NormalizInterface

GAP wrapper for Normaliz

0.9.8

07/05/2016

Sebastian Gutsche

Max Horn

Christof Söger

Sebastian Gutsche
Email: gutsche@mathematik.uni-kl.de
Homepage: http://wwwb.math.rwth-aachen.de/~gutsche/
Address: Department of Mathematics
University of Kaiserslautern
67653 Kaiserslautern
Germany

Max Horn
Email: max.horn@math.uni-giessen.de
Homepage: http://www.quendi.de/math
Address: AG Algebra
Mathematisches Institut
Justus-Liebig-Universität Gießen
Arndtstraße 2
35392 Gießen
Germany

Christof Söger
Email: csoeger@uos.de
Homepage: https://www.normaliz.uni-osnabrueck.de
Address: Institut für Mathematik
Albrechtstr. 28a
49076 Osnabrück
Germany

Contents

Chapter 1

Introduction

1.1 What is the purpose of the this package?

Normaliz [BIRS14] is a software for computations with rational cones and affine monoids. It pursues two main computational goals: finding the Hilbert basis, a minimal generating system of the monoid of lattice points of a cone; and counting elements degree-wise in a generating function, the Hilbert series. As a recent extension, Normaliz can handle unbounded polyhedra. The Hilbert basis computation can be considered as solving a linear diophantine system of inhomogeneous equations, inequalities and congruences.

This package encapsulates a libnormaliz cone and gives access to it in the GAP environment. In this way GAP can be used as interactive interface to libnormaliz.

Chapter 2

Functions

In this chapter we describe the functions offered by *NormalizInterface*. All functions supplied by this package start with "Nmz". For examples see the chapter 'Examples'.

2.1 Create a NmzCone

To create a cone object use *NmzCone*.

2.1.1 NmzCone

▷ NmzCone(*list*) (function)

Returns: NmzCone

Creates a NmzCone. The *list* argument should contain an even number of elements, alternating between a string and a integer matrix. The string has to correspond to a Normaliz input type string and the following matrix will be interpreted as input of that type. Currently the following strings are recognized:

- integral_closure,
- polyhedron,
- normalization,
- polytope,
- rees_algebra,
- inequalities,
- strict_inequalities,
- signs,
- strict_signs,
- equations,
- congruences,

- inhom_inequalities,
- inhom_equations,
- inhom_congruences,
- dehomogenization,
- lattice_ideal,
- grading,
- excluded_faces,
- lattice,
- saturation,
- cone,
- offset,
- vertices,
- support_hyperplanes,
- cone_and_lattice,
- subspace.

See the Normaliz manual for a detailed description.

Example

```
gap> cone := NmzCone(["integral_closure",[[2,1],[1,3]]]);
<a Normaliz cone>
```

2.2 Use a NmzCone

2.2.1 NmzHasConeProperty

▷ NmzHasConeProperty(*cone, property*) (function)

Returns: whether the cone has already computed the given property

See NmzConeProperty (2.2.6) for a list of recognized properties.

Example

```
gap> NmzHasConeProperty(cone, "ExtremeRays");
false
```

2.2.2 NmzKnownConeProperties

▷ NmzKnownConeProperties(*cone*) (function)

Returns: a list of strings representing the known (computed) cone properties

Given a Normaliz cone object, return a list of all properties already computed for the cone.

Example

```
gap> NmzKnownConeProperties(cone);
[ "Generators", "OriginalMonoidGenerators", "Sublattice" ]
```

2.2.3 NmzSetVerboseDefault

▷ `NmzSetVerboseDefault(`*verboseFlag*`)` (function)

Returns: the previous verbosity

Set the global default verbosity state in libnormaliz. This will influence all NmzCone created afterwards, but not any existing ones. See also `NmzSetVerbose` (2.2.4)

2.2.4 NmzSetVerbose

▷ `NmzSetVerbose(`*cone, verboseFlag*`)` (function)

Returns: the previous verbosity

Set the verbosity state for a cone. See also `NmzSetVerboseDefault` (2.2.3)

2.2.5 NmzCompute

▷ `NmzCompute(`*cone[, propnames]*`)` (function)

Returns: a boolean indicating success

Start computing properties of the given cone. The first parameter indicates a cone object, the second parameter is either a single string, or a list of strings, which indicate what should be computed.

The single parameter version is equivalent to `NmzCone(cone, ["DefaultMode"])`. See `NmzConeProperty` (2.2.6) for a list of recognized properties.

