THE LIBRARAY NORMALIZ.LIB VERSION 0.3

WINFRIED BRUNS

The library normaliz.lib provides an interface for the use of normaliz 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.

The library is loaded by

```
LIB "normaliz.lib";
```

(Of course it must be in a directory where Singular looks for libraries.)

The library creates some global variables holding path names, the value of options etc. Their names start with nmz_.

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

1. Paths and files

If normaliz is not in the search path for executables, then its path must be made known to Singular. Furthermore one can set the file name to be used for the exchange of data, set the path to the directory where normaliz and Singular exchange data, choose the executable (if enormalz 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.

The library defines the following functions for these purposes:

- set_nmz_exec_path(string s)

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

  ```
  > set_nmz_exec_path("d:/normaliz/bin"); // Windows
  > nmz_exec_path;  // the global variable holding the path name
  d:/normaliz/bin/  // the last / is added (if necessary)
  > set_nmz_exec_path("$HOME/normaliz/bin");  // Unix
  ```

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

- set_nmz_version(string s)

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

  ```
  > set_nmz_version("enormalz"); // choose enormalz
  > nmz_version;       // the global variable holding the version name
  ```

Date: 25 June 2006.

```
enormalz
> set_nmz_version("normaliz");  // back to normaliz
```

- `set_nmz_data_path(string s)`

  The function sets the directory for the exchange of data. By default it is the current directory (or the home directory of `Singular`). If this choice is o.k., nothing needs to be done.

```
> set_nmz_data_path("d:/normaliz/example/"); // Windows
> nmz_data_path;       // the global variable holding the path name
d:/normaliz/example/
set_nmz_data_path("$HOME/MyFiles/normaliz");  // Unix
```

- `write_nmz_paths()`

  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 written, but contains nothing.

```
> write_nmz_paths();
> int dummy=system("sh","cat nmz_sing_exec.path");
d:/normaliz/bin/
> dummy=system("sh","cat nmz_sing_data.path");
d:/normaliz/example/;
```

- `start_nmz()`

  This function reads the files written by `write_nmz_paths()`, 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.

```
> start_nmz();                              > write_nmz_paths();
nmz_exec_path is d:/normaliz/bin/           > start_nmz();
nmz_data_path is d:/normaliz/example/       nmz_exec_path is d:/normaliz/bin/
> set_nmz_data_path("");                    nmz_data_path not set
// ** redefining nmz_data_path
```

- `set_nmz_filename(string s)`

  The function sets the filename for the exchange of data. By default, the library creates a filename `nmz_sng_pid` where `pid` is the process identification of the current `Singular` process. If this choice is o.k., nothing needs to be done.

```
> set_nmz_filename("VeryInteresting");
> nmz_filename;       // the global variable holding the file name
VeryInteresting
> nmz_file;
d:/normaliz/example/VeryInteresting  // path + filename
                    // (not formed before normaliz is called)
```

- `rm_nmz_files()`

  This function removes the files created for and by `normaliz`, using the last filename created. These files are *not* removed automatically.

```
> rm_nmz_files();
rm: cannot remove ‘d:/normaliz/example/VeryInteresting.sup’: ... // not all the files
rm: cannot remove ‘d:/normaliz/example/VeryInteresting.val’: ... // are always created
```

2. Integral closures of monomial ideals and toric rings

There are 4 functions, corresponding to the modes 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 the volume option is set (see Section 3).

- normal_toric_ring(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!

```
> LIB "normaliz.lib";
// ** loaded normaliz.lib (1.58,2001/02/22)
// ** library normaliz.lib has old format. This format is still accepted,
// ** but for functionality you may wish to change to the new
// ** format. Please refer to the manual for further information.
> ring R=37,(x,y,t),dp;
> ideal I=x3,x2y,y3;
> set_nmz_exec_path("d:/normaliz/bin");
> normal_toric_ring(I);
x3 x2y y3 xy2
```

- intcl_toric_ring(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.

