInteriorRay (for IsCone)

▷ `RelativeInteriorRay(C)` (attribute)

Returns: a list

Returns a relative interior point (or ray) in the cone `C`.

4.2.8 HilbertBasis (for IsCone)

▷ `HilbertBasis(C)` (attribute)

Returns: a list

Returns the Hilbert basis of the cone `C`

4.2.9 HilbertBasisOfDualCone (for IsCone)

▷ `HilbertBasisOfDualCone(C)` (attribute)

Returns: a list

Returns the Hilbert basis of the dual cone of the cone `C`

4.2.10 LinealitySpaceGenerators (for IsCone)

▷ `LinealitySpaceGenerators(C)` (attribute)

Returns: a list

Returns a basis of the lineality space of the cone `C`.

4.2.11 ExternalCddCone (for IsCone)

▷ `ExternalCddCone(C)` (attribute)

Returns: a cdd object

Converts the cone to a cdd object. The operations of CddInterface can then be applied on this convex object.

4.2.12 ExternalNmzCone (for IsCone)

▷ `ExternalNmzCone(C)` (attribute)

Returns: an normaliz object

Converts the cone to a normaliz object. The operations of NormalizInterface can then be applied on this convex object.

4.2.13 AmbientSpaceDimension (for IsCone)

▷ `AmbientSpaceDimension(C)` (attribute)

Returns: an integer

The dimension of the ambient space of the cone, i.e., the space that contains the cone.

4.2.14 LatticePointsGenerators (for IsCone)

▷ `LatticePointsGenerators(C)` (attribute)

Returns: a list

See `LatticePointsGenerators` for polyhedrons. Please note that any cone is a polyhedron.

4.2.15 GridGeneratedByCone (for IsCone)

▷ `GridGeneratedByCone(C)` (attribute)

Returns: a homalg module

Returns the homalg $\mathbb{Z}$-module that is generated by the ray generators of the cone.

4.2.16 FactorGrid (for IsCone)

▷ `FactorGrid(C)` (attribute)

Returns: a homalg module

Returns the homalg $\mathbb{Z}$-module that is presented by the matrix whose raws are the ray generators of the cone.

4.2.17 FactorGridMorphism (for IsCone)

▷ `FactorGridMorphism(C)` (attribute)

Returns: a homalg morphism

Returns an epimorphism from a free $\mathbb{Z}$-module to `FactorGrid(C)`.

4.2.18 GridGeneratedByOrthogonalCone (for IsCone)

▷ `GridGeneratedByOrthogonalCone(C)` (attribute)

Returns: a homalg module

Returns the homalg $\mathbb{Z}$-module that is by generated the ray generators of the orthogonal cone on `C`.

4.3 Properties of Cones

4.3.1 IsRegularCone (for IsCone)

▷ `IsRegularCone(C)` (property)

Returns: true or false

Returns if the cone `C` is regular or not.

4.3.2 IsRay (for IsCone)

▷ `IsRay(C)` (property)

Returns: true or false

Returns if the cone `C` is ray or not.

4.3.3 IsZero (for IsCone)

▷ `IsZero(C)` (property)

Returns: true or false

Returns whether the cone is the zero cone or not.

4.4 Operations on cones

4.4.1 FourierProjection (for IsCone, IsInt)

▷ `FourierProjection(C, m)` (operation)

Returns: a cone

Returns the projection of the cone on the space (O, $x_1,\ldots,x_{m-1},x_{m+1},\ldots,x_n$).

4.4.2 IntersectionOfCones (for IsCone, IsCone)

▷ `IntersectionOfCones(C1, C2)` (operation)

Returns: a cone

Returns the intersection.

4.4.3 IntersectionOfCones (for IsList)

▷ `IntersectionOfCones(L)` (operation)

Returns: a cone

The input is a list of cones and the output is their intersection.

4.4.4 Contains (for IsCone, IsCone)

▷ `Contains(C1, C2)` (operation)

Returns: a true or false

Returns if the cone `C1` contains the cone `C2`.

4.4.5 IsRelativeInteriorRay (for IsList, IsCone)

▷ `IsRelativeInteriorRay(L, C)` (operation)

Returns: a true or false

Checks whether the input point (or ray) `L` is in the relative interior of the cone `C`.

