THE PACKAGE NORMALIZ FOR NORMALIZ 2.1

WINFRIED BRUNS, GESA KÄMPF

The package Normaliz provides an interface for the use of normaliz 2.1 within Macaulay 2. The structure is similar to the one of the corresponding Singular library Normaliz.lib. 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 Matrix. Therefore Macaulay 2 can be used as a comfortable environment for the work with normaliz.

The package is loaded by

```
i1 : loadPackage("Normaliz");
```

(Of course it must be in a directory where Macaulay 2 looks for packages.)

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

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) The path names for executables do not reflect the present normaliz distribution. They should be considered as examples.

1. PATHS AND FILES

If normaliz is not in the search path for executables, then its path must be made known to Macaulay 2. Furthermore one can set the file name to be used for the exchange of data, set the path to the directory where normaliz and Macaulay 2 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.

Note: The path names should not contain ~ or $, because Macaulay 2 seems to have problems with such paths.

The package defines the following functions for these purposes:

- setNmzExecPath(s)

  The function sets the path to the executable for normaliz. This is absolutely necessary if it is not in the search path. To retrieve the value stored in the global variable call getNmzExecPath().

  ```
  i2 : setNmzExecPath("d:/normaliz/bin"); -- Windows
  i3 : getNmzExecPath()  -- returns the global variable holding the path name
  o3 = d:/normaliz/bin/  -- the last / is added (if necessary)
  i2 : setNmzExecPath("../normaliz/bin");  -- Unix
  ```

  Please consult the installation instructions of Macaulay 2 for the conventions regarding path names under Windows.

Date: 29 January 2009.

- `setNmzVersion(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. To retrieve the value stored in the global variable call `getNmzVersion()`.

```
i4 : setNmzVersion("normbig"); -- choose normbig
i5 : getNmzVersion()       -- returns the global variable holding the version name
o5 = normbig
i6 : setNmzVersion("norm32");  -- now it is norm32
```

- `setNmzDataPath(s)`

  The function sets the directory for the exchange of data. By default it is the current directory (or the home directory of `Macaulay 2`). If this choice is o.k., nothing needs to be done. To retrieve the value stored in the global variable call `getNmzDataPath()`.

```
i7 : setNmzDataPath("d:/normaliz/example/"); -- Windows
i8 : getNmzDataPath()       -- returns the global variable holding the path name
o8 = d:/normaliz/example/
i7 : setNmzDataPath("../MyFiles/normaliz");  -- Unix
```

- `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 written, but contains nothing.

```
i9 : writeNmzPaths();
i10 : get("nmzM2Exec.path")
o10 = d:/normaliz/bin/
i11 : get("nmzM2Data.path")
o11 = d:/normaliz/example/;
```

```
i11 : get("nmzM2Data.path")
o11 = d:/normaliz/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.

```
i1 : startNmz();
nmzExecPath is d:/normaliz/bin/
nmzDataPath is d:/normaliz/example/
i2 : setNmzDataPath("");
```

```
i3 : writeNmzPaths();
i4 : startNmz();
nmzExecPath is d:/normaliz/bin/
nmzDataPath not set
```

- `setNmzFilename(s)`

  The function sets the filename for the exchange of data. By default, the package creates a filename `nmzM2_pid` where `pid` is the process identification of the current `Macaulay 2` process. If this choice is o.k., nothing needs to be done. Call `getNmzFilename()` to obtain the current filename.

```
i5 : setNmzFilename("VeryInteresting");
i6 : getNmzFilename()  -- the file name
o6 = VeryInteresting
```

```
i7 : setNmzFile()
o7 = d:/normaliz/
        example/VeryInteresting
```

- `rmNmzFiles()`

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

```
i8 : rmNmzFiles();
```

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).

- `normalToricRing(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!

```
i1 : loadPackage "Normaliz";
i2 : R=ZZ/37[x,y,t];
i3 : I=ideal(x^3,x^2*y,y^3);
o3 : Ideal of R
i4 : setNmzExecPath("d:/normaliz/bin");
i5 : normalToricRing(I)
             3   2    3     2
o5 = ideal (x , x y, y , x*y )
o5 : Ideal of R
```

- `intclToricRing(I)`

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

```
i6 : intclToricRing(I)
o6 = ideal (x, y)
o6 : Ideal of R
```

- `ehrhartRing(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 returns only one ideal, namely the monomials representing the lattice points of the polytope.

```
i7 : ehrhartRing(I)
               3   2    3     2           3    2      3      2
o7 = ideal (x , x y, y , x*y ), ideal (x t, x y*t, y t, x*y t)
o7 : List
i8 : J=I+ideal(x*y^2*t^7)
i9 : ehrhartRing(J)
              3   2    3     2 7   2      2   2   2   3     2     2      2 2
o9 = ideal (x , x y, y , x*y t , x y*t, x y*t , x y*t , x*y , x*y t, x*y t ,
     ------------------------------------------------------------------------
        2 3     2 4     2 5     2 6
     x*y t , x*y t , x*y t , x*y t )
```

- `intclMonIdeal(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, it returns only one ideal, namely the monomials generating the integral closure of the ideal.

