Normaliz 2.12

Winfried Bruns, Bogdan Ichim, Tim Römer and Christof Söger

http://www.math.uos.de/normaliz

mailto:normaliz@uos.de

Contents

1. Introduction

1.1. The objectives of Normaliz

The program Normaliz, version 2.11, is a tool for computing the Hilbert bases and enumerative data of rational cones, and more generally, sets of lattice points in rational polyhedra. The mathematical background and the terminology of this manual are explained in Appendix A. For a thorough treatment of the mathematics involved we refer the reader to [2] and [4]. The terminology follows [2]. For algorithms of Normaliz see [3], [5], [6] and [7].

A rational cone or a rational polyhedron can be given by

(1) a system of generators $\mathscr{G}$ in a lattice $\mathbb{Z}^n$;
(2) constraints: an (in)homogeneous linear system of equations and inequalities.

Affine monoids can also be defined by generators and binomial relations.

The Hilbert basis of a rational pointed cone C in $\mathbb{R}^n$ is defined with respect to a lattice $L \subset \mathbb{Z}^n$: it is the unique minimal system of generators of the monoid $C \cap L$. In the case of polyhedra Normaliz computes a system of generators of the set of lattice points in P over the Hilbert basis of the recession monoid.

The standard choice for L is $\mathbb{Z}^n$ itself, but for Normaliz this choice can be modified in two ways:

(1) L can be chosen to be the sublattice of $\mathbb{Z}^n$ generated by $\mathscr{G}$;
(2) L can be chosen to be the lattice of solutions of an (in)homogeneous system of congruences if the cone or the polyhedron is specified by equations and inequalities.

In particular, Normaliz solves combined systems of diophantine linear equations, inequalities and congruences. Conversely, Normaliz computes a system of constraints defining the cone or polyhedron and the lattice for which the Hilbert basis and system of generators have been computed.

Normaliz has special input types for lattice polytopes (represented by their vertices) and monomial ideals (represented by the exponent vectors of their generators). Via the specification of a grading, one can easily apply Normaliz also to rational polytopes.

The enumerative data computed by Normaliz depend on a grading of the monoid under consideration (see Section 3.6): if asked to do so, Normaliz computes the Hilbert series and the Hilbert quasipolynomial of the monoid or set of lattice points in a polyhedron. In polytopal terminology: Normaliz computes Ehrhart series and quasipolynomials of rational polyhedra. Via its offspring NmzIntegrate [8], Normaliz computes generalized Ehrhart series and Lebesgue integrals of polynomials over rational polytopes.

The computations can be restricted in several ways, for example to the support hyperplanes or the lattice points of a rational polytope.

Acknowledgement. The development of Normaliz is currently supported by the DFG SPP 1489 “Experimentelle Methoden in Algebra, Geometrie und Zahlentheorie”.

1.2. Platforms and access from other systems

Executables for Normaliz are provided for Mac OS, Linux and MS Windows. Normaliz is written in C++, and should be compilable on every system that has a GCC compatible compiler. It uses the standard packages Boost and GMP (see Section 10).

Normaliz can be accessed from the following systems:

- SINGULAR via the library `normaliz.lib`,
- MACAULAY 2 via the package `Normaliz.m2`,
- COCOA via an external library,
- GAP via the GAP package NORMALIZINTERFACE [10] which uses libnormaliz,
- POLYMAKE (thanks to the POLYMAKE team),
- SAGE via an optional package by A. Novoseltsev.

The Singular and Macaulay 2 interfaces are contained in the Normaliz distribution. At present, their functionality is limited to Normaliz 2.10.

Furthermore, Normaliz is used by the B. Burton's system REGINA.

1.3. Major changes relative to version 2.11

(1) Complete revision of the dual algorithm towards better performance in arithmetically complex situations.
(2) Additional internal parallelization of simplicial cones with large determinants.
(3) Linear algebra accelerated.
(4) Overflow checks improved.

There are no changes in the user interface.

1.4. Future extensions

(1) Redesign of input and output.
(2) Automatic choice of integer type.
(3) A programming interface (using the already existing library).
(4) Exploitation of symmetries.
(5) Access from further systems.

2. Getting started

Download

- the zip file with the Normaliz source, documentation, examples and further platform independent components, and
- the zip file containing the executable(s) for your system

from the Normaliz website

`http://www.math.uos.de/normaliz`

and unzip both in the same directory of your choice. In it, a directory `Normaliz2.12` (called Normaliz directory in the following) is created with several subdirectories. (Some versions of the Windows executables may need the installation of a runtime library; see our website.)

In the Normaliz directory open jNormaliz by clicking `jNormaliz.jar` in the appropriate way. (We assume that Java is installed on your machine.) In the jNormaliz file dialogue choose one

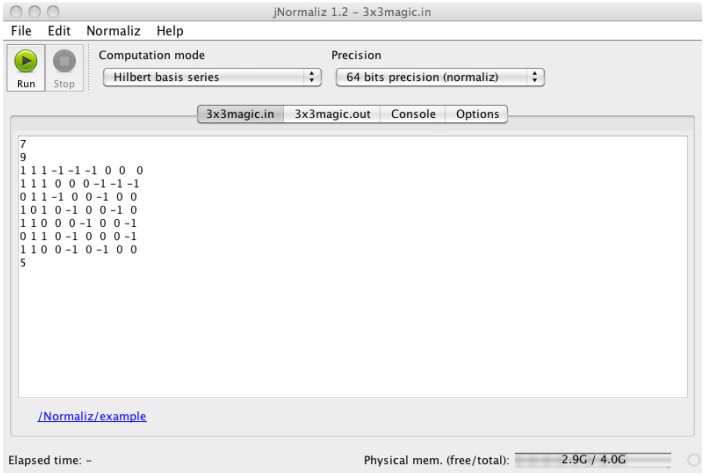

Figure 1: jNormaliz

of the input files in the subdirectory `example`, say `small.in`, and press `Run`. In the console window you can watch Normaliz at work. Finally inspect the output window for the results.

The menus and dialogues of jNormaliz are self explanatory, but you can also consult the documentation [1] via the help menu.

If the executables prepared cannot be run on your system, then you can compile Normaliz yourself (see Section 10).

Moreover, one can, and often will, run Normaliz from the command line. This is explained in Section 4.

If 64 bit integer precision is not sufficient, then one can switch jNormaliz to infinite precision (or use the option `-B` from the command line). Then Normaliz has no restrictions on the integer precision. See Section 4.6. (The integer precision has nothing to do with the address width (32 bit or 64 bit) of your operating system.)

3. The input file

The input file `<project>.in` consists of one or several matrices. Each matrix is built as follows:

(1) The first line contains the number of rows m.
(2) The second contains the number of columns n.
(3) The next m lines of n integers each contain the rows.
(4) The last line contains a single number or word specifying the type of input the matrix presents.

The line structure is irrelevant for the interpretation of the input, and only meant as a suggestion for a transparent structure of the file. The examples below are often typeset in two columns. It should always be obvious how to read them properly.

At the moment there are three major classes of input matrices, namely *generators*, *constraints*, and *relations*.

Generators and constraints themselves fall into two subclasses:

(1) *homogeneous*, defining cones and lattices,
(2) *inhomogeneous*, defining polyhedra and affine lattices.

Additional classes are *grading* and *dehomogenization*.

Remark. Normaliz 2.12 (like Normaliz 2.11) allows inhomogeneous input types. While polytopes are defined by inhomogeneous data, one usually wants to compute the cone over the polytope, its Hilbert basis and the Ehrhart series. These computations require *homogeneous* input types, and their application to polytopes is explained below. Also see Section 8.2.

Rules for the combination of input types:

(1) The order of the matrices does not matter.
(2) Matrices of the same type will be concatenated (and must have the same number of columns).
(3) The three classes generators, constraints and relations exclude each other (at present), with one exception: `excluded_faces` are allowed with homogeneous generators.
(4) There can be at most one type of generators.
(5) Homogeneous and inhomogeneous constraints can be mixed. As soon as an inhomogeneous type is present, Normaliz treats the input as being inhomogeneous, transforming the homogeneous types appropriately.
Exception: `excluded_faces` are forbidden in combination with inhomogeneous constraints for which they are replaced by `strict_inequalities`.

For each input type we specify two lattices: the *ambient lattice* $\mathbb{A}$ to which the input data refer and the *essential lattice* $\mathbb{E} \subset \mathbb{A}$ with respect to which all data are computed.

In this section we assume that Normaliz is run in a computation mode in which the Hilbert basis or the system of generators, respectively, is actually computed. (See Section 4 for computation modes.)

3.1. Generators of cones and lattices

These generator types are `integral_closure`, `normalization`, `polytope` and `rees_algebra`.

3.1.1. Type `integral_closure`

The rows of an $m \times n$ matrix of this type represent m vectors in the ambient lattice $\mathbb{A} = \mathbb{Z}^n$. The essential lattice $\mathbb{E}$ is the smallest direct summand of $\mathbb{Z}^n$ that contains the vectors in the matrix.

The vectors are considered as a system of generators $\mathscr{G}$ of a cone C, and Normaliz computes the Hilbert basis of C with respect to $\mathbb{E}$ (or, equivalently, $\mathbb{Z}^n$).

The nomenclature `integral_closure` is explained by the fact that the Hilbert basis generates the integral closure of the monoid $\mathbb{Z}_+\mathscr{G}$ in $\mathbb{Z}^n$.

A simple example:

```
Input            Hilbert basis
3                1 0
2                0 1
2 0
1 1
0 2
integral_closure
```

In this example, the three input vectors clearly generate the positive orthant $\mathbb{R}^2_+$ in $\mathbb{R}^2$, and the two unit vectors clearly are the Hilbert basis of $\mathbb{R}^2_+ \cap \mathbb{Z}^2$.

Example input files: `rproj2.in`, `small.in`.

3.1.2. Type `normalization`

The matrix is interpreted as as one of type `integral_closure`, however $\mathbb{E}$ is chosen as the sublattice of $\mathbb{Z}^n$ generated by $\mathscr{G}$.

The choice of the name `normalization` indicates that Normaliz computes the normalization of the monoid $\mathbb{Z}_+\mathscr{G}$. (The computation of such normalizations was the original goal of Normaliz, hence the name.)

We choose the same input vectors as above, but change the type to `normalization`:

```
Input            Hilbert basis
3                2 0
2                1 1
2 0              0 2
1 1
0 2
normalization
```

The cone has not changed, but the lattice has: $\mathbb{E}$ is now the sublattice of $\mathbb{Z}^2$ of all (z_1, z_2) with $z_1 + z_2 \equiv 0 \mod 2$.

Example input files: `rafa2416.in`, `A443.in`.

3.1.3. Type `polytope`

The rows of the matrix are interpreted as integral points of a lattice polytope in $\mathbb{R}^n$, which is their convex hull.

The cone C is the cone over the polytope, i.e. the cone with apex 0 in $\mathbb{R}^{n+1}$ generated by the vectors $(x,1)$ where x represents a row of the input matrix. We want to compute the *Ehrhart monoid* $C \cap \mathbb{Z}^{n+1}$.

The lattice $\mathbb{A}$ is $\mathbb{Z}^{n+1}$, and $\mathbb{E}$ is the smallest direct summand of $\mathbb{A}$ containing the generators of C.

Type `polytope` is only a variant of type `integral_closure`. One obtains the same results as in type `integral_closure` with the extended vectors $(x,1)$ as input.

Example input files: `polytop.in`, `FortuneCookie.in`, `lo6.in`.

3.1.4. Rational polytopes

Normaliz has no special input type for rational polytopes. In order to process them one uses type `integral_closure` together with a grading. Suppose the polytope is given by vertices

$$v_i = (r_{i1}, \ldots, r_{in}), \qquad i = 1, \ldots, m,\ r_{ij} \in \mathbb{Q}.$$

Then we write v_i with a common denominator:

$$v_i = \left(\frac{p_{i1}}{q_i}, \ldots, \frac{p_{in}}{q_i} \right), \quad p_{ij}, q_i \in \mathbb{Z},\ q_i > 0.$$

The generator matrix is given by the rows

$$\widetilde{v}_i = (p_{i1}, \ldots, p_{in}, q_i), \quad i = 1, \ldots, m.$$

We must add a grading since Normaliz cannot recognize it without help (unless all the q_i are equal). The grading linear form has coordinates $(0, \ldots, 0, 1)$. See 3.6 below for general information on gradings.

Let us look at a concrete example (contained in `rational.in`), the triangle P with vertices

$$(1/2, 1/2),\ (-1/3, -1/3),\ (1/4, -1/2).$$

In order to apply Normaliz to it one uses the following input:

```
3                          1
3                          3
1 1 2                      0 0 1
-1 -1 3                    grading
1 -2 4
integral_closure
```

The output will be discussed in 6.1.3.

3.1.5. Type `rees_algebra`

In this type the input vectors are considered as exponent vectors of the generators of a monomial ideal I in the polynomial ring $K[X_1,\dots,X_n]$. Normaliz computes the normalization of the Rees algebra of the ideal I (see [4] for the notion of Rees algebra.) This is a monomial subalgebra of the extended polynomial ring $K[X_1,\dots,X_n,T]$ with an auxiliary variable T. Normaliz computes the exponent vectors in $\mathbb{Z}^{n+1}$ of the system of generators. For an example, see Section 6.

In type `rees_algebra` one has $\mathbb{A} = \mathbb{E} = \mathbb{Z}^{n+1}$.

Example input file: `rees.in`.

3.1.6. Preparation of the generators

After the coordinate transformation to the lattice $\mathbb{E}$, Normaliz divides each generator by the greatest common divisor of its components. For example, the extreme rays listed will always be such $\mathbb{E}$-primitive vectors (re-transformed to $\mathbb{A}$ where they may not be primitive).

If a grading is present, the generators will be sorted by degree in ascending order. If no grading is available, they will be sorted by their 1-norm in the lattice $\mathbb{E}$. Those of the same degree will remain sorted as in the input file (or the result of a previous computation).

This preparation is also performed if the generators result from a system of constraints.

3.2. Homogeneous Constraints

Homogeneous inequalities, equations, and congruences defining the cone and the lattice are called *homogeneous constraints*. Matrices representing them are of types `equations`, `inequalities`, `signs`, `congruences` and `excluded_faces`.

In previous versions `inequalities` were called `hyperplanes`. The name is still allowed, but no longer recommended.

The numbers of columns must of course match: for the ambient lattice $\mathbb{A} = \mathbb{Z}^n$ the matrices of equations and inequalities must have n columns, and matrices of congruences must have $n+1$ columns. The essential lattice $\mathbb{E}$ is the smallest sublattice of $\mathbb{A}$ containing the solutions of the combined systems of constraints.

If there is no matrix of inequalities, then it is assumed that the user wants to compute the nonnegative solutions of the system represented by the equations and congruences.

3.2.1. Type `inequalities` (formerly `hyperplanes`)

A row $(\xi_1,\dots,\xi_n)$ of the input matrix of this type represents an inequality

$$\xi_1 x_1 + \cdots + \xi_n x_n \geq 0$$

for the vectors $(x_1,\dots,x_n)$ of $\mathbb{R}^n$.

Example:

```
Input            Hilbert basis
2                0 -1
2                1  1
1 0
1 -1
inequalities
```

Normaliz has computed the Hilbert basis of the cone defined by the inequalities $x_1 \geq 0$ and $x_1 - x_2 \geq 0$ with respect to the lattice $\mathbb{Z}^2$.

Example input file: `Condorcet.in`.

Remark: In previous versions this type was called `hyperplanes`, a name that can still be used.

3.2.2. Sign inequalities: `signs`

There is a shortcut for the input of inequalities $x_i \geq 0$ or $x_i \leq 0$. The input matrix of type `signs` has format $1 \times n$ and the entries of its single row are in $\{-1, 0, 1\}$:

-1 stands for $x_i \leq 0$,
1 stands for $x_i \geq 0$,
0 indicates that the sign of x_i is not restricted.

Example:

```
1
4
1 -1 0 1
signs
```

In this example we require that $x_1, x_4 \geq 0$ and $x_2 \leq 0$.

Example input file: `Condorcet.in`.

