NORMALIZ
VERSION 2.2

WINFRIED BRUNS AND BOGDAN ICHIM
WITH CONTRIBUTIONS BY CHRISTOF SÖGER

ABSTRACT. This manual describes the version 2.2 of NORMALIZ, a program for solving linear systems of inequalities. It is an upgrade of the program Normaliz, developed originally by Winfried Bruns and Robert Koch.

CONTENTS

1. INTRODUCTION

The program NORMALIZ, version 2.2, is mainly a tool for solving linear systems of inequalities or, in other terms, for computing the Hilbert basis of a rational cone. Several related computation tools are available. If you have previously worked with NORMALIZ before version 2.0 ("old version" in the following), you can use the same input files for NORMALIZ, version 2.2. Also the output files are similar since NORMALIZ maintains the same simple interface as its predecessor. However, the internal structure is different. Since version 2.0 NORMALIZ has been written in C++, and we have tried to maintain a clear design. We partly use new algorithms, see [3], obtaining a general increase in computation speed.

Using NORMALIZ, version 2.2, (in the following simply called NORMALIZ) one may compute the following:

(1) the Hilbert basis and the support hyperplanes of a rational cone. The cone may be given by:
   (i) a system of generators;
   (ii) a linear system of inequations (already in version 2.0);
   (iii) a linear system of equations (already in version 2.0);
(4) the lattice points and the support hyperplanes of an integral polytope;
(5) the generators of the integral closure of the Rees algebra of a monomial ideal $I \subseteq K[X_1,\dots,X_n]$ and the generators of the integral closure of I.

If the associated semigroup is homogeneous in a certain sense (see Section 8), then one may also compute the h-vector and Hilbert polynomial of the semigroup.

For the theory of affine semigroups and the notions of commutative algebra used in the following we refer the reader to [1] and [2]. For algorithms see [3] and [5].

We also provide a SINGULAR library `normaliz.lib` and the package `Normaliz.m2` for MACAULAY2 that make NORMALIZ accessible from SINGULAR and MACAULAY2 resp. Thus SINGULAR or MACAULAY2 can be used as a comfortable environment for the work with NORMALIZ, and, moreover, NORMALIZ can be applied directly to objects belonging to the classes of toric rings and monomial ideals.

NORMALIZ has been made accessible from POLYMAKE (thanks to Andreas Paffenholz).

2. CHANGES RELATIVE TO VERSION 2.0

Changes in version 2.1:

User control, input and output:

(1) The command line option `-i` forces NORMALIZ to ignore a potentially existing setup file. This is useful if an external program wants to keep complete control (see Section 7). In case the setup file does not exist, `-i` keeps NORMALIZ from issuing a warning message.

(2) In addition to the choice of the mode via a single digit in the last line of the input file, the mode can now be specified by a keyword (see Section 6).
(3) In the homogeneous case NORMALIZ also lists the "height 1" elements in the Hilbert basis (and writes them to a file with suffix `ht1` if requested); see Sections 8 and 10.
(4) The structure of the file with suffix `inv` (used for the communication with computer algebra systems) has been changed from a SINGULAR command to a neutral format.

Algorithms

(1) In modes 4 and 5 in which the input is given as a system of homogeneous linear inequalities or equations resp., it is often (but by no means always) better to use (a variant of) Pottier's algorithm. The user can choose this algorithm by the command line option `-d` representing "dual" (see Sections 7 and 11).
With the dual algorithm NORMALIZ cannot compute triangulations, multiplicities, and h-vectors.

Access from computer algebra systems

(1) a package for MACAULAY2.
(2) library for SINGULAR extended by functions for torus invariants and valuation rings.

Changes in version 2.2:

User control, input and output:

(1) New command line option `-e` to activate test for arithmetic errors.
(2) New command line option `-m` to save memory at the expense of computation time. This option replaces "optimize for speed" in version 2.1.
(3) New command line option `-?` to print a small help text.
(4) Name of setup file changed from `setup.txt` to `normaliz.cfg`.
(5) It is now possible to give the input file with the ending ".in" (but not recommended).
(6) Option "Abort by user" removed. The program exits if an error is detected.
(7) Renamed "Run mode type" to "Computation type" for clearer distinction to the (run) mode.
(8) Renamed "Testing number" to "Overflow Test Modulus", "Lifting constant" to "Lifting bound" and "Use control data" to "Verbose".
(9) File extension `.hom` changed to `.ht1`.
(10) In mode `2=polytope` the vectors in the file `.ext` are given as extreme rays of the cone over the polytope (vertices of the polytope in the previous version).