Example

```
gap> NmzKnownConeProperties(cone);
[ "Generators", "OriginalMonoidGenerators", "Sublattice" ]
gap> NmzCompute(cone, ["SupportHyperplanes", "IsPointed"]);
true
gap> NmzKnownConeProperties(cone);
[ "Generators", "ExtremeRays", "SupportHyperplanes", "IsPointed",
  "IsDeg1ExtremeRays", "OriginalMonoidGenerators", "Sublattice",
  "MaximalSubspace" ]
gap> NmzCompute(cone);
true
gap> NmzKnownConeProperties(cone);
[ "Generators", "ExtremeRays", "SupportHyperplanes", "TriangulationSize",
  "TriangulationDetSum", "HilbertBasis", "IsPointed", "IsDeg1ExtremeRays",
  "IsIntegrallyClosed", "OriginalMonoidGenerators", "Sublattice",
  "ClassGroup", "MaximalSubspace"]
```

2.2.6 NmzConeProperty

▷ `NmzConeProperty(`*cone, property*`)` (function)

Returns: the result of the computation, type depends on the property

Triggers the computation of the property of the cone and returns the result. If the property was already known, it is not recomputed. Currently the following strings are recognized as properties:

- `Generators` see `NmzGenerators` (2.3.12),
- `ExtremeRays` see `NmzExtremeRays` (2.3.11),
- `VerticesOfPolyhedron` see `NmzVerticesOfPolyhedron` (2.3.35),

- SupportHyperplanes see NmzSupportHyperplanes (2.3.31),
- TriangulationSize see NmzTriangulationSize (2.3.34),
- TriangulationDetSum see NmzTriangulationDetSum (2.3.33),
- Triangulation see NmzTriangulation (2.3.32),
- Multiplicity see NmzMultiplicity (2.3.26),
- RecessionRank see NmzRecessionRank (2.3.28),
- AffineDim see NmzAffineDim (2.3.6),
- ModuleRank see NmzModuleRank (2.3.25),
- HilbertBasis see NmzHilbertBasis (2.3.14),
- ModuleGenerators see NmzModuleGenerators (2.3.23),
- Deg1Elements see NmzDeg1Elements (2.3.8),
- HilbertSeries see NmzHilbertSeries (2.3.16),
- HilbertQuasiPolynomial see NmzHilbertQuasiPolynomial (2.3.15),
- Grading see NmzGrading (2.3.13),
- IsPointed see NmzIsPointed (2.3.21),
- IsDeg1ExtremeRays see NmzIsDeg1ExtremeRays (2.3.18),
- IsDeg1HilbertBasis see NmzIsDeg1HilbertBasis (2.3.19),
- IsIntegrallyClosed see NmzIsIntegrallyClosed (2.3.20),
- OriginalMonoidGenerators see NmzOriginalMonoidGenerators (2.3.27),
- IsReesPrimary see NmzIsReesPrimary (2.3.29),
- ReesPrimaryMultiplicity see NmzReesPrimaryMultiplicity (2.3.30),
- ExcludedFaces see NmzExcludedFaces (2.3.10),
- Dehomogenization see NmzDehomogenization (2.3.9),
- InclusionExclusionData see NmzInclusionExclusionData (2.3.17),
- ClassGroup see NmzClassGroup (2.3.7),
- ModuleGeneratorsOverOriginalMonoid see NmzModuleGeneratorsOverOriginalMonoid (2.3.24),
- Sublattice computes the efficient sublattice and returns a bool signaling whether the computation was successful. Actual data connected to it can be accessed by NmzRank (2.3.2), NmzEquations (2.3.4), NmzCongruences (2.3.5), and NmzBasisChange (2.3.36).

Additionally also the following compute options are accepted as property. They modify what and how should be computed, and return True after a successful computation.

- `Approximate` approximate the rational polytope by an integral polytope, currently only useful in combination with `Deg1Elements`.
- `BottomDecomposition` use the best possible triangulation (with respect to the sum of determinants) using the given generators.
- `DefaultMode` try to compute what is possible and do not throw an exception when something cannot be computed.
- `DualMode`activates the dual algorithm for the computation of the Hilbert basis and degree 1 elements. Includes `HilbertBasis`, unless `Deg1Elements` is set. Often a good choice if you start from constraints.
- `KeepOrder` forbids to reorder the generators. Blocks `BottomDecomposition`.