3.2.3. Polytopes by inequalities

Normaliz has no special input type for polytopes defined by inequalities since they can easily be specified via type `inequalities`. Suppose the polytope is given by inequalities

$$\alpha_{i1}x_1 + \cdots + \alpha_{in}x_n \geq \beta_i, \quad i = 1, \ldots, m, \ \alpha_{ij}, \beta_i \in \mathbb{Z}.$$

Then we homogenize the inequalities in the form

$$\alpha_{i1}x_1 + \cdots + \alpha_{in}x_n - \beta_i x_{n+1} \geq 0,$$

and use type `inequalities` for them in connection with the grading $(0, \ldots, 0, 1)$.

The file `poly_ineq.in` contains

```
3
3                          1
2  7 3                     3
-8 2 3                     0 0 1
1 -1 0                     grading
hyperplanes
```

It reproduces the triangle that we have discussed in 3.1.4.

3.2.4. Type `equations`

A row $(\xi_1,\dots,\xi_n)$ of the input matrix of this type represents an equation

$$\xi_1 x_1 + \cdots + \xi_n x_n = 0$$

for the vectors $(x_1,\dots,x_n)$ of $\mathbb{R}^n$.

Example:

```
Input             Hilbert basis
1                 2 0 1
3                 0 2 1
1 1 -2            1 1 1
equations
```

If the input file contains no further matrices, Normaliz has computed the Hilbert basis of the subcone of $\mathbb{R}^3_+$ defined by the equation $x_1 + x_1 - 2x_3 = 0$.

Example input files: `4x4.in`, `5x5.in`.

3.2.5. Type `congruences`

We consider the rows of a matrix of this type to have length $n+1$. Each row $(\xi_1,\dots,\xi_n,c)$ represents a congruence

$$\xi_1 z_1 + \cdots + \xi_n z_n \equiv 0 \mod c$$

for the elements $(z_1,\dots,z_n) \in \mathbb{Z}^n$.

Example:

```
Input             Hilbert basis
1                 2 0
3                 1 1
1 1 2             0 2
congruences
```

If no other matrix is in the input file, then Normaliz computes the Hilbert basis of the positive orthant intersected with the lattice of all integral vectors (z_1, z_2) such that $z_1 + z_2 \equiv 0 \mod 2$ and the result is the same as in 3.1.2 above.

Example input file: `3x3magiceven.in`.

3.2.6. Type `excluded_faces`

This type is useful for the computation of Hilbert series of semiopen cones. It is interpreted as follows:

(1) If used with input of type *generators*, the faces defined by `excluded_faces` are simply excluded. They do *not* restrict the cone.
(2) Otherwise `excluded_faces` has a twofold meaning: (a) they are additional inequalities defining the cone; (b) the faces they define are excluded from the Hilbert series computation.

The twofold interpretation in case (b) saves the user from including the `excluded_faces` twice.

An example of type (a):

```
4                       1
2                       3
0 0                     1 1 0
0 1                     excluded_faces
1 1
1 0
polytope
```

defines the unit square in $\mathbb{R}^2$, but $(0,0)$ excluded from the Hilbert series computation (See Section 6.4 for the discussion of the output.)

An example that combines `excluded_faces` with other constraints:

```
1 24
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
signs
3 24
1 1 1 1 1 1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 1 -1 1 1 -1 -1 1 -1
1 1 1 1 1 1 1 1 -1 -1 1 -1 -1 -1 -1 -1 -1 -1 1 1 1 -1 -1 -1
1 1 1 1 1 1 1 1 1 -1 -1 -1 1 1 1 -1 -1 -1 -1 -1 -1 -1 -1 -1
excluded_faces
```

The semiopen cone described by this input is the intersection of the positive orthant in $\mathbb{R}^{24}$ with 3 halfspaces that are defined by the linear forms given as `excluded_faces`, and the facets defined by these linear forms are then excluded.

Note that `excluded_faces` only affect the Hilbert series. They are ignored for all other computations.

Example input file: `SquareMinusVertex.in, CondParSemi.in`.

3.2.7. The constraints combined

Let L be the sublattice of $\mathbb{Z}^n$ that consists of the solutions of the system of congruences defined by the input matrix of type `congruences`. ($L = \mathbb{Z}^n$ if there is no such matrix.) Moreover, let A be the matrix of type `inequalities` (combined with the matrix representing the `signs`) and B be the matrix of type `equations`. Then the cone C is given by

$$C = \{x \in \mathbb{R}^n : Ax \geq 0,\ Bx = 0\},$$

and the Hilbert basis of $C \cap L$ is computed.

The ambient lattice $\mathbb{A}$ is $\mathbb{Z}^n$, and the essential lattice is $\mathbb{E} = L \cap \mathbb{R}C$.

If there is no matrix of type `equations`, then the system of equations is empty, satisfied by all vectors of $\mathbb{R}^n$.

If the input is of type constraints and there are no explicit inequalities contained in it, then Normaliz adds the $n \times n$ unit matrix of `inequalities` to restrict all computations to the non-negative orthant.

See Section 6.2.3 for an example combining types `equations` and `congruences`.

Example input file: `3x3magiceven.in`.

3.3. Relations

Relations do not select a sublattice of $\mathbb{Z}^n$ or a subcone of $\mathbb{R}^n$, but define a monoid as a quotient of $\mathbb{Z}^n_+$ modulo a system of congruences (in the semigroup sense!).

The rows of the input matrix of this type are interpreted as generators of a subgroup $U \subset \mathbb{Z}^n$, and Normaliz computes an affine monoid and its normalization as explained in Section A.5.

Set $G = \mathbb{Z}^n/U$ and $L = G/\mathrm{torsion}(G)$. Then the ambient lattice is $\mathbb{A} = \mathbb{Z}^r$, $r = \mathrm{rank}\, U$, and the essential lattice is L, realized as a sublattice of $\mathbb{A}$. Normaliz computes the image of $\mathbb{Z}^n_+$ in L and its normalization.

3.3.1. Type `lattice_ideal`

As an example we consider the binomials $X_1X_3 - X_2^2$, $X_1X_4 - X_2X_3$:

```
Input            Hilbert basis
2                3 0
4                2 1
1 -2  1 0        1 2
1 -1 -1 1        0 3
lattice_ideal
```

In this example $\mathbb{Z}^4/U$ is torsionfree, but we can replace each of the vectors in the input matrix by a nonzero integral multiple without changing the result.

The type `lattice_ideal` cannot be combined with any other input type (except `grading`)—such a combination would not make sense. (See Section 3.6.3 for the use of a grading in this case.)

Example input file: `lattice_ideal.in`.

3.4. Generators of polyhedra

3.4.1. Type polyhedron

A matrix of type polyhedron contains both the generators of the recession cone C and (a superset of) the vertices of the polyhedron. For a polyhedron in $\mathbb{R}^n$ the vectors have $n+1$ entries in $\mathbb{Z}$. The last component must be nonnegative. It is interpreted as follows:

(1) If it is 0, the vector is part of the system of generators of C.
(2) If it is a positive integer, it is considered as the denominator for the first n entries, which together define a (potential) vertex of P in $\mathbb{Q}^n$.

The order of the input vectors is irrelevant.

```
4 3
1 0 0
0 1 0
3 2 2
2 3 2
polyhedron
```

defines the polyhedron

$$P = \text{conv}((3/2,1),\ (1,3/2)) + \mathbb{R}^2_+$$

in $\mathbb{R}^2$.

Presently there is no possibility to restrict the lattice. If a polyhedron $P \subset \mathbb{R}^n$ is defined by a matrix of type `polyhedron`, then all lattice related computations will refer to the standard lattice $\mathbb{Z}^n$. Therefore $\mathbb{A} = \mathbb{E} = \mathbb{Z}^n$ for this input type.

Example input file: `polyhedron.in`

Remark: For computations of (rational) polytopes (i.e., bounded polyhedra) you almost always want to use the input types `polytope` or `integral_closure` (in conjunction with a grading). We discuss this point in Section 8.2.

3.5. Inhomogeneous constraints

What has been said about homogeneous constraints holds analogously for inhomogeneous ones. Note that we must accommodate a right hand side in inhomogeneous constraints, and therefore inhomogeneous constraints are one component longer than their homogeneous counterparts (with the exception of `strict_inequalities` and `strict_signs`).

If there are no explicit inequalities in the input, then, as in the homogeneous case, it is assumed that nonnegative solutions are to be computed.

The lattice $\mathbb{E}$ for the computation of the recession module is the sublattice of solutions of the homogeneous constraints associated to the given inhomogeneous constraints. (So it is defined even if the polyhedron is empty.) The affine lattice for the computation of lattice points in the polyhedron is the set of solutions of the given system of constraints.

3.5.1. Type `inhom_inequalities`

We consider inequalities

$$\xi_1 x_1 + \cdots + \xi_n x_n \geq \eta, \qquad \xi_i, \eta \in \mathbb{Z},$$

rewritten as

$$\xi_1 x_1 + \cdots + \xi_n x_n + (-\eta) \geq 0$$

and then represented by the input vector

$$(\xi_1, \ldots, \xi_n, -\eta).$$

```
2
3
1 0 -1
0 1 -2
inhom_inequalities
```

describes the polyhedron

$$P = \{x \in \mathbb{R}^2 : x_1 \geq 1, x_2 \geq 2\}.$$

Remark: For computations of (rational) polytopes (i.e., bounded polyhedra) you almost always want to use the input type `inequalities`. We discuss this point in Section 8.2.

Example input file: `SquareMinusVertexInhom.in, NonCMDivisor.in`.

3.5.2. Type `strict_inequalities`

These are shortcuts for inequalities of type

$$\xi_1 x_1 + \cdots + \xi_n x_n \geq 1.$$

```
2
2
1 0
0 1
strict_inequalities
```

describes the polyhedron

$$P = \{x \in \mathbb{R}^2 : x_1 \geq 1, x_2 \geq 1\}.$$

Example input file: `CondorcetInt.in`.

3.5.3. Type `strict_signs`

The components of a (one rowed) matrix of this type are interpreted as follows:

-1 stands for $x_i \leq -1$,
1 stands for $x_i \geq 1$,
0 indicates that x_i is not restricted.

```
1
3
1 -1 1
strict_signs
```

represents the inequalities

$$x_1 \geq 1,\ x_2 \leq -1,\ x_3 \geq 1.$$

Example input file: `CondorcetInt.in`.

3.5.4. Type `inhom_equations`

We consider equations

$$\xi_1 x_1 + \cdots + \xi_n x_n = \eta, \qquad \xi_i, \eta \in \mathbb{Z},$$

rewritten as

$$\xi_1 x_1 + \cdots + \xi_n x_n + (-\eta) = 0$$

and then represented by the input vector

$$\xi_1, \ldots, \xi_n, -\eta.$$

The input

```
2
4
1 2 3 -2
3 2 -2 5
inhom_equations
```

represents the system

$$\begin{aligned} x_1 + 2x_2 + 3x_3 &= 2, \\ 3x_1 + 2x_2 - 2x_3 &= -5. \end{aligned}$$

Example input file: `InhomIneq.in`.

3.5.5. Type `inhom_congruences`

We consider the rows of a matrix of type `inhom_congrueneces` to have length $n+2$. Each row $(\xi_1,\dots,\xi_n,-\eta,c)$ represents a congruence

$$\xi_1 z_1 + \cdots + \xi_n z_n \equiv \eta \mod c$$

for the elements $(z_1,\dots,z_n) \in \mathbb{Z}^n$.

```
2
4
1 2 -3 7
2 2 -4 13
inhom_congrueneces
```

represents the system

$$\begin{aligned} x_1 + 2x_2 &\equiv 3 \quad (7), \\ 2x_2 + x_2 &\equiv 4 \quad (13) \end{aligned}$$

of simultaneous congruences.

Example file: `ChineseRemainder.in`.

3.5.6. Mixing homogeneous and inhomogeneous constraints

Homogeneous and inhomogeneous constraints can appear in the same input file. Note that there must be at least one inhomogeneous type in order to indicate to Normaliz that the input is inhomogeneous.

The homogeneous constraints will simply be considered as inhomogeneous constraints with right hand side 0. An example:

```
2                          1
4                          3
0 1 0 -1                   1 0 0
0 0 1 -1                   inequalities
inhom_inequalities
```

defines the polyhedron

$$P = \{x \in \mathbb{R}^3 : x_1 \geq 0, x_2 \geq 1, x_3 \geq 1\}.$$

3.6. Grading

$\mathbb{Z}$-valued grading can be specified in two ways:

(1) *explicitly* by including a grading in the input, or

(2) *implicitly*. In this case Normaliz checks whether the extreme integral generators of the monoid lie in an (affine) hyperplane A. If so, then the (unique) primitive $\mathbb{Z}$-linear form λ that affords an equation $\lambda(x) = b$ for A is used as the grading.

Note: In previous versions we used *height* as a synonym for *degree*.

A grading is explicitly specified by an $1 \times n$ matrix for cones embedded in $\mathbb{R}^n$, and its type is fixed by the attribute `grading`, for example

```
1
2
3 2
grading
```

Normaliz checks whether all generators of the monoid have positive degree.

Before Normaliz can apply the degree, it must be restricted to the effective lattice $\mathbb{E}$. Even if the entries of the grading vector are coprime, it often happens that all degrees of vectors in $\mathbb{E}$ are divisible by a greatest common divisor $d > 1$. Then d is extracted from the degrees, and it will appear as `denominator` in the output file.

Special rules apply to some input types that we explain in the following.

3.6.1. `polytope`

Cones defined by lattice polytopes always have an implicit grading in which the lattice points in the polytope have degree 1 (roughly speaking). Therefore it is not possible to use an explicit grading together with this input type.

If it should be necessary to apply a different grading, then one converts the input of type `polytope` to `integral_closure` by appending `1` to each row of the input matrix and adds the grading to be used.

3.6.2. `rees_algebra`

Suppose that the rows of the input matrix specify vectors of length n. Then these are embedded into $\mathbb{R}^{n+1}$, and therefore the grading must have $n+1$ components. Example:

```
3                          1
3                          4
0 1 2                      1 1 1 -1
2 0 2                      grading
1 1 1
rees_algebra
```

Note that the Rees algebra has an implicit grading if and only if all the monomials have the same total degree, say g. Then the grading vector chosen automatically is $(1,\dots,1,-(g-1))$.

3.6.3. `lattice_ideal`

In this case the unit vectors correspond to generators of the monoid. Therefore the degrees assigned to them must be positive. Moreover, the vectors in the input represent binomial relations, and these must be homogeneous. In other words, both monomials in a binomial must have the same degree. This amounts to the condition that the input vectors have degree 0. Normaliz checks this condition. Example:

```
1 4                          1 4
1 1 -1 -1                     1 2 1 2
lattice_ideal                 grading
```

3.6.4. Inhomogeneous input

Recall that inhomogeneous data defining a polyhedron in $\mathbb{R}^n$ have length $n+1$ (or even $n+2$ in the case of inhomogeneous congruences). The grading assigns a degree to each of the n canonical basis elements of $\mathbb{Z}^n$, and therefore has length n:

```
2  3                            1 2
1 1 5                           1 1
1 -1 4                          grading
inhom_inequalities
```

is an example of consistent input.

3.7. Dehomogenization

Inhomogeneous input for objects in $\mathbb{R}^d$ is homogenized by an additional coordinate and then computed in $\mathbb{R}^{d+1}$, but with the additional condition $x_{d+1} \geq 0$, and then dehomogenizing all results: the substitution $x_{d+1} = 1$ acts as the *dehomogenization*, and the inhomogeneous input types implicitly choose this dehomogenization.

Like the grading, one can define the dehomogenization explicitly as in the following example:

```
2 3                       1 3
-1 1 0                    1 0 0
-1 0 1                    dehomogenization
inequalities
```

This input is equivalent to

```
2
3
1 0 -1
0 1 -1
inhom_inequalities
```

But the dehomogenization can be any linear form δ satisfying the condition $\delta(x) \geq 0$ on the cone that is truncated. (In combination with constraints, the condition $\delta(x) \geq 0$ is automatically satisfied since δ is added to the constraints.)

Since inhomogeneous input defines a dehomogenization implicitly, the type `dehomogenization` cannot be combined with any of the inhomogeneous input types. It is also forbidden for

`normalization`, `polytope`, `rees_algebra` and `lattice_ideal`.

(Note that `polytope` defines its own homogenization via the grading by the last coordinate.)

The input type `dehomogenization` makes the computation inhomogeneous, resulting in inhomogeneous output. The polyhedron computed is the intersection of the cone C (and the lattice L) defined by the remaining components of the input with the hyperplane given by $\delta(x) = 1$, and the recession cone is $C \cap \{x : \delta(x) = 0\}$.