Example
```
gap> P:= Cone( [ [ 2, 7 ], [ 0, 12 ], [ -2, 5 ] ] );
<A cone in |R^2>
gap> d:= DefiningInequalities( P );
[ [ -7, 2 ], [ 5, 2 ] ]
gap> Q:= ConeByInequalities( d );
<A cone in |R^2>
gap> P=Q;
true
gap> IsPointed( P );
true
gap> RayGenerators( P );
[ [ -2, 5 ], [ 2, 7 ] ]
gap> HilbertBasis( P );
[ [ -2, 5 ], [ -1, 3 ], [ 0, 1 ], [ 1, 4 ], [ 2, 7 ] ]
gap> HilbertBasis( Q );
[ [ -2, 5 ], [ -1, 3 ], [ 0, 1 ], [ 1, 4 ], [ 2, 7 ] ]
gap> P_dual:= DualCone( P );
<A cone in |R^2>
gap> RayGenerators( P_dual );
[ [ -7, 2 ], [ 5, 2 ] ]
gap> Dimension( P );
2
gap> List( Facets( P ), RayGenerators );
[ [ [ -2, 5 ] ], [ [ 2, 7 ] ] ]
gap> faces := FacesOfCone( P );
[ <A cone in |R^2>, <A cone in |R^2>, <A ray in |R^2>,
 <A ray in |R^2> ]
```

```
gap> RelativeInteriorRay( P );
[ -2, 41 ]
gap> IsRelativeInteriorRay( [ -2, 41 ], P );
true
gap> IsRelativeInteriorRay( [ 2, 7 ], P );
false
gap> LinealitySpaceGenerators( P );
[  ]
gap> IsRegularCone( P );
false
gap> IsRay( P );
false
gap> proj_x1:= FourierProjection( P, 2 );
<A cone in |R^1>
gap> RayGenerators( proj_x1 );
[ [ -1 ], [ 1 ] ]
gap> DefiningInequalities( proj_x1 );
[ [ 0 ] ]
gap> R:= Cone( [ [ 4, 5 ], [ -2, 1 ] ] );
<A cone in |R^2>
gap> T:= IntersectionOfCones( P, R );
<A cone in |R^2>
gap> RayGenerators( T );
[ [ -2, 5 ], [ 2, 7 ] ]
gap> W:= Cone( [ [-3,-4 ] ] );
<A ray in |R^2>
gap> I:= IntersectionOfCones( P, W );
<A cone in |R^2>
gap> RayGenerators( I );
[  ]
gap> Contains( P, I );
true
gap> Contains( W, I );
true
gap> Contains( P, R );
false
gap> Contains( R, P );
true
gap> cdd_cone:= ExternalCddCone( P );
< Polyhedron given by its V-representation >
gap> Display( cdd_cone );
V-representation
begin
3 X 3  rational

   0   2   7
   0   0  12
   0  -2   5
end
gap> Cdd_Dimension( cdd_cone );
2
gap> H:= Cdd_H_Rep( cdd_cone );
```

```
< Polyhedron given by its H-representation >
gap> Display( H );
H-representation
begin
   2 X 3  rational

   0   5   2
   0  -7   2
end
gap> P:= Cone( [ [ 1, 1, -3 ], [ -1, -1, 3 ], [ 1, 2, 1 ], [ 2, 1, 2 ] ] );
< A cone in |R^3>
gap> IsPointed( P );
false
gap> Dimension( P );
3
gap> IsRegularCone( P );
false
gap> P;
< A cone in |R^3 of dimension 3 with 4 ray generators>
gap> RayGenerators( P );
[ [ -1, -1, 3 ], [ 1, 1, -3 ], [ 1, 2, 1 ], [ 2, 1, 2 ] ]
gap> d:= DefiningInequalities( P );
[ [ -5, 8, 1 ], [ 7, -4, 1 ] ]
gap> facets:= Facets( P );
[ <A cone in |R^3>, <A cone in |R^3> ]
gap> faces := FacesOfCone( P );
[ <A cone in |R^3>, <A cone in |R^3>, <A cone in |R^3>,
 <A cone in |R^3>, <A cone in |R^3> ]
gap> FVector( P );
[ 1, 2, 1 ]
gap> List( faces, Dimension );
[ 0, 3, 2, 1, 2 ]
gap> L_using_4ti2 := [ [ [ 0, 0, 0 ] ], [ [ -2, -1, 10 ],
> [ 0, 0, 1 ], [ 2, 1, 2 ] ],  [ [ 1, 1, -3 ] ] ];;
gap> L_using_Normaliz := [ [ [ 0, 0, 0 ] ], [ [ -1, 0, 7 ],
> [ 0, 0, 1 ], [ 1, 0, 5 ] ], [ [ 1, 1, -3 ] ] ];;
gap> L := LatticePointsGenerators( P );;
gap> L = L_using_4ti2 or L = L_using_Normaliz;
true
gap> DualCone( P );
< A cone in |R^3>
gap> RayGenerators( DualCone( P ) );
[ [ -5, 8, 1 ], [ 7, -4, 1 ] ]
gap> Q_x1x3:= FourierProjection(P, 2 );
<A cone in |R^2>
gap> RayGenerators( Q_x1x3 );
[ [ -1, 3 ], [ 1, -3 ], [ 1, 1 ] ]
```

4.4.6 NonReducedInequalities (for IsCone)

▷ NonReducedInequalities(*C*) (operation)

Returns: a list

It returns a list of inequalities that define the cone.

Chapter 5

Fans

5.1 Constructors

5.1.1 Fan (for IsFan)

▷ `Fan(F)` (operation)

Returns: a fan object

If the input *F* is fan then return *F*.

5.1.2 Fan (for IsList)

▷ `Fan(C)` (operation)

Returns: a fan object

The input is a list of list C. the output is the fan defined by the cones $\{\mathrm{Cone}_i(C_i)\}_{C_i \in C}$.

5.1.3 Fan (for IsList, IsList)

▷ `Fan(R, C)` (operation)

Returns: a fan object

The input is two lists, R that indicates the rays and C that indicates the cones. The output is the fan defined by the cones $\{\mathrm{Cone}_i(\{R_j, j \in C_i\})\}_{C_i \in C}$.

Below we define the same fan in two different ways:

Example
```
gap> F1 := Fan( [ [ [ 2, 1 ], [ 1, 2 ] ], [ [ 2, 1 ], [ 1, -1 ] ],
>               [ [ -3, 1 ], [ -1, -3 ] ] ] );
<A fan in |R^2>
gap> F2 := Fan( [ [ 2, 1 ], [ 1, 2 ], [ -3, 1 ], [ -1, -3 ], [ 1, -1 ] ],
>          [ [ 1, 2 ], [ 1, 5 ], [ 3, 4 ] ] );
<A fan in |R^2>
gap> rays1 := RayGenerators( F1 );
[ [ -3, 1 ], [ -1, -3 ], [ 1, -1 ], [ 1, 2 ], [ 2, 1 ] ]
gap> rays2 := RayGenerators( F2 );
[ [ -3, 1 ], [ -1, -3 ], [ 1, -1 ], [ 1, 2 ], [ 2, 1 ] ]
gap> RaysInMaximalCones( F1 );
[ [ 0, 0, 0, 1, 1 ], [ 0, 0, 1, 0, 1 ], [ 1, 1, 0, 0, 0 ] ]
gap> RaysInMaximalCones( F2 );
[ [ 0, 0, 0, 1, 1 ], [ 0, 0, 1, 0, 1 ], [ 1, 1, 0, 0, 0 ] ]
```

```
gap> RaysInAllCones( F1 );
[ [ 0, 0, 0, 0, 0 ], [ 0, 0, 0, 1, 1 ], [ 0, 0, 0, 0, 1 ],
  [ 0, 0, 0, 1, 0 ], [ 0, 0, 1, 0, 1 ], [ 0, 0, 1, 0, 0 ],
  [ 1, 1, 0, 0, 0 ], [ 0, 1, 0, 0, 0 ], [ 1, 0, 0, 0, 0 ] ]
gap> FVector( F1 );
[ 5, 3 ]
gap> IsComplete( F1 );
false
gap> IsSimplicial( F1 );
true
gap> IsNormalFan( F1 );
false
gap> IsRegularFan( F1 );
false
gap> P1 := Polytope( [ [ 1 ], [ -1 ] ] );
<A polytope in |R^1>
gap> P1 := NormalFan( P1 );
<A complete fan in |R^1>
gap> RayGenerators( P1 );
[ [ -1 ], [ 1 ] ]
gap> P3 := P1 * P1 * P1;
<A fan in |R^3>
gap> RayGenerators( P3 );
[ [ -1, 0, 0 ], [ 0, -1, 0 ], [ 0, 0, -1 ], [ 0, 0, 1 ], [ 0, 1, 0 ],
  [ 1, 0, 0 ] ]
gap> RaysInMaximalCones( P3 );
[ [ 0, 0, 0, 1, 1, 1 ], [ 0, 0, 1, 0, 1, 1 ], [ 0, 1, 0, 1, 0, 1 ],
  [ 0, 1, 1, 0, 0, 1 ], [ 1, 0, 0, 1, 1, 0 ], [ 1, 0, 1, 0, 1, 0 ],
  [ 1, 1, 0, 1, 0, 0 ], [ 1, 1, 1, 0, 0, 0 ] ]
gap> RaysInAllCones( P3 );
[ [ 0, 0, 0, 0, 0, 0 ], [ 0, 0, 0, 1, 1, 1 ], [ 0, 0, 0, 1, 1, 0 ],
  [ 0, 0, 0, 1, 0, 0 ], [ 0, 0, 0, 0, 1, 0 ], [ 0, 0, 0, 1, 0, 1 ],
  [ 0, 0, 0, 0, 0, 1 ], [ 0, 0, 0, 0, 1, 1 ], [ 0, 0, 1, 0, 1, 1 ],
  [ 0, 0, 1, 0, 1, 0 ], [ 0, 0, 1, 0, 0, 0 ], [ 0, 0, 1, 0, 0, 1 ],
  [ 0, 1, 0, 1, 0, 1 ], [ 0, 1, 0, 1, 0, 0 ], [ 0, 1, 0, 0, 0, 0 ],
  [ 0, 1, 0, 0, 0, 1 ], [ 0, 1, 1, 0, 0, 1 ], [ 0, 1, 1, 0, 0, 0 ],
  [ 1, 0, 0, 1, 1, 0 ], [ 1, 0, 0, 1, 0, 0 ], [ 1, 0, 0, 0, 0, 0 ],
  [ 1, 0, 0, 0, 1, 0 ], [ 1, 0, 1, 0, 1, 0 ], [ 1, 0, 1, 0, 0, 0 ],
  [ 1, 1, 0, 1, 0, 0 ], [ 1, 1, 0, 0, 0, 0 ], [ 1, 1, 1, 0, 0, 0 ] ]
gap> IsNormalFan( P3 );
true
gap> Dimension( P3 );
3
gap> PrimitiveCollections( P3 );
[ [ 4, 3 ], [ 5, 2 ], [ 6, 1 ] ]
```

The following is an example for a fan that is complete but not normal.

```
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 ] ];;
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 ] ];;
gap> F := Fan( rays, cones );
<A fan in |R^3>
gap> IsComplete( F );
true
gap> IsNormalFan( F );
false
gap> PrimitiveCollections( F );
[ [ 7, 1 ], [ 7, 2 ], [ 7, 3 ], [ 7, 4 ], [ 7, 5 ], [ 7, 6 ],
  [ 9, 1 ], [ 9, 2 ], [ 9, 3 ], [ 9, 6 ], [ 10, 1 ], [ 10, 2 ],
  [ 10, 3 ], [ 10, 4 ], [ 8, 1 ], [ 8, 2 ], [ 8, 3 ], [ 8, 5 ],
  [ 6, 1 ], [ 5, 2 ], [ 4, 3 ], [ 9, 10, 8 ], [ 5, 6, 4 ] ]
```

5.1.4 FansFromTriangulation (for IsList)

▷ `FansFromTriangulation(R)` (operation)

Returns: a list of fans

The input is a list of ray generators R. Provided that the package TopcomInterface is available, this function computes the list of all fine and regular triangulations of these ray generators. It then returns the list of the associated fans of these triangulations.

5.1.5 FanFromTriangulation (for IsList)

▷ `FanFromTriangulation(R)` (operation)

Returns: a fan

The input is a list of ray generators R. Provided that the package TopcomInterface is available, this function computes a fine and regular triangulation of these ray generators and returns the associated fan.

The above methods construct fans from so-called triangulations. For a given list R of lists of integers, a triangulation is a fan whose ray generators are contained in the given list R.

A regular triangulation is such a fan, for which all cones are strictly convex. It is called a fine triangulation, iff all elements of R are ray generators of this fan.

Above we present two method which make this approach available in NConvex via the package TopcomInterface, which in turn rests on the program Topcom. Consequently, these methods are only available if the package TopcomInterface is available. They compute either all of the fine and regular triangulations or merely just a single such triangulation.

As an example inspired from toric geometry, let us use the ray generators of the fan of the resolved conifold (i.e. the total space of the bundle). This space is known to allow for two different triangulations. The code below reproduces this feature.

Code
```
gap> rays := [ [ 1, 0, 1 ], [ 1, 1, 0 ], [ 0, 0, -1 ], [ 0, -1, 0 ] ];;
gap> all_triangulations := FansFromTriangulation( rays );
[ <A fan in |R^3>, <A fan in |R^3> ]
gap> one_triangulation := FanFromTriangulation( rays );
<A fan in |R^3>
```

5.2 Attributes

5.2.1 RayGenerators (for IsFan)

▷ `RayGenerators(F)` (attribute)

Returns: a list

The input is a fan `F`. The output is the set of all ray generators of the maximal cones in the fan.

5.2.2 GivenRayGenerators (for IsFan)

▷ `GivenRayGenerators(F)` (attribute)

Returns: a list

The input is a fan `F`. The output is the given or defining set of ray generators of the maximal cones in the fan.

5.2.3 RaysInMaximalCones (for IsFan)

▷ `RaysInMaximalCones(F)` (attribute)

Returns: a list

The input is a fan `F`. The output is a list of lists. which represent an incidence matrix for the correspondence of the rays and the maximal cones of the fan `F`. The i'th list in the result represents the i'th maximal cone of `F`. In such a list, the j'th entry is 1 if the j'th ray is in the cone, 0 otherwise.

