Formatted manual of normaliz.lib

1 Singular libraries

1.1 normaliz_lib

```
-------BEGIN OF PART WHICH IS INCLUDED IN MANUAL-----
```

Library: normaliz.lib

Purpose: Provides an interface for the use of Normaliz 3.9.0 or newer within SINGULAR.

Authors: Winfried Bruns, Winfried.Bruns@Uni-Osnabrueck.de
Christof Soeger, Christof.Soeger@Uni-Osnabrueck.de

Overview: The library normaliz.lib provides an interface for the use of Normaliz 3.9.0 or newer within SINGULAR. The exchange of data is via files. 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. To some extent, SINGULAR can therefore be used as an environment for interactive access to Normaliz.
Please see the `Normaliz.pdf` (included in the Normaliz distribution) for a more extensive documentation of Normaliz.

Normaliz allows the use of a grading. In the Singular functions that access Normaliz the parameter grading is an intvec that assigns a (not necessarily positive) degree to every variable of the ambient polynomial ring. But it must give positive degrees to the generators given to the function.

Singular and Normaliz exchange data via files. These files are automatically created and erased behind the scenes. As long as one wants to use only the ring-theoretic functions there is no need for file management.
Note that the numerical invariants computed by Normaliz can be accessed in this "automatic file mode".
However, if Singular is used as a frontend for Normaliz or the user wants to inspect data not automatically returned to Singular, then an explicit filename and a path can be specified for the exchange of data. Moreover, the library provides functions for access to these files. Deletion of the files is left to the user. (Not all output files of Normaliz can be read by this library.)

Use of this library requires the program Normaliz to be installed. You can download it from `https://github.com/Normaliz/Normaliz/releases`. Please make sure that the executable is in the search path or use setNmzExecPath (Section 1.1 [setNmzExecPath], page 11).

Procedures:

1.1.0.1 intclToricRing

Procedure from library `normaliz.lib` (see Section 1.1 [normaliz_lib], page 1).

Usage: intclToricRing(ideal I);
intclToricRing(ideal I, intvec grading);

Return: The toric ring S is the subalgebra of the basering generated by the leading monomials of the elements of I (considered as a list of polynomials). The

function computes the integral
closure T of S in the basering and returns an ideal listing the algebra generators of T over the coefficient field.
The function returns the input ideal I if an option blocking the computation of Hilbert bases 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 1.1 [showNuminvs], page 8, Section 1.1 [exportNuminvs], page 8).

Note: A mathematical remark: the toric ring depends on the list of monomials given, and not only on the ideal they generate!

Example:

```
↦ LIB "normaliz.lib";
↦ ring R=37,(x,y,t),dp;
↦ ideal I=x3,x2y,y3;
↦ intclToricRing(I);
↦ showNuminvs();
↦ //now the same example with another grading
↦ intvec grading = 2,3,1;
↦ intclToricRing(I,grading);
↦ showNuminvs();
```

See also: Section 1.1 [ehrhartRing], page 3; Section 1.1 [intclMonIdeal], page 4; Section 1.1 [normalToricRing], page 2.

1.1.0.2 normalToricRing

Procedure from library `normaliz.lib` (see Section 1.1 [normaliz_lib], page 1).

Usage: normalToricRing(ideal I);
normalToricRing(ideal I, intvec grading);

Return: The toric ring S is the subalgebra of the basering generated by the leading monomials of the elements of I (considered as a list of polynomials). The function computes the
normalisation T of S and returns an ideal listing the algebra generators of T over the coefficient field.
The function returns the input ideal I if one of the options blocking the computation of Hilbert bases 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 1.1 [showNuminvs], page 8, Section 1.1 [exportNuminvs], page 8).

Note: A mathematical remark: the toric ring depends on the list of monomials given, and not only on the ideal they generate!

Example:

```
↦ LIB "normaliz.lib";
↦ ring  R = 37,(x,y,t),dp;
↦ ideal I = x3,x2y,y3;
↦ normalToricRing(I);
```

See also: Section 1.1 [ehrhartRing], page 3; Section 1.1 [intclMonIdeal], page 4; Section 1.1 [intclToricRing], page 1; Section 1.1 [normalToricRingFromBinomials], page 3.

1.1.0.3 normalToricRingFromBinomials

Procedure from library `normaliz.lib` (see Section 1.1 [normaliz_lib], page 1).

Usage: normalToricRingFromBinomials(ideal I);
normalToricRingFromBinomials(ideal I, intvec grading);

Return: The ideal I is generated by binomials of type $X^a - X^b$ (multiindex notation) in the surrounding polynomial ring $K[X] = K[X_1, ..., X_n]$. The binomials represent a congruence on the monoid Z^n with residue monoid M. Let N be the image of M in gp(M)/torsion. Then N is universal in the sense that every homomorphism from M to an affine monoid factors through N. If I is a prime ideal, then $K[N] = K[X]/I$. In general, $K[N] = K[X]/P$ where P is the unique minimal prime ideal of I generated by binomials of type $X^a - X^b$.