All the properties above can be given to `NmzCompute` (2.2.5). There you can combine different properties, e.g. give some properties that you would like to know and add some compute options.

See the Normaliz manual for a detailed description.

2.2.7 NmzPrintConeProperties

▷ `NmzPrintConeProperties(`*cone*`)` (function)

Returns:

Print an overview of all known properties of the given cone, as well as their values.

2.3 Cone properties

2.3.1 NmzEmbeddingDimension

▷ `NmzEmbeddingDimension(`*cone*`)` (function)

Returns: the embedding dimension of the cone

The embedding dimension is the dimension of the space in which the computation is done. It is the number of components of the output vectors. This value is always known directly after the creation of the cone.

2.3.2 NmzRank

▷ `NmzRank(`*cone*`)` (function)

Returns: the rank of the cone

The rank is the rank of the lattice generated by the lattice points of the cone.

This is part of the cone property “Sublattice”.

2.3.3 NmzIsInhomogeneous

▷ `NmzIsInhomogeneous(`*cone*`)` (function)

Returns: whether the cone is inhomogeneous

This value is always known directly after the creation of the cone.

2.3.4 NmzEquations

▷ `NmzEquations(cone)` (function)

Returns: a matrix whose rows represent the equations

The equations cut out the linear space generated by the cone. Together with the support hyperplanes and the congruences it describes the lattice points of the cone.

This is part of the cone property "Sublattice".

2.3.5 NmzCongruences

▷ `NmzCongruences(cone)` (function)

Returns: a matrix whose rows represent the congruences

Together with the support hyperplanes and the equations it describes the lattice points of the cone.

This is part of the cone property "Sublattice".

2.3.6 NmzAffineDim

▷ `NmzAffineDim(cone)` (function)

Returns: the affine dimension

The affine dimension of the polyhedron in inhomogeneous computations. Its computation is triggered if necessary.

This is an alias for `NmzConeProperty( cone, "AffineDim" );` see `NmzConeProperty` (2.2.6).

2.3.7 NmzClassGroup

▷ `NmzClassGroup(cone)` (function)

Returns: the class group in a special format

A normal affine monoid M has a well-defined divisor class group. It is naturally isomorphic to the divisor class group of $K[M]$ where K is a field (or any unique factorization domain). We represent it as a vector where the first entry is the rank. It is followed by sequence of pairs of entries n,m. Such two entries represent a free cyclic summand $(\mathbb{Z}/n\mathbb{Z})^m$. Not allowed in inhomogeneous computations.

This is an alias for `NmzConeProperty( cone, "ClassGroup" );` see `NmzConeProperty` (2.2.6).

2.3.8 NmzDeg1Elements

▷ `NmzDeg1Elements(cone)` (function)

Returns: a matrix whose rows are the degree 1 elements

Requires the presence of a grading. Not allowed in inhomogeneous computations.

This is an alias for `NmzConeProperty( cone, "Deg1Elements" );` see `NmzConeProperty` (2.2.6).

2.3.9 NmzDehomogenization

▷ `NmzDehomogenization(cone)` (function)

Returns: the dehomgenization vector

Only for inhomogeneous computations.

This is an alias for `NmzConeProperty( cone, "Dehomogenization" );` see `NmzConeProperty` (2.2.6).

2.3.10 NmzExcludedFaces

▷ `NmzExcludedFaces(`*cone*`)` (function)

Returns: a matrix whose rows represent the excluded faces

This is an alias for `NmzConeProperty( cone, "ExcludedFaces" );` see `NmzConeProperty` (2.2.6).

2.3.11 NmzExtremeRays

▷ `NmzExtremeRays(`*cone*`)` (function)

Returns: a matrix whose rows are the extreme rays

This is an alias for `NmzConeProperty( cone, "ExtremeRays" );` see `NmzConeProperty` (2.2.6).

2.3.12 NmzGenerators

▷ `NmzGenerators(`*cone*`)` (function)

Returns: a matrix whose rows are the generators

This is an alias for `NmzConeProperty( cone, "Generators" );` see `NmzConeProperty` (2.2.6).

2.3.13 NmzGrading

▷ `NmzGrading(`*cone*`)` (function)

Returns: the grading vector

This is an alias for `NmzConeProperty( cone, "Grading" );` see `NmzConeProperty` (2.2.6).

