Normaliz Short Reference

for Normaliz version 3.11.1

Useful links:

Manual `https://github.com/Normaliz/Normaliz/blob/master/doc/Normaliz.pdf`

Git Hub repository `https://github.com/Normaliz/Normaliz`

Home page `https://www.normaliz.uni-osnabrueck.de/`

Docker repository `https://hub.docker.com/r/normaliz/normaliz/`

Online exploration `https://mybinder.org/v2/gh/Normaliz/NormalizJupyter/master`

Support `mailto:normaliz@uos.de`

Mailing list `normaliz-subscribe@list.serv.uos.de`

For some entries we mention example input files: `example/small.in` refers to the file `small.in` in the subdirectory `example` of the Normaliz directory.

1 Introduction

Normaliz is a tool for discrete convex geometry. It computes several data of polyhedra and lattice points in them. The names of the Normaliz input types and computation goals are descriptive and self explaining. We recommend the user to experiment with the examples in the directory `example`. A large part of the manual is in tutorial style.

Ways to run Normaliz:

1. in a terminal of Linux, MacOS or MS Windows,
2. in a Docker container (effectively a Linux terminal),
3. in the GUI interface jNormaliz,
4. interactively via the Python interface PyNormaliz (see Appendix E of the Normaliz manual).

Moreover, Normaliz is used by several other packages, explicitly or implicitly. In this reference we assume that Normaliz is run in a terminal.

For a deeper understanding one must note the following. The output depends to some extent on the types of input and of computation that can be *homogeneous* or *inhomogeneous:*

1. Homogeneous input types define cones and lattices.
2. Inhomogeneous input types define polyhedra and an affine lattices.

For the computation, inhomogeneous input is homogenized, and the polyhedron and the affine lattice are selected by the *dehomogenization.* The computation is inhomogeneous if a dehomogenization is defined. Polytopes, i.e., bounded polyhedra, can be defined by homogeneous input if one adds a grading, or, as expected, by inhomogeneous input. These two approaches are almost equivalent.

Topics *not covered* by his short reference:

1. The use of high performance clusters for the computation of volumes by signed decomposition or the patching variant of project-and lift for lattice points. See Appendix G of the Normaliz manual.
2. the input types and computation goals for fusion rings. See Appendix H of the Normaliz manual.

Top Index

2 Command line

Normaliz is run in a terminal. The format of the command line is

`normaliz [options]` <project>

`project` defines the input file `<project.in`. The output files are `<project>.<suffix>`. The main out put file is `<project>.out`. Depending on your path settings and the place where `normaliz` is installed you may have to prefix `normaliz` by a path to it.

Options can be long options starting with `--` or short options, a single letter or number prefixed by `-`. Short options can be bundled into a string. A special case is `-x`, setting the parallelization. The order of options and `<project>` is irrelevant.

Examples, assuming that the Normaliz directory is the working directory and `normaliz` (or `normaliz.exe`) resides in it, MacOs or Linux:

```
./normaliz -c example/small

./normaliz -c -x=16 example/A553
```

For MS Windows the equivalent command lines are

```
normaliz.exe -c example\small

normaliz.exe -c -x=16 example\A553
```

Top Index

3 Input

The input file

<project>.in

contains: (i) the definition of the ambient space, (ii) the algebraic number field in the case of algebraic polyhedra, (iii) the definition of cones, polyhedra and lattices by generators or constraints, (iv) options for computation goals and algorithmic variants, (v) numerical parameters for certain computations, (vi) a polynomial for certain computations. One can insert C style comments *...*/ between input items.

There are 3 categories of input:

1. For discrete convex geometry Normaliz forms the cone defined by the generators and intersects it with the cone defined by the constraints. The same applies to lattices. See the manual for a more precise description.
2. Affine monoids defined by generators.
3. Binomial ideals saerving as lattice and toric iodeals.
4. Fusion rings. See Appendix H of the Normaliz manual.

Top Index

3.1 Ambient space

The first line of the input file sets the dimension of the ambient space:

`amb_space <d>`

or

`amb_space auto`

`auto` is only allowed if the first item, for which the dimension must be known, is a formatted vector or matrix.