A potential application is the adaptation of other input formats like that of polymake to Normaliz. For example, if the first coordinate is used as the homogenizing variable, then the system of inequalities above is given as

```
2                       1
3                       3
-1 1 0                  1 0 0
-1 0 1                  dehomogenization
inequalities
```

Example input file: `dehomogenization.in`.

3.8. Pointedness

For Hilbert basis computations and triangulations Normaliz requires the (recession) cone to be pointed ($x, -x \in C \implies x = 0$). Whenever the condition of pointedness is violated at a step where it is crucial, Normaliz will stop computations.

Pointedness is checked by testing whether the dual cone of C is full dimensional, and if not, then the constructor of the cone complains as follows:

```
Full Cone error: Matrix with rank = number of columns needed in
the constructor of the object Full_Cone. Probable reason: Cone
not full dimensional(<=> dual cone not pointed)!
```

3.9. The zero cone

The zero cone with an empty Hilbert basis is a legitimate object for Normaliz. Nevertheless a warning message is issued if the zero cone is encountered.

3.10. Additional input file for NmzIntegrate

NmzIntegrate, whether called by Normaliz or from the command line, needs an input file `<project>.pnm` that contains the polynomial for which the generalized Ehrhart series or the integral is to be computed. See [8].

3.11. Number codes for input types

For historical reasons some of the input types can be represented by numbers instead of names. We strongly advise the user to avoid them now despite that they can still be used. These codes are

0: `integral closure`
1: `normalization`
2: `polytope`
3: `rees_algebra`
4: `inequalties` (or `hyperplanes`)
5: `equations`
6: `congruences`
10: `lattice_ideal`

4. Running Normaliz

The simplest way to call Normaliz from the command line is

```
normaliz <project>
```

for example

```
normaliz rafa2416
```

The project name is `rafa2416`. Normaliz reads the input file `rafa2416.in` (hopefully existing), computes everything it can compute, and writes the output to `rafa2416.out`. (We assume that the executable `normaliz` or `normaliz.exe` is in the search path. Otherwise you have to prefix it with a suitable relative or absolute path.)

In the following we explain the various options of Normaliz. The full text names given appear in the help screen as well as in the menus of jNormaliz which allows you to choose options interactively.

In the default computation mode Normaliz will try to compute all data accessible to it, using the triangulation based *primal* algorithm. All options that can be activated are switched off by default.

The default mode is broken by any option that asks for a specific computation. These are all *except*

`ceBfax`.

4.1. Full syntax and basic rules

The full syntax for calling Normaliz from the command line is

```
normaliz [-sN1vq] [-nph] [-dr] [-tTy] [-ceB] [-fa] [-ELI] [-x=<T>] [<project>]
```

where the options and `<project>` are optional.

A help screen can be displayed by `normaliz -?`.

Basic rules for the use of options:

1. If no `<project>` is given, the program will ask you for it or display a help screen.
2. The option `-x` differs from the other ones: `<T>` represents a positive number assigned to `-x`; see Section 4.5.
3. Normaliz will look for `<project>.in` as input file.

   If you inadvertently typed `rafa2416.in` as the project name, then Normaliz will first look for `rafa2416.in.in` as the input file. If this file doesn't exist, `rafa2416.in` will be loaded.
4. Adding a pure output option, namely `-f` or `-a`, or an option controlling execution does not change the computation mode. In particular, it does not change the default computation mode.
5. The options can be given in arbitrary order. All options are accumulated, and there is no mutual exclusion.
6. However, not all options are allowed for inhomogeneous input data; see Section 4.3.
7. If Normaliz cannot perform a computation explicitly asked for by the user, it will terminate. Typically this happens if no grading is given although it is necessary.
8. In the default mode Normaliz does not complain about missing data (anymore). It will simply omit those computations that are impossible.
9. If a certain type of computation is not asked for explicitly, but can painlessly be produced as a side effect, Normaliz will compute it. For example, as soon as a grading is present and the Hilbert basis is computed, the degree 1 elements of the Hilbert basis are selected from it.

For example, if you input the command

```
normaliz -c -p -a rafa2416 -T -y    or    normaliz -cpaTy rafa2416
```

then the program will take the file `rafa2416.in` as input, control data will be displayed on your terminal, the support hyperplanes, the triangulation, the multiplicity, the Hilbert series and the Hilbert (quasi)polynomial will be computed and all the possible output files will be produced.

4.2. Computation modes: homogeneous input

4.2.1. Basic modes

The least that Normaliz can do is

`-s support hyperplanes`: only the constraints of the cone and the lattice under consideration and the extreme rays are computed.

All computation modes include `-s`.

For Hilbert basis computations one uses

`-N Hilbert basis`: computes the Hilbert basis.

The degree 1 elements of the Hilbert basis are computed by

`-1 degree 1 elements`: only degree 1 elements are computed.

Enumerative data are chosen by

`-v volume`: Normaliz computes the multiplicity (or normalized volume);
`-q Hilbert series`.

Remarks: (1) The Hilbert basis can also be computed by the dual algorithm; see Section 4.2.3. In the presence of a grading, `-N` includes the computation of the degree 1 elements.

(2) For degree 1 elements the dual algorithm can be applied as well. Moreover, the approximation of rational polytopes is available for this purpose; see Section 4.2.4.

(3) The options `-1vq` require the presence of a grading (implicit or explicit).

(4) `-N` and `-1` only need a partial triangulation, whereas the options `-vq` require a full triangulation.

4.2.2. Combined modes

For convenience, Normaliz provides shortcut modes that combine basic modes:

`-n Hilbert basis volume`: combines `-N` and `-v`;
`-p Hilbert series degree 1 elements`.This mode combines `-q` and `-1`;
`-h Hilbert basis series`: combines `-N` and `-q`. This computation mode yields the maximum information Normaliz can produce.

Remarks: (1) There is no mode combining `-v` and `-1`. Simply use `-v1`.

(2) The effect of `-h` is also reached by the default mode. However, `-h` will result in termination if a grading cannot be found.

4.2.3. The dual algorithm

If a cone is defined by constraints, it is often (but not always) faster to use a Hilbert basis algorithm originally due to Pottier [12] that we call the *dual* algorithm, in contrast to the primal (triangulation based) algorithm of Normaliz. (See [5] for our version of the dual algorithm.) The dual algorithm is invoked by

`-d dual Hilbert basis`: computes the Hilbert basis using the dual algorithm;

`-d1 dual degree 1`: computes only the degree 1 elements.

Remarks: (1) The dual algorithm can be used with all input types. See Section 8.6 for a comparison of performance on various examples.

(2) `-d1` is optimized for the computation of degree 1 elements.

(3) The dual algorithm can be combined with the primal algorithm. For example, `-d1v` or `-dq` make perfect sense, and are often useful.

(4) If `-d` is set, the dual algorithm has priority in the computation of Hilbert bases, but not necessarily in the computation of degree 1 elements. For example, `-d1q` will bypass the dual algorithm, since the degree 1 elements can be gotten as a byproduct of `-q`.

4.2.4. Approximation of rational polytopes

Even the computation of degree 1 elements can be an extremely difficult problem in the primal as well as in the dual approach. A major obstruction in the primal algorithm is the occurrence of gigantic determinants of the simplicial cones in the triangulation. In this case the approximation of a rational polytope by a lattice polytope offers a way out:

`-r approximate rat polytope`: computes the lattice points in the degree 1 cross-section by approximation.

It makes no sense to combine `-r` with the computation of a Hilbert basis or Hilbert series or with the dual mode. In such case `-r` is bypassed. However, `-rv` makes sense.

4.2.5. What option do I use ...

The following table gives an overview of the computation options introduced so far.

	No enumerative data	multiplicity	Hilbert series
only extreme rays	`-s`	`-v`	`-q`
degree 1 elements	`-1` `-d1, -r`	`-v1` `-d1v, -rv`	`-p`
Hilbert basis	`-d` `-N`	`-dv` `-n`	default, `-h, -dq`

4.2.6. Modes calling NmzIntegrate

NmzIntegrate is an independent executable, but it can be called by Normaliz. The options are exactly those that would be used for a command line call of NmzIntegrate:

- `-E Generalized Ehrhart series`: computation of generalized Ehrhart series,
- `-L Leading coefficient`: computation of leading coefficient of generalized Ehrhart quasi-polynomial,
- `-I Integral`: computation of Lebesgue integrals.

See [8] for the details of NmzIntegrate. The options `-c` and `-x=<T>` are forwarded to NmzIntegrate.

The option `-E` contains `-y`, and `-I` and `-L` both contain `-T`. See Section 4.2.7 for `-T` and `-y`.

Note: the option `-F` of NmzIntegrate cannot be accessed via Normaliz.

4.2.7. Triangulation and Stanley decomposition

In some applications it may be useful to base further computations on the triangulation or even the Stanley decomposition computed by Normaliz. As already mentioned, NmzIntegrate needs these data:

- `-T` computes the triangulation and writes it to the file `<project>.tri`.
- `-y` computes the Stanley decomposition and writes it to the file `<project>.dec`.

Both modes generate further output files, namely `<project.>inv` and `<project>.tgn`. The format of these files will be explained in Section 7.

There is a further mode computing the triangulation, but producing no output:

- `-t` Computes the triangulation.

Why this option is sometimes useful will be explained in Section 8.7.

4.3. Computation modes: inhomogeneous input

The following options are *not allowed* for inhomogeneous input:

```
1prTyELI
```

The other modes are completely analogous to those for cones when one replaces "Hilbert basis" by "system of generators and Hilbert basis of recession monoid" (see Section A.6). Whether Normaliz computes a cone or a polyhedron does only depend on the input file and not on the command line options. In order to keep jNormaliz simple, the same full text names of the options are used for polyhedra.

4.3.1. Basic modes

`-s support hyperplanes`: only the constraints of the polyhedron and its vertices are computed.

All computation modes include `-s`.

`-N Hilbert basis`: computes the system of generators, namely the Hilbert basis of the recession cone and the minimal system of module generators.

`-v volume`: Normaliz computes the multiplicity.

`-q Hilbert series`: what it says.

Remarks: (1) Systems of generators can also be computed by the dual algorithm; see Section 4.2.3.

(2) The options `-vq` require the presence of a grading.

(3) `-N` only needs a partial triangulation, whereas the options `-vq` require a full triangulation.

4.3.2. Combination modes

`-n Hilbert basis volume`: combines `-N` and `-v`;

`-h Hilbert basis series`: combines `-N` and `-q`. This computation mode yields the maximum information Normaliz can produce.

The effect of `-h` is also reached by the default mode. However, `-h` will result in termination if a grading cannot be found.

4.3.3. The dual algorithm

`-d dual Hilbert basis`: computes the system of generators using the dual algorithm.

4.3.4. What option do I use ...

The following table gives an overview of the computation options introduced so far.

	No enumerative data	multiplicity	Hilbert series
only vertices	`-s`	`-v`	`-q`
Hilbert basis	`-d` `-N`	`-dv` `-n`	default, `-h, -dq`

4.4. Control of output files

In the default setting Normaliz writes only the output file `<project>.out` (and the files produced by `-T` and `-y`). The amount of output files can be increased as follows:

`-f` Normaliz writes the additional output files with suffixes `gen`, `cst`, and `inv`, provided the data of these files have been computed.

`-a` includes `-f`, Normaliz writes all available output files except the triangulation or the Stanley decomposition.

In order to see all available output files one uses `-aTy`.

The triangulation and the Stanley decomposition are treated separately since they can become very large and may exhaust memory if they must be stored for output.

For the list of potential output files and their interpretation see Section 7.

4.5. Control of execution

The options that control the execution are:

`-c` activates the verbose ("console") behavior of Normaliz in which Normaliz writes additional information about its current activities to the standard output.

`-e` activates the overflow error check of Normaliz. Ignored if used with -B.

`-B` switches Normaliz to infinite precision.

`-x=<T>` Here `<T>` stands for a positive integer limiting the number of threads that Normaliz is allowed access on your system. The default value is set by the operating system. If you want to run Normaliz in a strictly serial mode, choose `-x=1`.

The number of threads can also be controlled by the environment variable `OMP_NUM_THREADS`. See Section 8.6 for further discussion.

4.6. Numerical limitations

Even in low dimensions, the range of 64 bit integers may not be sufficient for the computations of Normaliz. Therefore `normaliz` can be switched to infinite precision by the option `-B`.

Computations with `-B` typically run about 5 times slower than those without it. In examples that look critical, it may be useful first to try `normaliz` without `-B`, but with the error check option activated. This costs time, too, but hardly more than 50% extra.

The user should run the example `critical64.in` in the subdirectory `examples` with `normaliz -e` in order to see the failure of 64 bit arithmetic. (Running it with `-B` takes a while and requires much memory.)

4.7. Obsolete options

The options `-i` and `-m` of version 2.2 have become obsolete. They will be ignored if present.

The options `-SVHP` of versions 2.5 and 2.7 are now synonymous with`-svhp` and can still be used.

5. The output file

The data you will find in the output file `<project>.out` depend on the input type and on the computation mode. The output file starts with an "abstract" that collects various numerical and qualitative data, for example the number of elements in the Hilbert basis. The lists of vectors, equations etc. follow the abstract.

Note that the values of results computed by Normaliz depend only on the input file, except that the order of vectors in lists like the Hilbert basis can vary, for example because of the unpredictability of the parallelization. The command line parameters only determine which results are computed and what algorithm is used.

5.1. The homogeneous case

The output file will contain the following data as far as computed:

- only for type `lattice_ideal`: the original system of generators (see below);
- the Hilbert basis H computed;
- the extreme rays of the cone C generated by H;
- the rank of the lattice $\mathbb{E}$;
- the embedding dimension (the rank of $\mathbb{A}$);
- the index of the lattice generated by the original input vectors in $\mathbb{E}$;
- the support hyperplanes of C;
- a system of equations defining the vector space generated by C;
- a system of congruences defining $\mathbb{E}$ as a sublattice of $\mathbb{A}$ (together with the equations);
- the number of simplicial cones in the triangulation and the sum of the absolute values of their determinants.

In the presence of a grading the following extra data may be printed:

- the linear form defining the degree;
- the degree 1 elements of the Hilbert basis;
- the multiplicity;
- the Hilbert series and the coefficients of the Hilbert (quasi)polynomial;
- the excluded faces (if any).

The degrees of the extreme rays are listed in the abstract. If the whole Hilbert basis is of degree 1, this fact is indicated. Moreover, Normaliz tells you whether the original system of generators contains the Hilbert basis by indicating whether the original monoid is integrally closed.

Please note:

(1) The equations and support hyperplanes *together* define the cone C. While support hyperplanes will always be present (except for the zero cone), equations will only be printed if necessary, namely when $\dim C < \operatorname{rank} \mathbb{A}$.

Similarly, congruences will only be printed if the lattice $\mathbb{E}$ is not given by $\mathbb{R}C \cap \mathbb{A}$. This can only happen with input matrices of types `normalization` or `congruences`.

Even if the cone and the lattice are defined by constraints, the inequalities, equations and congruences of the input will in general not be reproduced, but replaced by an equivalent system.

(2) The extreme rays are given by the first points in $\mathbb{E}$ on them (the extreme integral generators with respect to $\mathbb{E}$).

(3) In order to lift the grading from $\mathbb{E}$ to $\mathbb{A}$ it may be necessary to replace it by a multiple (in order to avoid fractions as coefficients). The necessary factor appears as "denominator". The Hilbert series and the Hilbert (quasi)polynomial do always refer to the degree in $\mathbb{E}$.

(4) Input matrices of type *generators* contain an explicit system of generators. For the other types different from `lattice_ideal` the extreme rays computed by Normaliz take their place. For type `lattice_ideal` Normaliz first computes the monoid M generated by the residue classes of the canonical basis of $\mathbb{Z}^n$ (compare Section 3.3), and they are considered as the original system of generators.

In type `rees_algebra`, the data in the output file refer to the integral closure $\overline{\mathscr{R}}$ of the Rees algebra. In addition to what has been mentioned already, the following data are computed:

- the generators of the integral closure of the ideal;
- if the ideal is primary to the irrelevant maximal ideal, the multiplicity of the ideal (not to be confused with the multiplicity of the monoid).