3. Numerical limitations

The program comes in three executables: `norm32` works with 32 bit integers, `norm64` with 64 bit integers, and `normbig` uses integers of arbitrary precision. As a general rule, `norm32` is somewhat faster that `norm64` (say, by a factor of 1.5), which is significantly faster than `normbig` (say, by a factor of 10). We consider `norm64` as the standard choice.

We have implemented some arithmetic tests designed to catch errors due to arithmetic overflow (since version 2.0, see also Sections 7). They are quite good at doing that, however, some undetected errors may occur when running `norm32` and `norm64`. If you feel that an arithmetical problem could have arisen, we recommend using `normbig`, which is free from errors due to arithmetic overflow.

4. Distribution

We provide executables for Windows, Linux and Mac. Download the archive file corresponding to your system `Normaliz2.2<systemname>.zip` to a directory of your choice and unzip it. The names of the subdirectories created are self-explanatory.

- In the main directory `Normaliz2.2<systemname>` you should find the three executables, a file named `normaliz.cfg` and subdirectories.
- In the subdirectory `source/` you find the source files, a `Makefile` and again the file `normaliz.cfg`.
- Subdirectory `doc/` contains the file you are reading, as well as some documentation on the algorithms we use.
- In the subdirectory `example/` are the input and output files for some examples.
- The subdirectory `singular/` contains the SINGULAR library `normaliz.lib` and the documentation files `nmz_sing.tex` and `nmz_sing.pdf`.
- The subdirectory `macaulay2/` contains the MACAULAY2 package `Normaliz.m2` and the documentation files `nmzM2.tex` and `nmzM2.pdf`.

5. Compilation

If for some reason the executables we provide do not work on your system, you may want to compile the source files. Go to the `source/` directory. Then you can use `make` (see below) or compile the 32 and 64 bit executables by entering the commands

```
CompilerName -O3 N32.cc [-o TargetName]

CompilerName -O3 N64.cc [-o TargetName]
```

where "CompilerName" is the name of your favorite C++ compiler and "TargetName" is the name of the executable you want to build (this may depend on the operating system). The option `-O3` indicates that you should use the best available optimization.

For example, assuming that you work with the `GNU` compiler on a Windows system, the command to enter is

```
g++ -O3 N32.cc -o norm32.exe
```

while working with the `GNU` compiler on a Linux system the command to enter is

```
g++ -O3 N64.cc -o norm64
```

In order to compile the indefinite precision arithmetic executable it is crucial to have `GMP` installed on your system. When installing `GMP`, you must also build the `C++` class wrapper. This is not produced automatically by `GMP` at installation. The command which may help at installing `GMP` is

```
./configure --prefix="CompilerPath" --enable-cxx
```

where “CompilerPath” is the path to your compiler.

Now assuming that you have installed `GMP`, the command to produce the arbitrary precision arithmetic executable is

```
CompilerName -O3 NBig.cc [-o TargetName] -lgmpxx -lgmp
```

Assuming that you work with the `GNU` compiler on a Windows system the command to enter is

```
g++ -O3 NBig.cc -o normbig.exe -lgmpxx -lgmp
```

Note that Mac users may need to add the option `-m64` at this command line.

We also provide a `Makefile`. For Linux systems you can compile all 3 executables by calling

```
make
```

in the `source/` directory. Of course you need an C++ compiler and for the arbitrary precision version also `GMP`. Use

```
make [norm32|norm64|normbig]
```

to compile only a single version.

For Windows systems we have also added a `Visual C++` solution to the `source/` directory. If you want to use `Visual C++ Express` (which is freely available from Microsoft) to compile the source files, the directory `source/Visual C++/` contains the solution file `Normaliz.sln`. Assuming that you have installed `GMP` (unfortunately a difficult point), set the build configuration to ”Release” and build the solution `Normaliz`. You should find the executables in the directory `source/Visual C++/Release/`.

The executables for Windows have been compiled with `Visual C++`.

6. The input file

Each input is in fact a matrix, the rows of the matrix are interpreted according to a parameter called *mode*. The input file `<projectname>.in` is structured as follows.

