THE LIBRARY NORMALIZ.LIB (VERSION 2.2) FOR NORMALIZ 2.2

WINFRIED BRUNS

CONTENTS

The library normaliz.lib provides an interface for the use of Normaliz 2.2 within Singular. The exchange of data is via files, the only possibility offered by Normaliz in its present version. In addition to the top level functions that aim at objects of type ideal or ring, several other auxiliary functions allow the user to apply Normaliz to data of type intmat. Therefore Singular can be used as a comfortable environment for the work with Normaliz.

Note: (a) The order in which the vectors or monomials in the examples are computed, depends sometimes on random parameters. Therefore your own computations may produce them in a different order.

(b) In order to save space some of the examples below are typeset in two or three columns.

(c) In version 2.2 of the library there are some changes to meet the guidelines for Singular libraries. This includes the renaming of procedures: the underscores are removed and the following letter set to uppercase. Furthermore the old procedures for setting options are no longer available, see Section 6.

(d) Version 2.2 of normaliz.lib uses sed to transfer the normaliz output into a Singular conform format. So please make sure sed is in the search path of your system.

(e) A significant change in version 2.1.1 of the library is that intermediate files are erased automatically if the user does not set a filename explicitly.

1. PREPARATIONS

There are only two steps of which the second is optional:

Date: 16 July 2009.

(1) The library must be stored in a directory where `Singular` looks for libraries.
(2) The executable(s) for `Normaliz` should be stored in a directory on your search path.

Instead of storing the executable(s) on your search path you can also specify a path where `Singular` should look for them. See Section 8.

2. A SIMPLE EXAMPLE

The typical application of `Normaliz` in commutative algebra is the computation of the integral closures of monomial ideals and monomial subalgebras of polynomial ring. In the following example we consider the ideal $I = (x^2, y^2, z^3)$ in the polynomial ring $S = K[x, y, z]$ and want to compute the integral closure of the ideal and the integral closure of the Rees algebra $\mathscr{R}(I)$ of I in the ring $R = S[t]$ of which $\mathscr{R}(I)$ is naturally a subalgebra. (The output of `Singular` is partially typeset in 3 columns.)

```
                     SINGULAR                                 /
 A Computer Algebra System for Polynomial Computations       /   version 3-0-4
                                                            0<
     by: G.-M. Greuel, G. Pfister, H. Schoenemann             \   Nov 2007
FB Mathematik der Universitaet, D-67653 Kaiserslautern         \
> LIB "normaliz.lib";
// ** loaded normaliz.lib 2.2, 2009/07/14
> ring R=0,(x,y,z,t),dp;
> ideal I=x^2,y^2,z^3;
> intclMonIdeal(I);
[1]:                    [2]:
   _[1]=y2                 _[1]=z              _[7]=x
   _[2]=xy                 _[2]=y2t            _[8]=yz2t
   _[3]=x2                 _[3]=xyt            _[9]=xz2t
   _[4]=z3                 _[4]=x2t
   _[5]=yz2                _[5]=z3t
   _[6]=xz2                _[6]=y
```

The library is loaded by `Normaliz.lib` as usual.

We define R and I as indicated above, and compute what we wanted by `intclMonIdeal(I)`. The output is a list of two monomial ideals, of which the integral closure of I is the first. The second ideal is to be considered as the list of monomials generating the integral closure of the Rees algebra. we continue by retrieving the numerical information computed by `Normaliz`. The last two last lines tell us that our ideal I is primary to the maximal ideal of S with multiplicity 12. The other lines give information about the cone generated by (the exponent vectors of) the monomials in the Rees algebra. (Think about their ring theoretic interpretation!)

```
> showNuminvs();
hilbert_basis_elements : 9
number_extreme_rays : 6
rank : 4
index : 1
number_support_hyperplanes : 5
homogeneous : 0
primary : 1
ideal_multiplicity : 12
```

The exchange of data between Singular and Normaliz is via files. These files are created and erased automatically. (*A significant change in version 2.1.1.*) However, you may want to inspect them for additional information, and the library contains functions for this purpose. In this case you must change the treatment of files from "delete" to "keep". This is done by the explicit declaration of a filename; see Section 8.

