Normaliz 3.0

Winfried Bruns, Tim Römer, Richard Sieg and Christof Söger

Normaliz 2 team member: Bogdan Ichim

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

mailto:normaliz@uos.de

Contents

1. Introduction 5
- 1.1. The objectives of Normaliz . . . 5
- 1.2. Platforms, implementation and access from other systems . . . 6
- 1.3. Major changes relative to version 2.12 . . . 6
- 1.4. Future extensions . . . 7
- 1.5. Download and installation . . . 7

2. Normaliz by examples 7
- 2.1. Terminology . . . 7
- 2.2. Practical preparations . . . 8
- 2.3. A cone in dimension 2 . . . 9
  - 2.3.1. The Hilbert basis . . . 10
  - 2.3.2. The cone by inequalities . . . 12
  - 2.3.3. The interior . . . 12
- 2.4. A lattice polytope . . . 14
  - 2.4.1. Only the lattice points . . . 16
- 2.5. A rational polytope . . . 17
  - 2.5.1. The rational polytope by inequalities . . . 19
- 2.6. Magic squares . . . 20
  - 2.6.1. With even corners . . . 23
  - 2.6.2. The lattice as input . . . 25
- 2.7. Decomposition in a numerical semigroup . . . 26
- 2.8. A job for the dual algorithm . . . 26

2.9. A dull polyhedron . . . 27
2.9.1. Defining it by generators . . . 29
2.10. The Condorcet paradoxon . . . 29
2.10.1. Excluding ties . . . 30
2.10.2. At least one vote for every preference order . . . 31
2.11. Testing normality . . . 32
2.12. Inhomogeneous congruences . . . 33
2.12.1. Lattice and offset . . . 34
2.12.2. Variation of the signs . . . 35
2.13. Integral closure and Rees algebra of a monomial ideal . . . 35
2.13.1. Only the integral closure . . . 36
2.14. Only the convex hull . . . 37
2.15. Starting from a binomial ideal . . . 37

3. The input file 39
3.1. Input items . . . 40
3.1.1. The ambient space and lattice . . . 40
3.1.2. Matrices . . . 40
3.1.3. Vectors . . . 41
3.1.4. Computation goals and algorithmic variants . . . 41
3.1.5. Comments . . . 41
3.1.6. Restrictions . . . 42
3.1.7. Default values . . . 42
3.1.8. Normaliz takes intersections (almost always) . . . 42
3.2. Homogeneous generators . . . 43
3.2.1. Cones . . . 43
3.2.2. Lattices . . . 43
3.3. Homogeneous Constraints . . . 43
3.3.1. Cones . . . 43
3.3.2. Lattices . . . 44
3.4. Inhomogeneous generators . . . 44
3.4.1. Polyhedra . . . 44
3.4.2. Lattices . . . 45
3.5. Inhomogeneous constraints . . . 45
3.5.1. Cones . . . 45
3.5.2. Lattices . . . 45
3.6. Relations . . . 46
3.7. Unit vectors . . . 46
3.8. Grading . . . 46
3.8.1. `lattice_ideal` . . . 47
3.9. Dehomogenization . . . 47
3.10. Pointedness . . . 48
3.11. The zero cone . . . 48

3.12. Additional input file for NmzIntegrate 48

4. Computation goals and algorithmic variants 48
4.1. Default choices and basic rules 49
4.2. Computation goals 49
4.2.1. Lattice data 49
4.2.2. Support hyperplanes and extreme rays 49
4.2.3. Hilbert basis and related data 50
4.2.4. Enumerative data 50
4.2.5. Combined computation goals 50
4.2.6. The class group 50
4.2.7. Triangulation and Stanley decomposition 50
4.2.8. NmzIntegrate 50
4.2.9. Boolean valued computation goals 51
4.3. Algorithmic variants 51
4.4. Integer type 51
4.5. Control of computations 52

5. Running Normaliz 52
5.1. Basic rules 53
5.2. Info about Normaliz 53
5.3. Control of execution 53
5.4. Control of output files 54
5.5. Overriding the options in the input file 54

6. More examples 54
6.1. Lattice points by approximation 54
6.2. The bottom decomposition 56
6.3. Subdivision of large simplicial cones 57
6.4. Primal vs. dual – division of labor 58
6.5. Explicit dehomogenization 58
6.6. Exporting the triangulation 60
6.6.1. Nested triangulations 61
6.7. Exporting the Stanley decomposition 61
6.8. Module generators over the original monoid 62
6.9. Precomputed support hyperplanes 63
6.10. Shift, denominator, quasipolynomial and multiplicity 64
6.11. A very large computation 66

7. Optional output files 66
7.1. The homogeneous case 67
7.2. Modifications in the inhomogeneous case 68

8. Performance 68
8.1. Parallelization 68
8.2. Running large computations 69

9. Distribution and installation 70

10.Compilation 70
10.1. General 71
10.2. Linux 71
10.3. MacOs X 72
10.4. Windows 72

11.Copyright and how to cite 72

A. Mathematical background and terminology 74
A.1. Polyhedra, polytopes and cones 74
A.2. Cones 75
A.3. Polyhedra 75
A.4. Affine monoids 77
A.5. Affine monoids from binomial ideals 78
A.6. Lattice points in polyhedra 78
A.7. Hilbert series 79
A.8. The class group 81

B. Annotated console output 81
B.1. Primal mode 81
B.2. Dual mode 83

C. Normaliz 2 input syntax 85

D. Changes relative to version 2.12 86

1. Introduction

1.1. The objectives of Normaliz

The program Normaliz 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 [4] and [6]. The terminology follows [4]. For algorithms of Normaliz see [5], [7], [8] and [9]. Some new developments are briefly explained in this manual.

Both polyhedra and lattices can be given by

(1) systems of generators and/or
(2) constraints.

In addition to generators and constraints, affine monoids can be defined by lattice ideals, in other words, by binomial equations.

In order to define a rational polyhedron by generators, one specifies a finite set of vertices $x_1,\dots,x_n \in \mathbb{Q}^d$ and a set $y_1,\dots,y_m \in \mathbb{Z}^d$ generating a rational cone C. The polyhedron defined by these generators is

$$P = \mathrm{conv}(x_1,\dots,x_n) + C, \qquad C = \mathbb{R}_+ y_1 + \cdots + \mathbb{R}_+ y_m.$$

An affine lattice defined by generators is a subset of $\mathbb{Z}^d$ given as

$$L = w + L_0, \qquad L_0 = \mathbb{Z}z_1 + \cdots + \mathbb{Z}z_r, \qquad w, z_1, \dots, z_r \in \mathbb{Z}^d.$$

Constraints defining a polyhedron are affine-linear inequalities with integral coefficients, and the constraints for an affine lattice are affine-linear diophantine equations and congruences. The conversion between generators and constraints is an important task of Normaliz.

The first main goal of Normaliz is to compute a system of generators for

$$P \cap L.$$

The minimal system of generators of the monoid $M = C \cap L_0$ is the Hilbert basis $\mathrm{Hilb}(M)$ of M. The homogeneous case, in which $P = C$ and $L = L_0$, is undoubtedly the most important one, and in this case $\mathrm{Hilb}(M)$ is the system of generators to be computed. In the general case the system of generators consists of $\mathrm{Hilb}(M)$ and finitely many points $u_1,\dots,u_s \in P \cap L$ such that

$$P \cap L = \bigcup_{j=1}^{s} u_j + M.$$

The second main goal are enumerative data that depend on a grading of the ambient lattice. Normaliz computes the Hilbert series and the Hilbert quasipolynomial of the monoid or set of lattice points in a polyhedron. In combinatorial terminology: Normaliz computes Ehrhart

series and quasipolynomials of rational polyhedra. Via its offspring NmzIntegrate [11], Normaliz computes generalized Ehrhart series and Lebesgue integrals of polynomials over rational polytopes.

The computation goals of Normaliz can be set by the user. In particular, they can be restricted to subtasks, such as the lattice points in a polytope or the leading coefficient of the Hilbert (quasi)polynomial.

Performance data of Normaliz can be found in [8].

Acknowledgement. The development of Normaliz is currently supported by the DFG SPP 1489 "Algorithmische und experimentelle Methoden in Algebra, Geometrie und Zahlentheorie".

1.2. Platforms, implementation and access from other systems

Executables for Normaliz are provided for Mac OS, Linux and MS Windows. If the executables prepared cannot be run on your system, then you can compile Normaliz yourself (see Section 10).

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). The parallelization is based on OpenMP.

The executables provided by us use the integer optimization program SCIP [1] for certain subtasks, but the inclusion of SCIP must be activated at compile time.

Normaliz consists of two parts: the front end normaliz for input and output and the C++ library libnormaliz that does the computations.

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 and libnormaliz,
- GAP via the GAP package NORMALIZINTERFACE [12] 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 2.12

(1) A new, more comfortable input syntax (with backward compatibility).
(2) New input types, in particular generators for lattices.
(3) Free combination of input types.
(4) Improved linear algebra with much better protection against overflows.

(5) Automatic choice of integer type.
(6) Reduction of the arithmetical complexity by subdivision of large simplicial cones and bottom decomposition based on SCIP and approximation methods.
(7) Improvement of Fourier-Motzkin elimination by ordering the generators.
(8) An improved programming interface.
(9) Massive parallelization on Intel Xeon Phi cards (experimental).

1.4. Future extensions

(1) Exploitation of symmetries.
(2) Access from further systems.
(3) Gröbner and Graver bases.

1.5. Download and installation

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 `Normaliz3.0` (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.)

2. Normaliz by examples

2.1. Terminology

For the precise interpretation of parts of the Normaliz output some terminology is necessary, but this section can be skipped at first reading, and the user can come back to it when it becomes necessary. We will give less formal descriptions along the way.

As pointed out in the introduction, Normaliz "computes" intersections $P \cap L$ where P is a rational polyhedron in $\mathbb{R}^d$ and L is an affine sublattice of $\mathbb{Z}^d$. It proceeds as follows:

(1) If the input is inhomogeneous, then it is homogenized by introducing a homogenizing coordinate: the polyhedron P is replaced by the cone $C(P)$: it is the closure of $\mathbb{R}_+(P \times \{1\}$ in $\mathbb{R}^{d+1}$. Similarly L is replaced by $\widetilde{L} = \mathbb{Z}(L \times \{1\})$. In the homogeneous case in which P is a cone and L is a subgroup of $\mathbb{Z}^d$, we set $C(P) = P$ and $\widetilde{L} = L$.

(2) The computations take place in the *efficient lattice*

$$\mathbb{E} = \widetilde{L} \cap \mathbb{R}C(P).$$

where $\mathbb{R}C(P)$ is the linear subspace generated by $C(P)$. The internal coordinates are chosen with respect to a basis of $\mathbb{E}$. The *efficient cone* is

$$\mathbb{C} = \mathbb{R}_+(C(P) \cap \mathbb{E}).$$

(3) Inhomogeneous computations are truncated using the dehomogenization (defined implicitly or explicitly).

(4) The final step is the conversion to the original coordinates. Note that we must use the coordinates of $\mathbb{R}^{d+1}$ if homogenization has been necessary, simply because some output vectors may be non-integral.

Normaliz computes inequalities, equations and congruences defining $\mathbb{E}$ and $\mathbb{C}$. The output contains only those constraints that are really needed. They must always be used jointly: the equations and congruences define $\mathbb{E}$, and the equations and inequalities define $\mathbb{C}$. Altogether they define the monoid $M = \mathbb{C} \cap \mathbb{E}$. In the homogeneous case this is the monoid to be computed. In the inhomogeneous case we must intersect M with the dehomogenizing hyperplane to obtain $P \cap L$.

2.2. Practical preparations

You may find it comfortable to run Normaliz via the GUI jNormaliz [2]. 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 of the input files

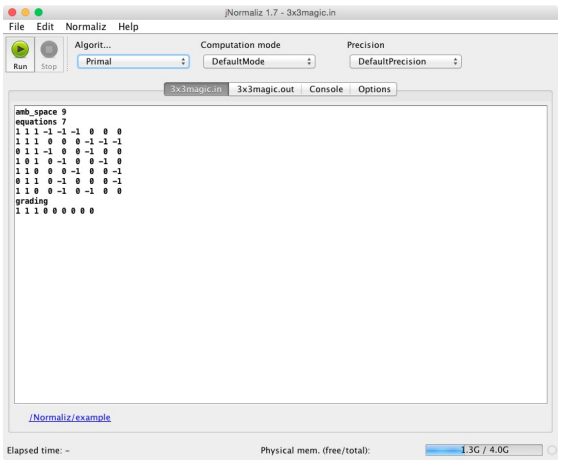


Figure 1: jNormaliz

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 [2] via the help menu.

Moreover, one can, and often will, run Normaliz from the command line. This is fully explained in Section 5. At this point it is enough to call Normaliz by typing

```
normaliz -c <project>
```

where `<project>` denotes for the project to be computed. Normaliz will load the file `project.in`. The option `-c` makes Normaliz to write a progress report on the terminal. Normaliz writes its results to `<project>.out`.

Note that you may have to prefix `normaliz` by a path name, and `<project>` must contain a path to the input file if it is not in the current directory. Suppose the Normaliz directory is the current directory and we are using a Linux or Mac system. Then

```
./normaliz -c example/small
```

will run `small.in` from the directory `example`. On Windows we must change this to

```
.\normaliz -c example\small
```

The commands given above will run Normaliz with the full parallelization that your system can deliver. For the very small examples in this tutorial you may want to add `-x=1` to suppress parallelization.

As long as you don't specify a computation goal on the command line or in the input file, Normaliz will use the *default computation goals*:

```
HilbertBasis
HilbertSeries
ClassGroup
```

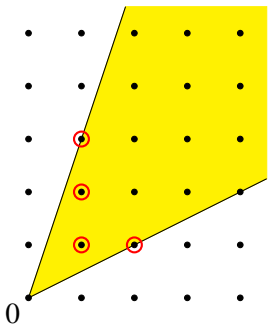
The computation of the Hilbert series requires the explicit or implicit definition of a grading. Normaliz does only complain that a computation goal cannot be reached if the goal has been set explicitly. For example, if you say `HilbertSeries` and there is no grading, an exception will be thrown and Normaliz terminates, but an output file with the already computed data will be written.

Normaliz will always print the results that are obtained on the way to the computation goals and do not require extra effort.