3.2 Rational numbers

All standard formats can be used: integers, fractions, decimal fractions, standard floating point notation. Some input types accept only integers:

```
lattice     cone_and_lattice    offset         open_facets
congruences inhom_congruences   rees_algebra   lattice_ideal
grading     dehomogenization    signs          strict_signs
```

3.3 Algebraic numbers

For algebraic polyhedra the definition of the number field must follow the definition of the ambient space:

```
number_field min_poly (<poly>) embedding [<center> +/- <radius>]
```

`<poly>` is a polynomial with rational coefficients (integers or fractions) in one indeterminate, the field generator. The latter is named by a single letter different from `e` and `x`. The zero of the minimal polynomial is located in the interval `<center>` $\pm$ `<radius>`.

Example:

```
number_field min_poly (a^2 - 5) embedding [ 2 +/- 1]
```

An algebraic number is a sum of terms, and each term is of type `<coeff>[*]<gen>^<exp>` with the usual simplifications for the exponents 0 and 1. Examples:

```
1/2*a^2+a-1   5a-6
```

Example: `example/icosahedron-v.in`

The following input types are NOT allowed for algebraic polytopes:

```
lattice            strict_inequalities     strict_signs    open_facets
cone_and_lattice   inhom_congruences       lattice_ideal   offset
congruences        hilbert_basis_rec_cone  rees_algebra    rational_lattice
rational_offset
```

Input Top Index

3.4 Vectors

A plain vector is given by

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

<T> denotes the type and <x> is the vector itself. The entries are separated by spaces. The number of components is determined by the type of the vector and the dimension of the ambient space. It can be sparse, given by

```
sparse <entries>;
```

where <entries> is a sequence of pairs <c>:<v>. In each pair <c> is the index of a coordinate and <v> is its value. <c> can also stand for a range of coordinates in the form <first>..<last>. The concluding semicolon is mandatory.

A formatted vector is given by

```
<T>
[<x>]
```

where <x> is a sequence of entries separated by spaces, commas or semicolons.

A special vector is

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

Examples:

```
amb_space 3     amb_space 3        amb_space 3        amb_space auto
grading         grading sparse     grading            grading
0 0 1           3:1;               unit_vector 3      [0, 0,1]
```

Input Top Index

3.5 Matrices

A plain matrix is built as follows:

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

<T> is the type and <m> the number of rows, the latter given by <x_1>...<x_m>. The number of columns is defined by the value of amb_space and the type.

The matrix can be transposed:

```
<T> transpose <c>
<x_1>
...
<x_m>
```

with <c> as the number of entries of each row of the input and <x_1>...<<x_m> are the columns of the resulting matrix. The number of rows of the input is calculated from the value of amb_space and the type.

Both matrices and transposed matrices can be sparse. The keyword sparse> follows <m> or <c>.

A formatted matrix is built as follows:

```
<T>
[ [<x_1>]
...
[<x_m>] ]
```

It can also be transposed.

The unit matrix can be given to every input type that expects a matrix:

unit_matrix

The number of rows is defined by amb_space and the type of the matrix, as usual.

Examples (with amb_space 3 or amb_space auto in the formatted case):

```
inequalities 4       inequalities        inequalities transpose 4
-1  0 2              [ [-1  0 2],        -1 1  2 -2
 1  1 1                [ 1  1 1],         0 1 -3 -1
 2 -3 4                [ 2 -3 4],         2 1  4  6
-2 -1 6                [-2 -1 6] ]
```

3.6 Generators

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

subspace is a matrix with d columns. The linear subspace generated by the rows is added to the cone. example/normface.in

polytope is a matrix with $d-1$ columns. It is internally converted to cone extending each row by an entry 1.This input type automatically sets NoGradingDenom and defines the grading $(0,\dots,0,1)$. Not allowed in combination with inhomogeneous types. example/polytope.in

cone_and_lattice The vectors of the matrix with d columns define both a cone and a lattice. If subspace is used in combination with cone_and_lattice, then the sublattice generated by its rows is added to the lattice generated by cone_and_lattice. example/A443.in

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

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

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

Note: vertices cone and subspace together define a polyhedron. If vertices is present in the input, then the default choices for cone and subspace are the empty matrices. example/icosahedron-v.in

monoid is a matrix with d columns. Its rows generate a positive affine monnoid.