2.3.14 NmzHilbertBasis

▷ `NmzHilbertBasis(`*cone*`)` (function)

Returns: a matrix whose rows are the Hilbert basis elements

This is an alias for `NmzConeProperty( cone, "HilbertBasis" );` see `NmzConeProperty` (2.2.6).

2.3.15 NmzHilbertQuasiPolynomial

▷ `NmzHilbertQuasiPolynomial(`*cone*`)` (function)

Returns: the Hilbert function as a quasipolynomial

The Hilbert function counts the lattice points degreewise. The result is a quasipolynomial Q, that is, a polynomial with periodic coefficients. It is given as list of polynomials $P_0,\ldots,P_{(p-1)}$ such that $Q(i) = P_{(i \bmod p)}(i)$.

This is an alias for `NmzConeProperty( cone, "HilbertQuasiPolynomial" );` see `NmzConeProperty` (2.2.6).

2.3.16 NmzHilbertSeries

▷ `NmzHilbertSeries(`*cone*`)` (function)

Returns: the Hilbert series as rational function

The result consists of a list with two entries. The first is the numerator polynomial. In inhomogeneous computations this can also be a Laurent polynomial. The second list entry represents the denominator. It is a list of pairs $[k_i, l_i]$. Such a pair represents the factor $(1-t^{k_i})^{l_i}$.

This is an alias for `NmzConeProperty( cone, "HilbertSeries" );` see `NmzConeProperty` (2.2.6).

2.3.17 NmzInclusionExclusionData

▷ `NmzInclusionExclusionData(`*cone*`)` (function)

Returns: inclusion-exclusion data

List of faces which are internally have been used in the inclusion-exclusion scheme. Given as a list pairs. The first pair entry is a key of generators contained in the face (compare also `NmzTriangulation` (2.3.32)) and the multiplicity with which it was considered. Only available with excluded faces or strict constraints as input.

This is an alias for `NmzConeProperty( cone, "InclusionExclusionData" );` see `NmzConeProperty` (2.2.6).

2.3.18 NmzIsDeg1ExtremeRays

▷ `NmzIsDeg1ExtremeRays(`*cone*`)` (function)

Returns: `true` if all extreme rays have degree 1; `false` otherwise

This is an alias for `NmzConeProperty( cone, "IsDeg1ExtremeRays" );` see `NmzConeProperty` (2.2.6).

2.3.19 NmzIsDeg1HilbertBasis

▷ `NmzIsDeg1HilbertBasis(`*cone*`)` (function)

Returns: `true` if all Hilbert basis elements have degree 1; `false` otherwise

This is an alias for `NmzConeProperty( cone, "IsDeg1HilbertBasis" );` see `NmzConeProperty` (2.2.6).

2.3.20 NmzIsIntegrallyClosed

▷ `NmzIsIntegrallyClosed(`*cone*`)` (function)

Returns: `true` if the cone is integrally closed; `false` otherwise

It is integrally closed when the Hilbert basis is a subset of the original monoid generators. So it is only computable if we have original monoid generators.

This is an alias for `NmzConeProperty( cone, "IsIntegrallyClosed" );` see `NmzConeProperty` (2.2.6).

2.3.21 NmzIsPointed

▷ `NmzIsPointed(`*cone*`)` (function)

Returns: `true` if the cone is pointed; `false` otherwise

This is an alias for `NmzConeProperty( cone, "IsPointed" );` see `NmzConeProperty` (2.2.6).

2.3.22 NmzMaximalSubspace

▷ `NmzMaximalSubspace(`*cone*`)` (function)

Returns: a matrix whose rows generate the maximale linear subspace

This is an alias for `NmzConeProperty( cone, "MaximalSubspace" );` see `NmzConeProperty` (2.2.6).

2.3.23 NmzModuleGenerators

▷ `NmzModuleGenerators(`*cone*`)` (function)

Returns: a matrix whose rows are the module generators

This is an alias for `NmzConeProperty( cone, "ModuleGenerators" );` see `NmzConeProperty` (2.2.6).

2.3.24 NmzModuleGeneratorsOverOriginalMonoid

▷ `NmzModuleGeneratorsOverOriginalMonoid(`*cone*`)` (function)

Returns: a matrix whose rows are the module generators over the original monoid

A minimal system of generators of the integral closure over the original monoid. Requires the existence of original monoid generators. Not allowed in inhomogeneous computations.

This is an alias for `NmzConeProperty( cone, "ModuleGeneratorsOverOriginalMonoid" );` see `NmzConeProperty` (2.2.6).