Appendix B helps you to read the console output that you have demanded by the option `-c`.

2.3. A cone in dimension 2

We want to investigate the cone $C = \mathbb{R}_+(2,1) + \mathbb{R}_+(1,3) \subset \mathbb{R}^2$:



This cone is defined in the input file `2cone.in`:

```
amb_space 2
cone 2
2 1
1 3
```

The input tells Normaliz that the ambient space is $\mathbb{R}^2$, and then a cone with 2 generators is defined, namely the cone C from above.

The figure indicates the Hilbert basis, and this is our first computation goal.

2.3.1. The Hilbert basis

In order to compute the Hilbert basis, we run Normaliz from jNormaliz or by

```
./normaliz -c example/2cone
```

and inspect the output file:

```
4 Hilbert basis elements
2 extreme rays
2 support hyperplanes
```

Self explanatory so far.

```
embedding dimension = 2
rank = 2 (maximal)
external index = 1
internal index = 5
original monoid is not integrally closed
```

The embedding dimension is the dimension of the space in which the computation is done. The rank is the rank of the lattice $\mathbb{E}$ (notation as in Section 2.1). In fact, in our example $\mathbb{E} = \mathbb{Z}^2$, and therefore has rank 2.

For subgroups $G \subset U \subset \mathbb{Z}^d$ we denote the order of the torsion subgroup of U/G by the *index* of G in U. The *external index* is the index of the lattice $\mathbb{E}$ in $\mathbb{Z}^d$. In our case $\mathbb{E} = \mathbb{Z}^d$, and therefore the external index is 1. Note: the external index is 1 exactly when $\mathbb{E}$ is a direct

summand of $\mathbb{Z}^d$.

For this example and many others the *original monoid* is well defined: the generators of the cone used as input are contained in $\mathbb{E}$. (This need not be the case if $\mathbb{E}$ is a proper sublattice of $\mathbb{Z}^d$, and we let the original monoid undefined in inhomogeneous computations.) Let G be the subgroup generated by the original monoid. The *internal index* is the index of G in $\mathbb{E}$.

The original monoid is *integrally closed* if and only if the it contains the Hilbert basis, and this is evidently false for our example. We go on.

```
size of triangulation   = 1
resulting sum of |det|s = 5
```

The primal algorithm of Normaliz relies on a (partial) triangulation. In our case the triangulation consists of a single simplicial cone, and (the absolute value of) its determinant is 5.

```
No implicit grading found
```

If you do not define a grading explicitly, Normaliz tries to find one itself: It is defined if and only if there is a linear form γ on $\mathbb{E}$ under which all extreme rays of the efficient cone $\mathbb{C}$ have value 1, and if so, γ is the implicit grading. Such does not exist in our case.

The last information before we come to the vector lists:

```
rank of class group = 0
finite cyclic summands:
5: 1
```

The class group of the monoid M has rank 0, in other words, it is finite. It has one finite cyclic summand of order 5.

This is the first instance of a multiset of integers displayed as a sequence of pairs

```
<n>: <m>
```

Such an entry says: the multiset contains the number `<n>` with multiplicity `<m>`.

Now we look at the vector lists (typeset in two columns to save space):

```
4 Hilbert basis elements:       2 extreme rays:
1 1                             1 3
1 2                             2 1
1 3
2 1                             2 support hyperplanes:
                                -1  2
                                 3 -1
```

The support hyperplanes are given by the linear forms (or inner normal vectors):

$$-x_1 + 2x_2 \geq 0,$$
$$3x_1 - x_2 \geq 0.$$

If the order is not fixed for some reason, Normaliz sorts vector lists as follows : (1) by degree if a grading exists and the application makes sense, (2) lexicographically.

2.3.2. The cone by inequalities

Instead by generators, we can define the cone by the inequalities just computed (`2cone_ineq.in`):

```
amb_space 2
inequalities 2
-1  2
3 -1
```

A matrix of input type `inequalities` contains *homogeneous* inequalities.

We get the same result as with `2cone.in` except that the data depending on the original monoid cannot be computed: the internal index and the information on the original monoid are missing since there is no original monoid.

2.3.3. The interior

Now we want to compute the lattice points in the interior of our cone. If the cone C is given by the inequalities $\lambda_i(x) \geq 0$ (within $\mathrm{aff}(C)$), then the interior is given by the inequalities $\lambda_i(x) > 0$. Since we are interested in lattice points, we work with the inequalities $\lambda_i(x) \geq 1$.

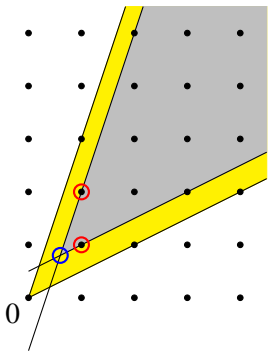
The input file `2cone_int.in` says

```
amb_space 2
strict_inequalities 2
-1  2
3 -1
```

The strict inequalities encode the conditions

$$-x_1 + 2x_2 \geq 1,$$
$$3x_1 - x_2 \geq 1.$$

This is our first example of inhomogeneous input.



Normaliz homogenizes inhomogeneous computations by introducing an auxiliary homogenizing coordinate x_{d+1}. The polyhedron is obtained by intersecting the homogenized cone with the hyperplane $x_{d+1} = 1$. The recession cone is the intersection with the hyperplane $x_{d+1} = 0$. The recession monoid is the monoid of lattice points in the recession cone, and the set of lattice points in the polyhedron is represented by its system of module generators over the recession monoid.

Note that the homogenizing coordinate serves as the denominator for rational vectors. In our example the recession cone is our old friend that we have already computed, and therefore we need not comment on it.

```
2 module generators
4 Hilbert basis elements of recession monoid
1 vertices of polyhedron
2 extreme rays of recession cone
2 support hyperplanes of polyhedron

embedding dimension = 3
affine dimension of the polyhedron = 2 (maximal)
rank of recession monoid = 2
```

The only surprise may be the embedding dimension: Normaliz always takes the dimension of the space in which the computation is done. It is the number of components of the output vectors. Because of the homogenization it has increased by 1.

```
size of triangulation   = 1
resulting sum of |det|s = 25
```

In this case the homogenized cone has stayed simplicial, but the determinant has changed.

```
dehomogenization:
0 0 1
```

The dehomogenization is the linear form δ on the homogenized space that defines the hyperplanes from which we get the polyhedron and the recession cone by the equations $\delta(x) = 1$ and $\delta(x) = 0$, respectively. It is listed since one can also work with a user defined dehomogenization.

```
module rank = 1
```

This is the rank of the module of lattice points in the polyhedron over the recession monoid. In our case the module is an ideal, and so the rank is 1.

The output of inhomogeneous computations is always given in homogenized form. The last coordinate is the value of the dehomogenization on the listed vectors, 1 on the module generators, 0 on the vectors in the recession monoid:

```
2 module generators:          4 Hilbert basis elements of recession monoid:
1 1 1                         1 1 0
1 2 1                         1 2 0
```

```
1 3 0
2 1 0
```

The module generators are $(1,1)$ and $(1,2)$.

```
1 vertices of polyhedron:
3 4 5
```

Indeed, the polyhedron has a single vertex, namely $(3/5,4/5)$.

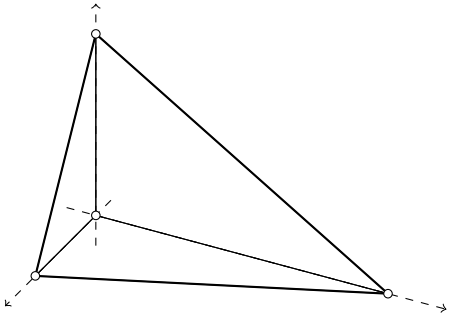
```
2 extreme rays of recession cone:     2 support hyperplanes of polyhedron:
1 3 0                                 -1  2 -1
2 1 0                                  3 -1 -1
```

The support hyperplanes are exactly those that we have used to define the polyhedron. This is obvious in our example, but need not always be true since one or more of the defining input hyperplanes may be superfluous.

2.4. A lattice polytope

The file `polytope.in` contains

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



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. The generators of the Ehrhart monoid are obtained by attaching a further coordinate 1 to the vertices, and this explains `amb_space 4`. In fact, the input type `polytope` is nothing but a convenient (perhaps superfluous) version of

```
amb_space 4
cone 4
0 0 0 1
2 0 0 1
0 3 0 1
0 0 5 1
```

Running `normaliz` produces the file `polytope.out`:

```
19 Hilbert basis elements
18 Hilbert basis elements of degree 1
```

```
4 extreme rays
4 support hyperplanes

embedding dimension = 4
rank = 4 (maximal)
external index = 1
internal index = 30
original monoid is not integrally closed
```

Perhaps a surprise: the lattice points of the polytope do not yield all Hilbert basis elements.

```
size of triangulation   = 1
resulting sum of |det|s = 30
```

Nothing really new so far. But now Normaliz finds a grading given by the last coordinate. See 3.8 below for general information on gradings.

```
grading:
0 0 0 1

degrees of extreme rays:
1: 4
```

Again we encounter the notation `<n>: <m>`: we have 4 extreme rays, all of degree 1.

```
Hilbert basis elements are not of degree 1
```

Perhaps a surprise: the polytope is not integrally closed as defined in [4]. Now we see the enumerative data defined by the grading:

```
multiplicity = 30

Hilbert series:
1 14 15
denominator with 4 factors:
1: 4

degree of Hilbert Series as rational function = -2

Hilbert polynomial:
1 4 8 5
with common denominator = 1
```

The polytope has $\mathbb{Z}^3$-normalized volume 30 as indicated by the multiplicity. The Hilbert (or Ehrhart) function counts the lattice points in kP, $k \in \mathbb{Z}_+$. The corresponding generating function is a rational function $H(t)$. For our polytope it is

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

The denominator is given in multiset notation: `1: 4` say that the factor $(1-t^1)$ occurs with multiplicity 4.

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

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

In our case it has integral coefficients, a rare exception. Therefore one usually needs a denominator.

Everything that follows has already been explained.

```
rank of class group = 0
finite cyclic summands:
30: 1

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

18 Hilbert basis elements of degree 1:
0 0 0 1
...
2 0 0 1

1 further Hilbert basis elements of higher degree:
1 2 4 2

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

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

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

2.4.1. Only the lattice points

Suppose we want to compute only the lattice points in our polytope. In the language of graded monoids these are the degree 1 elements, and so we add `Deg1Elements` to our input file (`polytope_deg1.in`):

```
amb_space 4
polytope 4
0 0 0
2 0 0
0 3 0
```

```
0 0 5
Deg1Elements
/* This is our first explicit computation goal*/
```

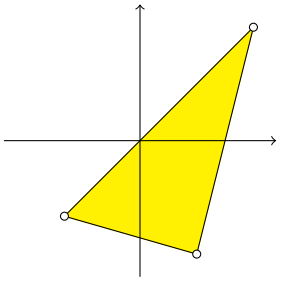
We have used this opportunity to include a comment in the input file.

We lose all information on the Hilbert series, and from the Hilbert basis we only retain the degree 1 elements.

2.5. A rational polytope

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

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



Then we write v_i with a common denominator:

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

The generator matrix is given by the rows

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

We must add a grading since Normaliz cannot recognize it without help (unless all the q_i are equal to 1). The grading linear form has coordinates $(0, \ldots, 0, 1)$.

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

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

For this example the procedure above yields the input file `rational.in`:

```
amb_space 3
cone 3
1 1 2
-1 -1 3
1 -2 4
grading
unit_vector 3
HilbertSeries
```

This is the first time that we used the shortcut `unit_vector <n>` which represents the nth unit vector $e_n \in \mathbb{R}^d$ and is only allowed for input types which require a single vector.

From the output file we only list the data of the Ehrhart series.

```
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

degree of Hilbert Series as rational function = -3

Hilbert series with cyclotomic denominator:
-1 -1 -1 -3 -4 -3 -2
cyclotomic denominator:
1: 3  2: 2  3: 1  4: 1

Hilbert quasi-polynomial of period 12:
0:   48 28 15                        7:   23 22 15
1:   11 22 15                        8:   16 28 15
2:  -20 28 15                        9:   27 22 15
3:   39 22 15                        10:   -4 28 15
4:   32 28 15                        11:    7 22 15
5:   -5 22 15                        with common denominator = 48
6:   12 28 15
```

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 $5/16$.

Unlike in the case of a lattice polytope, there is no canonical choice of the denominator of the Ehrhart series. Normaliz gives it in 2 forms. In the first form the numerator polynomial is

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

and the denominator is

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

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. 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.

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.

Warning: It is tempting, but not a good idea to define the polytope by the input type `vertices`. It would make Normaliz compute the lattice points in the polytope, but not in the cone over the polytope, and we need these to determine the Ehrhart series.

2.5.1. The rational polytope by inequalities

We extract the support hyperplanes of our polytope from the output file and use them as input (`poly_ineq.in`):

```
amb_space 3
inequalities 3
-8  2 3
1 -1 0
2  7 3
grading
unit_vector 3
HilbertSeries
```

These data tell us that the polytope, as a subset of $\mathbb{R}^2$, is defined by the inequalities

$$\begin{aligned} -8x_1 + 2x_2 + 3 &\geq 0, \\ x_1 - x_2 + 0 &\geq 0, \\ 2x_1 + 7x_2 + 3 &\geq 0. \end{aligned}$$

These inequalities are inhomogeneous, but we are using the homogeneous input type `inequalities` which amounts to introducing the grading variable x_3, as we have done it for the generators.

Why don't we define it by the "natural" inhomogeneous inequalities using `inhom_inequalities`? We could do it, but then only the polytope itself would be the object of computation, and we would have no access to the Ehrhart series. We could just compute the lattice points in the polytope. (Try it.)

2.6. Magic squares

Suppose that you are interested in the following 'type of '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 is encoded in the file `3x3magic.in`:

```
amb_space 9
equations 7
1 1 1 -1 -1 -1  0  0  0
1 1 1  0  0  0 -1 -1 -1
0 1 1 -1  0  0 -1  0  0
1 0 1  0 -1  0  0 -1  0
1 1 0  0  0 -1  0  0 -1
```

```
0 1 1  0 -1  0  0  0 -1
1 1 0  0 -1  0 -1  0  0
grading
1 1 1 0 0 0 0 0 0
```

The input type `equations` represents *homogeneous* equations. The first equation reads

$$x_1 + x_2 + x_3 - x_4 - x_5 - x_6 = 0,$$

and the other equations are to be interpreted analogously. The magic constant is a natural choice for the grading.

It seems that we have forgotten to define the cone. This may indeed be the case, but doesn't matter: if there is no input type that defines a cone, Normaliz chooses the positive orthant, and this is exactly what we want in this case.

The output file contains the following:

```
5 Hilbert basis elements
5 Hilbert basis elements of degree 1
4 extreme rays
4 support hyperplanes

embedding dimension = 9
rank = 3
external index = 1

size of triangulation   = 2
resulting sum of |det|s = 4

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

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.

```
degrees of extreme rays:
1: 4

Hilbert basis elements are of degree 1
```

This was not to be expected (and is no longer true for 4×4 squares).

```
multiplicity = 4

Hilbert series:
1 2 1
denominator with 3 factors:
```

```
1: 3

degree of Hilbert Series as rational function = -1

Hilbert polynomial:
1 2 2
with common denominator = 1
```

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 $\mathscr{M}$, provided 3 divides $\mathscr{M}$. (If $3 \nmid \mathscr{M}$, there is no magic square of magic constant $\mathscr{M}$.)

```
rank of class group = 1
finite cyclic summands:
2: 2
```

So the class group is $\mathbb{Z} \oplus (\mathbb{Z}/2\mathbb{Z})^2$.