5.2.4 MaximalCones (for IsFan)

▷ `MaximalCones(F)` (attribute)

Returns: a list

The input is a fan `F`. The output is a list of the maximal cones of `F`.

5.2.5 FVector (for IsFan)

▷ `FVector(F)` (attribute)

Returns: a list

Description

5.3 Properties

5.3.1 IsWellDefinedFan (for IsFan)

▷ `IsWellDefinedFan(F)` (property)

Returns: a true or false

It checks whether the constructed fan is well defined or not.

5.3.2 IsComplete (for IsFan)

▷ `IsComplete(F)` (property)

Returns: a true or false

Checks whether the fan is complete, i.e. if its support is the whole space.

5.3.3 IsPointed (for IsFan)

▷ `IsPointed(F)` (property)

Returns: a true or false

Checks whether the fan is pointed, i.e., that every cone it contains is pointed.

5.3.4 IsSmooth (for IsFan)

▷ `IsSmooth(F)` (property)

Returns: a true or false

Checks if the fan is smooth, i.e. if every cone in the fan is smooth.

5.3.5 IsSimplicial (for IsFan)

▷ `IsSimplicial(F)` (property)

Returns: a true or false

Checks if the fan is simplicial, i.e. if every cone in the fan is simplicial.

5.3.6 IsNormalFan (for IsFan)

▷ `IsNormalFan(F)` (property)

Returns: a true or false

Checks if the fan is normal as described in (Theorem 4.7, Combinatorial convexity and algebraic geometry, Ewald, Guenter).

5.3.7 IsRegularFan (for IsFan)

▷ `IsRegularFan(F)` (property)

Returns: a true or false

Synonyme to `IsNormalFan`

5.3.8 IsFanoFan (for IsFan)

▷ `IsFanoFan(F)` (property)

Returns: a true or false

Checks whether the fan is a fano fan.

5.4 Operations on fans

A star subdivision is a certain way of extending a fan. In toric geometry, its applications include blowups of varieties. The following examples correspond to blowups of the origin of the 2-dimensional and 3-dimensional affine space, respectively.

Example
```
gap> rays := [ [ 1,0 ], [ 0,1 ] ];;
gap> max_cones := [ [1,2] ];;
gap> fan_affine2 := Fan( rays, max_cones );;
gap> fan_blowup_affine2 := StarSubdivisionOfIthMaximalCone( fan_affine2, 1 );
<A fan in |R^2>
gap> Length( RaysInMaximalCones( fan_blowup_affine2 ) );
```

```
2
gap> rays := [ [ 1,0,0 ], [ 0,1,0 ], [0,0,1] ];;
gap> max_cones := [ [1,2,3] ];;
gap> fan_affine3 := Fan( rays, max_cones );;
gap> fan_blowup_affine3 := StarSubdivisionOfIthMaximalCone( fan_affine3, 1 );
<A fan in |R^3>
gap> Length( RaysInMaximalCones( fan_blowup_affine3 ) );
3
```

Chapter 6

Polyhedrons

6.1 Creating polyhedron

6.1.1 PolyhedronByInequalities (for IsList)

▷ `PolyhedronByInequalities(`*L*`)` (operation)

Returns: a `Polyhedron` Object

The function takes a list of lists `L`:$= [L_1, L_2, ...]$ where each L_j represents an inequality and returns the polyhedron defined by them. For example the j'th entry $L_j = [c_j, a_{j1}, a_{j2}, ..., a_{jn}]$ corresponds to the inequality $c_j + \sum_{i=1}^{n} a_{ji} x_i \geq 0$.

6.1.2 Polyhedron (for IsPolytope, IsCone)

▷ `Polyhedron(`*P*`, `*C*`)` (operation)

Returns: a `Polyhedron` Object

The input is a polytope `P` and a cone `C`. The output is the polyhedron defined by the Minkowski sum `P+C`.

6.1.3 Polyhedron (for IsList, IsCone)

▷ `Polyhedron(`*L*`, `*C*`)` (operation)

Returns: a `Polyhedron` Object

The input is a list `L` and a cone `C`. The output is the polyhedron defined by the Minkowski sum `P+C` where `P` is the polytope, i.e., the convex hull, defined by the points `L`.

6.1.4 Polyhedron (for IsPolytope, IsList)

▷ `Polyhedron(`*P*`, `*L*`)` (operation)

Returns: a `Polyhedron` Object

The input is a polytope `P` and a list `L`. The output is the polyhedron defined by the Minkowski sum `P+C` where `C` is the cone defined by the rays `L`.

6.1.5 Polyhedron (for IsList, IsList)

▷ `Polyhedron(`*P*`, `*C*`)` (operation)

Returns: a `Polyhedron` Object

The input is a list `P` and a list `C`. The output is the polyhedron defined by the Minkowski sum of the polytope defined by `P` and the cone defined by `C`.

6.2 Attributes

6.2.1 ExternalCddPolyhedron (for IsPolyhedron)

▷ `ExternalCddPolyhedron(`*P*`)` (attribute)

Returns: cdd Object

Converts the polyhedron to a cdd object. The operations of CddInterface can then be applied on this convex object.

6.2.2 ExternalNmzPolyhedron (for IsPolyhedron)

▷ `ExternalNmzPolyhedron(`*P*`)` (attribute)

Returns: normaliz Object

Converts the polyhedron to an normaliz object. The operations of NormalizInterface can then be applied on this convex object.

6.2.3 DefiningInequalities (for IsPolyhedron)

▷ `DefiningInequalities(`*P*`)` (attribute)

Returns: a list

Returns the Defining inequalities of the given polyhedron.

6.2.4 MainRatPolytope (for IsPolyhedron)

▷ `MainRatPolytope(`*P*`)` (attribute)

Returns: a `Polytope` Object

Returns the main rational polytope of the polyhedron.

6.2.5 MainPolytope (for IsPolyhedron)

▷ `MainPolytope(`*P*`)` (attribute)

Returns: a `Polytope` Object

Returns the main integral polytope of the given polyhedron.

6.2.6 VerticesOfMainRatPolytope (for IsPolyhedron)

▷ `VerticesOfMainRatPolytope(`*P*`)` (attribute)

Returns: a list

Returns the vertices of the main rational polytope of the polyhedron.