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

- ehrhart_ring(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.

```
> ehrhart_ring(I);
[1]:
   _[1]=x3
   _[2]=x2y
   _[3]=y3
   _[4]=xy2
[2]:
   _[1]=x3t
   _[2]=x2yt
   _[3]=y3t
   _[4]=xy2t
```

```
> ideal J=I,xy2t7;
> ehrhart_ring(J);
[1]:
   _[1]=x3
   _[2]=x2y
   _[3]=y3
   _[4]=xy2t7
   _[5]=x2yt
   _[6]=x2yt2
   _[7]=x2yt3
   _[8]=xy2t6
   _[9]=xy2t5
   _[10]=xy2t4
   _[11]=xy2t3
   _[12]=xy2t2
   _[13]=xy2t
   _[14]=xy2
```

- `intcl_mon_ideal(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.

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

3. SETTING OPTIONS

The library uses always the option `-f` for `normaliz`. The other options are set as follows:

- `set_hilb_option(int onoff)`

  Sets/resets the option for the computation of the Hilbert function. The default is `0=off`.

```
> set_hilb_option(1); // on
> nmz_hilb_switch;  // global variable holding the option
1
> set_hilb_option(0); // off again
```

- `set_vol_option(int onoff)`

  Sets/resets the option that keeps `normaliz` from the computation of generators. The default is `0=off`.

```
> set_vol_option(1); // on
> nmz_vol_switch;    // global variable holding the option
1
set_vol_option(0);   // off again
> nmz_vol_switch;
0
```

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

  These are completely analogous. The last function sets the `-a` option.

4. Retrieving numerical invariants

The following functions make the numerical invariants computed by `normaliz` accessible to `Singular` (as far as they are computed and available in the output file of `normaliz`). Note that some of the numerical invariants are also computed with the volume option. While the output file of `normaliz` interprets the numerical invariants according to the mode, such interpretation is not taken care of by the functions below.

- `get_numinvs()`

  The function returns a list whose length depends on the invariants available. The order of the elements in the list is always the same. Each list element has two parts. The first is a `string` describing the invariant, the second is the invariant, namely an `int` for rank, index, multiplicity, and an `intvec` for the weights, the h-vector and the Hilbert polynomial.

```
> set_hilb_option(1);
> list l= intcl_mon_ideal(I);
> get_numinvs();
[1]:                                [4]:
   [1]:                                [1]:
      rank of semigroup                   multiplicity
   [2]:                                [2]:
      3                                   4
[2]:                                [5]:
   [1]:                                [1]:
      index in integral closure           h-vector
   [2]:                                [2]:
      1                                   1,3,0
[3]:                                [6]:
   [1]:                                [1]:
      homogeneous w.r.t. weights          Hilbert poly (multiplied by (r-1)!)
   [2]:                                [2]:
      1,1,-2                              2,6,4
```

- `show_numinvs()`

  This function types the numerical invariants on the standard output, but returns nothing. (It calls `get_numinvs()`.)

```
> show_numinvs();
rank of semigroup : 3                multiplicity : 4
index in integral closure : 1        h-vector : 1,3,0
homogeneous w.r.t. weights : 1,1,-2  Hilbert poly (multiplied by (r-1)!) : 2,6,4
```

5. 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 Section 2 as well as the function `normaliz` below write and read their data automatically to and from the hard disk so that `write_nmz_data` will hardly ever be used explicitly.

- `write_nmz_data(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.

```
> intmat sgr[3][3]=1,2,3,4,5,6,7,8,10;
> write_nmz_data(sgr,1);
> nmz_file;
```

```
nmz_sing_1716  // neither data path nor filename explicitly set
> int dummy=system("sh","cat nmz_sing_1716.in");
```

```
3
3
1 2 3
4 5 6
7 8 10
1
```

- `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);
  1,2,3,
  4,5,6,
  7,8,10,
  3,4,5,
  2,3,4
  ```

- `read_nmz_data(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 in use and the suffix given to the function.

  ```
  > read_nmz_data("sup");
  1,-2,1,
  -4,11,-6,
  -2,-2,3
  ```

6. 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
  ```

7. Concluding remarks

I have tested the library under Solaris, Windows 2000 Pro, and Windows XP Pro. Under Windows XP Pro I get an error message of the kernel `‘‘Couldn’t duplicate my handle ...’’` or similar when the shell command running `normaliz` is issued. It seems to be irrelevant.