5.2. The inhomogeneous case

Note: All data are presented in *homogenized coordinates*. For inhomogeneous input types with implicit dehomogenization this means that all vectors with last component 0 belong to the recession cone, and those with last component > 0 represent a rational point in the polyhedron with denominator given by the last coordinate. In constraints the last coordinate represents the negative of the right hand side, as in the input.

If the dehomogenization is another coordinate, the interpretation is analogous. If the dehomogenization δ is not a coordinate, one must compute it by applying the truncating linear form to the vectors given: the denominator is $\delta(x)$ for the points of the polyhedron. Solving the linear equation $\delta(x) = 1$ for one coordinate and substituting the result into the constraints, one obtains inhomogeneous versions of the constraints if they should be needed.

In the inhomogeneous case we compute a polyhedron P. We can find the following lists of vectors in the output file describing the solution monoid:

- the dehomogenization,
- the generators of the solution module,
- the Hilbert basis of the solution module.

The convex-geometric data of the polyhedron are given by:

- the vertices of the polyhedron (with denominator $\delta(x)$),
- the extreme rays of the recession cone.

The support hyperplanes, equations and congruences have the same meaning as in the homogeneous case: they describe the polyhedron and the affine lattice with respect to which the solution monoid is computed.

The module rank and the rank of the recession monoid are always computed, as well as the affine dimension of the polyhedron P.

If a grading is given, then one can also compute

- the multiplicity of the solution monoid (the suitably normed leading coefficient of the Hilbert (quasi)polynomial),
- the Hilbert series given as a rational function with numerator and denominator as in the homogeneous case, however modified by
- a shift, and
- the Hilbert (quasi)polynomial, provided the period is not too large.

The shift corrects the Hilbert series as follows:

$$H_{\text{true}}(t) = t^{-\text{shift}} H_{\text{computed}}(t).$$

The Hilbert (quasi)polynomial must be shifted correspondingly:

$$q_{\text{true}}(k) = q_{\text{computed}}(k + \text{shift}).$$

At the end we find the system of constraints that defines the polyhedron P and the set of lattice points in it.

The remarks in the homogeneous case apply accordingly.

6. Examples

Note that the output you get by running the examples may differ from the one given below in the order of the vectors in lists like the Hilbert basis.

6.1. Generators of cones and lattices

6.1.1. Type `integral_closure`

The file `rproj2.in` contains the following:

```
16 7
 1 0 0 0 0 0 0     0 0 0 0 1 0 0     1 0 1 0 1 0 1     0 1 0 1 1 0 1
 0 1 0 0 0 0 0     0 0 0 0 0 1 0     1 0 0 1 0 1 1     0 1 0 0 1 1 1
 0 0 1 0 0 0 0     1 1 1 0 0 0 1     1 0 0 0 1 1 1     0 0 1 1 1 0 1
 0 0 0 1 0 0 0     1 1 0 1 0 0 1     0 1 1 0 0 1 1     0 0 1 1 0 1 1
                                                     integral_closure
```

This means that we wish to compute the cone C generated by the 16 vectors

$$(1,0,0,0,0,0,0)\,,\quad (0,1,0,0,0,0,0)\,,\quad \ldots\,,\quad (0,0,1,1,0,1,1)$$

in $\mathbb{R}^7$ with respect to the full lattice $\mathbb{Z}^7$, as indicated by the type `integral_closure`. (Actually, the vectors generate the full lattice so that a replacement of type `integral_closure` by type `normalization` would not change anything.)

Running `normaliz` with no option (or option `-h`, `Hilbert basis series`) produces the file `rproj2.out` which has the following content (partially typeset in 2 or 3 columns):

```
17 Hilbert basis elements                              multiplicity = 72
16 Hilbert basis elements of degree 1
16 extreme rays                                        Hilbert series:
24 support hyperplanes                                 1 9 31 25 6
                                                       denominator with 7 factors:
embedding dimension = 7                                1: 7
rank = 7 (maximal)
index = 1                                              Hilbert polynomial:
original monoid is not integrally closed               60 194 284 245 130 41 6
                                                       with common denominator = 60
size of triangulation   = 67
resulting sum of |det|s = 72

grading:
1 1 1 1 1 1 -2

degrees of extreme rays:
1: 16

Hilbert basis elements are not of degree 1

******************************************************************

17 Hilbert basis elements:                 24 support hyperplanes:
 0 0 0 0 0 1 0                               0  0  0  1  0  0  0
 0 0 0 0 1 0 0                               0  0  0  0  1  0  0
 0 0 0 1 0 0 0                               0  0  0  0  0  1  0
 0 0 1 0 0 0 0                               0  0  0  0  0  0  1
 0 0 1 1 0 1 1                               0  0  1  0  0  0  0
 0 0 1 1 1 0 1                               0  1  0  0  0  0  0
 0 1 0 0 0 0 0                               0  1  1  0  0  1 -1
 0 1 0 0 1 1 1                               0  1  0  0  1  1 -1
 0 1 0 1 1 0 1                               0  1  0  1  1  0 -1
 0 1 1 0 0 1 1                               0  0  1  1  0  1 -1
 1 0 0 0 0 0 0                               0  1  1  1  1  1 -2
 1 0 0 0 1 1 1                               0  0  1  1  1  0 -1
 1 0 0 1 0 1 1                               1  0  0  0  0  0  0
```

```
 1 0 1 0 1 0 1
 1 1 0 1 0 0 1
 1 1 1 0 0 0 1
 1 1 1 1 1 1 2

16 extreme rays:
 1 0 0 0 0 0 0
 0 1 0 0 0 0 0
 0 0 1 0 0 0 0
 0 0 0 1 0 0 0
 0 0 0 0 1 0 0
 0 0 0 0 0 1 0
 1 1 1 0 0 0 1
 1 1 0 1 0 0 1
 1 0 1 0 1 0 1
 1 0 0 1 0 1 1
 1 0 0 0 1 1 1
 0 1 1 0 0 1 1
 0 1 0 1 1 0 1
 0 1 0 0 1 1 1
 0 0 1 1 1 0 1
 0 0 1 1 0 1 1

 1  1  1  1  1  1 -3
 1  0  0  0  1  1 -1
 1  0  0  1  0  1 -1
 1  0  1  0  1  0 -1
 1  0  1  1  1  1 -2
 1  1  1  0  0  0 -1
 1  1  1  1  0  1 -2
 1  1  0  1  0  0 -1
 1  1  1  0  1  1 -2
 1  1  1  1  1  0 -2
 1  1  0  1  1  1 -2

16 degree 1 Hilbert basis elements:
 0 0 0 0 0 1 0
 0 0 0 0 1 0 0
 0 0 0 1 0 0 0       0 1 1 0 0 1 1
 0 0 1 0 0 0 0       1 0 0 0 0 0 0
 0 0 1 1 0 1 1       1 0 0 0 1 1 1
 0 0 1 1 1 0 1       1 0 0 1 0 1 1
 0 1 0 0 0 0 0       1 0 1 0 1 0 1
 0 1 0 0 1 1 1       1 1 0 1 0 0 1
 0 1 0 1 1 0 1       1 1 1 0 0 0 1
```

From this, we see that there are 17 elements in the Hilbert basis, of which 16 are of degree 1, and 16 extreme rays, that the sublattice generated by the input vectors has index 1 in $\mathbb{Z}^7$, and that the corresponding support hyperplanes are given by the linear forms $(0,0,0,1,0,0,0)$, $(0,0,0,0,1,0,0)$, $\ldots$, $(1,1,0,1,1,1,-2)$.

We are also given the information that there is a grading (defined implicitly) and what it is. The multiplicity with respect to this grading is 72. By definition, the multiplicity is the $\mathbb{E}$-normalized volume of the polytope obtained by intersecting the cone with the hyperplane at degree 1.

The degrees of the extreme rays are given in multiset notation:

```
1:  16
```

indicates that 16 extreme rays have degree 1. (The input file contains no explicit grading. The implicitly defined grading requires that all extreme rays have the same degree, but it need not be 1 as in this case.)

Since there is a grading, the degree 1 elements of the Hilbert basis, the Hilbert series and Hilbert polynomial of the monoid generated by the Hilbert basis are also computed. The Hilbert series is given as a rational function. Its numerator polynomial is

$$1+9t+31t^2+25t^3+6t^4$$

as we can see from the vector below the heading `Hilbert series`. The denominator is given in multiset notation: `1:  7` specifies the denominator

$$(1-t^1)^7.$$

More general cases will be discussed in 6.1.3 and 6.2.3 below.

The Hilbert polynomial is given by

$$P(k) = \frac{60}{60} + \frac{194}{60}k + \frac{284}{60}k^2 + \frac{245}{60}k^3 + \frac{130}{60}k^4 + \frac{41}{60}k^5 + \frac{6}{60}k^6\,.$$

The Hilbert polynomial gives the number of elements of degree k, starting from degree 0, as is always the case for normal monoids. Note that the multiplicity m can also be read from the leading coefficient c of the Hilbert polynomial:

$$c = \frac{m}{(r-1)!}, \qquad r = \text{rank}, \tag{1}$$

in our case

$$c = \frac{1}{10} = \frac{72}{720}.$$

The lines

```
size of triangulation = 67
resulting sum of |det|s = 72
```

give some information about Normaliz' (not so hard) work: It produced a triangulation of 67 simplicial cones, and the sum of the absolute values of the determinants of their generator matrices is 72. It is not surprising that this number equals the multiplicity. This is always the case if only degree 1 vectors appear in the generator matrix.

We omit an example of type `normalization` since it does not add anything new.

6.1.2. Type `polytope`

The file `polytop.in`contains

```
4
3
0 0 0
2 0 0
0 3 0
0 0 5
polytope
```

The Ehrhart monoid of the integral polytope with the 4 vertices

$$(0,0,0)\,, \quad (2,0,0)\,, \quad (0,3,0) \quad \text{and} \quad (0,0,5)$$

in $\mathbb{R}^3$ is to be computed. (Note the last line, indicating the polytope type.)

Running `normaliz` without an option (or option `-h`) produces the file `polytop.out`:

```
19 Hilbert basis elements                          multiplicity = 30
18 Hilbert basis elements of degree 1
4 extreme rays                                     Hilbert series:
4 support hyperplanes                              1 14 15
                                                   denominator with 4 factors:
embedding dimension = 4                            1: 4
rank = 4 (maximal)
index = 30                                         Hilbert polynomial:
original monoid is not integrally closed           1 4 8 5
                                                   with common denominator = 1
size of triangulation   = 1
resulting sum of |det|s = 30

grading:
0 0 0 1

degrees of extreme rays:
1: 4

Hilbert basis elements are not of degree 1

**********************************************************************

19 Hilbert basis elements:     18 Hilbert basis elements of degree 1:
 0 0 5 1                        0 0 5 1
 0 3 0 1                        0 3 0 1
 ...                            ...
 1 0 2 1                        1 0 2 1
 1 1 0 1                        1 1 0 1
 1 2 4 2

4 extreme rays:                4 support hyperplanes:
 0 0 0 1                        -15 -10  -6  30
 2 0 0 1                          1   0   0   0
 0 3 0 1                          0   1   0   0
 0 0 5 1                          0   0   1   0
```

For the polytopal interpretation one must observe that all data are in homogenized coordinates for which Normaliz has appended 1 to the input vectors that (in this case) are the vertices of the polytope. In the cone produced, the lattice points of the polytope are of degree 1. Therefore the 18 Hilbert basis elements of degree 1 represent the lattice points of the polytope, starting from $(0,0,5)$ and ending with $(1,1,0)$. The extreme rays represent the 4 vertices.

From the fact that there are 19 Hilbert basis elements, but only 18 of degree 1, we see that the lattice points in the polytope do not yield the Hilbert basis of the Ehrhart monoid (or the cone over the polytope).

That there is only one simplicial cone in the triangulation is not surprising since our polytope is a simplex.

The support hyperplanes give us a description of the polytope by inequalities: it is the solution of the system of the 4 inequalities

$$x_3 \geq 0\,, \quad x_2 \geq 0\,, \quad x_1 \geq 0 \quad \text{and} \quad 15x_1 + 10x_2 + 6x_3 \leq 30\,.$$

The dimension of the polytope is 3 since the cone over it has dimension 4. The polytope has $\mathbb{Z}^3$-normalized volume 30 as indicated by the multiplicity.

The Ehrhart series (we use the more general term Hilbert series) is

$$\frac{1 + 14t + 15t^2}{(1-t)^4}$$

and its Ehrhart polynomial (again we use a more general term in the output file) of the polytope is

$$p(k) = 1 + 4k + 8k^2 + 5k^3\,.$$

6.1.3. A rational polytope

We want to investigate the Ehrhart series of the triangle P with vertices

$$(1/2, 1/2),\ (-1/3, -1/3),\ (1/4, -1/2).$$

The input file is `rational.in`. Running Normaliz yields the following output:

```
8 Hilbert basis elements                      H...t s...s  cycl... denom...:
1 Hilbert basis elements of degree 1          -1 -1 -1 -3 -4 -3 -2
3 extreme rays                                cyclotomic denominator:
3 support hyperplanes                         1: 3  2: 2  3: 1  4: 1

embedding dimension = 3                       H...t quasi-pol...l of period 12:
rank = 3 (maximal)                              0:   48  28  15
index = 15                                      1:   11  22  15
original monoid is not integrally closed        2:  -20  28  15
                                                3:   39  22  15
size of triangulation   = 1                     4:   32  28  15
resulting sum of |det|s = 15                    5:   -5  22  15
                                                6:   12  28  15
grading:                                        7:   23  22  15
0 0 1                                           8:   16  28  15
                                                9:   27  22  15
degrees of extreme rays:                       10:   -4  28  15
2: 1  3: 1  4: 1                               11:    7  22  15
                                              with common denominator = 48
multiplicity = 5/8
```

```
Hilbert series:
1 0 0 3 2 -1 2 2 1 1 1 1 2
denominator with 3 factors:
1: 1  2: 1  12: 1

*********************************************************************

8 Hilbert basis elements:          3 extreme rays:
  1 -2  4                            1  1  2
 -1 -1  3                           -1 -1  3
  1  1  2                            1 -2  4
  0  0  1
  0 -1  3                          3 support hyperplanes:
  1  0  3                            2  7  3
  1 -1  4                           -8  2  3
  0 -2  5                            1 -1  0

                                   1 Hilbert basis elements of degree 1:
                                    0 0 1
```

The 3 extreme rays have reproduced the vertices (don't forget that the last coordinate can be interpreted as a denominator), and the 3 support hyperplanes represent the 3 inequalities that define the polytope as an intersection of affine halfspaces like in 6.1.2. The Hilbert basis element of degree 1 shows that there is a single lattice point in the polytope, namely the origin $(0,0)$. Except that P has non-integral vertices now, these data are completely analogous to those of the lattice polytope in 6.1.2.

The multiplicity is a rational number. Since in dimension 2 the normalized area (of full-dimensional polytopes) is twice the Euclidean area, we see that P has Euclidean area $5/16$.

The Hilbert (or Ehrhart) function counts the lattice points in kP, $k \in \mathbb{Z}_+$. The corresponding generating function is a rational function $H(t)$ with numerator

$$1+3t^3+2t^4-t^5+2t^6+2t^7+t^8+t^9+t^{10}+t^{11}+2t^{12}$$

and denominator

$$(1-t)(1-t^2)(1-t^{12}).$$

As a rational function, $H(t)$ has degree -3. This implies that $3P$ is the smallest integral multiple of P that contains a lattice point in its interior.

Normaliz gives also a representation as a quotient of coprime polynomials with the denominator factored into cyclotomic polynomials. The multiset notation lists the orders of the cyclotomic polynomials and their multiplicities. In this case we have

$$H(t) = -\frac{1+t+t^2+t^3+4t^4+3t^5+2t^6}{\zeta_1^3\zeta_2^2\zeta_3\zeta_4}$$

where ζ_i is the i-th cyclotomic polynomial ($\zeta_1 = t-1$, $\zeta_2 = t+1$, $\zeta_3 = t^2+t+1$, $\zeta_4 = t^2+1$).

Normaliz transforms the representation with cyclotomic denominator into one with denominator of type $(1-t^{e_1})\cdots(1-t^{e_r})$, $r =$ rank, by choosing e_r as the least common multiple of all the orders of the cyclotomic polynomials appearing, e_{r-1} as the lcm of those orders that have multiplicity ≥ 2 etc.