The first line contains the number of rows, the second line contains the number of columns n, which is always the dimension of the ambient lattice.

The next lines contain the rows of the matrix. The last line contains a single digit between 0 and 5, the *mode*. Alternatively you can use a label for the mode. The following list shows the modes, their labels and a short description of what NORMALIZ will compute in "Computation type" `normal`. (See Section 7 for other computation types. They restrict or extend the data computed by Normaliz.)

Modes in which the rows of the input matrix represent generators $x_1,\dots,x_m \in \mathbb{Z}^n$ (of a cone, polytope or monomial ideal):

0 `integral_closure`: Computes the Hilbert basis of the rational cone generated by $x_1,\dots,x_m$ with respect to the ambient lattice $\mathbb{Z}^n$;

1 `normalization`: The same as `0`, but with respect to the sublattice of $\mathbb{Z}^n$ generated by $x_1,\dots,x_m$;

2 `polytope`: Computes the integral points in the polytope spanned by $x_1,\dots,x_m$ and its Ehrhart semigroup;

3 `rees_algebra`: Computes the integral closure of the Rees algebra of the ideal I generated by the monomials with exponent vectors $x_1,\dots,x_m$.

Modes in which the rows of the input matrix A represents a system of constraints:

4 `hyperplanes`: Computes the Hilbert basis of the rational cone in $\mathbb{R}^m$ given by the system of homogeneous inequalities; $Ax \geq 0$

5 `equations`: Computes Hilbert basis of the rational cone given by the nonnegative solutions of the homogeneous system $Ax = 0$.

Note that the output of NORMALIZ depends on the lattices involved. Therefore we define the *ambient lattice* $\mathbb{A}$, the semigroup S and the *effective lattice* $\mathbb{E}$ as follows:

mode 0: $\mathbb{A} = \mathbb{Z}^n$ where n is the number contained in the second line of the input file;
S is the subsemigroup of $\mathbb{A}$ generated by the vectors in the input file;
$\mathbb{E}$ is the smallest direct summand of $\mathbb{A}$ containing the subgroup $\operatorname{gp}(S) \subset \mathbb{A}$ generated by S.

mode 1: $\mathbb{A}$ and S as for mode 0, but $\mathbb{E} = \operatorname{gp}(S)$.

mode 2 (polytopal application): $\mathbb{A} = \mathbb{Z}^{n+1}$;
S is generated by the vectors $(x,1)$ for the vectors x in the input file;
$\mathbb{E}$ as in mode 0.

mode 3 (Rees application): $\mathbb{A} = \mathbb{Z}^{n+1}$;
S is generated by the unit vectors $e_1,\dots,e_n$ (representing the indeterminates of the polynomial ring) and the vectors $(x,1)$ for the vectors x in the input file;
$\mathbb{E} = \mathbb{Z}^{n+1}$.

mode 4 (system of inequations): $\mathbb{A}$ and $\mathbb{E}$ as for mode 0, S is the semigroup of the solutions of the system of linear inequations $Mx \geq 0$, where M is the input matrix. In this mode the input matrix must be of maximal rank.

mode 5 (system of equations): $\mathbb{A}$ and $\mathbb{E}$ as for mode 0, S is the semigroup of the non-negative solutions of the system of linear equations $Mx = 0$, where M is the input matrix.

In each case the integral closure of S in $\mathbb{E}$ is computed (in computation type `normal`). On its way to the Hilbert basis NORMALIZ computes auxiliary data. These will also be printed to the output file or files in the directory of the project (see section 8).

7. RUNNING NORMALIZ

The syntax for calling NORMALIZ is

```
norm64 [-acdfhimnpsv] [<projectname>]
```

where the options and `<projectname>` are optional. On a Linux/Mac system it might be necessary to use `./norm64` instead of `norm64`. If no `<projectname>` is given, the program will enter an "interactive mode". In this case the program will ask you for the name of the project and after the computations the program will ask you again to press some key in order to quit. NORMALIZ will look for `<projectname>.in` as input file.

For example, if you input the command

```
norm64 -c -p -a rafa2416
```

then the program will take the file `rafa2416.in` as input, control data will be printed on screen, the support hyperplanes, the triangulation, the multiplicity, the h-vector and the Hilbert polynomial will be computed and all the possible output files will be produced.