Input Top Index

3.7 Constraints

Homogeneous constraints:

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

$$\xi_1 x_1 + \cdots + \xi_d x_d \ge 0$$

for the vectors $(x_1,\dots,x_d) \in \mathbb{R}^d$. example/Condorcet.in

nonnegative represents a system of inequalities cutting out the positive orthant. example/Condorcet.in

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

$$\xi_1 x_1 + \cdots + \xi_d x_d = 0$$

for the vectors $(x_1,\dots,x_d) \in \mathbb{R}^d$. example/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 $(z_1,\dots,z_d) \in \mathbb{Z}^d$. example/3x3magiceven.in

Inhomogeneous constraints:

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

$$\xi_1 x_1 + \cdots + \xi_d x_d \ge \eta, \qquad \text{equivalently}, \qquad \xi_1 x_1 + \cdots + \xi_d x_d + (-\eta) \ge 0,$$

represented by the input vectors $(\xi_1,\dots,\xi_d,-\eta)$. example/icosahedron-h.in

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

$$\xi_1 x_1 + \cdots + \xi_d x_d = \eta, \qquad \text{equivalently}, \qquad \xi_1 x_1 + \cdots + \xi_d x_d + (-\eta) = 0,$$

represented by the input vectors $(\xi_1,\dots,\xi_d,-\eta)$. example/truncated_dodecahedron_dual.in

inhom_congruences We consider a matrix with $d+2$ columns. Each 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 $(z_1,\dots,z_d) \in \mathbb{Z}^d$. example/ChineseRemainder.in

convert_equations converts equations in the input file into inequalities. Can be useful to avoid superfluous coorddinate transformations. Requires the addiion of `nonnegative` if no other inequalities are present. example/pet.in

If there are no cone generators or inequalities, Normaliz automatically assumes the sign inequalities defining the positive orthant positive orthant. This behavior can be broken by an empty matrix `inequalities 0` or by the directive

no_pos_orth_def

3.8 Tabular and symbolic constraints

constraints <n> allows the input of <n> equations, inequalities and congruences in a format that is close to standard notation. As for matrix types the keyword `constraints` is followed by the number of constraints. If $(\xi_1,\dots,\xi_d)$ is the vector on the left hand side and η the number on the right hand side, then the constraint defines the set of vectors $(x_1,\dots,x_d)$ such that the relation

$$\xi_1 x_1 + \dots + \xi_d x_d \ \texttt{rel} \ \eta$$

is satisfied, where `rel` can take the values $=,\le,\ge,<,>$ with the represented by input strings `=,<=,>=,<,>`, respectively.

A further choice for `rel` is `~`. It represents a congruence $\equiv$ and requires the additional input of a modulus: the right hand side becomes $\eta(c)$. It represents the congruence

$$\xi_1 x_1 + \dots \xi_d x_d \equiv \eta \pmod{c}.$$

A right hand side $\neq 0$ makes the input inhomogeneous, as well as the relations $<$ and $>$. Strict inequalities (not allowed for algebraic polyhedra) are always understood as conditions for integers. So

$$\xi_1 x_1 + \dots + \xi_d x_d < \eta$$

is interpreted as

$$\xi_1 x_1 + \dots \xi_d x_d \le \eta - 1.$$

Examples:

```
1 0 2 = 5          1 0 -2 >= 6          1 2 3 ~ 5 (12)
```

example/ChF_8_1024.in, example/CondorcetRange.in

constraints <n> symbolic where <n> is the number of constraints in symbolic form that follow.

Symbolic constraints are given in traditional mathematical form. Note that every symbolic constraint (including the last) must be terminated by a semicolon. The left and right hand side can be affine-linear expressions in the coordinates given by `x<[i>]` where <i> varies between 1 and d.