6.2.7 VerticesOfMainPolytope (for IsPolyhedron)

▷ `VerticesOfMainPolytope(`*P*`)` (attribute)

Returns: a list

Returns the vertices of the main integral polytope of the given polyhedron.

6.2.8 TailCone (for IsPolyhedron)

▷ `TailCone(P)` (attribute)

Returns: a `Cone Object`

Returns the tail cone of the polyhedron.

6.2.9 RayGeneratorsOfTailCone (for IsPolyhedron)

▷ `RayGeneratorsOfTailCone(P)` (attribute)

Returns: a list

Returns the Ray Generators of the tail cone.

6.2.10 LatticePointsGenerators (for IsPolyhedron)

▷ `LatticePointsGenerators(P)` (attribute)

Returns: a list

Returns the integral lattice generators of the polyhedron. The output is a list L of length 3. Any integral point in polyhedron can be written as $a+mb+nc$, where $a \in L[1], b \in L[2], c \in L[3], m \geq 0$.

6.2.11 BasisOfLinealitySpace (for IsPolyhedron)

▷ `BasisOfLinealitySpace(P)` (attribute)

Returns: a list

Returns a basis to the lineality space of the polyhedron. I.e., a basis to the vector space that is contained in `P`.

6.2.12 FVector (for IsPolyhedron)

▷ `FVector(P)` (attribute)

Returns: a list

Returns a list whose i'th entry is the number of faces of dimension $i-1$.

6.3 Properties

6.3.1 IsBounded (for IsPolyhedron)

▷ `IsBounded(P)` (property)

Returns: true or false

The input is a polyhedron P and the output is whether it is bounded or not.

6.3.2 IsPointed (for IsPolyhedron)

▷ `IsPointed(P)` (property)

Returns: true or false

The input is a polyhedron P and the output is whether its tail cone is pointed or not.

Example
```
gap> P := Polyhedron( [ [ 1, 1 ], [ 4, 7 ] ], [ [ 1, -1 ], [ 1, 1 ] ] );
<A polyhedron in |R^2>
gap> VerticesOfMainRatPolytope( P );
```

```
[ [ 1, 1 ], [ 4, 7 ] ]
gap> VerticesOfMainPolytope( P );
[ [ 1, 1 ], [ 4, 7 ] ]
gap> P := Polyhedron( [ [ 1/2, 1/2 ] ], [ [ 1, 1 ] ] );
<A polyhedron in |R^2>
gap> VerticesOfMainRatPolytope( P );
[ [ 1/2, 1/2 ] ]
gap> VerticesOfMainPolytope( P );
[ [ 1, 1 ] ]
gap> LatticePointsGenerators( P );
[ [ [ 1, 1 ] ], [ [ 1, 1 ] ], [  ] ]
gap> Dimension( P );
1
gap> Q := Polyhedron( [ [ 5, 0 ], [ 0, 6 ] ], [ [ 1, 2 ] , [ -1, -2 ] ] );
<A polyhedron in |R^2>
gap> VerticesOfMainRatPolytope( Q );
[ [ 0, 6 ], [ 5, 0 ] ]
gap> V := VerticesOfMainPolytope( Q );
[ [ 0, 6 ], [ 5, 0 ] ]
gap> L := LatticePointsGenerators( Q );
[ [ [ 0, -10 ], [ 0, -9 ], [ 0, -8 ], [ 0, -7 ], [ 0, -6 ],
[ 0, -5 ], [ 0, -4 ], [ 0, -3 ], [ 0, -2 ], [ 0, -1 ],
[ 0, 0 ], [ 0, 1 ], [ 0, 2 ], [ 0, 3 ], [ 0, 4 ],
[ 0, 5 ], [ 0, 6 ] ], [  ], [ [ 1, 2 ] ] ]
gap> Dimension( Q );
2
gap> RayGeneratorsOfTailCone( Q );
[ [ -1, -2 ], [ 1, 2 ] ]
gap> BasisOfLinealitySpace( Q );
[ [ 1, 2 ] ]
gap> DefiningInequalities( Q );
[ [ 6, 2, -1 ], [ 10, -2, 1 ] ]
gap> Q;
<A polyhedron in |R^2 of dimension 2>
```

Let us now find out if the equation $-2+3x+4y-7z=0$ has integer solutions.

Example

```
gap> P := PolyhedronByInequalities( [ [ -2, 3, 4, -7 ], -[ -2, 3, 4, -7 ] ] );
<A polyhedron in |R^3 >
gap> L := LatticePointsGenerators( P );
[ [ [ -4, 0, -2 ] ], [  ], [ [ 0, 7, 4 ], [ 1, 1, 1 ] ] ]
```

So the solutions set is $\{[-4,0,-2]+t_1*[1,1,1]+t_2*[0,7,4]; t_1,t_2 \in \mathbb{Z}\}$.
We know that $4x+6y=3$ does not have any solutions because $gcd(4,6)=2$ does not divide 3.

Example

```
gap> Q := PolyhedronByInequalities( [ [-3, 4, 6 ], [ 3, -4, -6 ] ] );
<A polyhedron in |R^2 >
gap> LatticePointsGenerators( Q );
[ [  ], [  ], [ [ 3, -2 ] ] ]
```

Let us solve the folowing linear system

$$2x+3y = 1 \bmod 2$$

$$7x+\ \ y = 3 \bmod 5.$$

which is equivalent to the sytem

$$-1+2x+3y+2u = 0$$

$$-3+7x+\ \ y+5v = 0$$

Example

```
gap> P := PolyhedronByInequalities( [ [ -1, 2, 3, 2, 0 ], [ -3, 7, 1, 0, 5 ],
>       [ 1, -2, -3, -2, 0 ], [ 3, -7, -1, 0, -5 ] ] );
<A polyhedron in |R^4 >
gap> L := LatticePointsGenerators( P );
[ [ [ -19, 1, 18, 27 ] ], [  ], [ [ 0, 10, -15, -2 ], [ 1, -2, 2, -1 ] ] ]
```

I.e., the solutions set is

$$\{[-19,1]+t_1*[1,-2]+t_2*[0,10];t_1,t_2 \in \mathbb{Z}\}$$

6.4 Solving Linear programs

The problem of solving linear programs can be solved in the gap package `CddInterface`, which is required by `NConvex`.