There are other ways to form a suitable denominator with 3 factors $1-t^e$, for example $g(t) = (1-t^2)(1-t^3)(1-t^4) = -\zeta_1^3\zeta_2^2\zeta_3\zeta_4$. Of course, $g(t)$ is the optimal choice in this case. However, P is a simplex, and in general such optimal choice may not exist. We will explain the reason for our standardization below.

Let $p(k)$ be the number of lattice points in kP. Then $p(k)$ is a quasipolynomial:

$$p(k) = p_0(k) + p_1(k)k + \cdots + p_{r-1}(k)k^{r-1},$$

where the coefficients depend on k, but only to the extent that they are periodic of a certain period $\pi \in \mathbb{N}$. In our case $\pi = 12$ (the lcm of the orders of the cyclotomic polynomials).

The table giving the quasipolynomial is to be read as follows: The first column denotes the residue class j modulo the period and the corresponding line lists the coefficients $p_i(j)$ in ascending order of i, multiplied by the common denominator. So

$$p(k) = 1 + \frac{7}{12}k + \frac{5}{16}k^2, \qquad k \equiv 0 \pmod{12},$$

etc. The leading coefficient is the same for all residue classes and equals the Euclidean volume as in equation (1).

Our choice of denominator for the Hilbert series is motivated by the following fact: e_i is the common period of the coefficients $p_{r-i}, \ldots, p_{r-1}$. The user should prove this fact or at least verify it by several examples.

6.1.4. Type `rees_algebra`

Next, let us discuss the example `rees.in`:

```
10
6                       0 1 1 0 0 1
1 1 1 0 0 0             0 1 0 1 1 0
1 1 0 1 0 0             0 1 0 0 1 1
1 0 1 0 1 0             0 0 1 1 1 0
1 0 0 1 0 1             0 0 1 1 0 1
1 0 0 0 1 1             rees_algebra
```

A comparison with the data in `rproj2.in` shows that `rees` is the origin of `rproj2`.

Here we want to compute the integral closure of the Rees algebra of the ideal generated by the monomials corresponding to the above 10 exponent vectors. The output in `rees.out` coincides with that in `rproj2.out`, up to terminology and the supplementary information on the integral closure of the ideal:

```
10 generators of integral closure of the ideal:
 0 0 1 1 0 1
 0 0 1 1 1 0
 0 1 0 0 1 1                1 0 0 1 0 1
 0 1 0 1 1 0                1 0 1 0 1 0
 0 1 1 0 0 1                1 1 0 1 0 0
 1 0 0 0 1 1                1 1 1 0 0 0
```

A brief look at `rproj2.out` shows that exactly the generators with the last coordinate 1 have been extracted. (So the ideal is integrally closed. This is not surprising because we have chosen squarefree monomials.)

6.2. Homogeneous constraints

6.2.1. Type `inequalities`

The file `dual.in` is

```
24
7
0 0 0 1 0 0  0             1 0 0 0 0 0  0
0 0 0 0 1 0  0             1 1 1 1 1 1 -3
...                        ...
0 0 1 1 0 1 -1             1 1 1 1 1 0 -2
0 1 1 1 1 1 -2             1 1 0 1 1 1 -2
                           inequalities
```

This means that we wish to compute the Hilbert basis of the cone cut out from $\mathbb{R}^7$ by the 24 inequalities. (It is the dual of the cone spanned by the 24 linear forms in $(\mathbb{R}^7)^*$.) The inequalities represent exactly the support hyperplanes from the file `rproj2.out`. The output in `dual.out` coincides with that in `rproj2.out`.

6.2.2. Type `equations`

Suppose that you have the following "square"

x_1	x_2	x_3
x_4	x_5	x_6
x_7	x_8	x_9

and the problem is to find nonnegative values for $x_1, \ldots, x_9$ such that the 3 numbers in all rows, all columns, and both diagonals sum to the same constant $\mathscr{M}$. Sometimes such squares are

called *magic* and $\mathscr{M}$ is the *magic constant*. This leads to a linear system of equations

$$\begin{aligned}
x_1+x_2+x_3 &= x_4+x_5+x_6;\\
x_1+x_2+x_3 &= x_7+x_8+x_9;\\
x_1+x_2+x_3 &= x_1+x_4+x_7;\\
x_1+x_2+x_3 &= x_2+x_5+x_8;\\
x_1+x_2+x_3 &= x_3+x_6+x_9;\\
x_1+x_2+x_3 &= x_1+x_5+x_9;\\
x_1+x_2+x_3 &= x_3+x_5+x_7.
\end{aligned}$$

This system of equations is contained in the file `3x3magic.in`. It has input type `equations`. (Don't forget that the sign conditions $x_i \geq 0$ are automatically included if there are no explicit inequalities.)

The magic constant is a natural choice for the grading, and therefore

```
1
9
1 1 1 0 0 0 0 0 0
grading
```

follows the equations.

The output file contains the following:

```
5 Hilbert basis elements                        multiplicity = 4
5 Hilbert basis elements of degree 1
4 extreme rays                                  Hilbert series:
4 support hyperplanes                           1 2 1
                                                denominator with 3 factors:
embedding dimension = 9                         1: 3
rank = 3
index = 2                                       Hilbert polynomial:
original monoid is not integrally closed        1 2 2
                                                with common denominator = 1
size of triangulation   = 2
resulting sum of |det|s = 4

grading:
1 1 1 0 0 0 0 0 0
with denominator = 3

degrees of extreme rays:
1: 4

Hilbert basis elements are of degree 1
```

```
**********************************************************************

5 Hilbert basis elements:
 1 0 2 2 1 0 0 2 1
 0 2 1 2 1 0 1 0 2
 2 0 1 0 1 2 1 2 0
 1 2 0 0 1 2 2 0 1
 1 1 1 1 1 1 1 1 1

4 extreme rays:
 1 2 0 0 1 2 2 0 1
 2 0 1 0 1 2 1 2 0
 0 2 1 2 1 0 1 0 2
 1 0 2 2 1 0 0 2 1

4 support hyperplanes:
  0 -1  0  0  2  0  0  0  0
  0  1  2  0 -2  0  0  0  0
  0 -1 -2  0  4  0  0  0  0
  0  1  0  0  0  0  0  0  0

6 equations:
 -2  1  4 -3  0  0  0  0  0
 -1  0  1 -1  1  0  0  0  0
 -2  0  2 -1  0  1  0  0  0
 -2  0  3 -2  0  0  1  0  0
  0  0 -2  1  0  0  0  1  0
 -1  0  2 -2  0  0  0  0  1

5 Hilbert basis elements of degree 1:
 1 0 2 2 1 0 0 2 1
 0 2 1 2 1 0 1 0 2
 2 0 1 0 1 2 1 2 0
 1 2 0 0 1 2 2 0 1
 1 1 1 1 1 1 1 1 1
```

The 5 elements of the Hilbert basis represent the magic squares

2	0	1
0	1	2
1	2	0

1	0	2
2	1	0
0	2	1

1	1	1
1	1	1
1	1	1

1	2	0
0	1	2
2	0	1

0	2	1
2	1	0
1	0	2

All other solutions are linear combinations of these squares with nonnegative integer coefficients.

Normaliz tells us that the cone generated by the magic squares can be described by 4 inequalities and 6 linear relations. The number of equations becomes clear when we look at the rank.

The input degree is the magic constant. However, as the denominator 3 shows, the magic constant is always divisible by 3, and therefore the effective degree is $\mathscr{M}/3$. This degree is used for the multiplicity and the Hilbert series.

The Hilbert series is

$$\frac{1+2t+t^2}{(1-t)^3}.$$

The Hilbert polynomial is

$$P(k) = 1+2k+2k^2,$$

and after substituting $\mathscr{M}/3$ for k we obtain the number of magic squares of magic constant $\mathscr{M}$.

6.2.3. Type `congruences`

We change our definition of magic square by requiring that the entries in the 4 corners are all even. Then we have to augment the input file as follows (`3x3magiceven.in`):

```
7                              4
9                              10
1 1 1 -1 -1 -1  0  0  0        1 0 0 0 0 0 0 0 0 2
1 1 1  0  0  0 -1 -1 -1        0 0 1 0 0 0 0 0 0 2
0 1 1 -1  0  0 -1  0  0        0 0 0 0 0 0 1 0 0 2
1 0 1  0 -1  0  0 -1  0        0 0 0 0 0 0 0 0 1 2
1 1 0  0  0 -1  0  0 -1        congruences
0 1 1  0 -1  0  0  0 -1        1
1 1 0  0 -1  0 -1  0  0        9
equations                      1 1 1 0 0 0 0 0 0
                               grading
```

The output changes accordingly:

```
9 Hilbert basis elements                        multiplicity = 1
0 Hilbert basis elements of degree 1
4 extreme rays                                  Hilbert series:
4 support hyperplanes                           1 -1 3 1
                                                denominator with 3 factors:
embedding dimension = 9                         1: 1  2: 2
rank = 3
index = 4                                       H...t s...s  cycl... denom...:
original monoid is not integrally closed        -1 1 -3 -1
                                                cyclotomic denominator:
size of triangulation   = 2                     1: 3  2: 2
resulting sum of |det|s = 8
                                                H...t quasi-pol...l of period 2:
grading:                                         0:   2  2  1
1 1 1 0 0 0 0 0 0                                1:  -1  0  1
with denominator = 3                            with common denominator = 2

degrees of extreme rays:
2: 4

**********************************************************************

9 Hilbert basis elements:        4 support hyperplanes:
 2 0 4 4 2 0 0 4 2                 1  0  1  0 -1  0  0  0  0
 0 4 2 4 2 0 2 0 4                -1  0  1  0  1  0  0  0  0
 4 0 2 0 2 4 2 4 0                 1  0 -1  0  1  0  0  0  0
 2 4 0 0 2 4 4 0 2                -1  0 -1  0  3  0  0  0  0
 2 2 2 2 2 2 2 2 2
```

```
 4 3 2 1 3 5 4 3 2
 2 5 2 3 3 3 4 1 4
 2 3 4 5 3 1 2 3 4
 4 1 4 3 3 3 2 5 2

4 extreme rays:
 2 4 0 0 2 4 4 0 2
 4 0 2 0 2 4 2 4 0
 0 4 2 4 2 0 2 0 4
 2 0 4 4 2 0 0 4 2
```

```
6 equations:
 -2  1  4 -3  0  0  0  0  0
 -1  0  1 -1  1  0  0  0  0
 -2  0  2 -1  0  1  0  0  0
 -2  0  3 -2  0  0  1  0  0
  0  0 -2  1  0  0  0  1  0
 -1  0  2 -2  0  0  0  0  1

2 congruences:
 0 0 1 0 0 0 0 0 0 2
 1 0 0 0 0 0 0 0 0 2

0 Hilbert basis elements of degree 1:
```

It is not surprising that the support hyperplanes have not changed after the introduction of the congruences, since the latter only modify the lattice $\mathbb{E}$. Similarly the number of extreme rays is the same, but the vectors are multiplied by the factor 2 since Normaliz chooses them in $\mathbb{E}$, and therefore these vectors must satisfy the congruences.

Its first representation tells us that the Hilbert series is

$$\frac{1-t+3t^2+t^3}{(1-t)(1-t^2)^2}.$$

As in 6.1.3, the second representation gives coprime numerator and denominator polynomials in which the denominator is a product of cyclotomic polynomials:

$$\frac{-1+t-3t^2-t^3}{\zeta_1^3\zeta_2^2}, \qquad \zeta_1 = t-1, \zeta_2 = t+1.$$

In this case, the two denominators differ by the factor -1. In general, the first representation is not coprime, as we have seen in 6.1.3.

The lattice point enumerator is a quasipolynomial of period 2:

$$|\{x : \deg x = k\}| = \begin{cases} 1+k+k^2/2, & k \equiv 0 \quad (2) \\ -1/2+k^2/2, & k \equiv 1 \quad (2). \end{cases}$$

In general one must expect a non-integral multiplicity if the period is > 1. That the multiplicity is integral, namely 1, in this case must be considered an exception.

As you can see, the equations make two of the input congruences superfluous: it is enough to require the two corners in the first row to be even. The first congruence is to be read as $x_1 \equiv 0 \mod 2$, the second as $x_3 \equiv 0 \mod 2$.

Another good example for Hilbert series and gradings is given by `Condorcet.in`. The reader should run it or have a look at the corresponding output file.

6.3. Relations

6.3.1. Type `lattice_ideal`

As an example, we consider the binomial ideal generated by

$$X_1^2X_2 - X_4X_5X_6,\ X_1X_4^2 - X_3X_5X_6,\ X_1X_2X_3 - X_5^2X_6.$$

We want to find an embedding of the toric ring it defines and the normalization of the toric ring.

The input ideal `lattice_ideal.in` is

```
3
6
2 1  0 -1 -1  -1
1 0 -1  2 -1  -1
1 1  1  0 -2  -1
lattice_ideal
```

It yields the output

```
6 original generators of the toric ring          multiplicity = 10
9 Hilbert basis elements
9 Hilbert basis elements of degree 1             Hilbert series:
5 extreme rays                                   1 6 3
5 support hyperplanes                            denominator with 3 factors:
                                                 1: 3
embedding dimension = 3
rank = 3 (maximal)                               Hilbert polynomial:
index = 1                                        1 3 5
original monoid is not integrally closed         with common denominator = 1

size of triangulation   = 3
resulting sum of |det|s = 10

grading:
1 -1 1

degrees of extreme rays:
1: 5

Hilbert basis elements are of degree 1

**********************************************************************

6 original generators:              5 support hyperplanes:
 0 1 2                                1  0  0
```

```
 3 2 0                                  0  1  0
 0 0 1                                  0  0  1
 1 1 1                                  3 -2  1
 1 0 0                                  6 -9  7
 1 3 3
                                      10 Hilbert basis elements of degree 1:
10 Hilbert basis elements:             1 3 3
 1 3 3                                 1 0 0
 ...                                   ...
 2 2 1                                 1 1 1
 1 1 1

5 extreme rays:
 0 1 2
 ...
 1 3 3
```

The 6 original generators correspond to the indeterminates $X_1, \ldots, X_6$ in the binomial equations. They represent an embedding of the affine monoid defined by the binomial equations.

6.4. Type `excluded_faces`

We choose the input file `SquareMinusVertex.in`:

```
4                       1
2                       3
0 0                     1 1 0
0 1                     excluded_faces
1 1
1 0
polytope
```

This defines the unit square Q in $\mathbb{R}^2$, and we want to compute the Hilbert function of the cone over Q minus the ray through the point $(0,0,1)$ which represents the left lower corner of the square.

Let us restrict ourselves to the part of the output file that differs from the one obtained without `excluded_faces` (4 Hilbert basis elements, all of degree 1 etc.):

```
1 excluded faces
...
Hilbert series:                 Hilbert polynomial:
0 3 -1                          0 2 1
denominator with 3 factors:     with common denominator = 1
1: 3
...
```

```
1 excluded faces:
 1 1 0
```

So the Hilbert series is

$$\frac{3t-t^2}{(1-t)^3} = \frac{1+t}{(1-t)^3} - \frac{1}{1-t}$$

what we could have computed by hand.

6.5. Generators of polyhedra

6.5.1. Type polyhedron

Let us realize the example of Section 6.4 via the type `polyhedron`. There are of course several solutions. We choose `SquareMinusVertexPolyh.in`:

```
7 4
0 0 1 0              1 3
1 0 1 0              0 0 1
1 1 1 0              grading
0 1 1 0
1 1 1 2
0 1 1 2
1 0 1 2
polyhedron
```

This input defines a polyhedron in $\mathbb{R}^3$ with three vertices, namely $(1/2,1/2,1/2)$, $(0,1/2,1/2)$, $(1/2,0,1/2)$, as given by the last 3 rows of the input matrix. (This choice of vertices has purely didactical reasons.) The recession cone is simply the cone over the unit square, and its four rays are determined by the first 4 rows of the input matrix. This yields the output

```
3 module generators                                  module rank = 1
4 Hilbert basis elements of recession monoid         multiplicity = 2
3 vertices of polyhedron
4 extreme rays of recession cone                     Hilbert series:
6 support hyperplanes of polyhedron                  0 3 -1
                                                     denominator with 3 factors:
embedding dimension = 4                              1: 3
affine dimension of the polyhedron = 3 (maximal)
rank of recession monoid = 3                         shift = 0

size of triangulation   = 5                         Hilbert polynomial:
resulting sum of |det|s = 4                          0 2 1
                                                     with common denominator = 1
dehomogenization:
0 0 0 1
```

```
grading:
0 0 1 0

**************************************************************************

3 module generators:                4 extreme rays of recession cone:
 0 1 1 1                             0 0 1 0
 1 0 1 1                             1 0 1 0
 1 1 1 1                             1 1 1 0
                                     0 1 1 0

4 Hilbert basis elements of recession monoid:
 0 0 1 0
 0 1 1 0
 1 0 1 0                          6 support hyperplanes of polyhedron:
 1 1 1 0                           -2  0  2  1
                                    0 -2  2  1
3 vertices of polyhedron:           1  0  0  0
 1 1 0 2                            2  2  0 -1
 0 1 0 2                            0  1  0  0
 1 0 0 2                            0  0  1  0
```

We see that the vertices of P, and the extreme rays and the Hilbert basis of the recession cone are as expected. The only comment they need is that they are given in homogenized coordinates with last component 0, although they are vectors in $\mathbb{R}^3$.