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

0 further Hilbert basis elements of higher degree:
```

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. One of these 5 squares is clearly in the interior:

```
4 extreme rays:                 4 support hyperplanes:
0 2 1 2 1 0 1 0 2               -2 -1 0 0  4 0 0 0 0
1 0 2 2 1 0 0 2 1               0 -1 0 0  2 0 0 0 0
1 2 0 0 1 2 2 0 1               0  1 0 0  0 0 0 0 0
2 0 1 0 1 2 1 2 0               2  1 0 0 -2 0 0 0 0
```

These 4 support hyperplanes cut out the cone generated by the magic squares from the linear subspace they generate. Only one is reproduced as a sign inequality. This is due to the fact that

the linear subspace has submaximal dimension and there is no unique lifting of linear forms to the full space.

```
6 equations:                    3 basis elements of lattice:
1 0 0 0 0  1 -2 -1  1           1 0 -1 -2 0  2  1  0 -1
0 1 0 0 0  1 -2  0  0           0 1 -1 -1 0  1  1 -1  0
0 0 1 0 0  1 -1 -1  0           0 0  3  4 1 -2 -1  2  2
0 0 0 1 0 -1  2  0 -2
0 0 0 0 1 -1  1  0 -1
0 0 0 0 0  3 -4 -1  2
```

So one of our equations has turned out to be superfluous (why?). Note that also the equations are not reproduced exactly. Finally, Normaliz lists a basis of the efficient lattice $\mathbb{E}$ generated by the magic squares.

2.6.1. With even corners

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 by the following (`3x3magiceven.in`):

```
congruences 4
1 0 0 0 0 0 0 0 0 2
0 0 1 0 0 0 0 0 0 2
0 0 0 0 0 0 1 0 0 2
0 0 0 0 0 0 0 0 1 2
```

The first 9 entries in each row represent the coefficients of the coordinates in the homogeneous congruences, and the last is the modulus:

$$x_1 \equiv 0 \mod 2$$

is the first congruence etc.

The output changes accordingly:

```
9 Hilbert basis elements
0 Hilbert basis elements of degree 1
4 extreme rays
4 support hyperplanes

embedding dimension = 9
rank = 3
external index = 4

size of triangulation   = 2
resulting sum of |det|s = 8

grading:
```

```
1 1 1 0 0 0 0 0 0
with denominator = 3

degrees of extreme rays:
2: 4

multiplicity = 1

Hilbert series:
1 -1 3 1
denominator with 3 factors:
1: 1  2: 2

degree of Hilbert Series as rational function = -2

Hilbert series with cyclotomic denominator:
-1 1 -3 -1
cyclotomic denominator:
1: 3  2: 2

Hilbert quasi-polynomial of period 2:
0:   2 2 1
1:  -1 0 1
with common denominator = 2
```

After the extensive discussion in Section 2.5 it should be easy for you to write down the Hilbert series and the Hilbert quasipolynomial. (But keep in mind that the grading has a denominator.)

```
rank of class group = 1
finite cyclic summands:
4: 2

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

0 Hilbert basis elements of degree 1:

9 further Hilbert basis elements of higher degree:
...

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

We have listed the extreme rays since they have changed after the introduction of the congruences, although the cone has not changed. The reason is that Normaliz always chooses the

extreme rays from the efficient lattice $\mathbb{E}$.

```
4 support hyperplanes:
...

6 equations:
...                              3 basis elements of lattice:
                                 2 0 -2 -4 0  4  2 0 -2
2 congruences:                   0 1  2  3 1 -1  0 1  2
1 0 0 0 0 0 0 0 0 2              0 0  6  8 2 -4 -2 4  4
0 1 0 0 1 0 0 0 0 2
```

The rank of the lattice has of course not changed, but after the introduction of the congruences the basis has changed.

2.6.2. The lattice as input

It is possible to define the lattice by generators. We demonstrate this for the magic squares with even corners. The lattice has just been computed (`3x3magiceven_lat.in`):

```
amb_space 9
lattice 3
2 0 -2 -4 0  4  2 0 -2
0 1  2  3 1 -1  0 1  2
0 0  6  8 2 -4 -2 4  4
grading
1 1 1 0 0 0 0 0 0
```

It produces the same output as the version starting from equations and congruences.

`lattice` has a variant that takes the saturation of the sublattice generated by the input vectors (`3x3magic_sat.in`):

```
amb_space 9
saturation 3
2 0 -2 -4 0  4  2 0 -2
0 1  2  3 1 -1  0 1  2
0 0  6  8 2 -4 -2 4  4
grading
1 1 1 0 0 0 0 0 0
```

Clearly, we remove the congruences by this choice and arrive at the output of `3x3magic.in`.

2.7. Decomposition in a numerical semigroup

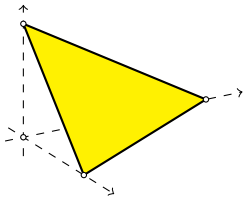
Let $S = \langle 6, 10, 15 \rangle$, the numerical semigroup generated by $6, 10, 15$. How can 97 be written as a sum in the generators?

In other words: we want to find all nonnegative integral solutions to the equation

$$6x_1 + 10x_2 + 15x_3 = 97$$

Input (`NumSemi.in`):

```
amb_space 3
inhom_equations 1
6 10 15 -97
```



The equation cuts out a triangle from the positive orthant.

The set of solutions is a module over the monoid M of solutions of the homogeneous equation $6x_1 + 10x_2 + 15x_3 = 0$. So $M = 0$.

```
6 module generators:
2 1 5 1
2 4 3 1
2 7 1 1
7 1 3 1
7 4 1 1
12 1 1 1

0 Hilbert basis elements of recession monoid:
```

The last line is as expected, and the 6 module generators are the goal of the computation.

Normaliz is smart enough to recognize that it must compute the lattice points in a polygon, and does exactly this. You can recognize it in the console output: it contains the line

```
Converting polyhedron to polytope
```

2.8. A job for the dual algorithm

We increase the size of the magic squares to 5×5. Normaliz can do the same computation as for 3×3 squares, but this will take some minutes. If we are only interested in the Hilbert basis, we should use the dual algorithm for this example. The input file is `5x5dual.in`:

```
amb_space 25
equations 11
1 1 1 1 1 -1 -1 -1 -1 -1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
...
1 1 1 1 0  0  0  0 -1  0  0  0 -1  0  0  0 -1  0  0  0 -1  0  0  0  0
```

```
DualMode
grading
1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

The choice of the dual algorithm implies the computation goal `HilbertBasis`. By adding `Deg1Elements` we could restrict the computation to the degree 1 elements.

The Hilbert basis contains 4828 elements, too many to be listed here.

If you want to run this example with default computation goals, use the file `5x5.in`. It will compute the Hilbert basis and the Hilbert series.

The size 6×6 is out of reach for the Hilbert series, but the Hilbert basis can be computed in dual mode. It takes some hours.

2.9. A dull polyhedron

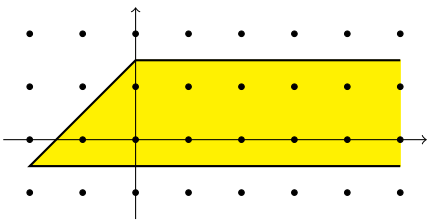
We want to compute the polyhedron defined by the inequalities

$$\begin{aligned} \xi_2 \geq -1/2 &\iff 0\xi_1 + 2\xi_2 + 1 \geq 0 \\ \xi_2 \leq 3/2 &\iff 0\xi_1 - 2\xi_2 + 3 \geq 0 \\ \xi_2 \leq \xi_1 + 3/2 &\iff 2\xi_1 - 2\xi_2 + 3 \geq 0 \end{aligned}$$

They are contained in the input file `InhomIneq.in`:

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

The grading says that we want to count points by the first coordinate.



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

embedding dimension = 3
affine dimension of the polyhedron = 2 (maximal)
rank of recession monoid = 1

size of triangulation   = 1
resulting sum of |det|s = 8

dehomogenization:
0 0 1

grading:
1 0 0
```

The interpretation of the grading requires some care in the inhomogeneous case. We have extended the input grading vector by an entry 0 to match the embedding dimension. For the computation of the degrees of *lattice points* in the ambient space you can either use only the first 2 coordinates or take the full scalar product of the point in homogenized coordinates and the extended grading vector.

```
module rank = 2
multiplicity = 2
```

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

```
Hilbert series:
1 1
denominator with 1 factors:
1: 1

shift = -1
```

We haven't seen a shift yet. It is always printed (necessarily) if the Hilbert series does not start in degree 0. In our case it starts in degree -1 as indicated by the shift -1. We thus get the Hilbert series

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

Note: We used the opposite convention for the shift in Normaliz 2.

Note that the Hilbert (quasi)polynomial is always computed for the unshifted monoid defined by the input data. (This was different in previous versions of Normaliz.)

```
degree of Hilbert Series as rational function = -1

Hilbert polynomial:
2
```

```
with common denominator = 1

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

2 module generators:
-1 0 1
0 1 1

1 Hilbert basis elements of recession monoid:
1 0 0

2 vertices of polyhedron:
-4 -1 2
0  3 2

1 extreme rays of recession cone:
1 0 0

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

2.9.1. Defining it by generators

If the polyhedron is given by its vertices and the recession cone, we can define it by these data (`InhomIneq_gen.in`):

```
amb_space 2
vertices 2
-4 -1 2
0  3 2
cone 1
1 0
grading
unit_vector 1
```

The output is identical to the version starting from the inequalities.

2.10. The Condorcet paradoxon

In social choice elections each of the k voters picks a preference order of the n candidates. There are $n!$ such orders.

We say that candidate A *beats* candidate B if the majority of the voters prefers A to B. As the

Marquis de *Condorcet* (and others) observed, "beats" is not transitive, and an election may exhibit the *Condorcet paradoxon*: there is no Condorcet winner. (See [10] and the references given there for more information.)

We want to find the probability for $k \to \infty$ that there is a Condorcet winner for $n = 4$ candidates. The event that A is the Condorcet winner can be expressed by linear inequalities on the election outcome (a point in 24-space). The wanted probability is the lattice normalized volume of the polytope cut out by the inequalities at $k = 1$. The file `Condorcet.in`:

```
amb_space 24
inequalities 3
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
nonnegative
total_degree
Multiplicity
```

The first inequality expresses that A beats B, the second and the third say that A beats C and D. (So far we do not exclude ties, and they need not be excluded for probabilities as $k \to \infty$.)

In addition to these inequalities we must restrict all variables to nonnegative values, and this is achieved by adding the attribute `nonnegative`. The grading is set by `total_degree`. It replaces the grading vector with 24 entries 1. Finally `Multiplicity` sets the computation goal.

From the output file we only mention the quantity we are out for:

```
multiplicity = 1717/8192
```

Since there are 4 candidates, the probability for the existence of a Condorcet winner is $1717/2048$.

We can refine the information on the Condorcet paradoxon by computing the Hilbert series. Either we delete `Multiplicity` from the input file or, better, we add `--HilbertSeries` (or simply `-q`) on the command line. The result:

```
Hilbert series:
1 5 133 363 4581 8655 69821 100915 ... 12346 890 481 15 6
denominator with 24 factors:
1: 1  2: 14  4: 9

degree of Hilbert Series as rational function = -25
```

2.10.1. Excluding ties

Now we are more ambitious and want to compute the Hilbert series for the Condorcet paradoxon, or more precisely, the number of election outcomes having A as the Condorcet winner depending on the number k of voters. Moreover, as it is customary in social choice theory, we want to exclude ties. The input file changes to `CondorcetSemi.in`:

```
amb_space 24
excluded_faces 3
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
nonnegative
total_degree
HilbertSeries
```

We could omit `HilbertSeries`, and the computation would include the Hilbert basis. The type `excluded_faces` only affects the Hilbert series. In every other respect it is equivalent to `inequalities`.

From the file `CondorcetSemi.out` we only display the Hilbert series:

```
Hilbert series:
6 15 481 890 12346 ... 100915 69821 8655 4581 363 133 5 1
denominator with 24 factors:
1: 1  2: 14  4: 9

shift = 1

degree of Hilbert Series as rational function = -24
```

Surprisingly, this looks like the Hilbert series in the previous section read backwards, roughly speaking. This is true, and one can explain it as we will see below.

It is justified to ask why we don't use `strict_inequalities` instead of `excluded_faces`. It does of course give the same Hilbert series. If there are many excluded faces, then it is better to use `strict_inequalities`. However, at present NmzIntegrate can only work with `excluded_faces`.

2.10.2. At least one vote for every preference order

Suppose we are only interested in elections in which every preference order is chosen by at least one voter. This can be modeled as follows (`Condorcet_one.in`):

```
amb_space 24
inequalities 3
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
strict_signs
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
total_degree
HilbertSeries
```

The entry 1 at position i of the vector `strict_signs` imposes the inequality $x_i \geq 1$. A -1 would impose the inequality $x_i \leq -1$, and the entry 0 imposes no condition on the i-th coordinate.