6.4.1 SolveLinearProgram (for IsPolyhedron, IsString, IsList)

▷ `SolveLinearProgram(P, max_or_min, target_func)` (operation)

Returns: a list or fail

The input is a polyhedron `P`, a string `max_or_min` ∈ {"max", "min"} and an objective function `target_func`, which we want to maximize or minimize. If the linear program has an optimal solution, the operation returns a list of two entries, the solution vector and the optimal value of the objective function, otherwise it returns fail.

6.4.2 SolveLinearProgram (for IsPolytope, IsString, IsList)

▷ `SolveLinearProgram(P, max_or_min, target_func)` (operation)

Returns: a list or fail

The input is a polytope `P`, a string `max_or_min` ∈ {"max","min"} and an objective function `target_func`, which we want to maximize or minimize. If the linear program has an optimal solution, the operation returns a list of two entries, the solution vector and the optimal value of the objective function, otherwise it returns fail.

To illustrate the using of this operation, let us solve the linear program:
$P(x,y) = 1-2x+5y$, with
$100 \leq x \leq 200$
$80 \leq y \leq 170$
$y \geq -x+200$

We bring the inequalities to the form $b+AX \geq 0$, we get:
$-100+x \geq 0$
$200-x \geq 0$
$-80+y \geq 0$
$170-y \geq 0$
$-200+x+y \geq 0$

Example
```
gap> P := PolyhedronByInequalities( [ [ -100, 1, 0 ], [ 200, -1, 0 ],
> [ -80, 0, 1 ], [ 170, 0, -1 ], [ -200, 1, 1 ] ] );;
gap> max := SolveLinearProgram( P, "max", [ 1, -2, 5 ] );
[ [ 100, 170 ], 651 ]
gap> min := SolveLinearProgram( P, "min", [ 1, -2, 5 ] );
[ [ 200, 80 ], 1 ]
gap> VerticesOfMainRatPolytope( P );
[ [ 100, 100 ], [ 100, 170 ], [ 120, 80 ], [ 200, 80 ], [ 200, 170 ] ]
```

So the optimal solutions are $(x = 100, y = 170)$ with maximal value $p = 1-2(100)+5(170) = 651$ and $(x = 200, y = 80)$ with minimal value $p = 1-2(200)+5(80) = 1$.

6.5 ZSolve

6.5.1 SolveEqualitiesAndInequalitiesOverIntergers

▷ `SolveEqualitiesAndInequalitiesOverIntergers(`*eqs, eqs_rhs, ineqs, ineqs_rhs[, signs]*`)` (function)

Returns: a list of three matrices

This function produces a basis of the system *eqs* = *eqs_rhs* and *ineqs* >= *ineqs_rhs*. It outputs a list containing three matrices. The first one is a list of points in a polytope, the second is the hilbert basis of a cone. The set of solutions is then the minkowski sum of the polytope generated by the points in the first list and the cone generated by the hilbert basis in the second matrix. The third one is the free part of the solution polyhedron. The optional argument *signs* must be a list of zeros and ones which length is the number of variables. If the ith entry is one, the ith variable must be >= 0. If the entry is 0, the number is arbitraty. Default is all zero.

To illustrate the using of this function, let us compute the integers solutions of the system:
$3x+5y = 8$
$4x-2y \geq 2$
$3x+y \geq 3$

Example
```
gap> SolveEqualitiesAndInequalitiesOverIntergers( [ [ 3, 5 ] ], [ 8 ],
> [ [ 4, -2 ], [ 3, 1 ] ], [ 2, 3 ] );
[ [ [ 1, 1 ] ], [ [ 5, -3 ] ], [  ] ]
gap> SolveEqualitiesAndInequalitiesOverIntergers( [ [ 3, 5 ] ], [ 8 ],
> [ [ 4, -2 ], [ 3, 1 ] ], [ 2, 3 ], [ 1, 1 ] );
[ [ [ 1, 1 ] ], [  ], [  ] ]
```

So the set of all solutions is given by $\{(1,1)+y*(5,-3), y \geq 0\}$ and it has only one positive solution $(1,1)$.

Chapter 7

Polytopes

7.1 Creating polytopes

7.1.1 PolytopeByInequalities (for IsList)

▷ `PolytopeByInequalities(L)` (operation)

Returns: a Polytope Object

The operation takes a list L of lists $[L_1, L_2, ...]$ where each L_j represents an inequality and returns the polytope defined by them (if they define a polytope). For example the j'th entry $L_j = [c_j, a_{j1}, a_{j2}, ..., a_{jn}]$ corresponds to the inequality $c_j + \sum_{i=1}^{n} a_{ji}x_i \geq 0$.

7.1.2 Polytope (for IsList)

▷ `Polytope(L)` (operation)

Returns: a Polytope Object

The operation takes the list of the vertices and returns the polytope defined by them.

7.2 Attributes

7.2.1 ExternalCddPolytope (for IsPolytope)

▷ `ExternalCddPolytope(P)` (attribute)

Returns: a CddPolyhedron

Converts the polytope to a CddPolyhedron. The operations of CddInterface can then be applied on this polyhedron.

7.2.2 LatticePoints (for IsPolytope)

▷ `LatticePoints(P)` (attribute)

Returns: a List

The operation returns the list of integer points inside the polytope.

7.2.3 RelativeInteriorLatticePoints (for IsPolytope)

▷ `RelativeInteriorLatticePoints(P)` (attribute)

Returns: a List

The operation returns the interior lattice points inside the polytope.

7.2.4 VerticesOfPolytope (for IsPolytope)

▷ `VerticesOfPolytope(P)` (attribute)

Returns: a list of lists

The operation returns the vertices of the polytope

7.2.5 Vertices (for IsPolytope)

▷ `Vertices(P)` (operation)

Returns: a list of lists

The same output as `VerticesOfPolytope`.

7.2.6 DefiningInequalities (for IsPolytope)

▷ `DefiningInequalities(P)` (attribute)

Returns: a list of lists

The operation returns the defining inequalities of the polytope. I.e., a list of lists $[L_1, L_2, ...]$ where each $L_j = [c_j, a_{j1}, a_{j2}, ..., a_{jn}]$ represents the inequality $c_j + \sum_{i=1}^{n} a_{ji} x_i \geq 0$. If L and $-L$ occur in the output then L is called a defining-equality of the polytope.