2.3.25 NmzModuleRank

▷ `NmzModuleRank(`*cone*`)` (function)

Returns: the rank of the module of lattice points in the polyhedron as a module over the recession monoid

Only for inhomogeneous computations.

This is an alias for `NmzConeProperty( cone, "ModuleRank" );` see `NmzConeProperty` (2.2.6).

2.3.26 NmzMultiplicity

▷ `NmzMultiplicity(`*cone*`)` (function)

Returns:

This is an alias for `NmzConeProperty( cone, "Multiplicity" );` see `NmzConeProperty` (2.2.6).

2.3.27 NmzOriginalMonoidGenerators

▷ `NmzOriginalMonoidGenerators(`*cone*`)` (function)

Returns: a matrix whose rows are the original monoid generators

This is an alias for `NmzConeProperty( cone, "OriginalMonoidGenerators" );` see `NmzConeProperty` (2.2.6).

2.3.28 NmzRecessionRank

▷ `NmzRecessionRank(`*cone*`)` (function)

Returns: the rank of the recession cone

Only for inhomogeneous computations.

This is an alias for `NmzConeProperty( cone, "RecessionRank" );` see `NmzConeProperty` (2.2.6).

2.3.29 NmzIsReesPrimary

▷ `NmzIsReesPrimary(`*cone*`)` (function)

Returns: `true` if is the monomial ideal is primary to the irrelevant maximal ideal, `false` otherwise

Only used with the input type `rees_algebra`.

This is an alias for `NmzConeProperty( cone, "IsReesPrimary" );` see `NmzConeProperty` (2.2.6).

2.3.30 NmzReesPrimaryMultiplicity

▷ `NmzReesPrimaryMultiplicity(`*cone*`)` (function)

Returns:

the multiplicity of a monomial ideal, provided it is primary to the maximal ideal generated by the indeterminates. Used only with the input type `rees_algebra`.

This is an alias for `NmzConeProperty( cone, "ReesPrimaryMultiplicity" );` see `NmzConeProperty` (2.2.6).

2.3.31 NmzSupportHyperplanes

▷ `NmzSupportHyperplanes(`*cone*`)` (function)

Returns: a matrix whose rows represent the support hyperplanes

The equations cut out the linear space generated by the cone. Together with the support hyperplanes and the congruences it describes the lattice points of the cone.

This is an alias for `NmzConeProperty( cone, "SupportHyperplanes" );` see `NmzConeProperty` (2.2.6).

2.3.32 NmzTriangulation

▷ `NmzTriangulation(`*cone*`)` (function)

Returns: the triangulation

It is given as a list of pairs representing the maximal simplicial cones in the triangulation. The first pair entry is the key of the simplex, i.e. the indices of the generators with respect th the generators obtained by `NmzGenerators` (2.3.12) (counting from 0). The second pair entry is the absolute value of the determinant of the generator matrix.

This is an alias for `NmzConeProperty( cone, "Triangulation" );` see `NmzConeProperty` (2.2.6).

2.3.33 NmzTriangulationDetSum

▷ `NmzTriangulationDetSum(`*cone*`)` (function)

Returns: sum of the absolute values of the determinants of the simplicial cones in the used triangulation

This is an alias for `NmzConeProperty( cone, "TriangulationDetSum" );` see `NmzConeProperty` (2.2.6).

2.3.34 NmzTriangulationSize

▷ `NmzTriangulationSize(`*cone*`)` (function)

Returns: the number of simplicial cones in the used triangulation

This is an alias for `NmzConeProperty( cone, "TriangulationSize" );` see `NmzConeProperty` (2.2.6).

2.3.35 NmzVerticesOfPolyhedron

▷ `NmzVerticesOfPolyhedron(`*cone*`)` (function)

Returns: a matrix whose rows are the vertices of the polyhedron

This is an alias for `NmzConeProperty( cone, "VerticesOfPolyhedron" );` see `NmzConeProperty` (2.2.6).

2.3.36 NmzBasisChange

▷ `NmzBasisChange(`*cone*`)` (function)

Returns: a record describing the basis change

The result record `r` has three components: `r.Embedding`, `r.Projection`, and `r.Annihilator`, where the embedding `A` and the projection `B` are matrices, and the annihilator `c` is an integer. They represent the mapping into the effective lattice $\mathbb{Z}^n \to \mathbb{Z}^r, u \mapsto (uB)/c$ and the inverse operation $\mathbb{Z}^r \to \mathbb{Z}^n, v \mapsto vA$.