```
Hilbert series:
1 5 133 363 4581 8655 69821 100915 ... 12346 890 481 15 6
denominator with 24 factors:
1: 1  2: 14  4: 9

shift = 24

degree of Hilbert Series as rational function = -1
```

Again we encounter (almost) the Hilbert series of the Condorcet paradoxon (without side conditions). It is time to explain this coincidence. Let C be the Condorcet cone defined by the nonstrict inequalities, M the monoid of lattice points in it, $I_1 \subset M$ the ideal of lattice points avoiding the 3 facets defined by ties, I_2 the ideal of lattice points with strictly positive coordinates, and finally I_3 the ideal of lattice points in the interior of C. Moreover, let $\mathbb{1} \in \mathbb{Z}^{24}$ be the vector with all entries 1.

Since $\mathbb{1}$ lies in the three facets defining the ties, it follows that $I_2 = M + \mathbb{1}$. This explains why we obtain the Hilbert series of I_2 by multiplying the Hilbert series of M by t^{24}, as just observed. Generalized Ehrhart reciprocity (see [4, Theorem 6.70]) then explains the Hilbert series of I_1 that we observed in the previous section. Finally, the Hilbert series of I_3 that we don't have displayed is obtained from that of M by "ordinary" Ehrhart reciprocity. But we can also obtain I_1 from I_3: $I_1 = I_3 - \mathbb{1}$, and generalized reciprocity follows from ordinary reciprocity in this very special case.

The essential point in these arguments (apart from reciprocity) is that $\mathbb{1}$ lies in all support hyperplanes of C except the coordinate hyperplanes.

You can easily compute the Hilbert series of I_3 by making all inequalities strict.

2.11. Testing normality

We want to test the monoid $A_{4\times4\times3}$ defined by $4 \times 4 \times 3$ contingency tables for normality (see [5] for the background). The input file is `A443.in`:

```
amb_space 40
cone_and_lattice 48
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
...
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1
HilbertBasis
```

Why `cone_and_lattice`? Well, we want to find out whether the monoid is normal, i.e., whether $M = C(M) \cap \mathrm{gp}(M)$. If M is even integrally closed in $\mathbb{Z}^{24}$, then it is certainly integrally closed in the evidently smaller lattice $\mathrm{gp}(M)$, but the converse does not hold in general, and therefore we work with the lattice generated by the monoid generators.

It turns out that the monoid is indeed normal:

```
original monoid is integrally closed
```

Actually the output file reveals that M is even integrally closed in $\mathbb{Z}^{24}$: the external index is 1, and therefore $\mathrm{gp}(M)$ is integrally closed in $\mathbb{Z}^{24}$.

The output files also shows that there is a grading on $\mathbb{Z}^{24}$ under which all our generators have degree 1. We could have seen this ourselves: Every generator has exactly one entry 1 in the first 16 coordinates. (This is clear from the construction of M.)

A noteworthy detail from the output file:

```
size of partial triangulation   = 48
```

It shows that Normaliz uses only a partial triangulation in Hilbert basis computations; see [5].

It is no problem to compute the Hilbert series as well if you are interested in it. Simply add `-q` to the command line or remove `HilberBasis` from the input file. Then a full triangulation is needed (size $2,654,272$).

Similar examples are `A543`, `A553` and `A643`. The latter is not normal. Even on a standard PC or laptop, the Hilbert basis computation does not take very long because Normaliz uses only a partial triangulation. The Hilbert series can still be determined, but the computation time will grow considerably since the it requires a full triangulation. See [8] for timings.

2.12. Inhomogeneous congruences

We want to compute the nonnegative solutions of the simultaneous inhomogeneous congruences

$$x_1 + 2x_2 \equiv 3 \quad (7),$$
$$2x_1 + 2x_2 \equiv 4 \quad (13)$$

in two variables. As usual we move the right hand side over to the left. The input file `InhomCong.in` is

```
amb_space 2
inhom_congruences 2
1 2 -3 7
2 2 -4 13
```

The first vector list in the output:

```
3 module generators:
 0 54 1
 1  1 1
80  0 1
```

Easy to check: if $(1,1)$ is a solution, then it must generate the module of solutions together with the generators of the intersections with the coordinate axes. Perhaps more difficult to

find:

```
6 Hilbert basis elements of recession monoid:
 0 91 0
 1 38 0
 3 23 0                                          1 vertices of polyhedron:
 5  8 0                                          0 0 91
12  1 0
91  0 0
```

Strange, why is $(0,0,1)$, representing the origin in $\mathbb{R}^2$, not listed as a vertex as well? Well the vertex shown represents an extreme ray in the lattice $\mathbb{E}$, and $(0,0,1)$ does not belong to $\mathbb{E}$.

```
2 extreme rays of recession cone:
 0 91 0
91  0 0

2 support hyperplanes of polyhedron:
0 1 0
1 0 0

1 congruences:
58 32 1 91
```

Normaliz has simplified the system of congruences to a single one.

```
3 basis elements of lattice:
1 0  33
0 1 -32
0 0  91
```

Again, don't forget that Normaliz prints a basis of the efficient lattice $\mathbb{E}$.

2.12.1. Lattice and offset

The set of solutions to the inhomogeneous system is an affine lattice in $\mathbb{R}^2$. The lattice basis of $\mathbb{E}$ above does not immediately let us write down the set of solutions in the form $w+L_0$ with a subgroup L_0, but we can easily transform the basis of $\mathbb{E}$: just add the first and the second vector to obtain $(1,1,1)$ – we already know that it belongs to $\mathbb{E}$ and any element in $\mathbb{E}$ with last coordinate 1 would do. Try the file `InhomCongLat.in`:

```
amb_space 2
offset
1 1
lattice 2
32 33
91 91
```

2.12.2. Variation of the signs

Suppose we want to solve the system of congruences under the condition that both variables are negative (`InhomCongSigns.in`):

```
amb_space 2
inhom_congruences 2
1 2 -3 7
2 2 -4 13
signs
-1 -1
```

The two entries of the sign vector impose the sign conditions $x_1 \leq 0$ and $x_2 \leq 0$.

From the output we see that the module generators are more complicated now:

```
4 module generators:
-11   0 1
 -4  -7 1
 -2 -22 1
  0 -37 1
```

The Hilbert basis of the recession monoid is simply that of the nonnegative case multiplied by -1.

2.13. Integral closure and Rees algebra of a monomial ideal

Next, let us discuss the example `MonIdeal.in` (typeset in two columns):

```
amb_space 5
rees_algebra 9
1 2 1 2                 1 0 3 4
3 1 1 3                 5 1 0 1
2 5 1 0                 2 4 1 5
0 2 4 3                 2 2 2 4
0 2 3 4
```

The input vectors are the exponent vectors of a monomial ideal I in the ring $K[X_1,X_2,X_3,X_4]$. We want to compute the normalization of the Rees algebra of the ideal. In particular we can extract from it the integral closure of the ideal. Since we must introduce an extra variable T, we have `amb_space 5`.

In the Hilbert basis we see the exponent vectors of the X_i, namely the unit vectors with last component 0. The vectors with last component 1 represent the integral closure $\bar{I}$ of the ideal. There is a vector with last component 2, showing that the integral closure of I^2 is larger than $\bar{I}^2$.

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

```
...
5 1 0 1 1
6 5 2 2 2

11 generators of integral closure of the ideal:
0 2 3 4
...
5 1 0 1
```

The output of the generators of $\bar{I}$ is the only place where we suppress the homogenizing variable for "historic" reasons. If we extract the vectors with last component 1 from the extreme rays, then we obtain the smallest monomial ideal that has the same integral closure as I.

```
10 extreme rays:
0 0 0 1 0
...
5 1 0 1 1
```

The support hyperplanes which are not just sign conditions describe primary decompositions of all the ideals $\overline{I^k}$ by valuation ideals. It is not hard to see that none of them can be omitted for large k (for example, see: W. Bruns and G. Restuccia, Canonical modules of Rees algebras. J. Pure Appl. Algebra 201, 189–203 (2005)).

```
23 support hyperplanes:
0 0 0  0   1
0 ...
6 0 1  3 -13
```

2.13.1. Only the integral closure

If only the integral closure of the ideal is to be computed, one can choose the input as follows (`IntClMonId.in`):

```
amb_space 4
vertices 9
1 2 1 2 1
...
2 2 2 4 1
cone 4
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
```

The generators of the integral closure appear as module generators in the output and the generators of the smallest monomial ideal with this integral closure are the vertices of the polyhedron.

2.14. Only the convex hull

Normaliz computes convex hulls as should be very clear by now, and the only purpose of this section is to emphasize that the computation can be restricted to it by setting an explicit computation goal. We choose the input of the preceding section and add the computation goal (`IntClMonIdSupp.in`):

```
amb_space 4
vertices 9
1 2 1 2 1
...
2 2 2 4 1
vertices
cone 4
1 0 0 0
...
0 0 0 1
SupportHyperplanes
```

As you can see from the output, the support hyperplanes of the polyhedron are computed as well as the extreme rays.

2.15. Starting from a binomial 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 vectors are obtained as the differences of the two exponent vectors in the binomials. So the input ideal `lattice_ideal.in` is

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

In order to avoid special input rules for this case in which our object is not defined as a subset of an ambient space, but as a quotient of type *generators/relations*, we abuse the name `amb_space`: it determines the space in which the input vectors live.

We get the output

```
6 original generators of the toric ring
```

namely the residue classes of the indeterminates.

```
9 Hilbert basis elements
9 Hilbert basis elements of degree 1
```

So the toric ring defined by the binomials is not normal. Normaliz found the standard grading on the toric ring. The normalization is generated in degree 1, too (in this case).

```
5 extreme rays
5 support hyperplanes

embedding dimension = 3
rank = 3 (maximal)
external index = 1
internal index = 1
original monoid is not integrally closed
```

We saw that already.

```
size of triangulation   = 5
resulting sum of |det|s = 10

grading:
-2 1 1
```

This is the grading on the ambient space (or polynomial ring) defining the standard grading on our subalgebra. The enumerative data that follow are those of the normalization!

```
degrees of extreme rays:
1: 5

Hilbert basis elements are of degree 1

multiplicity = 10

Hilbert series:
1 6 3
denominator with 3 factors:
1: 3

degree of Hilbert Series as rational function = -1

Hilbert polynomial:
1 3 5
with common denominator = 1

rank of class group = 2
class group is free
```

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

6 original generators:
 0 0 1
 3 5 2
 0 1 0
 1 2 1
 1 3 0
 1 0 3
```

This is an embedding of the toric ring defined by the binomials. There are many choices, and Normaliz has taken one of them. You should check that the generators in this order satisfy the binomial equations. Turning to the ring theoretic interpretation, we can say that the toric ring defined by the binomial equations can be embedded into $K[Y_1, Y_2, Y_3]$ as a monomial subalgebra that is generated by $Y_1^0Y_2^0Y_3^1, \ldots, Y_1^1Y_2^0Y_3^3$.

Now the generators of the normalization:

```
9 Hilbert basis elements of degree 1:    5 extreme rays:
0 0 1                                    0 0 1
0 1 0                                    0 1 0
1 0 3                                    1 0 3
1 1 2                                    1 3 0
1 2 1                                    3 5 2
1 3 0
2 3 2                                    5 support hyperplanes:
2 4 1                                    -15 7 5
3 5 2                                    -3 1 2
                                         0 0 1
                                         0 1 0
                                         1 0 0

0 further Hilbert basis elements of higher degree:
```

3. The input file

The input file `<project>.in` consists of one or several items. There are several types of items:

(1) definition of the ambient space,
(2) matrices with integer entries,
(3) vectors with integer entries,
(4) computation goals and algorithmic variants,
(5) comments.

An item cannot include another item. In particular, comments can only be included between other items, but not within another item.

Matrices and vectors are classified by the following attributes:

(1) generators, constraints, accessory,
(2) cone/polyhedron, (affine) lattice,
(3) homogeneous, inhomogeneous.

In this classification, equations are considered as constraints on the lattice because Normaliz treats them as such – for good reason: it is very easy to intersect a lattice with a hyperplane.

The line structure is irrelevant for the interpretation of the input, but it is advisable to use it for the readability of the input file.

The input syntax of Normaliz 2 can still be used. It is explained in Appendix C.

3.1. Input items

3.1.1. The ambient space and lattice

The ambient space is specified as follows:

```
amb_space <d>
```

where <d> stands for the dimension d of the ambient vector space $\mathbb{R}^d$ in which the geometric objects live. The *ambient lattice* $\mathbb{A}$ is set to $\mathbb{Z}^d$.

In the following the letter d will always denote the dimension set with amb_space.

An example:

```
amb_space 5
```

indicates that polyhedra and lattices are subobjects of $\mathbb{R}^5$. The ambient lattice is $\mathbb{Z}^5$.

The first non-comment input item must specify the ambient space. There are no further rules for the order of the items in the input file.

3.1.2. Matrices

A matrix is built as follows:

```
<T> <m>
<x_1>
...
<x_m>
```

Here <T> denotes the type of the matrix, <m> the number of rows, and <x_1>,...,<x_m> are the rows. The number of columns is implicitly defined by the dimension of the ambient space and the type of the matrix. Example (with amb_space 3):

```
cone 3
1 2 3
```

```
4 5 6
11 12 13
```

3.1.3. Vectors

A vector is built as follows:

```
<T>
<x>
```

Again <T> denotes the type and <x> is the vector itself. The number of components is determined by the type of the vector and the dimension of the ambient space. At present, all vectors have length d.

Example:

```
grading
1 0 0
```

For certain vectors there exist shortcuts. Examples:

```
total_degree
unit_vector 25
```

3.1.4. Computation goals and algorithmic variants

These are single or compound words, such as

```
HilbertBasis
Multiplicity
```

The file can contain several computation goals, as in this example.

3.1.5. Comments

A comment has the form

```
/* <text> */
```

where <text> stands for the text of the comment. It can have arbitrary length and stretch over several lines. Example:

```
/* This is a comment
*/
```

Comments are only allowed at places where also a new keyword would be allowed, especially not between the entries of a matrix or a vector. Comments can not be nested.