7.2.7 EqualitiesOfPolytope (for IsPolytope)

▷ `EqualitiesOfPolytope(P)` (attribute)

Returns: a list of lists

The operation returns the defining-equalities of the polytope.

7.2.8 FacetInequalities (for IsPolytope)

▷ `FacetInequalities(P)` (attribute)

Returns: a list of lists

The operation returns the list of the inequalities of the facets. Each defining inequality that is not defining-equality of the polytope is a facet inequality.

7.2.9 VerticesInFacets (for IsPolytope)

▷ `VerticesInFacets(P)` (attribute)

Returns: a list of lists

The operation returns list of lists L. The entries of each L_j in L consists of 0's or 1's. For instance, if $L_j = [1,0,0,1,0,1]$, then The polytope has 6 vertices and the vertices of the j'th facet are $\{V_1, V_4, V_6\}$.

7.2.10 NormalFan (for IsPolytope)

▷ `NormalFan(P)` (attribute)

Returns: a fan

The operation returns the normal fan of the given polytope.

7.2.11 FaceFan (for IsPolytope)

▷ `FaceFan(`*P*`)` (attribute)

Returns: a fan

The operation returns the face fan of the given polytope. Remember that the face fan of a polytope is isomorphic to the normal fan of its polar polytope.

7.2.12 AffineCone (for IsPolytope)

▷ `AffineCone(`*P*`)` (attribute)

Returns: a cone

If the ambient space of the polytope is R^n, then the output is a cone in R^{n+1}. The defining rays of the cone are $[a_{j1}, a_{j2}, ..., a_{jn}, 1]_j$ such that $V_j = [a_{j1}, a_{j2}, ..., a_{jn}]$ is a vertex in the polytope.

7.2.13 PolarPolytope (for IsPolytope)

▷ `PolarPolytope(`*P*`)` (attribute)

Returns: a Polytope

The operation returns the polar polytope of the given polytope.

7.2.14 DualPolytope (for IsPolytope)

▷ `DualPolytope(`*P*`)` (attribute)

Returns: a Polytope

The operation returns the dual polytope of the given polytope.

7.3 Properties

7.3.1 IsEmpty (for IsPolytope)

▷ `IsEmpty(`*P*`)` (property)

Returns: a true or false

Returns whether the polytope empty or not

7.3.2 IsLatticePolytope (for IsPolytope)

▷ `IsLatticePolytope(`*P*`)` (property)

Returns: a true or false

Returns whether the polytope is lattice polytope or not.

7.3.3 IsVeryAmple (for IsPolytope)

▷ `IsVeryAmple(`*P*`)` (property)

Returns: a true or false

Returns whether the polytope is very ample or not.

7.3.4 IsNormalPolytope (for IsPolytope)

▷ `IsNormalPolytope(P)` (property)

Returns: a true or false

Returns whether the polytope is normal or not.

7.3.5 IsSimplicial (for IsPolytope)

▷ `IsSimplicial(P)` (property)

Returns: a true or false

Returns whether the polytope is simplicial or not.

7.3.6 IsSimplexPolytope (for IsPolytope)

▷ `IsSimplexPolytope(P)` (property)

Returns: a true or false

Returns whether the polytope is simplex polytope or not.

7.3.7 IsSimplePolytope (for IsPolytope)

▷ `IsSimplePolytope(P)` (property)

Returns: a true or false

Returns whether the polytope is simple or not.

7.3.8 IsReflexive (for IsPolytope)

▷ `IsReflexive(P)` (property)

Returns: a true or false

Returns whether the polytope is reflexive or not, i.e., if its dual polytope is lattice polytope.

7.3.9 IsFanoPolytope (for IsPolytope)

▷ `IsFanoPolytope(P)` (property)

Returns: a true or false

returns whether the polytope is Fano or not. Fano polytope is a full dimensional lattice polytope whose vertices are primitive elements in the containing lattice, i.e., each vertex is not a positive integer multiple of any other lattice element.

7.3.10 IsCanonicalFanoPolytope (for IsPolytope)

▷ `IsCanonicalFanoPolytope(P)` (property)

Returns: a true or false

returns whether the polytope is canonical Fano or not. A canonical Fano polytope is a full dimensional lattice polytope whose relative interior contains only one lattice point, namely the origin.

7.3.11 IsTerminalFanoPolytope (for IsPolytope)

▷ `IsTerminalFanoPolytope(P)` (property)

Returns: a true or false

returns whether the polytope is terminal Fano or not. A terminal Fano polytope is a full dimensional lattice polytope whose lattice points are its vertices and the origin.

7.3.12 IsSmoothFanoPolytope (for IsPolytope)

▷ `IsSmoothFanoPolytope(P)` (property)

Returns: a true or false

Returns whether the polytope is smooth fano polytope or not, i.e, if the vertices in each facet form a basis for the containing lattice or not. polytope.

7.4 Operations on polytopes

7.4.1 \+ (for IsPolytope, IsPolytope)

▷ `\+(P1, P2)` (operation)

Returns: a polytope

The output is Minkowski sum of the input polytopes.

7.4.2 * (for IsInt, IsPolytope)

▷ `\*(n, P)` (operation)

Returns: a polytope

The output is Minkowski sum of the input polytope with itself *n* times.

7.4.3 IntersectionOfPolytopes (for IsPolytope, IsPolytope)

▷ `IntersectionOfPolytopes(P1, P2)` (operation)

Returns: a polytope

The output is the intersection of the input polytopes.

7.4.4 RandomInteriorPoint (for IsPolytope)

▷ `RandomInteriorPoint(P)` (operation)

Returns: a list

Returns a random interior point in the polytope.

7.4.5 IsInteriorPoint (for IsList,IsPolytope)

▷ `IsInteriorPoint(M, P)` (operation)

Returns: true or false

Checks if the given point is interior point of the polytope.