The vertices of the polyhedron are exactly as expected. (A priori it may not be clear whether each of the points in the input is a vertex of the polyhedron.)

Also the interpretation of the support hyperplanes should be obvious. For example, `2  2  0 -1` represents the inequality

$$2\xi_1 + 2\xi_2 - 1 \geq 0.$$

(Which of the support hyperplanes define(s) a compact face?)

As a new type of data we see a shift. We cannot avoid using it since negative degrees are not excluded, and we want to avoid true Laurent polynomials in the numerator of the Hilbert series. We know already that

$$\frac{3t - t^2}{(1-t)^3}$$

is the Hilbert series, whereas the raw result just computed is

$$\frac{3-t}{(1-t)^3}.$$

So the correction is

$$H_{\text{true}}(t) = t^{-\text{shift}} H_{\text{computed}}(t).$$

The Hilbert (quasi)polynomial must be shifted accordingly:

$$q_{\text{true}}(k) = q_{\text{computed}}(k + \text{shift}).$$

The module rank is 1, and the multiplicity is the same as that of the recession monoid since the leading terms in the Hilbert polynomial agree.

6.6. Inhomogeneous constraints

6.6.1. Type `inhom_inequalities`

Let $x = (-2,-1/2)$, $y = (0,3/2)$ and let Q be the line segment joining x and y. We want to model the polyhedron $Q + \mathbb{R}_+(1,0)$ by inequalities. These are

$$\begin{aligned} \xi_2 &\geq -1/2, \\ \xi_2 &\leq 3/2, \\ \xi_2 &\leq \xi_1 + 3/2 \end{aligned}$$

and yield the input `InhomIneq.in`:

```
3                   1
3                   2
0  2  1             1 0
0 -2  3             grading
2 -2  3
inhom_inequalities
```

It yields the output

```
2 module generators
1 Hilbert basis elements of recession monoid
2 vertices of polyhedron
1 extreme rays of recession cone
3 support hyperplanes of polyhedron                 module rank = 2
                                                    multiplicity = 2
embedding dimension = 3
affine dimension of the polyhedron = 2 (maximal)    Hilbert series:
rank of recession monoid = 1                        0 1 1
                                                    denominator with 1 factors:
size of triangulation   = 1                         1: 1
resulting sum of |det|s = 8
                                                    shift = 2
dehomogenization:
0 0 1                                               Hilbert polynomial:
                                                    2
grading:                                            with common denominator = 1
1 0 0
```

```
**************************************************************************

2 module generators:            1 extreme rays of recession cone:
 -1  0  1                        1 0 0
  0  1  1

1 Hilbert basis elements of recession monoid:
 1 0 0

2 vertices of polyhedron:       3 support hyperplanes of polyhedron:
 -4 -1  2                         0  2  1
  0  3  2                         0 -2  3
                                   2 -2  3
```

The module rank is 2 in this case since we have two "layers" in the solution module that are parallel to the recession monoid. The Hilbert series is

$$t^{-2}\,\frac{t+t^2}{1-t} = \frac{t^{-1}+1}{1-t}$$

It reflects the disjoint decomposition $((-1,0)+\mathbb{Z}_+(1,0))\cup((0,1)+\mathbb{Z}_+(1,0))$ of the solution module: $(-1,0)$ has degree -1, $(0,1)$ has degree 0.

6.6.2. Type `strict_inequalities`

Once more we choose the example of Section 6.4, this time realized with a strict inequality (`SquareMinusVertexStrict.in`):

```
4                   1
3                   3
0 -1  1             1 1 0
1  0  0             strict_inequalities
-1  0  1            1
0  1  0             3
inequalities        0 0 1
                    grading
```

There is nothing to comment on the output since it is identical to that of Section 6.5.1. The strict inequality is only a shortcut for

```
1
4
1 1 0 -1
inhom_inequalities
```

The two inequalities represented by `1 0 0` and `0 1 0` could also be given by

```
1 3
1 1 0
signs
```

6.6.3. Type `inhom_equations`

We want to compute the nonnegative solutions of the system

$$\begin{aligned} \xi_1+2\xi_2+3\xi_3-4\xi_4-5\xi_5-6\xi_6-7\xi_7 &= -3,\\ -3\xi_2-2\xi_3-\ \xi_4-\ \xi_5+2\xi_6+13x_7 &= -7. \end{aligned}$$

A suitable input is `InhomEq.in`:

```
2 8
1  2  3 -4 -5 -6 -7 3
0 -3 -2 -1  1  2 13 7
inhom_equations
```

This is only to demonstrate the use of this type. We restrict ourselves to the abstract of the output file (default computation mode):

```
69 module generators
236 Hilbert basis elements of recession monoid
7 vertices of polyhedron                          size of triangulation   = 46
13 extreme rays of recession cone                 r...g sum of |det|s = 10790791
7 support hyperplanes of polyhedron
                                                  dehomogenization:
embedding dimension = 8                           0 0 0 0 0 0 0 1
affine dimension of the polyhedron = 5
rank of recession monoid = 5                      module rank = 1

size of triangulation   = 46
resulting sum of |det|s = 10790791
```

The example shows that the determinants of the simplicial cones in the triangulation can be quite large for seemingly harmless examples. It is then no surprise that the computation in the dual mode is significantly faster, yielding the same information.

6.6.4. Type `inhom_congruences`

The input file `ChineseRemainder.in` shows that Normaliz can even be used for elementary number theory:

```
3
3
2 1 5
1 -1 7
-1 5 11
inhom_congruences
```

This means that we want to solve the simultaneous congruences

$$\begin{aligned} 2x &\equiv -1 \quad (5),\\ x &\equiv 1 \quad (7),\\ -x &\equiv -5 \quad (11). \end{aligned}$$

The condition $x \geq 0$ is implicit, but it doesn't harm.

The solution is $x \equiv 302 \ \ (385)$:

```
1 module generators:
 302   1

1 Hilbert basis elements of recession monoid:
 385
```

7. Optional output files

When one of the options `-f`, `-a`, `-T`, `-y` or an option calling NmzIntegrate is activated, Normaliz writes additional output files whose names are of type `<project>.<type>`. The format of most files is completely analogous to that of the input file, except that there is usually no last line denoting the type. The main purpose of these files is to give the other systems easy access to the results of Normaliz without complicated parsing. The packages for Singular and Macaulay 2 use the extra output files to retrieve the results of Normaliz. Furthermore they provide additional information not contained in the standard output file.

7.1. The homogeneous case

The option `-f` makes Normaliz write the following files:

`gen` contains the Hilbert basis.

`cst` contains the constraints defining the cone and the lattice in the same format as they would appear in the input: matrices of types *constraints* following each other. Each matrix is concluded by the type of the constraints. Empty matrices are indicated by 0 as the number of rows. Therefore there will always be at least 3 matrices.

If a grading is defined, it will be appended. Therefore this file (with suffix `in`) as input for Normaliz will reproduce the Hilbert basis and all the other data computed, at least in principle.

`inv` contains all the information from the file `out` that is not contained in any of the other files.

If `-a` is activated, then the following files are written *additionally:*

`typ` This is the product of the matrices corresponding to `egn` and `esp`. In other words, the linear forms representing the support hyperplanes of the cone C are evaluated on the Hilbert basis. The resulting matrix, with the generators corresponding to the rows and the support hyperplanes corresponding to the columns, is written to this file.
The suffix `typ` is motivated by the fact that the matrix in this file depends only on the isomorphism type of monoid generated by the Hilbert basis (up to row and column permutations). In the language of [2] it contains the *standard embedding*.

`ext` contains the extreme rays of the cone.

`ht1` contains the degree 1 elements of the Hilbert basis if a grading is defined.

`egn,esp` These contain the Hilbert basis and support hyperplanes in the coordinates with respect to a basis of $\mathbb{E}$. `egn` contains the grading in the coordinates of $\mathbb{E}$ if one exists. Note that no equations for $C \cap \mathbb{E}$ or congruences for $\mathbb{E}$ are necessary.

7.2. Triangulation and Stanley decomposition

The option `-T` (independently from `-f` or `-a`) writes `inv` and the triangulation data:

`tgn, tri` These files together describe the triangulation computed by Normaliz.
The file `tri` lists the simplicial subcones as follows: The first line contains the number of simplicial cones in the triangulation, and the next line contains the number $m+1$ where $m = \operatorname{rank}\mathbb{E}$. Each of the following lines specifies a simplicial cone Δ: the first m numbers are the indices (with respect to the order in the file `tgn`) of those generators that span Δ, and the last entry is the multiplicity of Δ in $\mathbb{E}$, i. e. the absolute value of the determinant of the matrix of the spanning vectors (as elements of $\mathbb{E}$).
If `-t` is combined with `-T`, then the determinants have not been computed, and the last entry of each row is 0 (a forbidden value for the determinant).
The file `tgn` contains a matrix of vectors (in the coordinates of $\mathbb{A}$) spanning the simplicial cones in the triangulation.

The following example is the 2-dimensional cross polytope with one excluded face (`cross2.in`)

```
4 2                 1 3
 1  0               1  1 -1
 0  1               excluded_faces
-1  0
 0 -1
polytope
```

Its `tgn` and `tri` files are

```
tgn             tri
4               2
3               4
1 0 1           1 2 3 2
0 1 1           1 3 4 2
-1 0 1
0 -1 1
```

We see the 4 vertices $v_1, \ldots, v_4$ in homogenized coordinates in `tgn` and the 2 simplices (or the simplicial cones over them) in `tri`: both have multiplicity 2.

The option `-y` (independently from `-f` or `-a`) writes `inv`, `tgn` and the Stanley decomposition:

`dec` This file contains two types of data: (a) the information resulting from the excluded faces in connection with the sieve formula for inclusion/exclusion, (b) the Stanley decomposition.

(a) If there are any excluded faces, the file starts with the word `in_ex_data`. The next line contains the number of such data that follow. Each of these lines contains the data of a face and the coefficient with which the face is to be counted: the first number lists the number of generators that are contained in the face, followed by the indices of the generators relative to the `tgn` file and the last number is the coefficient.

(b) The second block (the first if there are no excluded faces) starts with the word `Stanley_dec`, followed by the number of simplicial cones in the triangulation.

For each simplicial cone Δ in the triangulation this file contains a block of data:

(i) a line listing the indices $i_1, \ldots, i_m$ of the generators $v_{i_1}, \ldots, v_{i_m}$ relative to the order in `tgn` (as in `tri`, $m = \operatorname{rank} \mathbb{E}$);

(ii) a $\mu \times m$ matrix where μ the multiplicity of Δ (see above).

In the notation of [6], each line lists an "offset" $x + \varepsilon(x)$ by its coordinates with respect to $v_{i_1}, \ldots, v_{i_m}$ as follows: if $(a_1, \ldots, a_m)$ is the line of the matrix, then

$$x + \varepsilon(x) = \frac{1}{\mu}(a_1 v_{i_1} + \cdots + a_m v_{i_m}).$$

The `dec` file of the example above is

```
in_ex_data
1
2 1 2 -1
Stanley_dec
2
1 3 4             1 2 3
2                 2
3                 3
0 0 2             0 0 0
1 1 2             1 0 1
```

There is 1 face in `in_ex_data` (namely the excluded one), it contains the 2 generators v_1 and v_2 and appears with multiplicity -1. The Stanley decomposition consists of 4 components of which each of the simplicial cone contains 2. The second offset in the second simplicial cone is

$$\frac{1}{2}(1v_1 + 0v_2 + 1v_3) = (0,0,1).$$

The file `3x3magiceven.in` has been processed with the option `-ahTy` activated. We recommend you to inspect all the output files in the subdirectory `example` of the distribution.

7.3. Modifications in the inhomogeneous case

The types are a subset of the types that can be produced in the homogeneous case. The main difference is that the generators of the solution module and the Hilbert basis of the recession monoid appear together in the file `gen`. They can be distinguished by the last component, as discussed already, and the same applies to the vertices of the polyhedron and extreme rays of the recession cone. The file `cst` contains the constraints defining the polyhedron and the recession module in conjunction with the dehomogenization which is also contained in the `cst` file, following the constraints.

With `-a` the files `typ`, `egn` and `esp` are produced, but `typ` has no meaning. The other two files contain `gen` in the coordinates of the efficient homogenized lattice and the support hyperplanes of the homogenized cone in the coordinates of $\mathbb{E}$.

8. Advanced topics

8.1. Primal vs. dual

It has been pointed out several times above that the dual algorithm can be much faster if the number of support hyperplanes is small relative to the dimension. This is in particular true for (in)homogeneous linear systems of equations where the number of support hyperplanes is bounded above by the number of indeterminates ($+1$ in the inhomogeneous case).

The paper [6] contains computation times for many examples that can help the user to choose the right algorithm.

Note that the dual algorithm is arithmetically much more stable than the primal algorithm since it basically only requires addition of vectors.

8.2. Polytope vs. polyhedron

Every polytope is a polyhedron, and therefore the input type `polyhedron` or inhomogeneous constraints seems to be a reasonable choice in order to compute polytopes. In general it is not since in homogenized coordinates all computations are truncated at degree 1, and the information hidden in the cone over the polytope cannot be obtained.

If a polytope is defined via inhomogeneous data, the computations are essentially limited to the lattice points in the polytope, and the multiplicity computed this way is their number.

8.3. Lattice points in polytopes

If only the lattice points in a rational polytope are to be computed, there are essentially three options:

`-1` Use the primal algorithm on the original polytope.
`-r` Use the primal algorithm on an integral approximation.
`-d1` Use the dual algorithm on the original polytope.

For lattice polytopes `-1` and `-r` are identical. The choice between the primal and the dual algorithm has been discussed above. Note that `-d1` is *not* a Hilbert basis computation via `-d` with a subsequent selection of degree 1 elements. On the contrary, all computations are truncated in degree 1.

In order to compare the approaches the user should try to compute the degree 1 points for the example `hickerson-18.in` (taken from the LattE distribution [9]): with `-rB` it is a matter of minutes, with `-d1` it is doable with some patience, and `-1B` seems to be out of reach presently. (Note that `-r` needs the inclusion of `B`. On a powerful machine `-d` is doable.)

8.4. Semiopen vs. inhomogeneous

The user may have noticed that the type `excluded_faces` is in principle superfluous since the Hilbert series of semiopen cones can also be computed via inhomogeneous input using `strict_inequalities`. However, the two types represent different algorithmic approaches:

(1) The computation with excluded faces are done in the "original" cone without the introduction of an additional coordinate: the Stanley decomposition is restricted to the excluded faces and all faces that arise from them as intersections. The result is then obtained by the sieve formula for inclusion/exclusion.
(2) For inhomogeneous input we introduce a homogenizing coordinate, and the Stanley decomposition of the homogenized monoid is restricted to the hyperplane $\delta(x) = 1$ (δ is the dehomogenization).

As a rule of thumb, `excluded_faces` should be preferred if only few faces are excluded. Moreover, at present they present the only way for applying NmzIntegrate to semiopen cones.

8.5. Ordering the generators

The computation time of Normaliz for input of type generators depends often on the order of the generators. The following discussion may help to deal with this problem.

In computations with partial or full triangulation the crucial measure of complexity is the sum of the determinants of the simplicial cones in the (partial) triangulation.

If a grading is defined and all extreme rays have the same degree, then the determinant sum of a full triangulation is independent of the order of the generators. The typical example is a lattice polytope given by (a superset of) its vertices. In such a case the order can only influence the combinatorial complexity of the triangulation. But the latter is impossible to predict, and since it plays no essential role, Normaliz processes the generators in the input order.