3.1.6. Restrictions

Input items can almost freely be combined, but there are some restrictions:

(1) Every input type can appear only once.
(2) There can be at most one matrix of homogeneous cone generators. The types
`cone, cone_and_lattice, polytope, rees_algebra`
exclude each other mutually.
(3) There can be at most one matrix of homogeneous lattice generators. The types
`lattice`, `saturation`, `cone_and_lattice`
exclude each other mutually.
(4) `polytope` can not be combined with `grading`.
(5) The only type that can be combined with `lattice_ideal` is `grading`.
(6) The following types cannot be combined with inhomogeneous types or `dehomogenization`:
`polytope, rees_algebra, excluded_faces`
(7) The following types cannot be combined with inhomogeneous types:
`dehomogenization, support_hyperplanes`

Apart from these restrictions, homogeneous and inhomogeneous types can be combined as well as generators and constraints. A single inhomogeneous type or `dehomogenization` in the input triggers an inhomogeneous computation.

3.1.7. Default values

If there is no lattice defining item, Normaliz (virtually) inserts the the unit matrix as an input item of type `lattice`. If there is no cone defining item, the unit matrix is (additionally) inserted as an input item of type `cone`.

If the input is inhomogeneous, then Normaliz provides default values for vertices and the offset as follows:

(1) If there is an input matrix of lattice type lattice, but no `offset`, then the offset 0 is inserted.
(2) If there is an input matrix of type cone, but no `vertices`, then the vertex 0 is inserted.

3.1.8. Normaliz takes intersections (almost always)

The input may contain several cone defining items and several lattice defining items.

The sublattice L defined by the lattice input items is the *intersection* of the sublattices defined by the single items. The polyhedron P is defined as the intersection of all polyhedra defined by the single polyhedron defining items. The object then computed by Normaliz is

$$P \cap L.$$

There are two notable exceptions to the rule that Normaliz takes intersections:

(1) `vertices` and `cone` form a unit. Together they define a polyhedron.
(2) The same applies to `offset` and `lattice` that together define an affine lattice.

3.2. Homogeneous generators

3.2.1. Cones

The main type is `cone`. The other two types are added for special computations.

`cone` is a matrix with d columns. Every row represents a vector, and they define the cone generated by them. Section 2.3, `2cone.in`

`polytope` is a matrix with $d-1$ columns. It is internally converted to `cone` extending each row by an entry 1. Section 2.4, `polytope.in`

`rees_algebra` is a matrix with $d-1$ columns. It is internally converted to type `cone` in two steps: (i) each row is extended by an entry 1 to length d. (ii) The first $d-1$ unit vectors of length d are appended. Section 2.13, `MonIdeal.in`.

Moreover, it is possible to define a cone and a lattice by the same matrix:

`cone_and_lattice` The vectors of the matrix with d columns define both a cone and a lattice. Section 2.11, `A443.in`.

The Normaliz 2 types `integral_closure` and `normalization` can still be used. They are synonyms for `cone` and `cone_and_lattice`, respectively.

3.2.2. Lattices

There are 3 types:

`lattice` is a matrix with d columns. Every row represents a vector, and they define the lattice generated by them. Section 2.6.2, `3x3magiceven_lat.in`

`saturation` is a matrix with d columns. Every row represents a vector, and they define the *saturation* of the lattice generated by them. Section 2.6.2, `3x3magic_sat.in`.

`cone_and_lattice` See Section 3.2.1.

3.3. Homogeneous Constraints

3.3.1. Cones

`inequalities` is a matrix with d columns. Every row $(\xi_1,\dots,\xi_d)$ represents a homogeneous inequality

$$\xi_1 x_1 + \cdots + \xi_d x_d \geq 0, \qquad \xi_i \in \mathbb{Z},$$

for the vectors $(x_1,\dots,x_d) \in \mathbb{R}^d$. Sections 2.3.2, 2.5.1 , `2cone_ineq.in`, `poly_ineq.in`

`signs` is a vector with d entries in $\{-1,0,1\}$. It stands for a matrix of type `inequalities` composed of the sign inequalities $x_i \geq 0$ for the entry 1 at the i-th component and the

inequality $x_i \leq 0$ for the entry -1. The entry 0 does not impose an inequality. See 2.12.2, `InhomCongSigns.in`.

`nonnegative` It stands for a vector of type `sign` with all entries equal to 1. See Section 2.10, `Condorcet.in`.

`excluded_faces` is a matrix with d columns. Every row $(\xi_1,\dots,\xi_d)$ represents an inequality

$$\xi_1 x_1 + \cdots + \xi_d x_d > 0, \qquad \xi_i \in \mathbb{Z},$$

for the vectors $(x_1,\dots,x_d) \in \mathbb{R}^d$. It is considered as a homogeneous input type though it defines inhomogeneous inequalities. The faces of the cone excluded by the inequalities are excluded from the Hilbert series computation, but `excluded_faces` behaves like `inequalities` in every other respect . Section 2.10.1, `CondorcetSemi.in`.

`support_hyperplanes` is a matrix with d columns. It requires homogeneous input. It is the input type for precomputed support hyperplanes. Therefore Normaliz checks if all input generators satisfy the inequalities defined by them. Apart from this extra check, it behaves like `inequalities`. Section 6.9, `2cone_supp.in`.

3.3.2. Lattices

`equations` is a matrix with d columns. Every row $(\xi_1,\dots,\xi_d)$ represents an equation

$$\xi_1 x_1 + \cdots + \xi_d x_d = 0, \qquad \xi_i \in \mathbb{Z},$$

for the vectors $(x_1,\dots,x_d) \in \mathbb{R}^d$. Section 2.6, `3x3magic.in`

`congruences` is a matrix with $d+1$ columns. Each row $(\xi_1,\dots,\xi_d,c)$ represents a congruence

$$\xi_1 z_1 + \cdots + \xi_d z_d \equiv 0 \mod c, \qquad \xi_i, c \in \mathbb{Z},$$

for the elements $(z_1,\dots,z_d) \in \mathbb{Z}^d$. Section 2.6.1, `3x3magiceven.in`.

3.4. Inhomogeneous generators

3.4.1. Polyhedra

`vertices` is a matrix with $d+1$ columns. Each row $(p_1,\dots,p_d,q)$, $q>0$, specifies a generator of a polyhedron (not necessarily a vertex), namely

$$v_i = \left(\frac{p_1}{q},\dots,\frac{p_n}{q}\right), \qquad p_i \in \mathbb{Z}, q \in \mathbb{Z}_{>0},$$

Section 2.9.1, `InhomIneq_gen.in`

Note: `vertices` and `cone` together define a polyhedron. If `vertices` is present in the input, then the default choice for `cone` is the empty matrix.

The Normaliz 2 input type `polyhedron` can still be used.

3.4.2. Lattices

offset is a vector with d entries. It defines the origin of the affine lattice. Section 2.12.1, `InhomCongLat.in`.

Note: `offset` and `lattice` (or `saturation`) together define an affine lattice. If `offset` is present in the input, then the default choice for `lattice` is the empty matrix.

3.5. Inhomogeneous constraints

3.5.1. Cones

inhom_inequalities is a matrix with $d+1$ columns. We consider inequalities

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

rewritten as

$$\xi_1 x_1 + \cdots + \xi_d x_d + (-\eta) \geq 0$$

and then represented by the input vectors

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

Section 2.9, `InhomIneq.in`.

strict_inequalities is a matrix with d columns. We consider inequalities

$$\xi_1 x_1 + \cdots + \xi_d x_d \geq 1, \qquad \xi_i \in \mathbb{Z},$$

represented by the input vectors

$$(\xi_1, \ldots, \xi_d).$$

Section 2.3.3, `2cone_int.in`.

strict_signs is a vector with d components in $\{-1, 0, 1\}$. It is the "strict" counterpart to `signs`. An entry 1 in component i represents the inequality $x_i > 0$, an entry -1 the opposite inequality, whereas 0 imposes no condition on x_i. 2.10.2, `Condorcet_one.in`

3.5.2. Lattices

inhom_equations is a matrix with $d+1$ columns. We consider equations

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

rewritten as

$$\xi_1 x_1 + \cdots + \xi_d x_d + (-\eta) = 0$$

and then represented by the input vectors

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

See 2.7`NumSemi.in`.

inhom_congruences We consider a matrix with $d+2$ columns. Each the row $(\xi_1,\dots,\xi_d,-\eta,c)$ represents a congruence

$$\xi_1 z_1 + \cdots + \xi_d z_d \equiv \eta \mod c, \qquad \xi_i,\eta,c \in \mathbb{Z},$$

for the elements $(z_1,\dots,z_d) \in \mathbb{Z}^d$. Section 2.12, `InhomCong.in`.

3.6. Relations

Relations do not select a sublattice of $\mathbb{Z}^d$ or a subcone of $\mathbb{R}^d$, but define a monoid as a quotient of $\mathbb{Z}^d_+$ 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 $U \subset \mathbb{Z}^d$, and Normaliz computes an affine monoid and its normalization as explained in Section A.5.

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

lattice_ideal is a matrix with d columns containing the generators of the subgroup U. Section 2.15, `lattice_ideal.in`.

The type `lattice_ideal` cannot be combined with any other input type (except `grading`)—such a combination would not make sense. (See Section 3.8.1 for the use of a grading in this case.)

3.7. Unit vectors

A grading or a dehomogenization is often given by a unit vector:

unit_vector <n> represents the nth unit vector in $\mathbb{R}^d$ where n is the number given by `<n>`.

This shortcut cannot be used as a row of a matrix. It can be used whenever a single vector is asked for, namely after `grading`, `dehomogenization`, `signs` and `strict_signs`. See Section 2.5, `rational.in`

3.8. Grading

This type is accessory. A $\mathbb{Z}$-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 given by an equation $\lambda(x) = 1$ with a $\mathbb{Z}$-linear form λ. If so, then λ is used as the grading.
Implicit gradings are only possible for homogeneous computations.

Explicit definition of a grading:

grading is a vector of length d representing the linear form that gives the grading. Section 2.5, `rational.in`.

total_degree represents a vector of length d with all entries equal to 1. Section 2.10, `Condorcet.in`.

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 $g > 1$. Then g is extracted from the degrees, and it will appear as `denominator` in the output file.

Normaliz checks whether all generators of the (recession) monoid have positive degree. Vertices of polyhedra may have degrees ≤ 0.

3.8.1. 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.

3.9. Dehomogenization

Like `grading` this is an accessory type.

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:

dehomogenization is a vector of length d representing the linear form δ.

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.)

The input type `dehomogenization` can only be combined with homogeneous input types, but makes the computation inhomogeneous, resulting in inhomogeneous output. The polyhedron computed is the intersection of the cone $\mathbb{C}$ (and the lattice $\mathbb{E}$) with the hyperplane given by $\delta(x) = 1$, and the recession cone is $\mathbb{C} \cap \{x : \delta(x) = 0\}$.

A potential application is the adaptation of other input formats to Normaliz. The output must then be interpreted accordingly.

Section 6.5, `dehomogenization.in`.