Example
```
gap> P:= Polytope( [ [ 0, 0, 0 ], [ 1, 0, 0 ], [ 0, 1, 0 ], [ 1, 1, 2 ] ] );
<A polytope in |R^3>
gap> IsNormalPolytope( P );
```

```
false
gap> IsVeryAmple( P );
false
gap> Q:= Polytope( [ [ 0, 0, 0 ], [ 1, 0, 0 ], [ 0, 1, 0 ], [ 1, 1, 1 ] ] );
<A polytope in |R^3>
gap> IsNormalPolytope( Q );
true
gap> IsVeryAmple( Q );
true
gap> Q;
<A normal very ample polytope in |R^3 with 4 vertices>
gap> T:= Polytope( [ [ 0, 0, 0 ], [ 1, 0, 0 ], [ 0, 1, 0 ], [ 1, 1, 4 ] ] );
<A polytope in |R^3>
gap> I:= Polytope( [ [ 0, 0, 0 ], [ 0, 0, 1 ] ] );
<A polytope in |R^3>
gap> J:= T + I;
<A polytope in |R^3>
gap> IsVeryAmple( J );
true
gap> IsNormalPolytope( J );
false
gap> J;
<A very ample polytope in |R^3 with 8 vertices>
gap> # Example 2.2.20 Cox, Toric Varieties
> A:= [ [1,1,1,0,0,0], [1,1,0,1,0,0], [1,0,1,0,1,0], [ 1,0,0,1,0,1],
> [ 1,0,0,0,1,1], [ 0,1,1,0,0,1], [0,1,0,1,1,0], [0,1,0,0,1,1],
> [0,0,1,1,1,0], [0,0,1,1,0,1] ];
[ [ 1, 1, 1, 0, 0, 0 ], [ 1, 1, 0, 1, 0, 0 ], [ 1, 0, 1, 0, 1, 0 ],
[ 1, 0, 0, 1, 0, 1 ], [ 1, 0, 0, 0, 1, 1 ], [ 0, 1, 1, 0, 0, 1 ],
 [ 0, 1, 0, 1, 1, 0 ], [ 0, 1, 0, 0, 1, 1 ], [ 0, 0, 1, 1, 1, 0 ],
[ 0, 0, 1, 1, 0, 1 ] ]
gap> H:= Polytope( A );
<A polytope in |R^6>
gap> IsVeryAmple( H );
true
gap> IsNormalPolytope( H );
false
gap> H;
<A very ample polytope in |R^6 with 10 vertices>
gap> l:= [ [ 0, 0, 1 ], [ 0, 0, 0 ], [ 1, 0, 0 ], [ 1, 0, 1 ], [ 0, 1, 0 ],
> [ 0, 1, 1 ], [ 1, 1, 4 ], [ 1, 1, 5 ] ];;
gap> P:= Polytope( l );
<A polytope in |R^3>
gap> IsNormalPolytope( P );
false
gap> lattic_points:= LatticePoints( P );
[ [ 0, 0, 0 ], [ 0, 0, 1 ], [ 0, 1, 0 ], [ 0, 1, 1 ], [ 1, 0, 0 ], [ 1, 0, 1 ],
[ 1, 1, 4 ], [ 1, 1, 5 ] ]
gap> u:= Cartesian( lattic_points, lattic_points );;
gap> k:= Set( List( u, u-> u[1]+u[2] ) );
[ [ 0, 0, 0 ], [ 0, 0, 1 ], [ 0, 0, 2 ], [ 0, 1, 0 ], [ 0, 1, 1 ], [ 0, 1, 2 ],
[ 0, 2, 0 ], [ 0, 2, 1 ], [ 0, 2, 2 ], [ 1, 0, 0 ], [ 1, 0, 1 ], [ 1, 0, 2 ],
[ 1, 1, 0 ], [ 1, 1, 1 ], [ 1, 1, 2 ], [ 1, 1, 4 ], [ 1, 1, 5 ], [ 1, 1, 6 ],
[ 1, 2, 4 ], [ 1, 2, 5 ], [ 1, 2, 6 ], [ 2, 0, 0 ], [ 2, 0, 1 ], [ 2, 0, 2 ],
[ 2, 1, 4 ], [ 2, 1, 5 ], [ 2, 1, 6 ], [ 2, 2, 8 ], [ 2, 2, 9 ], [ 2, 2, 10 ] ]
gap> Q:= 2*P;
<A polytope in |R^3 with 8 vertices>
gap> LatticePoints( Q );
[ [ 0, 0, 0 ], [ 0, 0, 1 ], [ 0, 0, 2 ], [ 0, 1, 0 ], [ 0, 1, 1 ], [ 0, 1, 2 ],
[ 0, 2, 0 ], [ 0, 2, 1 ], [ 0, 2, 2 ], [ 1, 0, 0 ],
  [ 1, 0, 1 ], [ 1, 0, 2 ], [ 1, 1, 0 ], [ 1, 1, 1 ], [ 1, 1, 2 ], [ 1, 1, 3 ],
[ 1, 1, 4 ], [ 1, 1, 5 ], [ 1, 1, 6 ], [ 1, 2, 4 ], [ 1, 2, 5 ], [ 1, 2, 6 ],
[ 2, 0, 0 ], [ 2, 0, 1 ], [ 2, 0, 2 ], [ 2, 1, 4 ],
  [ 2, 1, 5 ], [ 2, 1, 6 ], [ 2, 2, 8 ], [ 2, 2, 9 ], [ 2, 2, 10 ] ]
gap> P:= Polytope( [ [ 1, 1 ], [ 1, -1 ], [ -1, 1 ], [ -1, -1 ] ] );
<A polytope in |R^2>
gap> Q:= PolarPolytope( P );
<A polytope in |R^2>
gap> Vertices( Q );
[ [ -1, 0 ], [ 0, -1 ], [ 0, 1 ], [ 1, 0 ] ]
gap> T := PolarPolytope( Q );
<A polytope in |R^2>
gap> Vertices( T );
[ [ -1, -1 ], [ -1, 1 ], [ 1, -1 ], [ 1, 1 ] ]
gap> P:= Polytope( [ [ 0, 0 ], [ 1, -1], [ -1, 1 ], [ -1, -1 ] ] );
<A polytope in |R^2>
gap> # PolarPolytope( P );;
```

Let us now find out if the vertices of the polytope defined by the following inequalities:

$$x_2 \geq 0, 1 - x_1 - x_2 \geq 0, 1 + x_1 - x_2 \geq 0.$$

Example

```
gap> P := PolytopeByInequalities( [ [ 0, 0, 1 ], [ 1, -1, -1 ], [ 1, 1, -1 ] ] );
<A polytope in |R^2>
gap> Vertices( P );
[ [ -1, 0 ], [ 0, 1 ], [ 1, 0 ] ]
```

Index