The function computes the normalization of $K[N]$ and returns a newly created polynomial ring of the same Krull dimension, whose variables are $x(1), ..., x(n-r)$, where r is the rank of the matrix with rows $a - b$. (In general there is no canonical choice for such an embedding.) Inside this polynomial ring there is an ideal I which lists the algebra generators of the normalization of $K[N]$.

The function returns the input ideal I if an option blocking the computation of Hilbert bases 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 1.1 [showNuminvs], page 8, Section 1.1 [exportNuminvs], page 8).

Example:

```
↦ LIB "normaliz.lib";
↦ ring R = 37,(u,v,w,x,y,z),dp;
↦ ideal I = u2v-xyz, ux2-wyz, uvw-y2z;
↦ def S = normalToricRingFromBinomials(I);
↦ setring S;
↦ I;
```

See also: Section 1.1 [ehrhartRing], page 3; Section 1.1 [intclMonIdeal], page 4; Section 1.1 [intclToricRing], page 1; Section 1.1 [normalToricRing], page 2.

1.1.0.4 ehrhartRing

Procedure from library `normaliz.lib` (see Section 1.1 [normaliz_lib], page 1).

Usage: ehrhartRing(ideal I);

Return: The exponent vectors of the leading monomials of the elements of I are considered as points of a lattice polytope P.
The Ehrhart ring of a (lattice) polytope P is the monoid algebra defined by the monoid of lattice points in the cone over the polytope P; see Bruns and Gubeladze, Polytopes, Rings, and K-theory, Springer 2009, pp. 228, 229.
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 all the lattice points of the polytope, the

second containing the algebra generators of the Ehrhart ring over the coefficient field.
(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.

The function returns the a list containing the input ideal I if an option blocking the computation of the Hilbert basis 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 1.1 [showNuminvs], page 8, Section 1.1 [exportNuminvs], page 8).

Note: A mathematical remark: the Ehrhart ring depends on the list of monomials given, and not only on the ideal they generate!

Example:

```
↦ LIB "normaliz.lib";
↦ ring R=37,(x,y,t),dp;
↦ ideal J=x3,x2y,y3,xy2t7;
↦ ehrhartRing(J);
```

See also: Section 1.1 [intclMonIdeal], page 4; Section 1.1 [intclToricRing], page 1; Section 1.1 [normalToricRing], page 2.

1.1.0.5 intclMonIdeal

Procedure from library `normaliz.lib` (see Section 1.1 [normaliz_lib], page 1).

Usage: intclMonIdeal(ideal I);
intclMonIdeal(ideal I, intvec grading);

Return: The exponent vectors of the leading monomials of the elements of I are considered as generators of a monomial ideal for which the normalization of its Rees algebra is computed. For a Definition of the Rees algebra (or Rees ring) see Bruns and Herzog, Cohen-Macaulay rings, Cambridge University Press 1998, p. 182.
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 algebra generators of the normalization 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.
The function returns the a list containing the input ideal I if an option blocking the computation of Hilbert bases 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 1.1 [showNuminvs], page 8, Section 1.1 [exportNuminvs], page 8).

Example:

```
↦ LIB "normaliz.lib";
↦ ring R=0,(x,y,z,t),dp;
↦ ideal I=x^2,y^2,z^3;
↦ list l=intclMonIdeal(I);
↦ l[1]; // integral closure of I
↦ l[2];  // monomials generating the integral closure of the Rees algebra
```

See also: Section 1.1 [ehrhartRing], page 3; Section 1.1 [intclToricRing], page 1; Section 1.1 [normalToricRing], page 2.

1.1.0.6 torusInvariants

Procedure from library `normaliz.lib` (see Section 1.1 [normaliz_lib], page 1).

Usage: torusInvariants(intmat A);
torusInvariants(intmat A, intvec grading);

Return: Returns an ideal representing the list of monomials generating the ring of invariants as an algebra over the coefficient field. R^T.
The function returns the ideal given by the input matrix A if one of the options `supp`, `triang`, `volume`, or `hseries` 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 1.1 [showNuminvs], page 8, Section 1.1 [exportNuminvs], page 8).

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

$$X_j \mapsto \lambda_1^{a_{1,j}} \cdots \lambda_r^{a_{r,j}} X_j, \quad j = 1, \ldots, n.$$

In order to compute the ring of invariants R^T one must specify the matrix $A = (a_{i,j})$.

Example:

```
↦ LIB "normaliz.lib";
↦ ring R=0,(x,y,z,w),dp;
↦ intmat E[2][4] = -1,-1,2,0, 1,1,-2,-1;
↦ torusInvariants(E);
```

See also: Section 1.1 [diagInvariants], page 6; Section 1.1 [finiteDiagInvariants], page 5; Section 1.1 [intersectionValRingIdeals], page 7; Section 1.1 [intersectionValRings], page 7.