There is also the possibility to set options via the config file `normaliz.cfg`. The following table shows the options in the config file, their possible values and the matching command line parameter.

name	possible values	parameter
Run tests for arithmetic overflow	YES NO (default)	`-e`
Overflow test modulus	natural number (default 10403)	
Lifting bound	natural number (default 9000)	
Verbose	YES NO (default)	`-c`
Save memory	YES NO (default)	`-m`

Computation type	`support hyperplanes`
	`triangulation`
	`normal` (default)
	`hilbert_polynomial`
	`hilbert_basis_polynomial`
	`dual`
		`-s`
		`-v`
		`-n`
		`-p`
		`-h`
		`-d`

Write .out file	YES (default), NO	
Write .inv, .ext, .esp, .typ, .egn, .gen, .sup, .tri, .ht1 file	YES, NO (default)	

In addition to the command line parameters in the table above, there are three more:

-i: the setup file will be ignored
-f: the write `.out, .gen, .inv, .typ, .sup` file options are set to YES, the others to NO
-a: all write file options are set to YES

Note: (a) Command line options will override the options written in the config file.

(b) Do **not change the order of the lines in the config file** — in the present version the program evaluates the lines in the given order!

Note also that the config file must be in your working directory. This allows you to use different directories for different computations, and you can save the settings for each directory in the config file.

"Run tests for arithmetic overflow": When set to YES the arithmetic tests will be performed, in order to assure that no arithmetic errors do occur. This may slow down the computations. Depending on the particular example, it may double the running time. For `normbig`, this option is set to NO to get some extra speed, since in this case no arithmetic errors can occur.

"Overflow test modulus": If you see that on some particular example the arithmetic tests are falling, you may try to change this number. Up to now we do not have such an example.

"Lifting bound": The lifting bound is a number used to do a random lifting of the cone. Large numbers assure the success of the lifting, but may lead to arithmetic overflow. If while running the program you get an error message which says that lifting has failed, you should increase this number.

"Verbose": This will give you some access to 'control' data during the computation. It is designed for users who run complex examples and wish to see how far the program has come (and if it is still running at all). When set to YES data will be printed on screen. You may want to set this to NO, for example if you call NORMALIZ from another program.

"Save memory": When set to YES some computed data will not be saved in memory. These have do be computed again in some cases, increasing the run time of the program.

"Computation type": This is for sure the most important option to consider. It controls the computations performed by the program. The following choices are possible:

(1) `support_hyperplanes`: Only the support hyperplanes are computed.
(2) `triangulation`: Computes the support hyperplanes, the triangulation and the multiplicity.
(3) `normal`: Computes the support hyperplanes, the triangulation, the multiplicity and the Hilbert basis.
(4) `hilbert_polynomial`: Computes the support hyperplanes, the triangulation, the multiplicity, the h-vector and the Hilbert polynomial.
(5) `hilbert_basis_polynomial`: Performs all implemented computations.
(6) `dual`: Computes the Hilbert basis using Pottier's algorithm [6]. It is available only in *mode* "4" and "5" (see Section 6).

If you run some complex example, it may be important to choose the right option here since it may greatly affect the running time. On some particular example it may take seconds to run the program in the `triangulation` mode and hours (days, years ...) with the `hilbert_basis_polynomial` mode. Try the examples `big.in` and `huge.in` (provided in the directory `example/`) to get a feeling for the computation time.

The remainder are "write" options: When set to YES the output file with the corresponding termination will be produced. Note that the information needed to write the output file should be available after computations (this depends on the option "Computation type"), or no output file will be produced. See also Section 9 for information on the possible output files.

Finally some remarks for users who want to run complex examples. Computing the Hilbert basis may be the most time consuming part. If you intend to compute the Hilbert basis, it may be useful to get an estimate of the running time first. If the example is homogeneous (see Section 8) this is possible by making a "multiplicity test". First run `normbig` with the option "Computation type" set to `triangulation` (parameter `-v` in the command line). This runs fast and the multiplicity is computed. Well, the multiplicity is the number of vectors that have to be reduced in order to compute the Hilbert basis. For example an input file with multiplicity around 10.000.000 may take some hours to compute with `normbig`. Also note that there are two stages of reduction, one local (fast), and one global (slow), and that information about the local and global reduction is printed on the screen if option "Verbose"is set to YES. Next you may try to compute your example with `norm64`, which is much faster than `normbig`. Set option "Run tests for arithmetic overflow" to YES, since arithmetic overflow is possible when running `norm64`. If `norm64` fails because of arithmetic overflow, run `normbig`.