This is part of the cone property “Sublattice”.

Chapter 3

Examples

3.1 Generators

Example

```
gap> C := NmzCone(["integral_closure",[[2,1],[1,3]]]);
<a Normaliz cone>
gap> NmzHasConeProperty(C,"HilbertBasis");
false
gap> NmzHasConeProperty(C,"SupportHyperplanes");
false
gap> NmzConeProperty(C,"HilbertBasis");
[ [ 1, 1 ], [ 1, 2 ], [ 1, 3 ], [ 2, 1 ] ]
gap> NmzHasConeProperty(C,"SupportHyperplanes");
true
gap> NmzConeProperty(C,"SupportHyperplanes");
[ [ -1, 2 ], [ 3, -1 ] ]
```

3.2 System of equations

Example

```
gap> D := NmzCone(["equations",[[1,2,-3]], "grading",[[0,-1,3]]]);
<a Normaliz cone>
gap> NmzCompute(D,["DualMode","HilbertSeries"]);
true
gap> NmzHilbertBasis(D);
[ [ 1, 1, 1 ], [ 0, 3, 2 ], [ 3, 0, 1 ] ]
gap> NmzHilbertSeries(D);
[ t^2-t+1, [ [ 1, 1 ], [ 3, 1 ] ] ]
gap> NmzHasConeProperty(D,"SupportHyperplanes");
true
gap> NmzSupportHyperplanes(D);
[ [ 1, 0, 0 ], [ 1, 3, -3 ] ]
gap> NmzEquations(D);
[ [ 1, 2, -3 ] ]
```

3.3 System of inhomogeneous equations

Example

```
gap> P := NmzCone(["inhom_equations",[[1,2,-3,1]], "grading", [[1,1,1]]]);
<a Normaliz cone>
gap> NmzIsInhomogeneous(C);
false
gap> NmzIsInhomogeneous(P);
true
gap> NmzHilbertBasis(P);
[ [ 1, 1, 1, 0 ], [ 3, 0, 1, 0 ], [ 0, 3, 2, 0 ] ]
gap> NmzModuleGenerators(P);
[ [ 0, 1, 1, 1 ], [ 2, 0, 1, 1 ] ]
```

3.4 Combined input

Normaliz also allows the combination of different kinds of input, e.g. multiple constraint types, or additional data like a grading. Suppose that you have a 3 by 3 "square" of nonnegative integers such that the 3 numbers in all rows, all columns, and both diagonals sum to the same constant M. Sometimes such squares are called magic and M is the magic constant. This leads to a linear system of equations. The magic constant is a natural choice for the grading. Additionally we force here the 4 corner to have even value by adding congruences.

Example

```
gap> Magic3x3even := NmzCone(["equations",
> [ [1, 1, 1,  -1, -1, -1,   0,  0,  0],
>   [1, 1, 1,   0,  0,  0,  -1, -1, -1],
>   [0, 1, 1,  -1,  0,  0,  -1,  0,  0],
>   [1, 0, 1,   0, -1,  0,   0, -1,  0],
>   [1, 1, 0,   0,  0, -1,   0,  0, -1],
>   [0, 1, 1,   0, -1,  0,   0,  0, -1],
>   [1, 1, 0,   0, -1,  0,  -1,  0,  0] ],
> "congruences",
> [ [1, 0, 0,   0, 0, 0,   0, 0, 0,  2],
>   [0, 0, 1,   0, 0, 0,   0, 0, 0,  2],
>   [0, 0, 0,   0, 0, 0,   1, 0, 0,  2],
>   [0, 0, 0,   0, 0, 0,   0, 0, 1,  2] ],
> "grading",
> [ [1, 1, 1,   0, 0, 0,   0, 0, 0] ] ] );
<a Normaliz cone>
gap> NmzHilbertBasis(Magic3x3even);
[ [ 0, 4, 2, 4, 2, 0, 2, 0, 4 ], [ 2, 0, 4, 4, 2, 0, 0, 4, 2 ],
  [ 2, 2, 2, 2, 2, 2, 2, 2, 2 ], [ 2, 4, 0, 0, 2, 4, 4, 0, 2 ],
  [ 4, 0, 2, 0, 2, 4, 2, 4, 0 ], [ 2, 3, 4, 5, 3, 1, 2, 3, 4 ],
  [ 2, 5, 2, 3, 3, 3, 4, 1, 4 ], [ 4, 1, 4, 3, 3, 3, 2, 5, 2 ],
  [ 4, 3, 2, 1, 3, 5, 4, 3, 2 ] ]
gap> NmzHilbertSeries(Magic3x3even);
[ t^3+3*t^2-t+1, [ [ 1, 1 ], [ 2, 2 ] ] ]
gap> NmzHilbertQuasiPolynomial(Magic3x3even);
[ 1/2*t^2+t+1, 1/2*t^2-1/2 ]
```