The interpretation of homogeneity follows the same rules as for tabular constraints.

Examples

```
x[1] + 2x[3] = 5;     x[1] >= x[2] + 6;     x[1] + 2x[2] + 3x[]3] ~ 5 (12);
```

example/cube_3.in, example/NumSemi.in

3.9 Grading and dehomogenization

grading is a vector of length d representing the linear form that gives the grading. (Special rules for lattice ideal input.) example/3x3magiceven.in

total_degree is the grading with all coordinates equal to 1. example/Condorcet.in

dehomogenization is a vector of length d representing the linear form that gives the dehomogenization. example/600cell-dual.in

3.10 Polynomials

3.10.1 Polynomial weights

For the computation of weighted Ehrhart series and integrals Normaliz needs the input of a polynomial with rational coefficients:

polynomial <poly expression>

The polynomial is first read as a string. For the computation the string is converted by the input function of CoCoALib. Therefore any string representing a valid CoCoA expression is allowed. However the names of the indeterminates are fixed: `x[1]`,...,`x[<d>` where `dN>]` is the value of `amb_space`. The polynomial must be concluded by a semicolon.

Example:

```
1/2*((x[1] + 2*x[2])^2 - x[3]);
```

example/j462.in

3.10.2 Polynomial constraints

Polynomial equations of type $f(x) = 0$ and uinequalities of type $f(x) \geq 0$ can be set by

polynomial_equations <n>

polynomial_inequalities <n>

where <n> is the number of polynomials that follow. Everey polynomial must be concuded by a semicolon. example/pet.in

Input Top Index

3.11 Binomial ideals

lattice_ideal is a matrix with d columns containing the generators of a lattice ideal L in the Laurent polynomial ring. It defines the intersection of L with the polynomial ring. example/non_toric.in

toric_ideal is a matrix with d columns containing the generators of a lattice ideal in the Laurent polynomial ring. It defines the smallest toric ideal T in the polynomial ring containing the generators and the affine monoid underlying the quotient of the polynomial ring modulo T. example/toric_ideal.in

normal_toric_ideal is a matrix with d columns containing the generators of a lattice ideal in the Laurent polynomial ring. It It goes one step further than `toric_ideal` by passing further to the normalizaion of the affime monoid. example/normal_toric_ideal.in

3.12 Numerical parameters

expansion_degree <n> where <n> is the number of coefficients in a series expansion to be computed and printed.

nr_coeff_quasipol <n> where <n> is the number of highest coefficients in a quasipolynomial to be printed.

face_codim_bound <n> where <n> is the bound for the codimension of the faces to be computed.

decimal_digits <n> where <n> sets the precision to 10^{-n} (for computation with signed decomposition).

block_size_hollow_tri <n> sets the block size for distributed computation to <n>.

gb_degree_bound <n> sets the upper bound for elements computed in a Gröbner or Markov basis to <n>.

gb_min_degree <n> sets the lower bound for elements in a Gröbner or Markov basis to <n>.

3.13 Types for precomputed data

Precomputed types are used for the recycling of data from previous computations, namely extreme rays and support hyperplanes; furthermore, the coordinate transformations represented by the generated sublattice (or subspace) and the maximal subspace contained in the cone. The latter are only required if they are different from the default values $\mathbb{Z}^d$ (or $\mathbb{R}^d$) and $\{0\}$, respectively.

extreme_rays is a matrix with d columns.

maximal_subspace is a matrix with d columns.

generated_lattice is a matrix with d columns.

support_hyperplanes is a matrix with d columns.

Further admitted types for precomputed data: `grading`, `dehomogenization`.

example/InhomIneq_prec.in

3.14 Additional input types

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. Not allowed in combination with inhomogeneous types.

rational_lattice is a matrix with d columns. Its entries can be fractions. Every row represents a vector, and they define the sublattice of $\mathbb{Q}^d$ generated by them.

saturation is a matrix with d columns. Every row represents a vector, and they define the lattice $U \cap \mathbb{Z}^d$ where U is the subspace generated by them. (If the vectors are integral, then $U \cap \mathbb{Z}^d$ is the saturation of the lattice generated by them.)