3.10. 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)!
```

3.11. 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.

3.12. 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 [11].

4. Computation goals and algorithmic variants

The library `libnormaliz` contains a class `ConeProperties` that collects computation goals, algorithmic variants and additional data that are used to control the work flow in `libnormaliz` as well as the communication with other programs. The latter are not important for the Normaliz user, but are listed as a reference for `libnormaliz`.

All computation goals and algorithmic variants can be communicated to Normaliz in two ways:

(1) in the input file, for example `HilbertBasis`,
(2) via a verbatim command line option, for example `--HilbertBasis`.

For the most important choices there are single letter command line options, for example `-N` for `HilbertBasis`. The single letter options ensure backward compatibility to Normaliz 2. In jNormaliz they are also accessible via their full names.

Some computation goals apply only to homogeneous computations, and some others make sense only for inhomogeneous computations.

Some single letter command line options combine two or more computation goals, and some algorithmic variants imply computation goals.

Normaliz can call NmzIntegrate. The three computation goals that require NmzIntegrate do not belong to the class `ConeProperties`.

4.1. Default choices and basic rules

If several computation goals are set, all of them are pursued. In particular, computation goals in the input file and on the command line are accumulated. But

`--ignore, -i` on the command line switches off the computation goals and algorithmic variants set in the input file.

The default computation goal is set if neither the input file nor command line contains a computation goal or an algorithmic variant that implies a computation goal. It is

`HilbertBasis` + `HilbertSeries` + `ClassGroup`.

If set explicitly in the input file or on the command line the following adds these computation goals:

`DefaultMode`

It is possible to set `DefaultMode` explicitly in addition to other computation goals. If it is set, implicitly or explicitly, Normaliz will not complain about unreachable computation goals.

Normaliz collects all computation goals and algorithmic variants set in the input file and on the command line. Note that some options may block others during the computation. For example, `KeepOrder` blocks `BottomDecomposition`. Moreover, Normaliz tries to omit superfluous computations. For example, if the dual mode is called for the degree 1 elements, but also the Hilbert series is to be computed, then the dual mode will be skipped since the degree 1 elements can be got as a cheap byproduct of the Hilbert series.

4.2. Computation goals

The computation goal `Sublattice` does not imply any other computation goal. All other computation goals include `Sublattice` and `SupportHyperplanes`.

4.2.1. Lattice data

`Sublattice, -S` (upper case S) asks Normaliz to compute the data of the efficient sublattice.

4.2.2. Support hyperplanes and extreme rays

`SupportHyperplanes, -s` triggers the computation of support hyperplanes and extreme rays.

Normaliz tries to find a grading. In the inhomogeneous case the module rank is computed.

4.2.3. Hilbert basis and related data

`HilbertBasis, -N` triggers the computation of the Hilbert basis. In inhomogeneous computations it asks for the Hilbert basis of the recession monoid *and* the module generators.

`Deg1Elements, -1` restricts the computation to the degree 1 elements of the Hilbert basis. Requires the presence of a grading. Forbidden in inhomogeneous computations.

`ModuleGeneratorsOverOriginalMonoid, -M` computes a minimal system of generators of the integral closure over the original monoid (see Section 6.8). Requires the existence of original monoid generators. Forbidden in inhomogeneous computations.

4.2.4. Enumerative data

The computation goals in this section require a grading. They include `SupportHyperplanes`.

`HilbertSeries,-q` triggers the computation of the Hilbert series.

`Multiplicity, -v` restricts the computation to the multiplicity.

4.2.5. Combined computation goals

Can only be set by single letter command line options:

- **`-n`** `HilbertBasis` + `Multiplicity`
- **`-h`** `HilbertBasis` + `HilbertSeries`
- **`-p`** `Deg1Elements` + `HilbertSeries`

4.2.6. The class group

`ClassGroup, -C` is self explanatory, includes `SupportHyperplanes`. Not allowed in inhomogeneous computations.

4.2.7. Triangulation and Stanley decomposition

`Triangulation, -T` makes Normaliz to compute, store and export the full triangulation.

`TriangulationSize, -t` makes Normaliz count the simplicial cones in the full triangulation.

`TriangulationDetSum` makes Normaliz additionally sum the absolute values of their determinants.

`StanleyDec, -y` makes Normaliz compute, store and export the Stanley decomposition. Only allowed in homogeneous computations.

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.

4.2.8. NmzIntegrate

The computation goals that require calling NmzIntegrate can only be set on the command line.

-E makes Normaliz compute a generalized Ehrhart series.

-L makes Normaliz compute the leading coefficient of a generalized Ehrhart series.

-I makes Normaliz compute an integral.

These computation goals require a homogeneous computation.

4.2.9. Boolean valued computation goals

They tell Normaliz to find out the answers to the questions they ask.

IsPointed : is the efficient cone $\mathbb{C}$ pointed?

IsDeg1ExtremeRays : do the extreme rays have degree 1?

IsDeg1HilbertBasis : do the Hilbert basis elements have degree 1?

IsIntegrallyClosed : is the original monoid integrally closed?.

IsReesPrimary : for the input type `rees_algebra`, is the monomial ideal primary to the irrelevant maximal ideal?

These computation goals are not really useful for Normaliz since they will be answered automatically. Note that they may trigger extensive computations.

4.3. Algorithmic variants

The default choice is the Normaliz primal algorithm that is based on a (partial) triangulation.

DualMode, -d activates the dual algorithm for the computation of the Hilbert basis and degree 1 elements. Includes `HilbertBasis`, unless `Deg1Elements` is set.

Approximate, -r activates the approximation algorithm for the computation of degree 1 elements. Nevertheless it does *not* imply `Deg1Elements` (we don't want to block potential other applications of `Approximate`.)

BottomDecomposition, -b tells Normaliz to use bottom decomposition in the primal algorithm.

KeepOrder, -k forbids Normaliz to reorder the generators of the efficient cone $\mathbb{C}$. Only useful if original monoid generators are defined. Blocks `BottomDecomposition`.

4.4. Integer type

There is no need to worry about the integer type chosen by Normaliz. All preparatory computations use indefinite precision. The main computation is then tried with 64 bit integers. If it fails, it will be restarted with indefinite precision. The 64 bit attempt can be surpassed by

BigInt, -B

Note that Normaliz tries to avoid overflows by intermediate results. If such overflow should happen, the computation is repeated locally with indefinite precision. (The number of such GMP transitions is shown in the terminal output.) If a final result is too large, Normaliz must globally restart the computation.

`BigInt` is not a cone property.

4.5. Control of computations

In addition to the computation goals in Section 4.2, the following elements of `ConeProperties` control the work flow in `libnormaliz` and can be used by programs calling Normaliz to ensure the availability of the data that are controlled by them.

`ModuleGenerators` controls the module generators in inhomogeneous computation.

`ExtremeRays` controls the extreme rays.

`VerticesOfPolyhedron` controls the vertices of the polyhedron in the inhomogeneous case.

`HilbertFunction` controls the Hilbert (quasi)polynomial.

`RecessionRank` controls the rank of the recession monoid in inhomogeneous computations.

`AffineDim` controls the affine dimension of the polyhedron in inhomogeneous computations.

`ModuleRank` in inhomogeneous computations it controls the rank of the module of lattice points in the polyhedron as a module over the recession monoid.

`ExcludedFaces` controls the excluded faces.

`InclusionExclusionData` controls data derived from the excluded faces.

`Grading` controls the grading.

`Dehomogenization` controls the dehomogenization.

`Generators` controls the generators of the efficient cone.

`OriginalMonoidGenerators` controls the generators of the original monoid.

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

5. Running Normaliz

The standard form for calling Normaliz is

```
normaliz [options] <project>
```

where `<project>` is the name of the project, and the corresponding input file is `<project>.in`. Note that `normaliz` may require to be prefixed by a path name, and the same applies to `<project>`. A typical example on a Linux or Mac system:

```
./normaliz --verbose -x=5 ../example/big
```

that for MS Windows must be converted to

```
.\normaliz --verbose -x=5 example\big
```

Normaliz uses the standard conventions for calls from the command line:

(1) the order of the arguments on the command line is arbitrary.
(2) Single letter options are prefixed by the character `-` and can be grouped into one string.
(3) Verbatim options are prefixed by the characters `--`.

The options for computation goals and algorithmic variants have been described in Section 4. In this section the remaining options for the control of execution and output are discussed, together with some basic rules on the use of the options.

5.1. Basic rules

The options for computation goals and algorithms variants have been explained in Section 4. The options that control the execution and the amount of output will be explained in the following. Basic rules for the use of options:

1. If no `<project>` is given, the program will terminate.
2. The option `-x` differs from the other ones: `<T>` in `-x=<T>` represents a positive number assigned to `-x`; see Section 5.3.
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. The options can be given in arbitrary order. All options, including those in the input file, are accumulated, and syntactically there is no mutual exclusion. However, some options may block others during the computation. For example, `KeepOrder` blocks `BottomDecomposition`.
5. 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.
6. In the options include `DefaultMode`, Normaliz does not complain about missing data (anymore). It will simply omit those computations that are impossible.
7. 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.

5.2. Info about Normaliz

`--help, -?` displays a help screen listing the Normaliz options.
`--version` displays information about the Normaliz executable.

5.3. Control of execution

The options that control the execution are:

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

-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 8.1 for further discussion.

5.4. Control of output files

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

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

--all-files, -a includes Files, Normaliz writes all available output files (except typ, the triangulation or the Stanley decomposition, unless these have been requested).

--<suffix> chooses the output file with suffix <suffix>.

For the list of potential output files, their suffixes and their interpretation see Section 7. There may be several options --<suffix>.

5.5. Overriding the options in the input file

Since Normaliz accumulates options, one cannot get rid of settings in the input file by command line options unless one uses

--ignore, -i This option disables all settings in the input file.

6. More examples

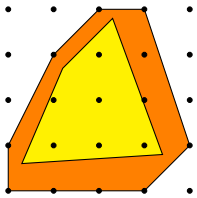
In this section we discuss some advanced features of Normaliz.

6.1. Lattice points by approximation

In order to find the lattice points in a polytope P, old versions of Normaliz created the same vectors that are needed for the Hilbert basis computations, but used that no reduction is necessary to find the degree 1 points. They are automatically part of the Hilbert basis. While this observation speeded up the computations considerably, the number of vectors to be created could simply make the computation impossible. This was especially true for rational polytopes whose vertices have large denominators.

Normaliz 3 takes the following approach: simplicial subcones S with rational vertices are approximated by integral polytopes, and only the lattice points in S are kept. This is done

automatically. Since version 2.11 the user can also choose a global approximation of the input polytope P by the option `Approximate`, and this is often a very good choice. It is not the default choice of Normaliz since the number of vertices of P should not be too large.



Another option that is often favorable is `DualMode` in connection with `Deg1Elements`. So there are three choices for the computation of lattice points in polytopes:

`Deg1Elements, -1` with local approximation of rational polytopes,

`Approximate Deg1Elements, -r1` combined with global approximation,

`DualMode Deg1Elements, -d1` dual mode optimized for the computation of degree 1 elements.

Our demonstration example is `max_polytope_cand` with the input file

```
amb_space 5
inequalities 11
-3  -3 -10  14  88
26  -3  19  14 361
-4  25   6 -49 133
2  -3   1   5  17
-2  -3  -7  11  61
-6  23   9 -30 233
-2   7   3  -9  81
-8  17  12 -65 183
8 -15   4  23  65
-8 -13 -28  45 241
-8  27  12 -35 321
grading
unit_vector 5
Approximate
Deg1Elements
```

set up for global approximation. (This is not a random input file; it has come up in connection with the paper "Quantum jumps of normal polytopes" by W. Bruns, J. Gubeladze and M. Michałek, arXiv:1504.01036.)

Despite of the large number of lattice points, the computation is extremely fast. Now remove `Approximate`. This will take considerably longer, but will not overstretch your patience. Note that Normaliz will pass to GMP integers, which becomes necessary because of the huge determinants of the simplicial cones (without approximation).

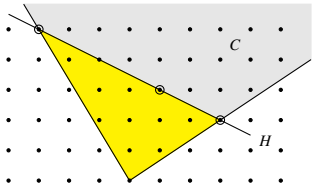
The third choice using the dual algorithm is still o.k. though it takes the most time in this case. In other cases it may very well be the fastest.

You can also use inhomogeneous input with the default computation mode or dual mode. This is equivalent to homogeneous input with `Deg1Elements` and `DualMode`, respectively, in addition.

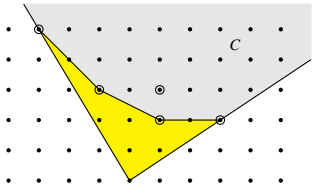
6.2. The bottom decomposition

The triangulation size and the determinant sum of the triangulation are critical size parameters in Normaliz computations. Normaliz always tries to order the generators in such a way that the determinant sum is close to the minimum, and on the whole this works out well. The use of the bottom decomposition by `BottomDecomposition, -b` enables Normaliz to compute a triangulation with the optimal determinant sum for the given set of generators, as we will explain in the following.

The determinant sum is independent of the order of the generators of the cone C if they lie in a hyperplane H. Then the determinant sum is exactly the normalized volume of the polytope spanned by 0 and $C \cap H$. The triangulation itself depends on the order, but the determinant sum is constant.



This observation helps to find a triangulation with minimal determinant sum in the general case. We look at the *bottom* (the union of the compact faces) of the polyhedron generated by $x_1, \ldots, x_n$ as vertices and C as recession cone, and take the volume underneath the bottom:



With the option `BottomDecomposition, -b`, Normaliz 3.0 computes a triangulation that respects the bottom facets. This yields the optimal determinant sum for the given generators. If one can compute the Hilbert basis by the dual algorithm, it can be used as input, and then one obtains the absolute bottom of the cone, namely the compact facets of the convex hull of all nonzero lattice points.

Normaliz does not always use the bottom decomposition by default since its computation requires some time and administrative overhead. However, as soon as the input "profile" is considered to be "rough" it is invoked. The measure of roughness is the ratio between the maximum degree (or L_1 norm without a grading) and the minimum. A ratio ≥ 10 activates the bottom decomposition.

The bottom decomposition is part of the subdivision of large simplicial cones discussed in the next section.

Also see Section 6.11.

Note: the bottom decomposition cannot be activated by Normaliz if `KeepOrder` is used. In this case `BottomDecomposition` must be set explicitly if it is wanted.

6.3. Subdivision of large simplicial cones

Especially in computations with rational polytopes one encounters very large determinants that can keep the Normaliz primal algorithm from terminating in reasonable time. As an example we take `hickerson-18.in` from the LattE distribution [3]. It is simplicial and the complexity is totally determined by the large determinant $\approx 4.17 \times 10^{14}$ (computed with `-v`).

If we are just interested in the degree 1 points, Normaliz uses the approximation method of Section 6.1 and finds 44 degree 1 points very quickly. If we use these points together with the extreme rays of the simplex, then the determinant sum decreases to $\approx 1.3 \times 10^{12}$, and the computation of the Hilbert basis and the Hilbert series is in reach. But it is better to pursue the idea of subdividing large simplicial cones systematically. Normaliz employs two methods:

(1) computation of subdivision points by the IP solver SCIP,
(2) computation of candidate subdivision points by approximation of the given simplicial cone by an overcone that is generated by vectors of "low denominator".

Normaliz tries to subdivide a simplicial cone if it has determinant $\geq 10^8$. Both methods are used recursively via stellar subdivision until simplicial cones with determinant $< 10^6$ have been reached or no further improvement is possible. Furthermore, if some big simplices are still remaining, method (2) is applied again in both cases with a higher approximation level. All subdivision points are then collected, and the start simplicial cone is subdivided with bottom decomposition, which in general leads to substantial further improvement.

The use of SCIP requires a Normaliz executable that is compiled with the option `SCIP=yes` after the installation of SCIP. Without SCIP only the approximation method is used. However, it can happen that SCIP fails because the required precision cannot be reached by floating point calculations. In this vase the approximation method will be tried as well.

The following table contains some performance data for subdivisions based on SCIP (parallelization with 20 threads).

	`hickerson-16`	`hickerson-18`	`knapsack_11_60`
simplex volume	9.83×10^7	4.17×10^{14}	2.8×10^{14}
volume under bottom	8.10×10^5	3.86×10^7	2.02×10^7
volume used	3.93×10^6	5.47×10^7	2.39×10^7
runtime without subdivision	2 s	>12 d	>8 d
runtime with subdivision	0.5 s	46 s	5.1 s

Performance data for approximation:

	hickerson-16	hickerson-18	knapsack_11_60
volume used	3.9×10^6	9.1×10^7	2.3×10^9
runtime with subdivision	0.7 s	58 s	2 m 36 s

A good nonsimplicial example showing the subdivision at work is `hickerson_18plus1.in`.

Note: After subdivision the decomposition of the cone may no longer be a triangulation in the strict sense, but a decomposition that we call a *nested triangulation*; see 6.6.1.

6.4. Primal vs. dual – division of labor

The choice between the primal and the dual algorithm for Hilbert basis computations is presently left to the user. Normaliz does not make a guess which algorithm might be faster. If the number of support hyperplanes is small relative to the dimension, then the dual algorithm can be much faster. 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 [8] contains computation times for many examples that can help the user to choose the right algorithm. Note that the dual algorithm is arithmetically less critical than the primal algorithm since it basically only requires addition of vectors.

When both Hilbert basis and Hilbert series are to be computed, the best solution can be the combination of both algorithms. We recommend `2equations.in` as a demonstration example which combines the algorithmic variant `DualMode` and the computation goal `HilbertSeries`:

```
amb_space 9
equations 2
1 6 -7 -18 25 -36 6 8 -9
7 -13 15 6 -9 -8 11 12 -2
total_degree
DualMode
HilbertSeries
```

The computation time (20 parallel threads) is 2 m 54 s whereas the default mode needs 4 h 4 m 20 s.

In addition to showing that the combination of the primal and dual algorithm can be very useful, it also shows the power of the the subdivision of large simplicial cones and bottom decomposition: while the "raw" determinant sum is $\approx 7.7 \times 10^{11}$ (you can compute it with the options `-iv`), the determinant sum of the Hilbert series computation is only $\approx 8.4 \times 10^9$.

6.5. Explicit dehomogenization

Inhomogeneous input for data in $\mathbb{R}^d$ is homogenized by an extra $(d+1)$th coordinate. The dehomogenization sets the last coordinate equal to 1. Other systems may prefer the first co-

ordinate. By choosing an explicit dehomogenization Normaliz can be adapted to such input. The file `dehomogenization.in`

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

indicates that in this case the first variable is the homogenizing one. The output file

```
1 module generators
2 Hilbert basis elements of recession monoid
1 vertices of polyhedron
2 extreme rays of recession cone
2 support hyperplanes of polyhedron

embedding dimension = 3
affine dimension of the polyhedron = 2 (maximal)
rank of recession monoid = 2

size of triangulation   = 0
resulting sum of |det|s = 0

dehomogenization:
1 0 0


module rank = 1

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

1 module generators:
1 1 1

2 Hilbert basis elements of recession monoid:
0 0 1
0 1 0

1 vertices of polyhedron:                   2 support hyperplanes of polyhedron:
1 1 1                                       -1 0 1
                                            -1 1 0
2 extreme rays of recession cone:
0 0 1
0 1 0
```

shows that Normaliz does the computation in the same way as with implicit dehomogenization, except that now the first coordinate decides what is in the polyhedron and what belongs to the recession cone, roughly speaking.

Note that the dehomogenization need not be a coordinate. It can be any linear form that is nonnegative on the cone generators.

6.6. Exporting the triangulation

The option `-T` asks Normaliz to export the triangulation by writing the files `<project>.tgn` and `<project>.tri`:

`tgn` The file `tgn` contains a matrix of vectors (in the coordinates of $\mathbb{A}$) spanning the simplicial cones in the triangulation.

`tri` 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 $\mathbb{E}$).

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

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

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          plain
0 -1 1
```

We see the 4 vertices $v_1, \dots, v_4$ in homogenized coordinates in `tgn` and the 2 simplices (or the simplicial cones over them) in `tri`: both have multiplicity 2. The last word `plain` indicates that Normaliz has computed a triangulation in the strict sense, namely a simplicial subdivision in which neighboring simplicial cones match along common faces. The alternative is `nested` that we discuss below.

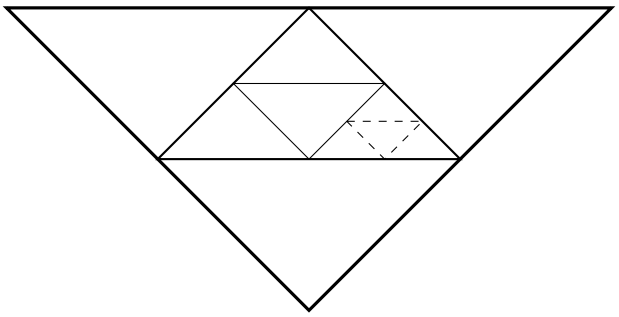
In addition to the files <project>.tgn and <project>.tri, also the file <object>.inv is written. It contains the data of the file <project>.out above the line of stars in a human and machine readable format.

6.6.1. Nested triangulations

If Normaliz has subdivided a simplicial cone of a triangulation of the cone C, the resulting decomposition of C may no longer be a triangulation in the strict sense. It is rather a *nested triangulation*, namely a map from a rooted tree to the set of full-dimensional subcones of C with the following properties:

(1) the root is mapped to C,
(2) every other node is mapped to a full dimensional simplicial subcone,
(3) the simplicial subcones corresponding to the branches at a node x form a triangulation of the simplicial cone corresponding to x.

The following figure shows a nested triangulation:



For the Normaliz computations, nested triangulations are as good as ordinary triangulations, but in other applications the difference may matter. With the option -T, Normaliz prints the leaves of the nested triangulation to the tri file. They constitute the simplicial cones that are finally evaluated by Normaliz.

The triangulation is always plain if -T is the only computation goal or if it is just combined with -v. Otherwise it can only fail to be plain if it contains determinants $\geq 10^8$.

6.7. Exporting the Stanley decomposition

The option -y makes Normaliz write the files <project>.tgn, <project>.dec and <project>.inv. Stanley decomposition is contained in the file with the suffix dec. But this file also contains the inclusion/exclusion data if there are excluded faces:

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

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

In the notation of [8], each line lists an "offset" $x + \varepsilon(x)$ by its coordinates with respect to $v_{i_1}, \dots, v_{i_m}$ as follows: if $(a_1, \dots, a_m)$ 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

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

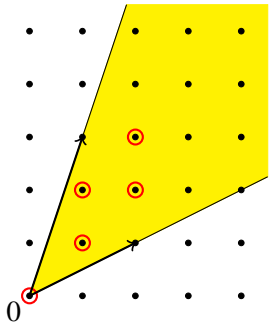
The file `3x3magiceven.in` has been processed with the option `-ahTy` activated. We recommend you to inspect all the output files in the subdirectory `example` of the distribution.

6.8. Module generators over the original monoid

Suppose that the original generators are well defined in the input. This is always the case when these consists just of a `cone` or a `cone_and_lattice`. Let M be the monoid generated by them. Then Normaliz computes the integral closure N of M in the effective lattice $\mathbb{E}$. It is often interesting to understand the difference set $N \setminus M$. After the introduction of a field K of coefficients, this amounts to understanding $K[N]$ as a $K[M]$-module. With the option `ModuleGeneratorsOverOriginalMonoid, -M` Normaliz computes a minimal generating set T of this module. Combinatorially this means that we find an irreducible cover

$$N = \bigcup_{x \in T} x + M.$$

Note that $0 \in T$ since $M \subset N$.



As an example, we can run `2cone.in` with the option `-M` on the command line. This yields the output

```
...
4 Hilbert basis elements:
 1 1
 1 2
 1 3
 2 1

2 extreme rays:
 1 3
 2 1

5 module generators over original monoid:
 0 0
 1 1
 1 2
 2 2
 2 3
```

6.9. Precomputed support hyperplanes

Computing the support hyperplanes can be a very time consuming task, and if it has been the first step in the exploration of a difficult example, it may be desirable, to use the support hyperplanes as additional input in order to save computation time. This is especially true if Normaliz must do an intermediate computation of the support hyperplanes because of a large number of simplicial cones to be evaluated. The file `2cone_supp.in` is just a toy example:

```
amb_space 2
cone 2
2 1
1 3
support_hyperplanes 2
-1  2
 3 -1
```

As pointed out in Section 3.3.1, Normaliz must trust you—here is no way of checking the correctness of this input without recomputing it.

6.10. Shift, denominator, quasipolynomial and multiplicity

In this section we discuss the interplay of shift, denominator of the grading and the quasipolynomial. As long as the denominator is 1, the situation is very simple and no ambiguity arises. See Section 2.9. We modify the example from that section as follows (`InhomIneq_7.in`):

```
amb_space 2
inhom_inequalities 3
0  2  1
0 -2  3
2 -2  3
grading
7 0
```

The output related to the grading is

```
grading:
7 0 0
with denominator = 7


module rank = 2
multiplicity = 2

Hilbert series:
1 1
denominator with 1 factors:
1: 1

shift = -1

degree of Hilbert Series as rational function = -1

Hilbert polynomial:
2
with common denominator = 1
```

The Hilbert series computed by hand is

$$\frac{t^{-7}+1}{1-t^7}.$$

We obtain it from the output as follows. The printed series is

$$\frac{1+t}{1-t}.$$

Now the shift is applied and yields

$$\frac{t^{-1}+1}{1-t}.$$

Finally we make the substitution $t \mapsto t^7$, and obtain the desired result.

Now we add the complication $x_1 + x_2 \equiv -1 \mod 8$ (`InhomIneq_7_8.in`):

```
amb_space 2
inhom_inequalities 3
0  2  1
0 -2  3
2 -2  3
grading
7 0
inhom_congruences 1
1 1 1 8
```

The result:

```
grading:
7 0 0
with denominator = 7


module rank = 2
multiplicity = 1/4

Hilbert series:
1 0 0 0 0 0 0 1
denominator with 1 factors:
8: 1

shift = -1

degree of Hilbert Series as rational function = -2

Hilbert series with cyclotomic denominator:
-1 1 -1 1 -1 1 -1
cyclotomic denominator:
1: 1  4: 1  8: 1

Hilbert quasi-polynomial of period 8:
0:  0           4:  0
1:  0           5:  0
2:  0           6:  1
3:  0           7:  1
                with common denominator = 1
```

The printed Hilbert series is

$$\frac{1+t^7}{1-t^8}.$$

The application of the shift yields

$$\frac{t^{-1}+t^6}{1-t^8}.$$

the correct result for the divided grading. *The Hilbert quasipolynomial is computed for the divided grading*, as already explained in Section 2.6.1. As a last step, we can apply the substitution $t \mapsto t^7$ in order obtain the Hilbert series

$$\frac{t^{-7}+t^{42}}{1-t^{56}}$$

for the original grading.

Like the quasipolynomial, *the multiplicity is computed for the divided grading*.

6.11. A very large computation

The example `CondEffPlur.in` is one of the largest that Normaliz has mastered. The goal is to compute the Condorcet efficiency of plurality voting for 4 candidates (see [10] and the references therein), even in the refined form of the Hilbert series. The Hilbert basis computation is extremely fast in `DualMode` so that it can be used as input. We tried three different setups (each with 30 parallel threads):

input	triangulation size	determinant sum	computation time
inequalities	347,225,775,338	4,111,428,313,448	112:43:17 h
inequalities, `-b`	288,509,390,884	1,509,605,641,358	84:26:19 h
Hilbert basis, `-b`	335,331,680,623	1,433,431,230,802	97:50:05 h

The results show that the determinant sum of the bottom decomposition based on the extreme rays (resulting from dualizing the input cone) comes very close to the absolute minimum that is reached by the Hilbert basis as input. The latter leads to a finer triangulation, which, despite of the smaller determinant sum, increases the computation time.

As long as the determinants are rather small (as in this computation), the two cases $\det = 1$ and $\det > 1$ make the difference in computation complexity. In the first case Normaliz must solve 1 or 2 linear systems for the simplicial cone, in the latter 2 or 3.

Note: though Normaliz can compute this example, it is really a case for NmzIntegrate; see [8]. It converts the computation time from days to hours.

7. Optional output files

When one of the options `Files,-f` or `AllFiles, -a` is activated, Normaliz writes additional output files whose names are of type `<project>.<type>`. (Note that the options `-T, Triangulation` and `-y, StanleyDec` as well as the options calling NmzIntegrate also write files in addition to `<project>.out`.) Moreover one can select the optional output files individually via command line options. Most of these files contain matrices in a simple format:

```
<m>
<n>
<x_1>
...
<x_m>
```

where each row has <n> entries. Exceptions are the files with suffixes cst, inv, esp.

7.1. The homogeneous case

The option -f makes Normaliz write the following files:

gen contains the Hilbert basis. If you want to use this file as an input file and reproduce the computation results, then you must make it a matrix of type cone_and_lattice (and add the dehomogenization in the inhomogeneous case).

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:*

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 $\mathbb{E}$. esp contains the grading and the dehomogenization in the coordinates of $\mathbb{E}$. Note that no equations for $\mathbb{C} \cap \mathbb{E}$ or congruences for $\mathbb{E}$ are necessary.

lat contains the basis of the lattice $\mathbb{E}$.

mod contains the module generators of the integral closure modulo the original monoid.

In order to select one or more of these files individually, add an option of type --<suffix> to the command line where <suffix> can take the values

```
gen, cst, inv, ext, ht1, egn, esp, lat, mod, typ
```

The type typ is not contained in Files or AllFiles since it can be extremely large. It is the matrix format described above. It is the product of the matrices corresponding to egn and the transpose of 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 [4] it contains the *standard embedding*.

Note: the explicit choice of an optional output file does *not* imply a computation goal. Output files that would contain unknown data are simply not written without a warning or error message.

7.2. Modifications in the inhomogeneous case

The optional output files are a subset of those 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 evaluating the dehomogenization on them (simply the last component with inhomogeneous input), 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 `egn` and `esp` are produced. These files contain `gen` and the support hyperplanes of the homogenized cone in the coordinates of $\mathbb{E}$, as well as the dehomogenization.