If the extreme rays have different degrees, then it is desirable to cover as much ground as possible by the generators of low degree, and Normaliz therefore orders the generators by ascending degree, for full triangulations as well as for partial ones. (In the absence of a grading, the L_1-norm is used.) This strategy has proved very useful. In addition to keeping the determinants small, it also helps to find a small partial triangulation.

In mode `-s` Normaliz does not sort the generators in any way. Especially in combinatorial situations the user may have found a clever order of the generators, and it would be foolish to destroy it.

It can be helpful to order the generators lexicographically if no better order is a priori available. Lexicographic ordering can easily be done by a text editor. In particular for `-s` one may not have a better choice. The following bidualization strategy often makes sense:

(1) Compute the dual of the input cone using option `-s`.
(2) If the number of support hyperplanes is not too large, use the constraints found in step (1) as input and compute the Hilbert basis with `-N`.

Many examples support our view that the order of the generators in step (2) is rather "regular", even if it is not always optimal.

As a case study, the user may compute the examples

```
cut_poly_7.in, cut_poly_7_lex.in, cut_poly_7_bidual.in, cut_poly_7_mysort.in.
```

with `-s` and `-N`. Start with `cut_poly_7_bidual.in`.

After step (1) above, one can of course also consider the dual algorithm for the computation of the Hilbert basis. It is instructive to apply `-d` to one of the input files above and compare computation times.

8.6. Performance and parallelization

The executables of Normaliz have been compiled for parallelization on shared memory systems with OpenMP. Parallelization reduces the "real" time of the computations considerably, even on relatively small systems. However, one should not underestimate the administrational overhead involved.

- It is not a good idea to use parallelization for very small problems.
- On multi-user systems with many processors it may be wise to limit the number of threads for Normaliz somewhat below the maximum number of cores.

The number of parallel threads can be limited by the Normaliz option `-x` (see Section 4.5) or by the commands

```
export OMP_NUM_THREADS=<T>      (Linux/Mac)
```

or

```
set OMP_NUM_THREADS=<T>      (Windows)
```

where <T> stands for the maximum number of threads accessible to Normaliz. For example, we often use

```
export OMP_NUM_THREADS=16
```

on a multi-user system system with 24 cores.

Limiting the number of threads to 1 forces a strictly serial execution of Normaliz.

The paper [6] contains extensive data on the effect of parallelization. On the whole Normaliz scales very well. However, the dual algorithm often performs best with mild parallelization, say with 4 or 6 threads.

8.7. Running large computations

Normaliz can cope with very large examples, but it is usually difficult to decide a priori whether an example is very large, but nevertheless doable, or simply impossible. Therefore some exploration makes sense.

See [6] for some very large computations. The following hints reflect the authors' experience with them.

(1) Run Normaliz with the option `-cs` and pay attention to the terminal output. The number of extreme rays, but also the numbers of support hyperplanes of the intermediate cones are useful data.

(2) In many cases the most critical size parameter is the number of simplicial cones in the triangulation. It makes sense to determine it as the next step. Even with the fastest potential evaluation (option `-v`), finding the triangulation takes less time, say by a factor between 3 and 10. Thus it makes sense to run the example with `-t` in order to explore the size.

As you can see from [6], Normaliz has successfully evaluated triangulations of size $\approx 5 \cdot 10^{11}$ in dimension 24.

(3) Another critical parameter are the determinants of the generator matrices of the simplicial cones. To get some feeling for their sizes, one can restrict the input to a subset (of the extreme rays computed in (1)) and use the option `-v`.

The output file contains the number of simplicial cones as well as the sum of the absolute values of the determinants. The latter is the number of vectors to be processed by Normaliz in triangulation based calculations.

The number includes the zero vector for every simplicial cone in the triangulation. The zero vector does not enter the Hilbert basis calculation, but cannot be neglected for the Hilbert series.

Normaliz has mastered calculations with $> 10^{15}$ vectors.

(4) If the triangulation is small, we can add the option `-T` in order to actually see the triangulation in a file. Then the individual determinants become visible.

(5) If a cone is defined by inequalities and/or equations consider the dual mode for Hilbert basis calculation, even if you also want the Hilbert series.

(6) The size of the triangulation and the size of the determinants are *not* dangerous for memory by themselves (unless `-T` or `-y` are set). Critical magnitudes can be the number of support hyperplanes, Hilbert basis candidates, or degree 1 elements.

9. Distribution and installation

In order to install Normaliz you should first download the basic package containing the documentation, examples, source code, jNormaliz, NmzIntegrate and the packages for Singular and Macaulay2. Then unzip the downloaded file `Normaliz2.12.zip` in a directory of your choice. (Any other downloaded zip file for Normaliz should be unzipped in this directory, too.)

This process will create a directory `Normaliz2.12` (called Normaliz directory) and several subdirectories in `Normaliz2.12`. The names of the subdirectories created are self-explanatory. Nevertheless we give an overview:

- In the main directory `Normaliz2.12` you should find `jNormaliz.jar`, `Copying` and subdirectories.
- The subdirectory `source` contains the source files and a `Makefile` for compilation with GCC. The subdirectory `genEhrhart` contains the NmzIntegrate source.
- The subdirectory `doc` contains the file you are reading and further documentation.
- In the subdirectory `example` are the input and output files for some examples. It contains all input files of examples of this documentation, except the toy examples of Section 3. Some very large output files are contained in an extra zip file accessible from the Normaliz home page.
- The subdirectory `singular` contains the SINGULAR library `normaliz.lib` and a PDF file with documentation.
- The subdirectory `macaulay2` contains the MACAULAY2 package `Normaliz.m2`.
- The subdirectory `lib` contains libraries for jNormaliz.

We provide executables for Windows, Linux (each in a 32 bit and a 64 bit version) and Mac. Download the archive file corresponding to your system `Normaliz2.12<systemname>.zip` and unzip it. This process will store the executables of Normaliz and NmzIntegrate in the directory `Normaliz2.12`. In case you want to run Normaliz from the command line or use it from other systems, you may have to copy the executables to a directory in the search path for executables.

Please remove old versions of `normaliz`, `norm64` and `normbig` from your search path.

Running NmzIntegrate requires the additional download of its executable for your system.

10. Compilation

We only describe the compilation of Normaliz. See the documentation of NmzIntegrate for its compilation.

10.1. GCC

Produce the executables by calling `make` in the subdirectory `source`. You may have to transport the executables to a directory in your search path. **jNormaliz expects them in its own directory.**

Note that `normaliz` needs GMP (including the C++ wrapper) and the Boost collection. Therefore you must install them first.

We are using OpenMP 3.0. Please make sure that your GCC version is compatible with it (version $\geq$ 4.4).

Note the following **exceptions:**

1. One can compile Windows executables with the Cygwin port of GCC. Unfortunately it is not compatible to OpenMP.
2. Mac versions of GCC older than 4.5 have a bug that makes it impossible to use OpenMP.

In any case, or if you want to avoid parallelization, you can call `make OPENMP=no`.

10.2. Visual Studio project

The Windows executables provided by us have been compiled with MS Visual Studio and Intel C++ Composer XE. (Visual C++ itself can only be used without OpenMP.)

If you want to compile Normaliz yourself in this way, please unzip the corresponding zip file on the Normaliz home page. This will create a subdirectory `Visual Studio` of the Normaliz directory. This directory contains the predefined project. We have provided

1. two configurations: `Release` (with OpenMP) and `ReleaseSerial` (without OpenMP), and
2. two platforms, `Win32` and `x64`.

Instead of GMP we use the MPIR library for the Windows version of `normaliz`. For convenience, the MPIR files have been included in the distribution (in the subdirectory `MPIR` of `Visual Studio`). Please

- copy the library files for Win32 into the `lib` subdirectory of the Visual C++ compiler,
- the library files for x64 to the subdirectory `amd64` (or `x64`) of `lib`, and
- the two header files to the `include` subdirectory of the compiler.

Moreover, you must install the Boost collection available from http://www.boost.org/. We only use Boost libraries that are entirely implemented in their headers. So the only preparation beyond downloading and unzipping is to add the Boost root directory to the list of include

paths. In the Visual Studio C++ IDE, click “Tools | Options... | Projects | VC++ directories”. Then, in “Show Directories for”, select “Include files” and add the path to the Boost root directory.

After the compilation with the Intel compiler you must copy the executable to the directories where they are expected (the Normaliz directory or a directory in the search path).

The source files for Visual Studio are identical to those for GCC.

11. Copyright and how to cite

Please refer to Normaliz in any publication for which it has been used:

W. Bruns, B. Ichim, T. Römer and C. Söger: Normaliz. Algorithms for rational cones and affine monoids. Available from http://www.math.uos.de/normaliz.

It is now customary to evaluate mathematicians by such data as numbers of publications, citations and impact factors. The data bases on which such dubious evaluations are based do not list mathematical software. Therefore we ask you to cite the article [6] in addition. This is very helpful for the younger members of the team.

A. Mathematical background and terminology

For a coherent and thorough treatment of the mathematical background we refer the reader to [2].

A.1. Polyhedra, polytopes and cones

An *affine halfspace* of $\mathbb{R}^d$ is a subset given as

$$H_\lambda^+ = \{x : \lambda(x) \geq 0\},$$

where λ is an affine form, i.e., a non-constant map $\lambda : \mathbb{R}^d \to \mathbb{R}$, $\lambda(x) = \alpha_1 x_1 + \cdots + \alpha_d x_d + \beta$ with $\alpha_1, \ldots, \alpha_d, \beta \in \mathbb{R}$. If $\beta = 0$ and λ is therefore linear, then the halfspace is called *linear*. The halfspace is *rational* if λ is *rational*, i.e., has rational coordinates. If λ is rational, we can assume that it is even *integral*, i.e., has integral coordinates, and, moreover, that these are coprime. Then λ is uniquely determined by H_λ^+. Such integral forms are called *primitive*, and the same terminology applies to vectors.

Definition 1. A (rational) *polyhedron* P is the intersection of finitely many (rational) halfspaces. If it is bounded, then it is called a *polytope*. If all the halfspaces are linear, then P is a cone.

The *dimension* of P is the dimension of the smallest affine subspace $\mathrm{aff}(P)$ containing P.

A support hyperplane of P is an affine hyperplane H that intersects P, but only in such a way that H is contained in one of the two halfspaces determined by H. The intersection $H \cap P$ is called a *face* of P. It is a polyhedron (polytope, cone) itself. Faces of dimension 0 are called *vertices*, those of dimension 1 are called *edges* (in the case of cones *extreme rays*), and those of dimension $\dim(P) - 1$ are *facets*.

When we speak of *the* support hyperplanes of P, then we mean those intersecting P in a facet. Their halfspaces containing P cut out P from $\mathrm{aff}(P)$. If $\dim(P) = d$, then they are uniquely determined (up to a positive scalar).

The constraints by which Normaliz describes polyhedra are

(1) linear equations for $\mathrm{aff}(P)$ and
(2) linear inequalities (simply called support hyperplanes) cutting out P from $\mathrm{aff}(P)$.

In other words, the constraints are given by a linear system of equations and inequalities, and a polyhedron is nothing else than the solution set of a linear system of inequalities and equations. It can always be represented in the form

$$Ax \geq b, \qquad A \in \mathbb{R}^{m \times d}, b \in \mathbb{R}^m,$$

if we replace an equation by two inequalities.

A.2. Cones

The definition describes a cone by constraints. One can equivalently describe it by generators:

Theorem 2 (Minkowski-Weyl)**.** *The following are equivalent for $C \subset \mathbb{R}^d$;*

1. *C is a (rational) cone;*
2. *there exist finitely many (rational) vectors $x_1, \ldots, x_n$ such that*

$$C = \{a_1x_1 + \cdots + a_nx_n : a_1, \ldots, a_n \in \mathbb{R}_+\}.$$

By $\mathbb{R}_+$ we denote the set of nonnegative real numbers; $\mathbb{Q}_+$ and $\mathbb{Z}_+$ are defined in the same way.

The conversion between the description by constraints and that by generators is one of the basic tasks of Normaliz. It uses the *Fourier-Motzkin elimination*.

A cone is *pointed* if $x \in C$ and $-x \in C$ is only possible with $x = 0$. If a rational cone is pointed, then it has uniquely determined *extreme integral generators*. These are the primitive integral vectors spanning the extreme rays. These can also be defined with respect to a sublattice L of $\mathbb{Z}^d$, provided C is contained in $\mathbb{R}L$.

The *dual cone* C^* is given by

$$C^* = \{\lambda \in (\mathbb{R}^d)^* : \lambda(x) \geq 0 \text{ for all } x \in C\}.$$

Under the identification $\mathbb{R}^d = (\mathbb{R}^d)^{**}$ one has $C^{**} = C$. Let C_0 be the set of those $x \in C$ for which $-x \in C$ as well. It is the largest vector subspace contained in C. Then one has

$$\dim C_0 + \dim C^* = d.$$

In particular, C is pointed if and only if C^* is full dimensional, and this is the criterion for pointedness used by Normaliz. Linear forms $\lambda_1, \ldots, \lambda_n$ generate C^* if and only if C is the intersection of the halfspaces $H^+_{\lambda_i}$. Therefore the conversion from constraints to generators and its converse are the same task, except for the exchange of $\mathbb{R}^d$ and its dual space.

A.3. Polyhedra

In order to transfer the Minkowski-Weyl theorem to polyhedra it is useful to homogenize coordinates by embedding $\mathbb{R}^d$ as a hyperplane in $\mathbb{R}^{d+1}$, namely via

$$\kappa : \mathbb{R}^d \to \mathbb{R}^{d+1}, \qquad \kappa(x) = (x, 1).$$

If P is a (rational) polyhedron, then the closure of the union of the rays from 0 through the points of $\kappa(P)$ is a (rational) cone $C(P)$, called the *cone over* P. The intersection $C(P) \cap (\mathbb{R}^d \times \{0\})$ can be identified with the *recession* (or tail) *cone*

$$\text{rec}(P) = \{x \in \mathbb{R}^d : y + x \in P \text{ for all } y \in P\}.$$

It is the cone of unbounded directions in P. The recession cone is pointed if and only if P has at least one bounded face, and this is the case if and only if it has a vertex.

The theorem of Minkowski-Weyl can then be generalized as follows:

Theorem 3 (Motzkin)**.** *The following are equivalent for a subset $P \neq \emptyset$ of $\mathbb{R}^d$:*

1. *P is a (rational) polyhedron;*
2. *$P = Q + C$ where Q is a (rational) polytope and C is a (rational) cone.*

If P has a vertex, then the smallest choice for Q is the convex hull of its vertices, and $C = \mathrm{rec}(P)$ is uniquely determined.

The *convex hull* of a subset $X \in \mathbb{R}^d$ is

$$\mathrm{conv}(X) = \{a_1x_1 + \cdots + a_nx_n : n \geq 1, x_1, \ldots, x_n \in X, a_1, \ldots, a_n \in \mathbb{R}_+, a_1 + \cdots + a_n = 1\}.$$

Clearly, P is a polytope if and only if $\mathrm{rec}(P) = \{0\}$, and the specialization to this case one obtains Minkowski's theorem: a subset P of $\mathbb{R}^d$ is a polytope if and only if it is the convex hull of a finite set. A *lattice polytope* is distinguished by having integral points as vertices.

Normaliz computes the recession cone and the polytope Q if P is defined by constraints. Conversely it finds the constraints if the vertices of Q and the generators of C are specified.

Suppose that P is given by a system

$$Ax \geq b, \qquad A \in \mathbb{R}^{m\times d},\ b \in \mathbb{R}^m,$$

of linear inequalities (equations are replaced by two inequalities). Then $C(P)$ is defined by the *homogenized system*

$$Ax - x_{d+1}b \geq 0$$

whereas the $\mathrm{rec}(P)$ is given by the *associated homogeneous system*

$$Ax \geq 0.$$

It is of course possible that P is empty if it is given by constraints since inhomogeneous systems of linear equations and inequalities may be unsolvable. By abuse of language we call the solution set of the associated homogeneous system the recession cone of the system.