`FortuneCookie` is an interesting example. You should compute it with `normbig`, at least if the "Computation type" is set to `hilbert_basis_polynomial`.

8. THE OUTPUT FILE

First note that the data you will find in the output file depend on the option "Computation type". For example, if "Computation type" is set to `normal`, you can not find the h-vector and the coefficients of the Hilbert polynomial because they are not computed.

In mode ≤ 1 and mode ≥ 4 the output file `<projectname>.out` may contain the following data:

- the generators of the integral closure of S in $\mathbb{E}$;
- the extreme rays of the cone C generated by S;
- $\operatorname{rank}\operatorname{gp}(S)$;
- the index of $\operatorname{gp}(S)$ in $\mathbb{E}$;
- if $\operatorname{rank}\mathbb{E} = \operatorname{rank}\mathbb{A}$ the (unique) support hyperplanes of C;
- if S is homogeneous the height 1 generators of the integral closure;
- if S is homogeneous the multiplicity;
- if S is homogeneous, the h-vector and the coefficients of the Hilbert polynomial.

The extreme rays are given by the elements in S that define the extreme rays.

The support hyperplanes are not listed if $\operatorname{rank}\mathbb{E} < \operatorname{rank}\mathbb{A}$. However, in this case the support hyperplanes of the cone generated by S in $\mathbb{R}\mathbb{E}$ is available in an optional output file (see Section 9).

We call S *homogeneous* if there is an integer-valued linear form φ on $\mathbb{E}$ such that all the generators v of S given in the input file satisfy $\varphi(v) = 1$. For instance, the examples `rafa2416`, `squaref1`, `rproj2`, `small`, `medium`, `big` and `huge` from the directory `example` are homogeneous. (Sometimes φ must be multiplied by a positive integer before the lifting from $\mathbb{A}$ to $\mathbb{E}$! Therefore you may get a value > 1 when evaluating the linear form in the output file on the height 1 elements.)

We ask for S to be homogeneous when printing the height 1 generators of the integral closure (that is $\varphi(g) = 1$), the multiplicity, the h-vector and the coefficients of the Hilbert polynomial to the output file.

Note that it can very well happen that the computation of the integral closure of S runs without problems, but an overflow occurs in the Hilbert polynomial computation. You may use `normbig` with "Computation type" set to `hilbert_polynomial` to check the results obtained with `norm32` or `norm64` (this is relatively fast, and a good test for arithmetic problems).

If "mode $= 2$", the following data may be found in the output file:

- the generators of the semigroup determined by the polytope, called Ehrhart semigroup in the following;
- the lattice points of the polytope;

- the extreme points;
- the support hyperplanes if it is of full dimension;
- the normalized volume;
- the *h*-vector and the coefficients of the Ehrhart polynomial.

In “mode = 3”, the output file may contain the following:

- the generators of the integral closure $\bar{\mathscr{R}}$ of the Rees algebra;
- the extreme rays;
- the generators of the integral closure of the ideal;
- the support hyperplanes;
- the height 1 generators of the integral closure if it is homogeneous;
- the multiplicity of the semigroup if it is homogeneous;
- if the ideal is primary to the irrelevant maximal ideal, the multiplicity of the ideal (not to be confused with the multiplicity of the semigroup);
- the *h*-vector and the coefficients of the Hilbert polynomial of $\bar{\mathscr{R}}$.

9. Optional output files

When the “write” options are set to YES, NORMALIZ writes additional output files whose names are of type `<projectname>.<type>`. The format of the files (with the exception of `inv`) is completely analogous to that of the input file, except that there is no last line denoting the mode.

The following files may be written, provided certain conditions are satisfied and the information that should go into them is available (we denote the files simply by their types):

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

`ext` The file `ext` contains the extreme rays, provided they have been computed.

`gen` The generators of the integral closure are written to this file (provided they have been computed).

`sup` If $\operatorname{rank}\mathbb{A} = \operatorname{rank}\mathbb{E}$, then the support hyperplanes are written.

`egn,esp` These are defined as `gen` and `sup`, however with respect to the lattice $\mathbb{E}$ and a basis of $\mathbb{E}$. Note that these data provide a description of the integral closure of S in the form $\mathbb{E} \cap C$.