3. INTEGRAL CLOSURES OF MONOMIAL IDEALS AND TORIC RINGS

There are 4 functions for the computation of integral closures, corresponding to the modes 0,1,2,3 of Normaliz. In all cases the parameter of the function is an ideal. Its elements need not be monomials: the exponent vectors of the leading monomials form the input of Normaliz. Note: the functions return nothing if one of the options supp, triang, or hvect has been activated. However, in this case some numerical invariants are computed, and some other data may be contained in files that you can read into Singular (see Section 9).

- normalToricRing(ideal I)

  Computes the normalization of the toric ring generated by the leading monomials of the elements of I. The function returns an ideal listing the generators of the normalization.

  A mathematical remark: the toric ring (and the other rings computed) depends on the list of monomials given, and not only on the ideal they generate!

```
> ring R=37,(x,y,t),dp;
> ideal I=x3,x2y,y3;
> normalToricRing(I);
x3 x2y y3 xy2
```

- intclToricRing(ideal I)

  Computes the integral closure of the toric ring generated by the leading monomials of the elements of I in the basering. The function returns an ideal listing the generators of the integral closure.

```
> intclToricRing(I);
x y
```

- ehrhartRing(ideal I)

  The exponent vectors of the leading monomials of the elements of I are considered as generators of a lattice polytope. The function returns a list of ideals:

  (i) If the last ring variable is not used by the monomials, it is treated as the auxiliary variable of the Ehrhart ring. The function returns two ideals, the first containing the monomials representing the lattice points of the polytope, the second containing the generators of the Ehrhart ring.

  (ii) If the last ring variable is used by the monomials, the list returned contains only one ideal, namely the monomials representing the lattice points of the polytope.

```
> ehrhartRing(I);
[1]:
   _[1]=x3
   _[2]=x2y
   _[3]=y3
   _[4]=xy2
[2]:
   _[1]=x3t
   _[2]=x2yt
```

```
> ideal J=I,xy2t7;
> ehrhartRing(J);
[1]:
   _[1]=x3
   _[2]=x2y
   _[3]=y3
   _[4]=xy2t7
   _[5]=x2yt
   _[6]=x2yt2
   _[9]=xy2t5
   _[10]=xy2t4
   _[11]=xy2t3
   _[12]=xy2t2
   _[13]=xy2t
   _[14]=xy2
```

```
   _[3]=y3t                _[7]=x2yt3
   _[4]=xy2t               _[8]=xy2t6
```

- `intclMonIdeal(ideal I)`

  The exponent vectors of the leading monomials of the elements of `I` are considered as generators of a monomial ideal whose Rees algebra is computed. The function returns a list of ideals:

  (i) If the last ring variable is not used by the monomials, it is treated as the auxiliary variable of the Rees algebra. The function returns two ideals, the first containing the monomials generating the integral closure of the monomial ideal, the second containing the generators of the Rees algebra.

  (ii) If the last ring variable is used by the monomials, the list returned contains only one ideal, namely the monomials generating the integral closure of the ideal.

```
> intclMonIdeal(I);
[1]:                  [2]:                    > intclMonIdeal(J);
   _[1]=x3               _[1]=x                 [1]:
   _[2]=x2y              _[2]=y                    _[1]=x3
   _[3]=y3               _[3]=x3t                  _[2]=x2y
   _[4]=xy2              _[4]=x2yt                 _[3]=y3
                         _[5]=y3t                  _[4]=xy2
                         _[6]=xy2t
```

4. TORUS INVARIANTS, VALUATION RINGS AND IDEALS

Let $T = (K^*)^r$ be the r-dimensional torus acting on the polynomial ring $R = K[X_1,\dots,X_n]$ diagonally. Such an action can be described as follows: there are integers a_{ij}, $i = 1,\dots,r$, $j = 1,\dots,n$, such that $(\lambda_1,\dots,\lambda_r) \in T$ acts by the substitution

$$X_j \mapsto \lambda_1^{a_{1j}} \cdots \lambda_r^{a_{rj}} X_j, \qquad j = 1,\dots,n.$$

In order to compute the ring of invariants R^T one must specify the matrix (a_{ij}).