Via the concept of dehomogenization, Normaliz allows for a more general approach. The *dehomogenization* is a linear form δ on $\mathbb{R}^{d+1}$. For a cone $\widetilde{C}$ in $\mathbb{R}^{d+1}$ and a dehomogenization δ, Normaliz computes the polyhedron $P = \{x \in \widetilde{C} : \delta(x) = 1\}$ and the recession cone $C = \{x \in \widetilde{C} : \delta(x) = 0\}$. In particular, this allows other choices of the homogenizing coordinate. (Often one chooses x_0, the first coordinate then.)

In the language of projective geometry, $\delta(x) = 0$ defines the hyperplane at infinity.

A.4. Affine monoids

An *affine monoid* M is a finitely generated submonoid of $\mathbb{Z}^d$ for some $d \geq 0$. This means: $0 \in M$, $M+M \subset M$, and there exist $x_1, \ldots, x_n$ such that

$$M = \{a_1x_1 + \cdots + a_nx_n : a_1, \ldots, a_n \in \mathbb{Z}_+\}.$$

We say that $x_1, \ldots, x_n$ is a *system of generators* of M. A monoid M is positive if $x \in M$ and $-x \in M$ implies $x = 0$. An element x in a positive monoid M is called *irreducible* if it has no decomposition $x = y + z$ with $y, z \in M$, $y, z \neq 0$. The *rank* of M is the rank of the subgroup $\mathrm{gp}(M)$ of $\mathbb{Z}^d$ generated by M. (Subgroups of $\mathbb{Z}^d$ are also called sublattices.)

Theorem 4 (van der Corput)**.** *Every positive affine monoid M has a unique minimal system of generators, given by its irreducible elements.*

We call the minimal system of generators the *Hilbert basis* of M. Normaliz computes Hilbert bases of a special type of affine monoid:

Theorem 5 (Gordan's lemma)**.** *Let $C \subset \mathbb{R}^d$ be a (pointed) rational cone and let $L \subset \mathbb{Z}^d$ be a sublattice. Then $C \cap L$ is a (positive) affine monoid.*

Let $M \subset \mathbb{Z}^d$ be an affine monoid, and let $N \supset M$ be an overmonoid (not necessarily affine), for example a sublattice L of $\mathbb{Z}^d$ containing M.

Definition 6. The *integral closure* (or *saturation*) of M in N is the set

$$\widehat{M}_N = \{x \in N : kx \in M \text{ for some } k \in \mathbb{Z}, k > 0\}.$$

If $\widehat{M}_N = M$, one calls M *integrally closed* in N.
The integral closure $\overline{M}$ of M in $\mathrm{gp}(M)$ is its *normalization*. M is *normal* if $\overline{M} = M$.

The integral closure has a geometric description:

Theorem 7.

$$\widehat{M}_N = \mathrm{cone}(M) \cap N.$$

Combining the theorems, we can say that Normaliz computes integral closures of affine monoids in lattices, and the integral closures are themselves affine monoids as well. (More generally, $\widehat{M}_N$ is affine if M and N are affine.)

In order to specify the intersection $C \cap L$ by constraints we need a system of homogeneous inequalities for C. Every sublattice of $\mathbb{Z}^d$ can be written as the solution set of a combined system of homogeneous linear diophantine equations and a homogeneous system of congruences (this follows from the elementary divisor theorem). Thus $C \cap L$ is the solution set of a homogeneous linear diophantine system of inequalities, equations and congruences. Conversely, the solution set of every such system is a monoid of type $C \cap L$.

A.5. Affine monoids from binomial ideals

Let U be a subgroup of $\mathbb{Z}^n$. Then the natural image M of $\mathbb{Z}^n_+ \subset \mathbb{Z}^n$ in the abelian group $G = \mathbb{Z}^n/U$ is a submonoid of G. In general, G is not torsionfree, and therefore M may not be an affine monoid. However, the image N of M in the lattice $L = G/\mathrm{torsion}(G)$ is an affine monoid. Given U, Normaliz chooses an embedding $L \hookrightarrow \mathbb{Z}^r$, $r = n - \mathrm{rank}\, U$, such that N becomes a submonoid of $\mathbb{Z}^r_+$. In general there is no canonical choice for such an embedding, but one can always find one, provided N has no invertible element except 0.

The typical starting point is an ideal $J \subset K[X_1,\dots,X_n]$ generated by binomials

$$X_1^{a_1}\cdots X_n^{a_n} - X_1^{b_1}\cdots X_n^{b_n}.$$

The image of $K[X_1,\dots,X_n]$ in the residue class ring of the Laurent polynomial ring $S = K[X_1^{\pm 1},\dots,X_n^{\pm 1}]$ modulo the ideal JS is exactly the monoid algebra $K[M]$ of the monoid M above if we let U be the subgroup of $\mathbb{Z}^n$ generated by the differences

$$(a_1,\dots,a_n) - (b_1,\dots,b_n).$$

Ideals of type JS are called lattice ideals if they are prime. Since Normaliz automatically passes to $G/\mathrm{torsion}(G)$, it replaces JS by the smallest lattice ideal containing it.

A.6. Lattice points in polyhedra

Let $P \subset \mathbb{R}^d$ be a rational polyhedron and $L \subset \mathbb{Z}^d$ be an *affine sublattice*, i.e., a subset $w + L_0$ where $w \in \mathbb{Z}^d$ and $L_0 \subset \mathbb{Z}^d$ is a sublattice. In order to investigate (and compute) $P \cap L$ one again uses homogenization: P is extended to $C(P)$ and L is extended to $\mathscr{L} = L_0 + \mathbb{Z}(w,1)$. Then one computes $C(P) \cap \mathscr{L}$. Via this "bridge" one obtains the following inhomogeneous version of Gordan's lemma:

Theorem 8. *Let P be a rational polyhedron with vertices and $L = w + L_0$ an affine lattice as above. Set* $\mathrm{rec}_L(P) = \mathrm{rec}(P) \cap L_0$. *Then there exist $x_1,\dots,x_m \in P \cap L$ such that*

$$P \cap L = \{(x_1 + \mathrm{rec}_L(P)) \cap \cdots \cap (x_m + \mathrm{rec}_L(P))\}.$$

If the union is irredundant, then $x_1,\dots,x_m$ are uniquely determined.

The Hilbert basis of $\mathrm{rec}_L(P)$ is given by $\{x : (x,0) \in \mathrm{Hilb}(C(P) \cap \mathscr{L})\}$ and the minimal system of generators can also be read off the Hilbert basis of $C(P) \cap \mathscr{L}$: it is given by those x for which $(x,1)$ belongs to $\mathrm{Hilb}(C(P) \cap \mathscr{L})$. (Normaliz computes the Hilbert basis of $C(P) \cap L$ only at "levels" 0 and 1.)

We call $\mathrm{rec}_L(P)$ the *recession monoid* of P with respect to L (or L_0). It is justified to call $P \cap L$ a *module* over $\mathrm{rec}_L(P)$. In the light of the theorem, it is a finitely generated module, and it has a unique minimal system of generators.

After the introduction of coefficients from a field K, $\mathrm{rec}_L(P)$ is turned into an affine monoid algebra, and $N = P \cap L$ into a finitely generated torsionfree module over it. As such it has a

well-defined *module rank* $\operatorname{mrank}(N)$, which is computed by Normaliz via the following combinatorial description: Let $x_1,\dots,x_m$ be a system of generators of N as above; then $\operatorname{mrank}(N)$ is the cardinality of the set of residue classes of $x_1,\dots,x_m$ modulo $\operatorname{rec}_L(P)$.

Clearly, to model $P\cap L$ we need linear diophantine systems of inequalities, equations and congruences which now will be inhomogeneous in general. Conversely, the set of solutions of such a system is of type $P\cap L$.

A.7. Hilbert series

Normaliz can compute the Hilbert series and the Hilbert (quasi)polynomial of a graded monoid. A *grading* of a monoid M is simply a homomorphism $\deg : M\to\mathbb{Z}^g$ where $\mathbb{Z}^g$ contains the degrees. The *Hilbert series* of M with respect to the grading is the formal Laurent series

$$H(t)=\sum_{u\in\mathbb{Z}^g}\#\{x\in M:\deg x=u\}t_1^{u_1}\cdots t_g^{u_g}=\sum_{x\in M}t^{\deg x},$$

provided all sets $\{x\in M:\deg x=u\}$ are finite. At the moment, Normaliz can only handle the case $g=1$, and therefore we restrict ourselves to this case. We assume in the following that $\deg x>0$ for all nonzero $x\in M$ and that there exists an $x\in\operatorname{gp}(M)$ such that $\deg x=1$. (Normaliz always rescales the grading accordingly.)

The basic fact about $H(t)$ in the $\mathbb{Z}$-graded case is that it is the Laurent expansion of a rational function at the origin:

Theorem 9 (Hilbert, Serre; Ehrhart)**.** *Suppose that M is a normal affine monoid. Then*

$$H(t)=\frac{R(t)}{(1-t^e)^r},\qquad R(t)\in\mathbb{Z}[t],$$

where r is the rank of M and e is the least common multiple of the degrees of the extreme integral generators of $\operatorname{cone}(M)$*. As a rational function, $H(t)$ has negative degree.*

The statement about the rationality of $H(t)$ holds under much more general hypotheses.

Usually one can find denominators for $H(t)$ of much lower degree than that in the theorem, and Normaliz tries to give a more economical presentation of $H(t)$ as a quotient of two polynomials. One should note that it is not clear what the most natural presentation of $H(t)$ is in general (when $e>1$). We discuss this problem in [6, Section 4] and in 6.1.3. The examples in Section 6, especially 6.1.3 and 6.2.3, may serve as an illustration.

A rational cone C and a grading together define the rational polytope $Q=C\cap A_1$ where $A_1=\{x:\deg x=1\}$. In this sense the Hilbert series is nothing but the Ehrhart series of Q. The following description of the Hilbert function $H(M,k)=\#\{x\in M:\deg x=k\}$ is equivalent to the previous theorem:

Theorem 10. *There exists a quasipolynomial q with rational coefficients, degree* $\operatorname{rank}M-1$ *and period π dividing e such that $H(M,k)=q(k)$ for all $q\geq 0$.*

The statement about the quasipolynomial means that there exist polynomials $q^{(j)}$, $j = 0, \dots, \pi - 1$, of degree $\operatorname{rank} M - 1$ such that

$$q(k) = q^{(j)}(k), \qquad j \equiv k \quad (\pi),$$

and

$$q^{(j)}(k) = q_0^{(j)} + q_1^{(j)} k + \cdots + q_{r-1}^{(j)} k^{r-1}, \qquad r = \operatorname{rank} M,$$

with coefficients $q_i^{(j)} \in \mathbb{Q}$. It is not hard to show that in the case of affine monoids all components have the same degree $r-1$ and the same leading coefficient:

$$q_{r-1} = \frac{\operatorname{vol}(Q)}{(r-1)!},$$

where vol is the lattice normalized volume of Q (a lattice simplex of smallest possible volume has volume 1). It is called the *multiplicity* of M.

Suppose now that P is a rational polyhedron in $\mathbb{R}^d$, $L \subset \mathbb{Z}^d$ is an affine lattice, and we consider $N = P \cap L$ as a module over $M = \operatorname{rec}_L(P)$. If $\mathbb{Z}^d$ is endowed with a grading whose restriction to M satisfies our conditions, then the Hilbert series

$$H_N(t) = \sum_{x \in N} t^{\deg x}$$

is well-defined, and the qualitative statement above about rationality remain valid. However, in general the quasipolynomial gives the correct value of the Hilbert function only for $k \gg 0$. The leading coefficient is still constant and given by

$$q_{r-1} = \operatorname{mrank}(N) \frac{\operatorname{vol}(Q)}{(r-1)!}, \qquad Q = \operatorname{rec}(P) \cap A_1.$$

The *multiplicity* of N is $\operatorname{mrank}(N) \operatorname{vol}(Q)$.

Since N may have generators in negative degrees, Normaliz shifts the degrees into $\mathbb{Z}_+$ by adding a constant, called the *shift*. (The shift may also be negative.)

B. Changes relative to version 2.5

For the history of changes starting from 2.0 see the manual of version 2.7 (still accessible on the web site). Note that some changes have become obsolete later on.

Changes in version 2.7:

User control, input and output:

1. Only one executable `normaliz`. Precision controlled by option `-B`.
2. Slight changes in the wording of the main output file.
3. Introduction of options for large problems. (Obsolete.)

Algorithms and implementation:

1. Separation of front end and kernel (implemented as a library).
2. Pyramid based algorithms for large problems (see [6]).
3. New algorithm for h-vector. No computation of line shellings in this version.
4. Dual mode accessible from all input types.
5. General improvement of memory use (and speed) by more efficient data types.

Changes in version 2.8:

User control, input and output:

1. Use of arbitrary $\mathbb{Z}$-gradings which make rational polytopes accessible.
2. Implied changes in the output files.
3. Simplification of the command line options. ("Large" modes now superfluous.)

Algorithms and implementation:

1. Handling of arbitrary $\mathbb{Z}$-gradings.
2. Substantial improvement of parallelization, based on thorough use of pyramid decompositions (see [6]).
3. Faster evaluation of simplicial cones (see [6]).
4. General overhaul of the code.

Changes in version 2.9:

User control, input and output:

1. Introduction of type `signs`.
2. Options for calling NmzIntegrate.
3. Corresponding output options and output files.

Algorithms and implementation:

1. Introduction of NmzIntegrate (independent executable).
2. Faster volume computation by using the heights of simplicial cones attached to unimodular ones.
3. Parallelization of pyramid decomposition also for support hyperplane computation.

Changes in version 2.10:

User control, input and output:

1. Corrections in the output forwarded to NmzIntegrate.

Algorithms and implementation:

1. Normaliz now avoids the production of duplicates of candidates for the Hilbert basis. At the expense of some computation time, this strategy saves much memory.

Version 2.10.1 is only a bug fix.

Changes in version 2.11:

User control, input and output:

1. Addition of inhomogeneous input types.
2. Hilbert series of semiopen cones.

Algorithms and implementation:

1. Corresponding extension of algorithms.
2. Integral approximation of rational polytopes.
3. Lattice points in polytopes via the dual algorithm.
4. Improvement in Fourier-Motzkin elimination by better use of pyramid decomposition.
5. Substantial improvement in computing "large" simplicial cones.

Versions 2.11.1 and 2.11.2 are mainly bug fixes.

Changes in version 2.12:

Algorithms and implementation:

1. Dual algorithm thoroughly revised.
2. Internal parallelization of simplicial cones with large determinants.
3. Improvement of linear algebra.

References

[1] V. Almendra and B. Ichim. *jNormaliz 1.5*. Available from `http://www.mathematik.uni-osnabrueck.de/normaliz/Normaliz2.11/jNormaliz1.5Documentation.pdf`

[2] W. Bruns and J. Gubeladze. *Polytopes, rings, and K-theory*. Springer 2009.

[3] W. Bruns, R. Hemmecke, B. Ichim, M. Köppe and C. Söger. *Challenging computations of Hilbert bases of cones associated with algebraic statistics* . Exp. Math.20 (2011), 25–33.

[4] W. Bruns and J. Herzog. *Cohen-Macaulay rings*. Rev. ed. Cambridge University Press 1998.

[5] W. Bruns and B. Ichim. *Normaliz: algorithms for rational cones and affine monoids.* J. Algebra **324** (2010) 1098–1113.

[6] W. Bruns, B. Ichim and C. Söger. *The power of pyramid decompositions in Normaliz*. Preprint arXiv:1206.1916.

[7] W .Bruns and R. Koch. *Computing the integral closure of an affine semigroup*. Univ. Iagell. Acta Math. **39** (2001), 59–70.

[8] W. Bruns and C. Söger. *NmzIntegrate 1.2.* Available from `http://www.mathematik.uni-osnabrueck.de/normaliz/Normaliz2.11/NmzIntegrate_1.2.pdf`

[9] J.A. De Loera, M. Köppe et al., *LattE integrale*. Available at `http://www.math.ucdavis.edu/~latte/`

[10] S. Gutsche, M. Horn, C. Söger, *NormalizInterface for GAP*. Available at `https://github.com/fingolfin/NormalizInterface`.

[11] M. Köppe and S. Verdoolaege. *Computing parametric rational generating functions with a primal Barvinok algorithm.* Electron. J. Comb. 15, No. 1, Research Paper R16, 19 p. (2008).

[12] L. Pottier. *The Euclide algorithm in dimension n*. Research report, ISSAC 96, ACM Press 1996.