```
i10 : intclMonIdeal(I)
               3   2    3     2                  3    2      3      2
o10 = ideal (x , x y, y , x*y ), ideal (x, y, x t, x y*t, y t, x*y t)
o10 : List
i11 : intclMonIdeal(J)
              3   2    3     2
o11 = ideal (x , x y, y , x*y )
o11 : Ideal of R
```

3. SETTING OPTIONS

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

- `setHilbOption(onoff)`

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

```
i1 : setHilbOption(true); -- on      o2 : true
i2 : hilbOption()                    i3 : setHilbOption(false); -- off again
```

- `setVolOption(onoff)`

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

```
i1 : setVolOption(true); -- on       o2 : true
i2 : VolOption()                     i3 : setVolOption(false); -- off again
```

- `setSOption(onoff)`
- `setNOption(onoff)`
- `setPOption(onoff)`
- `setIOption(onoff)`
- `setDOption(onoff)`
- `setCOption(onoff)`
- `setAllfOption(onoff)`

  These are completely analogous. The last function sets the `-a` option. Note that it does not make sense to turn on more than one of the options `s,v,n,p,h,d`.

4. RETRIEVING NUMERICAL INVARIANTS

The following functions make the numerical invariants computed by normaliz accessible to Macaulay 2 (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.

- getNumInvs()

  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 Integer for rank, index, multiplicity, a Sequence for the weights, the h-vector and the Hilbert polynomial and a Boolean for homogeneity and primary (to the maximal ideal).

```
i1 : setHilbOption(true);
i2 : intclMonIdeal(I);
i3 : getNumInvs()
o3 = {{hilbert basis elements, 6}, {number extreme rays, 4}, {rank, 3}, {index,
     ------------------------------------------------------------------------
     1}, {number support hyperplanes, 4}, {homogeneous, true}, {height 1
     ------------------------------------------------------------------------
     elements, 6}, {homogeneous weights, (1, 1, -2)}, {multiplicity, 4},
     ------------------------------------------------------------------------
     {h-vector, (1, 3, 0)}, {hilbert polynomial, (2, 6, 4)}, {primary, true},
     ------------------------------------------------------------------------
     {ideal multiplicity, 9}}
o3 : List
```

  Note: Only the data computed by normaliz are read. There are no "blank" entries in the result of getNumInvs().

- showNumInvs()

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

```
i4 : showNumInvs()                  height 1 elements : 6
hilbert basis elements : 6          homogeneous weights : (1,1,-2)
number extreme rays : 4             multiplicity : 4
rank : 3                            h-vector : (1,3,0)
index : 1                           hilbert polynomial : (2,6,4)
number support hyperplanes : 4      primary : true
homogeneous : true                  ideal multiplicity : 9
```

- exportNumInvs()

  This function exports the data read by getNumInvs() into numerical Macaulay 2 data that can be accessed directly. For each invariant a variable of type Integer, Sequence or Boolean is created whose name is the first entry of each list element shown above, prefixed by nmz. If the Print option is set to true, the variables are created and printed to the standard output. The default value of the Print option is false.

```
i5 : exportNumInvs()                nmzNumberSupportHyperplanes=4
i6 : nmzHilbertBasisElements        nmzHomogeneous=true
o6 = 6                              nmzHeight1Elements=6
i7 : nmzHomogeneousWeights          nmzHomogeneousWeights=(1,1,-2)
```

```
o7 = (1, 1, -2)                    nmzMultiplicity=4
o7 : Sequence                      nmzHVector=(1,3,0)
i8: exportNumInvs(Print=>true)     nmzHilbertPolynomial=(2,6,4)
nmzHilbertBasisElements=6          nmzPrimary=true
nmzNumberExtremeRays=4             nmzIdealMultiplicity=9
nmzRank=3                          i9 : nmzHilbertBasisElements
nmzIndex=1                         o9 = 6
```

5. RUNNING normaliz ON DATA OF TYPE Matrix

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 writeNmzData will hardly ever be used explicitly.

- writeNmzData(sgr,nmzMode)

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

```
i1 : sgr=matrix(1,2,3,4,5,6,7,8,10);
i2 : writeNmzData(sgr,1);
i3 : get(setNmzFile()|".in")
o3 = 3
     3
     1 2 3
     4 5 6
     7 8 10
     1
```

- normaliz(sgr, nmzMode)

  The function applies normaliz to the parameter sgr in the mode set by nmzMode. The function returns the Matrix defined by the file with suffix gen.

```
i4 : normaliz(sgr,0)
o4 = | 7 8 10 |
     | 1 2 3  |
     | 2 3 4  |
     | 3 4 5  |
     | 4 5 6  |
               5        3
o4 : Matrix ZZ  <--- ZZ
```

- readNmzData(suffix)

  Reads an output file of normaliz containing an integer matrix and returns it as a Matrix. 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.

```
i5 : readNmzData("sup")
o5 = | -2 -2 3  |
     | -4 11 -6 |
     | 1  -2 1  |
               3        3
o5 : Matrix ZZ  <--- ZZ
```

6. MONOMIALS TO/FROM Matrix

The transformation of data between an Ideal and a Matrix is carried out by the following functions:

- mons2intmat(I)

  Returns the Matrix whose rows represent the leading exponents of the elements of I. The length of each row is the numbers of variables of the ring of I.

```
i1 : m=mons2intmat(J)
o1 = | 3 0 0 |
     | 2 1 0 |
     | 0 3 0 |
     | 1 2 7 |
               4        3
o1 : Matrix ZZ  <--- ZZ
```

- intmat2mons(m,R)

  The converse operation. The ring whose elements the monomials shall be has to be specified by R.

```
i2 : intmat2mons(m,R)
                3   2    3     2 7
o2 = ideal (x , x y, y , x*y t )
o2 : Ideal of R
```