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.

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

$$\xi_1 x_1 + \cdots + \xi_d x_d > 0$$

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` behave like `inequalities` in almost every other respect.

offset is a vector with d integer entries. It defines the origin of the affine lattice.

rational_offset is a vector with d rational entries. It defines the origin of the rational affine lattice.

strict_inequalities is a matrix with d columns. We consider inequalities

$$\xi_1 x_1 + \cdots + \xi_d x_d \geq 1,$$

represented by the input vectors $(\xi_1,\dots,\xi_d)$.

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.

inhom_excluded_faces is a matrix with $d+1$ columns. Every row $(\xi_1,\dots,\xi_d,-\eta)$ represents an inequality

$$\xi_1 x_1 + \cdots + \xi_d x_d > \eta$$

for the vectors $(x_1,\dots,x_d) \in \mathbb{R}^d$. The faces of the polyhedron excluded by the inequalities are excluded from the Hilbert and Ehrhart series series computation, but `inhom_excluded_faces` behave like `inhom_inequalities` in almost every other respect.

hom_constraints for the input of equations, non-strict inequalities and congruences in the same format as `constraints`, except that these constraints are meant to be for a homogeneous computation. It is clear that the left hand side has only $d-1$ entries now. Also allowed for symbolic constraints.

projection_coordinates It is a 0-1 vector of length d.

The entries 1 mark the coordinates of the image of the projection. The other coordinates give the kernel of the projection.

open_facets is a vector of length d with entries $\in \{0,1\}$. (See manual)

hilbert_basis_rec_cone is a matrix with d columns. It contains the precomputed Hilbert basis of the recession cone.

gb_weight is a weight vector for the monomial order of Gröbner bases. Its length depends on other input data: it is `amb_space` for `lattice ideal` and `toric_ideal`. For `monoid` it is the number of generators of the monoid.

Input Top Index

4 Computation

4.1 Integer type

Normaliz chooses the integer type for computations automatically. But there can be reasons for the user to fix it:

`BigInt, -B` forces Normaliz to do all computations in indefinite precision.
`LongLong` forces 64 bit integers in all computations.

The probability that Normaliz does not notice an overflow in 64 bit computations is extremely small, but in critical cases it may be wise to ask for `Bigint`.

Computation Top Index

4.2 Computation goals

Depending on the input Normaliz chooses default computation goals and algorithmic variants. Most computation goals include `Sublattice` and `SupportHyperplanes`. If you are in doubt whether your desired data will be computed, add an explicit computation goal.

Computation gaols by themes:

4.2.1 Support hyperplanes and extreme rays

`SupportHyperplanes, -s` triggers the computation of support hyperplanes and extreme rays.
`example/cyclicpolytope30-15.in`

`IntegerHull, -H` computes the integer hull of a polyhedron. Implies the computation of the lattice points in it. More precisely: in homogeneous computations it implies `Deg1Elements`, in inhomogeneous computations it implies `HilbertBasis`. `example/InhomIneqIH.in`

`ProjectCone` Normaliz projects the cone defined by the input data onto a subspace generated by selected coordinate vectors and computes the image with the goal `SupportHyperplanes`.
`example/small_proj.in`

For the following we only need the support hyperplanes and the lattice:

`IsGorenstein, -G` : is the monoid of lattice points Gorenstein? In addition to answering this question, Normaliz also computes the generator of the interior of the monoid (the canonical module) if the monoid is Gorenstein. (Only in homogeneous computations.)
`example/5x5Gorenstein.in`

4.2.2 Hilbert basis and lattice points

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. `example/5x5.in`, `example/A443.in`

IsIntegrallyClosed : is the original monoid integrally closed? Normaliz stops the Hilbert basis computation as soon as it can decide whether the original monoid contains the Hilbert basis. Normaliz tries to find the answer as quickly as possible. This may include the computation of a witness, but not necessarily. If you need a witness, use `WitnessNotIntegrallyClosed, -w`. `example/A643.in`

Deg1Elements, -1 restricts the computation to the degree 1 elements of the Hilbert basis in homogeneous computations (where it requires the presence of a grading). `example/max_polytope_cand.in`

LatticePoints is identical to `Deg1Elements` in the homogeneous case, but implies `NoGradingDenom`. In inhomogeneous computations it is a synonym for `HilbertBasis`. `example/ChF_8_1024.in`

ModuleGeneratorsOverOriginalMonoid, -M computes a minimal system of generators of the integral closure over the original monoid. Requires the existence of original monoid generators.