`typ` This is the product of the matrices corresponding to `egn` and `esp`. In this case, the support hyperplanes of the cone C are evaluated (as linear forms) on the generators. The resulting matrix, with the generators corresponding to the rows and the support hyperplanes corresponding to the columns, is written to this file. Note that this file replaces the files `val` and `evl` from the old version. This the only significant change to the output we did in the new version.

`tri` The file `tri` contains a triangulation of the cone C computed by NORMALIZ. 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 input file) of those generators of S 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 $\mathbb{E}$).

`ht1` If S is *homogeneous*, the file `ht1` contains the height 1 elements of the cone.

10. Examples

Note: When you run Normaliz yourself on one of the examples in the distribution, it may happen that the lists of the vectors in the output appear in a different order since the order sometimes depends on random choices of the program.

The file `rproj2.in` contains the following (here typeset in 2 columns):

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

This means that we wish to compute the integral closure of the semigroup generated by the 16 vectors

$$[1,0,0,0,0,0,0], \quad [0,1,0,0,0,0,0], \quad \dots, \quad [0,0,1,1,0,1,1]$$

in dimension 7. We compute it in the ambient lattice $\mathbb{Z}^7$, which is indicated by the final digit 0.

Running `norm64` (option "Computation type" set to `"hilbert_basis_polynomial"`) produces the file `rproj2.out` which has the following content (here typeset in 2 columns):

```
17 generators of integral closure:    16 extreme rays
 1 0 0 0 0 0 0                         1 0 0 0 0 0 0
 0 1 0 0 0 0 0                         0 1 0 0 0 0 0
 0 0 1 0 0 0 0                         0 0 1 0 0 0 0
 0 0 0 1 0 0 0                         0 0 0 1 0 0 0
 0 0 0 0 1 0 0                         0 0 0 0 1 0 0
 0 0 0 0 0 1 0                         0 0 0 0 0 1 0
 1 1 1 0 0 0 1                         1 1 1 0 0 0 1
 1 1 0 1 0 0 1                         1 1 0 1 0 0 1
 1 0 1 0 1 0 1                         1 0 1 0 1 0 1
 1 0 0 1 0 1 1                         1 0 0 1 0 1 1
 1 0 0 0 1 1 1                         1 0 0 0 1 1 1
 0 1 1 0 0 1 1                         0 1 1 0 0 1 1
 0 1 0 1 1 0 1                         0 1 0 1 1 0 1
 0 1 0 0 1 1 1                   0 1 0 0 1 1 1
 0 0 1 1 1 0 1                   0 0 1 1 1 0 1
 0 0 1 1 0 1 1                   0 0 1 1 0 1 1
 1 1 1 1 1 1 2

 (original) semigroup has rank 7 (maximal)
 (original) semigroup is of index 1

24 support hyperplanes:          16 height 1 generators of
integral closure:
  0  0  0  1  0  0  0             1 0 0 0 0 0 0
  0  0  0  0  1  0  0             0 1 0 0 0 0 0
  0  0  0  0  0  1  0             0 0 1 0 0 0 0
  0  0  0  0  0  0  1             1 1 1 0 0 0 1
  0  0  1  0  0  0  0             0 0 0 0 0 1 0
  0  1  0  0  0  0  0             0 1 1 0 0 1 1
  0  1  0  1  1  0 -1             0 0 0 0 1 0 0
  0  1  0  0  1  1 -1             1 0 1 0 1 0 1
  0  1  1  0  0  1 -1             1 0 0 0 1 1 1
  0  0  1  1  1  0 -1             0 1 0 0 1 1 1
  0  0  1  1  0  1 -1             0 0 0 1 0 0 0
  0  1  1  1  1  1 -2             1 1 0 1 0 0 1
  1  0  0  0  0  0  0             1 0 0 1 0 1 1
  1  1  1  1  1  1 -3             0 0 1 1 0 1 1
  1  0  0  1  0  1 -1             0 1 0 1 1 0 1
  1  0  0  0  1  1 -1             0 0 1 1 1 0 1
  1  0  1  0  1  0 -1
  1  0  1  1  1  1 -2
  1  1  0  1  0  0 -1
  1  1  1  0  0  0 -1
  1  1  1  1  0  1 -2
  1  1  1  0  1  1 -2
  1  1  1  1  1  0 -2
  1  1  0  1  1  1 -2

 (original) semigroup is homogeneous via the linear form:
 1 1 1 1 1 1 -2

multiplicity = 72

h-vector = 1 9 31 25 6 0 0

Hilbert polynomial : 1/1 97/30 71/15 49/12 13/6 41/60 1/10
```