- `torusInvariants(intmat T)`

  The function returns an ideal representing the list of monomials generating R^T where in the function `T` stands for the matrix (a_{ij}).

```
> ring R=0,(x,y,z,w),dp;
> intmat T[2][4]=-1,-1,2,0,  1,1,-2,-1;
> torusInvariants(T);
x2z xyz y2z
```

It is of course possible that $R^T = K$. At present, `Normaliz` cannot deal with the zero cone and will issue the (wrong) error message that the cone is not pointed. The function also gives an error message if the matrix `T` has the wrong number of columns.

A discrete monomial valuation v on R (as above) is determined by the values $v(X_j)$ of the indeterminates. The following function computes the subalgebra $S = \{f \in R : v_i(f) \geq 0, i = 1,\dots,r\}$ for several such valuations v_i, $i = 1,\dots,r$. It needs the matrix $V = (v_i(X_j))$ as its input.

- `valRing(intmat V)`

  The function returns a monomial ideal, to be considered as the list of monomials generating S.

```
> ring R=0,(x,y,z,w),dp;
> intmat V0[2][4]=0,1,2,3, -1,1,2,1;
> valRing(V0);
w xw y xy z xz x2z
```

Again it is possible that $S = K$, and then the same (wrong) error message as above will be issued.

One can simultaneously determine the S-submodule $M = \{f \in R : v_i(f) \geq w_i, i = 1,\dots,r\}$ for integers $w_1,\dots,w_r$. (If $w_i \geq 0$ for all i, M is an ideal of S.) The numbers w_i form the $(n+1)$th column of the input matrix:

```
> intmat V[2][5]=0,1,2,3,4, -1,1,2,1,3;
> valRingIdeal(V);
[1]:                [2]:
   _[1]=y              _[1]=zw              _[7]=y4
   _[2]=xy             _[2]=xz2             _[8]=xy4
   _[3]=w              _[3]=z2              _[9]=yw2
   _[4]=xw             _[4]=y2w             _[10]=w3
   _[5]=z              _[5]=y2z
   _[6]=xz             _[6]=xy2z
   _[7]=x2z
```

We have just seen an example for

- `valRingIdeal(intmat V)`

  The function returns two ideals, both to be considered as lists of monomials. The first is the system of monomial generators of S, the second the system of generators of M.

Note: The functions in this section use the modes 4 and 5 of `Normaliz`. For large matrices T or V it could be useful to set the `dual` option. See Section 6.

5. RETRIEVING NUMERICAL INVARIANTS

The following functions make the numerical invariants computed by `Normaliz` accessible to `Singular`. Some of the these invariants are always computed,regardless of the option(s) set.

- `showNuminvs()`

  This function types the numerical invariants on the standard output, but returns nothing.

  ```
  > showNuminvs();
  hilbert_basis_elements : 73
  number_extreme_rays : 24
  rank : 16
  index : 1
  number_support_hyperplanes : 457
  homogeneous : 1
  height_1_elements : 24
  homogeneous_weights : 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
  multiplicity : 2531
  h_vector : 1,8,32,88,207,412,660,670,365,84,4,0,0,0,0,0
  ```

- `exportNuminvs()`

  This function exports the data read by `getNuminvs()` into numerical `Singular` data that can be accessed directly. For each invariant a variable of type `int` or `intvec` is created whose name is the first entry of each list element shown above, prefixed by `nmz_`.

```
> exportNuminvs();
> nmz_h_vector;
1,8,32,88,207,412,660,670,365,84,4,0,0,0,0,0
> nmz_index;
1
```

Note: (a) The `inv` file also contains the coefficients of the Hilbert (or Ehrhart) polynomial. Even for medium sized examples they may be too large for the type `int`. Therefore these coefficients are not imported.