Computation Top Index

4.2.3 Enumerative data, volumes and integrals

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

`HilbertSeries,-q` triggers the computation of the Hilbert series. `example/CondorcetSemi.in`

`EhrhartSeries` computes the Ehrhart series of a polytope, regardless of whether it is defined by homogeneous or inhomogeneous input. In the homogeneous case it is equivalent to `HilbertSeries` + `NoGradingDenom`, but not in the inhomogeneous case. `example/rational_inhom.in`

`Multiplicity, -v` restricts the computation to the multiplicity. `example/CondEffPlur.in`

`Volume, -V` computes the lattice normalized and the Euclidean volume of a polytope given by homogeneous or inhomogeneous input (implies `Multiplicity` in the homogeneous case, but also sets `NoGradingDenom`). `example/dodecahedron-v.in`

`NumberLatticePoints` finds the number of lattice points in a polytope. They are not stored. `example/CondorcetRange.in`

The following computation goals need the input of a polynomial:

`WeightedEhrhartSeries, -E` makes Normaliz compute a generalized Ehrhart series.

`VirtualMultiplicity, -L` makes Normaliz compute the virtual multiplicity of a weighted Ehrhart series.

`Integral, -I` makes Normaliz compute an integral over a polytope. Implies `NoGradingDenom`. `example/j462.in`

Computation goals Computation Top Index

4.2.4 Triangulation

`Triangulation, -T` makes Normaliz compute, store and export the full triangulation. `example/cross2.in`

`AllGeneratorsTriangulation` makes Normaliz compute and store a triangulation that uses all generators.

`LatticePointTriangulation` makes Normaliz compute and store a triangulation that uses all lattice points in a polytope.

`UnimodularTriangulation` makes Normaliz compute and store a unimodular triangulation.

4.2.5 Face structure

`FVector` computes the f-vector of a polyhedron. `example/icosahedron_prec.in`

`FaceLattice` computes the set of faces.

The range can be restricted by the numerical parameter `face_codim_bound`. There are dual cone we have:

`DualFVector`

`DualFaceLattice` `example/cube_3_dual_fac.in`

4.2.6 Automorphism groups

`Automorphisms` computes the integral automorphisms of monoids and rational polyhedra and the algebraic automorphisms of algebraic polytopes. `example/pythagoras_int.in`

`RationalAutomorphisms` computes the rational automorphisms of rational polytopes. `example/pythagoras_rat.in`

`EuclideanAutomorphisms` computes the euclidean automorphisms of rational and algebraic polytopes. `example/icosahedron-v.in`

`CombinatorialAutomorphisms` computes ate combinatorial automorphisms of polyhedra. `example/pentagon.in`

`AmbientAutomorphisms` computes automorphisms induce by permutations of coordinates of the ambient space. `example/A443.in`

`InputAutomorphisms` computes rational (or algebraic) automorphisms based solely on the input and initial coordinate transformations. `example/halfspace3inhom-input.in`

4.2.7 M;arkov and Gröbner bases

`MarkovBasis` computes a dsystem of generators of a givemn lattice or toric ideal defining an affine monoid. `example/monoid.in`

`GroebnerBasis` computes a Gröbner basis of a lttice or toric ideal. `example/toric_ideal_grb.in`

4.2.8 Additional computation goals

Sublattice, -S (upper case S) asks Normaliz to compute the coordinate transformation to and from the efficient sublattice.