3.5 Using the dual mode

For solving systems of equations and inequalities it is often faster to use the dual Normaliz algorithm. We demonstrate how to use it with an inhomogeneous system of equations and inequalities. The input consists of a system of 8 inhomogeneous equations in R^3. The first row of the matrix M encodes the inequality $8x+8y+8z+7 \geq 0$. Additionally we say that x, y, z should be non-negative by giving the sign vector and use the total degree.

Example

```
gap> M := [
>  [ 8,  8,  8,  7 ],
>  [ 0,  4,  0,  1 ],
>  [ 0,  1,  0,  7 ],
>  [ 0, -2,  0,  7 ],
>  [ 0, -2,  0,  1 ],
>  [ 8, 48,  8, 17 ],
>  [ 1,  6,  1, 34 ],
>  [ 2,-12, -2, 37 ],
>  [ 4,-24, -4, 14 ]
> ];
[ [ 8, 8, 8, 7 ], [ 0, 4, 0, 1 ], [ 0, 1, 0, 7 ], [ 0, -2, 0, 7 ],
  [ 0, -2, 0, 1 ], [ 8, 48, 8, 17 ], [ 1, 6, 1, 34 ],
  [ 2, -12, -2, 37 ], [ 4, -24, -4, 14 ] ]
gap> D := NmzCone(["inhom_inequalities", M,
>               "signs", [[1,1,1]],
>               "grading", [[1,1,1]]]);
<a Normaliz cone>
gap> NmzCompute(D,["DualMode","HilbertBasis","ModuleGenerators"]);
true
gap> NmzHilbertBasis(D);
[ [ 1, 0, 0, 0 ], [ 1, 0, 1, 0 ] ]
gap> NmzModuleGenerators(D);
[ [ 0, 0, 0, 1 ], [ 0, 0, 1, 1 ], [ 0, 0, 2, 1 ], [ 0, 0, 3, 1 ] ]
```

As result we get the Hilbert basis of the cone of the solutions to the homogeneous system and the module generators which are the base solutions to the inhomogeneous system.

Chapter 4

Installing NormalizInterface

4.1 Compiling

NormalizInterface supports GAP 4.8.0 or later, and Normaliz 3.0.0 or later.

For technical reasons, installing and using NormalizInterface requires that your version of GAP is compiled in a special way. Specifically, GAP must be compiled against the exact same version of the GMP library as Normaliz. By default, GAP compiles its own version of GMP; however, we cannot use that, as it lacks C++ support, which is required by Normaliz.

Thus as the very first step, please install a version of GMP in your system. On most Linux and BSD distributions, there should be a GMP package available with your system's package manager. On Mac OS X, you can install GMP via Fink, MacPorts or Homebrew.

Next, make sure your GAP installation is compiled against the system wide GMP installation. To do so, switch to the GAP root directory, and enter the following commands:

```
make clean
./configure --with-gmp=system
make
```

Next you need to compile a recent version of Normaliz. This requires the presence of several further system software packages, which you install via your system's package manager. At least the following are required:

- git
- cmake
- boost

Once you have installed these, you can build Normaliz by using the build-normaliz.sh script we provide. It takes a single, optional parameter: the location of the GAP root directory.

```
./build-normaliz.sh GAPDIR
```

Once it completed successfully, you can then build NormalizInterface like this:

```
./configure --with-gaproot=GAPDIR
make
```

If you need to customize the normaliz compilation, please have a look at Normaliz.git/source/INSTALL. Remember to use the same compiler and GMP version as for GAP.

References

[BIRS14] W. Bruns, B. Ichim, T. Römer, and C. Söger. Normaliz, a tool for computations in affine monoids, vector configurations, lattice polytopes, and rational cones, Version 3.1.0. https://www.normaliz.uni-osnabrueck.de, 2014. 3

Index