From this, we see that there are 17 generators of the integral closure of the semigroup in $\mathbb{Z}^7$ and 16 extreme rays, that the semigroup has index 1 in $\mathbb{Z}^7$, and that the corresponding support hyperplanes are given by the linear forms $[0,0,0,1,0,0,0]$, $[0,0,0,0,1,0,0]$, $\dots$,

$[1,1,0,1,1,1,-2]$. We are also given the information that the semigroup is homogeneous and that its multiplicity is 72.

Since we are in the homogeneous case the height 1 generators of integral closure, the h-vector and Hilbert polynomial are also computed. The h-vector of $\bar{S}$ is

$$(h_0,h_1,\dots,h_6) = (1,9,31,25,6,0,0),$$

and the Hilbert polynomial of $\bar{S}$ is given by

$$P_{\bar{S}}(t) = \frac{1}{1} + \frac{97}{30}t + \frac{71}{15}t^2 + \frac{49}{12}t^3 + \frac{13}{6}t^4 + \frac{41}{60}t^5 + \frac{1}{10}t^6 .$$

Here is another example from the file `polytop.in`:

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

The lattice points 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 $\mathbb{R}^3$ are to be computed. (Note the last line, indicating the polytopal mode 2.)

Running `norm64` (option “Computation type” set to `"hilbert_basis_polynomial"`) produces the file `polytop.out`:

```
19 generators of Ehrhart ring:    18 lattice points in polytope:
 0 0 0 1                           0 0 0
 2 0 0 1                           2 0 0
 0 3 0 1                           0 3 0
 0 0 5 1                           0 0 5
 0 0 1 1                           0 0 1
 0 0 2 1                           0 0 2
 0 0 3 1                           0 0 3
 0 0 4 1                           0 0 4
 0 1 0 1                           0 1 0
 0 1 1 1                           0 1 1
 0 1 2 1                           0 1 2
 0 1 3 1                           0 1 3
 0 2 0 1                           0 2 0
 0 2 1 1                           0 2 1
 1 0 0 1                           1 0 0
 1 0 1 1                           1 0 1
 1 0 2 1                           1 0 2
 1 1 0 1                           1 1 0
 1 2 4 2
```

```
4 extreme points of polytope:     4 support hyperplanes:
 0 0 0                             -15 -10  -6 >= -30
 2 0 0                               1   0   0 >=   0
 0 3 0                               0   1   0 >=   0
 0 0 5                               0   0   1 >=   0

normalized volume = 30

h-vector = 1 14 15 0

Ehrhart polynomial : 1/1 4/1 8/1 5/1
```

The desired lattice points are the 18 ones listed above. To complete the picture, we also provide all the generators of the Ehrhart ring of the polytope. (There are 19 of them in this example.) Furthermore, the original polytope is the solution of the system of the 4 inequalities

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

and has normalized volume 30.

The last two lines provide the information that the h-vector of the Ehrhart ring is

$$(h_0, h_1, h_2, h_3) = (1, 14, 15, 0),$$

and its Ehrhart polynomial is

$$P(t) = 1 + 4t + 8t^2 + 5t^3.$$

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

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

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

Here we want to compute the integral closure of the Rees algebra of the ideal generated by the monomials corresponding to the above 10 exponent vectors. (Note again the last line, containing 3 in this case.) The output in `rees.out` coincides with that in `rproj2.out`, up to notions and the supplementary information on the integral closure of the ideal:

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

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

The file dual.in looks like:

```
24
7
0 0 0 1 0 0  0              1 0 0 0 0 0  0
0 0 0 0 1 0  0              1 1 1 1 1 1 -3
0 0 0 0 0 1  0              1 0 0 1 0 1 -1
0 0 0 0 0 0  1              1 0 0 0 1 1 -1
0 0 1 0 0 0  0              1 0 1 0 1 0 -1
0 1 0 0 0 0  0              1 0 1 1 1 1 -2
0 1 0 1 1 0 -1              1 1 0 1 0 0 -1
0 1 0 0 1 1 -1              1 1 1 0 0 0 -1
0 1 1 0 0 1 -1              1 1 1 1 0 1 -2
0 0 1 1 1 0 -1              1 1 1 0 1 1 -2
0 0 1 1 0 1 -1              1 1 1 1 1 0 -2
0 1 1 1 1 1 -2              1 1 0 1 1 1 -2
                            hyperplanes
```

The last line, contains 4 in this case. This means that we wish to compute the Hilbert basis of the dual of the cone spanned by the 24 generators. The entries we have chosen are exactly the support hyperplanes from the file rproj2.out. The output in dual.out coincides with that in rproj2.out.