VerticesFloat converts the format of the vertices to floating point. It implies `SupportHyperplanes`.
SuppHypsFloat converts the format of the support hyperplanes to floating point.
ExtremeRaysFloat does the same for the extreme rays.

WitnessNotIntegrallyClosed, -w Normaliz stops the Hilbert basis computation as soon it has found a witness confirming that the original monoid is not integrally closed.

ClassGroup, -C is self explanatory, includes `SupportHyperplanes`. .

ConeDecomposition, -D Normaliz computes a disjoint decomposition of the cone into semi-open simplicial cones. Implies `Triangulation`.
TriangulationSize, -t makes Normaliz count the simplicial cones in the full triangulation.
TriangulationDetSum Normaliz additionally sums the absolute values of their determinants.
StanleyDec, -y makes Normaliz compute, store and export the Stanley decomposition.
PlacingTriangulation combinatorially defined triangulation, see manual.
PullingTriangulation ditto.

Incidence computes the incidence of extreme rays and facets.
DualIncidence the transpose of `Incidence`.

IsEmptySemiopen checks whether a semiopen polyhedron is empty.

IsPointed : is the efficient cone C pointed? This computation goal is sometimes useful to give Normaliz a hint that a nonpointed cone is to be expected.
IsDeg1ExtremeRays : do the extreme rays have degree 1?
IsDeg1HilbertBasis : do the Hilbert basis elements have degree 1?
IsReesPrimary : for the input type `rees_algebra`, is the monomial ideal primary to the irrelevant maximal ideal?

Representations asks for the representations of the reducible generators of a monoid in terms of the irreducible ones.

SingularLocus computes the singular locus of a monoid algebra.
CodimSingularLocus just the codimension.
IsSerreR1 checks whether a mo noid alebra satisfies the Serre condition (R_1).
IsLatticeIdealToric asks this question.

WritePreComp Computes and writes file with suffix `precomp.in` that can be used for the input of precomputed data.

There are several more computation goals that are used internally by Normaliz and the communication with external packages. See manual.

4.3 Major algorithmic variants

For several computation goals there exist more than a single algorithm in Normaliz. The program tries to choose the best variant, but it sometimes needs help by the user.

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

`PrimalMode, -P` blocks the use of the dual algorithm.

`Projection, -j` chooses project-and-lift for lattice points in polytopes.

`NoProjection` blocks it.

`Descent, -F` chooses descent in the face lattice for volume computations.

`NoDescent` blocks it.

`SignedDec` chooses signed decomposition for volume computations.

`NoSignedDec` blocks it.

`Symmetrize, -Y` lets Normaliz compute the multiplicity and/or the Hilbert series via symmetrization (or just compute the symmetrized cone).

`NoSymmetrization` blocks symmetrization.

`KeepOrder, -k` forces Normaliz to insert the generators (for generator input) or the inequalities (for constraint input) in the input order.

`Lex` ask for the lexicographic monomial order vfor the Gröbner basis.

`Lex` ask for the degree lexicographic monomial order vfor the Gröbner basis.

`RevLex` ask for the degree reverse lexicographic monomial order vfor the Gröbner basis. (Default choice)

Computation Top Index

4.4 Minor algorithmic variants

ProjectionFloat, -J , project-and-lift with floating point arithmetic.

NoLLL blocks the use of LLL.

NoRelax blocks relaxation in project-and-lift.

NoCoarseProjection blocks coarse projection.

NoPatching blocks the patching variant of project-and-lift.

Patching forces the patching variant of project-and-lift whenever possible.

Approximate, -r , approximation of rational polytopes for lattice point computation.

UseWeightsPatching "Weights of coordinates" are used in scheduling the patching.

LinearOrderPatches Polynomial equations are disregarded in scheduling the patching.

CongOrderPatches makes Normaliz use the congruences in scheduling the patching.

MinimizePolyEquations aks for the minimization of the polynomial equations (may take very long).

BottomDecomposition, -b forces the bottom decomposition in the primal algorithm.

NoBottomDec, -o forbids Normaliz to use it.

ExploitAutomsVectors exploits the automorphism group for computing Hilbert bases or degree 1 points by the primal algorithm.