(b) Only the data computed by `normaliz` are read. There are no "blank" entries in the result of `getNuminvs()`.

(c) The numerical invariants are stored in a variable of the library. You can inspect the variable by typing `Normaliz::NumInvs`.

6. SETTING OPTIONS

The library always uses the options `-f` and `-i` for `Normaliz` (the latter can be deactivated; see below). The options are set as follows:

- `setNmzOption(string optname, int onoff)`
  If `onoff=1` the option is activated, and if `onoff=0` it is deactivated.
- `showNmzOptions()`
  Returns the string of activated options.

The `Normaliz` options are accessible via the following names:

```
-s: supp                -a: allf
-v: triang              -c: control
-p: hvect               -i: ignore
-n: normal              -e: errorcheck
-h: hilb                -m: savememory
-d: dual
```

Note: (a) It makes no sense to activate more than one of the options in the left column. The option `normal` is hardly ever necessary because `Normaliz` uses it automatically (provided the setup file does not say something else—but that is ignored, unless you deactivate `ignore`).

(b) The option `allf` makes only sense if a filename has been set explicitly; see Section 8.

```
> setNmzOption("hilb",1);
> setNmzOption("hulb",1);
Invalid option  hulb
> showNmzOptions();
 -f -h -i
```

The old functions

- `set_hilb_option(int onoff)`
- `set_vol_option(int onoff)`
- `set_c_option(int onoff)`
- `set_allf_option(int onoff)`

are no longer available. Please use `setNmzOption()`. The old option `vol` is now represented by `triang` and `c` by `control`.

7. MONOMIALS TO/FROM INTMAT

The transformation of data between an `ideal` and an `intmat` is carried out by the following functions:

- `mons2intmat(ideal I)`

  Returns the `intmat` whose rows represent the leading exponents of the elements of `I`. The length of each row is `nvars(basering)`.

```
> intmat m=mons2intmat(J);
> m;
3,0,0, 2,1,0, 0,3,0, 1,2,7
```

- `intmat2mons(intmat m)`

  The converse operation.

```
> intmat2mons(m);
x3 x2y y3 xy2t7
```

8. PATHS AND FILES

The most important function in this section is `setNmzFilename` since the specification of a (nonempty) filename causes the library to change its treatment of intermediate files. The functions `readNmzData` and `writeNmzData` discussed in Section 9 require such an explicit filename.

If `Normaliz` is not in the search path for executables, then its path must be made known to `Singular`. Furthermore one can set the path to the directory where `Normaliz` and `Singular` exchange data, choose the executable (if `normbig` is needed), and remove the files created.

The path names need to be defined only once since they can be written to the hard disk and retrieved from there in subsequent sessions.

- `setNmzFilename(string s)`

  The function sets the filename for the exchange of data. As said above, it is mandatory to set a nonempty filename if you want to keep the files created for and by `Normaliz`. Unless a path is set by `setNmzDataPath`, files will be created in the current directory.

```
> setNmzFilename("VeryInteresting");
> Normaliz::nmz_filename;       // the variable holding the file name
VeryInteresting
```

  Note: Resetting the filename by `setNmzFilename("")` forces the library to return to deletion of temporary files, but the files created while the filename had been set will not be erased.

- `rmNmzFiles()`

  This function removes the files created for and by `Normaliz`, using the last filename specified. It needs an explicit filename.

- `setNmzExecPath(string s)`

  The function sets the path to the executable for `Normaliz`. This is absolutely necessary if it is not in the search path.

```
> setNmzExecPath("d:/Normaliz2.1Windows"); // Windows
> Normaliz::nmz_exec_path;  // the variable holding the path name
d:/Normaliz2.1Windows/  // the last / is added (if necessary)
> setNmzExecPath("$HOME/Normaliz2.1Linux");  // Linux
```

Please consult the installation instructions of `Singular` for the conventions regarding path names under Windows.