8. Performance

8.1. 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 5.3) 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=20
```

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 [8] 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.

8.2. 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 [8] 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 [8], 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` or the computation goal `TriangulationDetSum` if there is no grading.

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 `Normaliz3.0.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 `Normaliz3.0` (called Normaliz directory) and several subdirectories in `Normaliz3.0`. The names of the subdirectories created are self-explanatory. Nevertheless we give an overview:

- In the main directory `Normaliz3.0` you should find `jNormaliz.jar`, `Copying` and subdirectories.
- The subdirectory `source` contains the source files. 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.
- Automated tests which run Normaliz on different inputs and options are contained in the subdirectory `test`.
- 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 `Normaliz3.0<systemname>.zip` and unzip it. This process will store the executables of Normaliz and NmzIntegrate in the directory `Normaliz3.0`. 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.

10. Compilation

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

10.1. General

For compiling Normaliz the following libraries are needed:

- GMP including the C++ wrapper (libgmpxx and libgmp)
- boost (headers only)
- OpenMP 3.0 enabled compiler (to use parallelization, optional)

Furthermore we require some C++11 features (e.g. `std::exception_ptr`), supported by:

- GNU g++ 4.4,
- clang++ 2.9,
- Intel icpc 12.0

See https://github.com/Normaliz/Normaliz/issues/26 for a more detailed discussion.

The mentioned compilers are also able to handle OpenMP 3.0, with the exception of clang++, there the first OpenMP support was introduced in 3.7.

In the following we describe the configuration/build process using cmake. This has capabilities to find libraries and allows to change settings in a nice interface. It supports the compilation of the library as a static and a dynamic version, installation of the library. Furthermore it can also be used to compile Normaliz including SCIP or offloads to Intel Xeon Phi cards.

We will only handle the basic use of cmake for compilation, see the file `source/INSTALL` for additional information, especially on how to use customized paths.

10.2. Linux

On Ubuntu the following packages should be installed:

```
sudo apt-get install g++ libgmp-dev libboost-dev cmake cmake-curses-gui
```

We assume you start in the normaliz root dir (with subdirs source, example, ...).

1. Create a build directory where normaliz will be build and cd into it, e.g.

   ```
   mkdir BUILD; cd BUILD
   ```

2. Initial configuration, have a look at the next sections for further information about the configuration.

   ```
   cmake ../source
   ```

3. [Optional] Check configuration and maybe edit something (note the two `'c'`s at the beginning)

   ```
   ccmake ../source
   ```

   In the ccmake interface you can use this work flow:

   - `c` (configure)
   - change entries if you like (use cursor keys and enter), then press `c` again
   - `g` (generate and exit)

4. compile

   ```
   make
   ```

5. install it

   ```
   make install
   ```

   This will install the library headers, the produced libnormaliz and the normaliz executable. The installation path can be changed with ccmake (`CMAKE_INSTALL_PREFIX`).

10.3. MacOs X

Currently Apple does not supply a compiler which supports OpenMP. We recommend the use of a current gcc. Mac versions older than 4.5 have a bug that makes it impossible to use OpenMP.

You can then follow the instructions for Linux.

10.4. Windows

One can compile Windows executables with the Cygwin port of GCC. Unfortunately it is not compatible to OpenMP.

Using Visual Studio is a bit tricky. Microsoft's C++ compiler does not support OpenMP 3.0. Creating a Normaliz Visual Studio project via cmake is currently not fully supported. The supplied executables are compiled with icpc with a manually created project. Please contact us if you want to build Normaliz on Windows.

11. Copyright and how to cite

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

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

The corresponding `\bibitem`:

```
\bibitem W. Bruns, B. Ichim, R. Sieg, T. R\"omer and C. S\"oger:
Normaliz. Algorithms for rational cones and affine monoids.
Available at http://www.math.uos.de/normaliz.
```

A BibTeX entry:

```
@Misc{Normaliz,
author = {W. Bruns and B. Ichim and R. Sieg and T. R\"omer and C. S\"oger},
title = Normaliz. Algorithms for rational cones and affine monoids,
howpublished ={Available at \texttt{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 [8] 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 [4].

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.) For certain aspects of monoid theory it is very useful (or even necessary) to introduce coefficients from a field K (or a more general commutative ring) and consider the monoid algebra $K[M]$.

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 $C \cap L$.

In the situation of Theorem 7, if $\mathrm{gp}(N)$ has finite rank as a $\mathrm{gp}(M)$-module, $\widehat{M}_N$ is even a finitely generated module over M. I.e., there exist finitely many elements $y_1,\dots,y_m \in \widehat{M}_N$ such that $\widehat{M}_N = \bigcup_{i=1}^m M + y_i$. Normaliz computes a minimal system $y_1,\dots,y_m$ and lists the nonzero y_i as a system of module generators of $\widehat{M}_N$ modulo M. We must introduce coefficients to make this precise: Normaliz computes a minimal system of generators of the $K[M]$-module $K[\widehat{M}_N]/K[M]$.

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

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 $x \in \mathrm{gp}(M)$ 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 $\mathrm{cone}(M)$*. 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 [8, Section 4]. The examples 2.5 and 2.6.1, 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 $q^{(j)}$, $j = 0, \ldots, \pi - 1$, 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). The *multiplicity* of M, denoted by $e(M)$ is $(r-1)!q_{r-1} = \operatorname{vol}(Q)$.

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 $M = \operatorname{rec}_L(P)$. Then we must give up the condition that deg takes the value 1 on $\operatorname{gp}(M)$ (see Section 6.10 for an example). But 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 $k > r$ where r is the degree of the Hilbert series as a rational function.

Let m be the gcd of the numbers $\deg x$, $x \in M$. (For $M = \{0\}$ we set $m = 1$.) Then we must define $e(M) = e'(M)/m$ where $e'(M)$ is the multiplicity of M with respect to the normalized grading $\deg/m$. The multiplicity of N is given by

$$e(N) = \operatorname{mrank}(N) e(M).$$

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

A.8. The class group

A normal affine monoid M has a well-defined divisor class group. It is naturally isomorphic to the divisor class group of $K[M]$ where K is a field (or any unique factorization domain); see [4, 4.F], and especially [4, 4.56]. The class group classifies the divisorial ideals up to isomorphism. It can be computed from the standard embedding that sends an element x of $\mathrm{gp}(M)$ to the vector $\sigma(x)$ where σ is the collection of support forms $\sigma_1, \ldots, \sigma_s$ of M: $\mathrm{Cl}(M) = \mathbb{Z}^s/\sigma(\mathrm{gp}(M))$. Finding this quotient amounts to an application of the Smith normal form to the matrix of σ.

B. Annotated console output

B.1. Primal mode

With

```
./normaliz -c example/A443
```

we get the following terminal output.

```
                                                    \.....|
                    Normaliz 3.0                     \....|
                                                      \...|
    (C) The Normaliz Team, University of Osnabrueck    \..|
                  September 2015                        \.|
                                                         \|
************************************************************
Compute: DefaultMode
************************************************************
starting primal algorithm with full triangulation ...
Roughness 1
Generators sorted by degree and lexicographically
Generators per degree:
1: 48
```

Self explanatory so far (see Section 6.2 for the definition of roughness). Now the generators are inserted.

```
Start simplex 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 19 22 25 26 27 28 31 34
37 38 39 40 43 46
```

Normaliz starts by searching linearly independent generators with indices as small as possible. They span the start simplex in the triangulation. The remaining generators are inserted successively. (If a generator does not increase the cone spanned by the previous ones, it is not listed, but this does not happen for `A443`.)

```
gen=17, 39 hyp, 4 simpl
```

We have now reached a cone with 39 support hyperplanes and the triangulation has 3 simplices so far. We omit some generators until something interesting happens:

```
gen=35, 667 hyp, 85 pyr, 13977 simpl
```

In view of the number of simplices in the triangulation and the number of support hyperplanes, Normaliz has decided to build pyramids and to store them for later triangulation.

```
gen=36, 723 hyp, 234 pyr, 14025 simpl
...
gen=48, 4948 hyp, 3541 pyr, 14856 simpl
************************************************
level 0 pyramids remaining: 3541
************************************************
```

Now these pyramids must be triangulated. They may produce not only simplices, but also pyramids of higher level, and indeed they do so:

```
************************************************
all pyramids on level 0 done!
************************************************
level 1 pyramids remaining: 5935
************************************************
************************************************
all pyramids on level 1 done!
************************************************
level 2 pyramids remaining: 1567
************************************************
1180 pyramids remaining on level 2, evaluating 2503294 simplices
```

At this point the preset size of the evaluation buffer for simplices has been exceeded. Normaliz stops the processing of pyramids, and empties the buffer by evaluating the simplices.

```
||||||||||||||||||||||||||||||||||||||||||||||||
2503294 simplices, 0 HB candidates accumulated.
************************************************
all pyramids on level 2 done!
************************************************
level 3 pyramids remaining: 100
************************************************
************************************************
all pyramids on level 3 done!
```

This is a small computation, and the computation of pyramids goes level by level without the necessity to return to a lower level. But in larger examples the buffer for level $n+1$ may be filled before level n is finished. Then it becomes necessary to go back.

```
************************************************
Pointed since graded
```

Fortunately our cone is pointed. Some simplices remaining in the buffer are now evaluated:

```
evaluating 150978 simplices
||||||||||||||||||||||||||||||||||||||||||||||||||
2654272 simplices, 0 HB candidates accumulated.
Adding 1 denominator classes... done.
```

Since our generators form the Hilbert basis, we do not collect any further candidates. If all generators are in degree 1, we have only one denominator class in the Hilbert series, but otherwise there may be many. The collection of the Hilbert series in denominator classes reduces the computations of common denominators to a minimum.

```
Total number of pyramids = 14137, among them simplicial 2994
```

Some statistics of the pyramid decomposition.

```
Select extreme rays via comparison ... done.
```

Normaliz knows two methods for finding the extreme rays. Instead of "comparison" you may see "rank".

```
-------------------------------------------------------
transforming data... done.
```

A typical pair of lines that you will see for other examples is

```
auto-reduce 539511 candidates, degrees <= 1 3 7
reducing 30 candidates by 73521 reducers
```

It tells you that Normaliz has found a list of 539511 new candidates for the Hilbert basis, and this list is reduced against itself (auto-reduce). Then the 30 old candidates is reduced against the 73521 survivors of the auto-reduction.

B.2. Dual mode

Now we give an example of a computation in dual mode. It is started by the command

```
./normaliz -cd example/5x5
```

The console output:

```
                                                \.....|
                  Normaliz 3.0                   \....|
                                                  \...|
    (C) The Normaliz Team, University of Osnabrueck   \..|
                  September 2015                       \.|
```

```
                                                          \|

************************************************************
Compute: DualMode
No inequalities specified in constraint mode, using non-negative orthant.
```

Indeed, we have used equations as the input.

```
************************************************************
computing Hilbert basis ...
================================================
cut with halfspace 1 ...
Final sizes: Pos 1 Neg 1 Neutral 0
```

The cone is cut out from the space of solutions of the system of equations (in this case) by successive intersections with halfspaces defined by the inequalities. After such an intersection we have the positive half space, the "neutral" hyperplane and the negative half space. The final sizes given are the numbers of Hilbert basis elements strictly in the positive half space, strictly in the negative half space, and in the hyperplane. This pattern is repeated until all hyperplanes have been used.

```
================================================
cut with halfspace 2 ...
Final sizes: Pos 1 Neg 1 Neutral 1
```

We leave out some hyperplanes ...

```
================================================
cut with halfspace 20 ...
auto-reduce 1159 candidates, degrees <= 13 27
Final sizes: Pos 138 Neg 239 Neutral 1592
================================================
cut with halfspace 21 ...
Positive: 1027  Negative: 367
................................................
Final sizes: Pos 1094 Neg 369 Neutral 1019
```

Sometimes reduction takes some time, and then Normaliz may issue a message on "auto-reduction" organized by degree (chosen for the algorithm, not defined by the given grading). The line of dots is printed is the computation of new Hilbert basis candidates takes time, and Normaliz wants to show you that it is not sleeping. Normaliz shows you the number of positive and negative partners that must be pared produce offspring. TODO!!!

```
================================================
cut with halfspace 25 ...
Positive: 1856  Negative: 653
................................................
auto-reduce 1899 candidates, degrees <= 19 39
```

```
Final sizes: Pos 1976 Neg 688 Neutral 2852
```

All hyperplanes have been taken care of.

```
Find extreme rays
Find relevant support hyperplanes
```

Well, in connection with the equations, some hyperplanes become superfluous. In the output file Normaliz will list a minimal set of support hyperplanes that together with the equations define the cone.

```
Hilbert basis 4828
```

The number of Hilbert basis elements computed is the sum of the last positive and neutral numbers.

```
Find degree 1 elements
```

The input file contains a grading.

```
transforming data... done.
```

The computation of the new Hilbert basis after the intersection with the new hyperplane proceeds in rounds, and there be many rounds ...(not in the above example). then you can see terminal output like

```
Round 100
Round 200
Round 300
Round 400
Round 500
```

C. Normaliz 2 input syntax

A Normaliz 2 input file contains a sequence of matrices. Comments or options are not allowed in it. A matrix has the format

```
<m>
<n>
<x_1>
...
<x_m>
<type>
```

where <m> denotes the number of rows, <n> is the number of columns and <x_1>...<x_n> are the rows with <m> entries each. All matrix types of Normaliz 3 are allowed (with Normaliz 3), also grading and dehomogenization. These vectors must be encoded as matrices with 1 row.

The optional output files of with suffix `cst` are still in this format. Just create one and inspect it.

D. Changes relative to version 2.12

For the history of changes starting from 2.0 see the manuals of versions 2.7 and 2.12 (still accessible at the web site). Note that some changes have become obsolete later on.

Documentation:

1. Manual completely rewritten.

User control, input and output:

1. New syntax for input files and command line options (with backward compatibility).
2. Balance between generator input and constraints input.
3. Generator and constraints input can be mixed.
4. Output augmented and order of vectors in lists standardized.
5. Console output improved.

Computation goals:

1. Minimal module generators of integral closure can be computed.
2. Computation of class group added.
3. Implicit grading restricted to the case in which all extreme rays have degree 1 (instead of constant degree).

Algorithms and implementation:

1. Improved linear algebra, especially better coordinate transformations.
2. Local GMP transitions if intermediate results do not fit 64 bit integers.
3. Global GMP transition if final results do not fit 64 bit integers.
4. Local approximation of rational simplicial cones for degree 1 points.
5. Subdivision of large simplicial cones via SCIP and approximation.
6. Bottom decomposition introduced.
7. Recognition of polytopes in the computation of polyhedra.

References

[1] T. Achterberg. *SCIP: Solving constraint integer programs*. Mathematical Programming Computation 1 (2009), 1–41. Available from `http://mpc.zib.de/index.php/MPC/article/view/4`

[2] V. Almendra and B. Ichim. *jNormaliz 1.7*. Available from `http://www.mathematik.uni-osnabrueck.de/normaliz/Normaliz3.0/jNormaliz1.7Documentation.pdf`

[3] V. Baldoni, N. Berline, J.A. De Loera, B. Dutra, M. Köppe, S. Moreinis, G. Pinto, M. Vergne, J. Wu, *A User's Guide for LattE integrale v1.7.2, 2013.* Software package LattE is available at `http://www.math.ucdavis.edu/~latte/`

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

[5] 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.

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

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

[8] W. Bruns, B. Ichim and C. Söger. *The power of pyramid decompositions in Normaliz*. J. Sym. Comp., to appear. Preprint arXiv:1206.1916.

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

[10] W. Bruns and C. Söger. *The computation of generalized Ehrhart series in Normaliz.* J. Symb. Comp. 68 (2015), 75–86.

[11] W. Bruns and C. Söger. *NmzIntegrate 1.3.* Available from `http://www.mathematik.uni-osnabrueck.de/normaliz/Normaliz3.0/NmzIntegrate.pdf`

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

[13] 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).

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