Descent ExploitIsosMult chooses exploitation of isomorphism types in the descent algorithm for volumes.

StrictTypeChecking forbids Normaliz to use SHA256 hash values for the identification of isomorphism types.

DistributedComp asks for distributed computation of volumes by signed decomposition.

FixedPrecision sets fixed precision for volume computation by signed decomposition.

HSOP lets Normaliz compute the degrees in a homogeneous system of parameters and the induced representation of the Hilbert or Ehrhart series series.

NoPeriodBound This option removes the period bound that Normaliz sets for the computation of the Hilbert quasipolynomial (presently 10^6).

NoQuasiPolynomial suppresses the computation of the Hilbert or Ehrhart quasipolynimial.

OnlyCyclotomicHilbSer restricts the output of the Hilbert or Ehrhart series to the representation with the cyclotomic denominator.

NoGradingDenom forces Normaliz to keep the original grading if it would otherwise divide it by the grading denominator. It is implied by several computation goals for polytopes.

GradingIsPositive tells Normaliz that there is no need to check the grading for positivity. Useful in connection with `SignedDec`.

Computation Top Index

5 Execution

Normaliz is run in a terminal. The format of the command line is

`normaliz [options]` <project>

`project` defines the input file `<project.in`. The output files are `<project>.<suffix>`. The main out put file is `<project>.out`. Depending on your path settings and the place where `normaliz` is installed you may have to prefix `normaliz` by a path to it.

Options can be long options starting with `--` or short options, a single letter or number prefixed by `-`. Short options can be bundled into a string. A special case is `-x`, setting the parallelization. The order of options and `<project>` is irrelevant.

```
./normaliz -c example/small
./normaliz -c -x=16 example/A553
./normaliz -c example/Condorcet --HilbertSeries --NoMatricesOutput
```

Note that on MS Windows the slash `/` must be replaced by a backslash `\`.

All computation goals and algorithmic variants can be given as long options on the command line. There are other options for execution and output.

`--help, -?` displays a help screen listing the Normaliz options.

`--version` displays information about the Normaliz executable.

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

`--talk` gives more terminal output. At present only implemented for the patching variant of project-and-lift for lattice points.

`-list_polynomials` lists the input polynomials on the terminal for easier debugging. Can be part of the input file without `--`.

`-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 8.
-x=1] forces serial execution. `-x=0` switches off the limit set by Normaliz.

`parallel_threads <n>` sets the thread limit in the input file.

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

Top Index

6 Output

The main output file is `<project>.out`. However, some data are written to extra files, either because they can be very large, or contain the data of "derived" cones. They have suffixes:

- **tri** contains the triangulation.
- **tgn** reference generators for the triangulation.
- **aut** contains the automorphism group.
- **dec** contains the Stanley decomposition.
- **fac** contains the face lattice.
- **inc** contains the (dual) incidence matrix.
- **mrk** contains the Markov basis.
- **grb** contains the Gröbner basis.
- **rep** contains the representations of the reducible generators of a monoid by the irreducible ones.
- **ogn** contains the refernce generators for Markov bases, Gröbner bases and represntations.
- **sng** contains the singular locus.
- **proj.out** contains the projected cone.
- **inthull.out** contains the integer hull.
- **symm.out** contains the symmetrized cone.

Moreover, there are truly optional output files that serve as a file interface. See manual.

The content and the writing of output files can be controlled by further options:

- **--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 face lattice, the triangulation or the Stanley decomposition, unless these have been requested).
- **--OutputDir=<outdir>** The path `<outdir>` is an absolute path or a path relative to the current directory (which is not necessarily the directory of `<project>.in`.)
- **NoExtRaysOutput** suppresses the output of extreme rays in the `out` file.
- **NoHilbertBasisOutput** does the same for Hilbert bases and lattice points.
- **NoSuppHypsOutput** suppresses the output of support hyperplanes in the `out` file.
- **NoMatricesOutput** restricts the `out` file to the "preamble".
- **NoOutputOnInterrupt** prevents the output of data after Ctrl-C.

Top Index