- `setNmzVersion(string s)`

  The function chooses the version of the executable for `Normaliz`. The default is `norm64`, and nothing needs to be done if it is sufficient.

```
> setNmzVersion("normbig"); // choose normbig
> Normaliz::nmz_version;       // the variable holding the version name
normbig
> setNmzVersion("norm32");  // now it is norm32
```

- `setNmzDataPath(string s)`

  The function sets the directory for the exchange of data. It will only be used if a non-empty filename has been specified.

```
> setNmzDataPath("d:/Normaliz2.1Windows/example"); // Windows
> Normaliz::nmz_data_path;    // the variable holding the path name
d:/Normaliz2.1Windows/example/
setNmzDataPath("../MyFiles/normaliz");  // Linux
```

  Note: It seems that `Singular` cannot use filenames starting with `~` or `$HOME` in its input/output functions.

  You must also avoid path names starting with `/` if you work under Cygwin, since `Singular` and `Normaliz` interpret them in different ways.

- `writeNmzPaths()`

  The function writes the path names into two files in the current directory. If one of the names has not been defined, the corresponding file is created, but contains nothing.

```
> writeNmzPaths();
> int dummy=system("sh","cat nmz_sing_exec.path");
d:/Normaliz2.1Windows/
> dummy=system("sh","cat nmz_sing_data.path");
d:/Normaliz2.1Windows/example/
```

- `startNmz()`

  This function reads the files written by `writeNmzPaths()`, retrieves the path names, and types them on the standard output (as far as they have been set). Thus, once the path names have been stored, a `Normaliz` session can simply be opened by this function.

```
> startNmz();
nmz_exec_path is d:/Normaliz2.1Windows/
nmz_data_path is d:/Normaliz2.1Windows/example/
> setNmzDataPath("");
> writeNmzPaths();
> startNmz();
nmz_exec_path is d:/Normaliz2.1Windows/
nmz_data_path not set
```

9. RUNNING NORMALIZ ON DATA OF TYPE INTMAT

There are functions to write and read files created for and by Normaliz. Note that all functions in Sections 3 and 4 as well as the function normaliz below write and read their data automatically to and from the hard disk so that writeNmzData will hardly ever be used explicitly.

Note: In order to access files created by Normaliz you must have specified a filename. Also writeNmzData needs an explicit filename.

- writeNmzData(intmat sgr, int nmz_mode)

  Creates an input file for Normaliz. The rows of sgr are considered as the generators of the semigroup. The parameter nmz_mode sets the mode.

```
> setNmzFilename("VeryInteresting");
> intmat sgr[3][3]=1,2,3,4,5,6,7,8,10;
> writeNmzData(sgr,1);
> int dummy=system("sh","cat VeryInteresting.in");
3
3
1 2 3
4 5 6
7 8 10
0
```

- normaliz(intmat sgr, int nmz_mode)

  The function applies Normaliz to the parameter sgr in the mode set by nmz_mode. The function returns the intmat defined by the file with suffix gen.

```
> normaliz(sgr,0);
> print(normaliz(sgr,0));
     7     8    10
     1     2     3
     2     3     4
     3     4     5
     4     5     6
```

- readNmzData(string suffix)

  Reads an output file of Normaliz containing an integer matrix and returns it as an intmat. For example, this function is useful if one wants to inspect the support hyperplanes. The filename is created from the current filename and the suffix given to the function.

```
> setNmzFilename("VeryInteresting");
> intmat sgr[3][3]=1,2,3, 4,5,6, 7,8,10;
> intmat sgrnormal=normaliz(sgr,0);
> readNmzData("sup");
1,-2,1,
-4,11,-6,
-2,-2,3
> readNmzData("typ");
0,0,1,
3,0,0,
2,1,0,
1,2,0,
0,3,0
```