Finally, the mode 5 is presented in the next section.

11. MAGIC SQUARES

Suppose that you have the following square

a	b	c
d	e	f
h	i	j

and the problem is to find nonnegative values for a, b, ..., j such that the 3 numbers in all rows, all columns, and both diagonals sum to the same constant $\mathscr{M}$ (called the magic constant). This leads to a linear system of equations

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

The associated system of equations in this case is contained in the file `3x3magic.in`. The output file contains the information you need to write the solution:

```
5 generators of integral closure:
 1 2 0 0 1 2 2 0 1
 2 0 1 0 1 2 1 2 0
 0 2 1 2 1 0 1 0 2
 1 0 2 2 1 0 0 2 1
 1 1 1 1 1 1 1 1 1
```

The generators of the integral closure are in fact generators of the solution cone for the given problem. Each row describes a solution. That means that the 5 squares

1	2	0
0	1	2
2	0	1

2	0	1
0	1	2
1	2	0

0	2	1
2	1	0
1	0	2

1	0	2
2	1	0
0	2	1

1	1	1
1	1	1
1	1	1

are solution for the problem, and all other solutions are linear combinations of these squares with nonnegative integer coefficients.

The next question one may rise is: Given a constant $\mathscr{M}$, how many magic square are with magic constant $\mathscr{M}$? First all generators have magic constant 3, so there are no magic squares if $\mathscr{M} \neq 3t$. If $\mathscr{M} = 3t$, then the answer (in this particular case) is given by the Hilbert polynomial

$$P(t) = 1+2t+2t^2\,.$$

There are certain cases in *mode* 4 and 5 when computing the Hilbert basis with Pottier's algorithm is faster than the algorithm previously implemented by us (for example the magic squares). This is why we have added also the Pottier's algorithm as "Run_mode_type" `dual`. The following table contains test data we have obtained for computing the Hilbert basis (time is given in seconds).

	normal	dual
dual.in	0.005	0.004
cut.in	0.5	6.6
rafad.in	580	∞
4x4.in	0.01	0.003
6x6.in	∞	97000

12. PROBLEM

In the present version NORMALIZ cannot deal adequately with the zero cone. It is very unlikely that the zero cone comes up in modes 0, 1, 2 or 3, but in modes 4 and 5 this may very well happen. In this case you will get strange error messages.

COPYRIGHT

Please refer to NORMALIZ in the following manner in any publication for which it has been used:

W. Bruns and B. Ichim NORMALIZ. Computing normalizations of affine semigroups. With contributions by C. Söger. Available from http://www.math.uos.de/normaliz.

REFERENCES

[1] W.Bruns and J. Gubeladze. *Polytopes, rings, and K-theory*. Springer 2009.
[2] W.Bruns and J. Herzog. *Cohen-Macaulay Rings*. Rev. ed. Cambridge University Press 1998.
[3] W.Bruns and B. Ichim. *Algorithms for rational cones and affine monoids.* In preparation.
[4] W.Bruns, R. Koch et al. NORMALIZ, *Computing normalizations of affine semigroups*. (1998–2006).
[5] W.Bruns and R. Koch. *Computing the integral closure of an affine semigroup*. Uni. Iaggelonicae Acta Math. **39** (2001), 59–70.
[6] L. Pottier. *The Euclide algorithm in dimension n*. Research report, ISSAC 96, ACM Press 1996.

UNIVERSITÄT OSNABRÜCK, FB MATHEMATIK/INFORMATIK, 49069 OSNABRÜCK, GERMANY

E-mail address: `winfried@math.uos.de`

INSTITUTE OF MATHEMATICS "SIMION STOILOW" OF THE ROMANIAN ACADEMY, 010702 BUCHAREST, ROMANIA

E-mail address: bogdan.ichim@imar.ro