1.1.0.7 finiteDiagInvariants

Procedure from library `normaliz.lib` (see Section 1.1 [normaliz_lib], page 1).

Usage: finiteDiagInvariants(intmat U);
finiteDiagInvariants(intmat U, intvec grading);

Return: This function computes the ring of invariants of a finite abelian group G acting diagonally on the surrounding polynomial ring $K[X_1, ..., X_n]$. The group is the direct product of cyclic groups generated by finitely many elements $g_1, ..., g_w$.

The element g_i acts on the indeterminate X_j by $g_i(X_j) = \lambda_i^{u_{ij}} X_j$ where λ_i is a primitive root of unity of order equal to $ord(g_i)$. The ring of invariants is generated by all monomials satisfying the system $u_{i1}a_1 + \ldots + u_{in}a_n \equiv 0 \bmod \mathrm{ord}(g_i)$, $i = 1, \ldots, w$. The input to the function is the $w \times (n+1)$ matrix U with rows $u_{i1} \ldots u_{in}$ ord(gi), $i = 1, \ldots, w$. The output is a monomial ideal listing the algebra generators of the subalgebra of invariants $R^G = \{f \in R : g_i f = f$ for all $i = 1, \ldots, w\}$.

The function returns the ideal given by the input matrix C if one of the options `supp`, `triang`, `volume`, or `hseries` 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 1.1 [showNuminvs], page 8, Section 1.1 [exportNuminvs], page 8).

Note:

Example:

```
↦ LIB "normaliz.lib";
↦ ring R = 0,(x,y,z,w),dp;
↦ intmat C[2][5] = 1,1,1,1,5, 1,0,2,0,7;
↦ finiteDiagInvariants(C);
```

See also: Section 1.1 [diagInvariants], page 6; Section 1.1 [intersectionValRingIdeals], page 7; Section 1.1 [intersectionValRings], page 7; Section 1.1 [torusInvariants], page 5.

1.1.0.8 diagInvariants

Procedure from library `normaliz.lib` (see Section 1.1 [normaliz_lib], page 1).

Usage: diagInvariants(intmat A, intmat U);
diagInvariants(intmat A, intmat U, intvec grading);

Return: This function computes the ring of invariants of a diagonalizable group $D = T \times G$ where T is a torus and G is a finite abelian group, both acting diagonally on the polynomial ring $K[X_1, \ldots, X_n]$. The group actions are specified by the input matrices A and U. The first matrix specifies the torus action, the second the action of the finite group. See torusInvariants and finiteDiagInvariants for more detail. The output is a monomial ideal listing the algebra generators of the subalgebra of invariants.

The function returns the ideal given by the input matrix A if one of the options `supp`, `triang`, `volume`, or `hseries` 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 1.1 [showNuminvs], page 8, Section 1.1 [exportNuminvs], page 8).

Example:

```
↦ LIB "normaliz.lib";
↦ ring R=0,(x,y,z,w),dp;
↦ intmat E[2][4] = -1,-1,2,0, 1,1,-2,-1;
↦ intmat C[2][5] = 1,1,1,1,5, 1,0,2,0,7;
↦ diagInvariants(E,C);
```

See also: Section 1.1 [finiteDiagInvariants], page 5; Section 1.1 [intersectionValRingIdeals], page 7; Section 1.1 [intersectionValRings], page 7; Section 1.1 [torusInvariants], page 5.

1.1.0.9 intersectionValRings

Procedure from library `normaliz.lib` (see Section 1.1 [normaliz_lib], page 1).

Usage: intersectionValRings(intmat V, intvec grading);

Return: The function returns a monomial ideal, to be considered as the list of monomials generating S as an algebra over the coefficient field.

Background:

A discrete monomial valuation v on $R = K[X_1, \ldots, X_n]$ is determined by the values $v(X_j)$ of the indeterminates. This function computes the subalgebra $S = \{f \in R : v_i(f) \geq 0,\ i = 1, \ldots, r\}$ for several such valuations v_i, $i = 1, \ldots, r$. It needs the matrix $V = (v_i(X_j))$ as its input.

The function returns the ideal given by the input matrix V if one of the options `supp`, `triang`, `volume`, or `hseries` 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 1.1 [showNuminvs], page 8, Section 1.1 [exportNuminvs], page 8).

Example:

```
↦ LIB "normaliz.lib";
↦ ring R=0,(x,y,z,w),dp;
↦ intmat V0[2][4]=0,1,2,3, -1,1,2,1;
↦ intersectionValRings(V0);
```

See also: Section 1.1 [diagInvariants], page 6; Section 1.1 [finiteDiagInvariants], page 5; Section 1.1 [intersectionValRingIdeals], page 7; Section 1.1 [torusInvariants], page 5.

1.1.0.10 intersectionValRingIdeals

Procedure from library `normaliz.lib` (see Section 1.1 [normaliz_lib], page 1).

Usage: intersectionValRingIdeals(intmat V);
intersectionValRingIdeals(intmat V, intvec grading);

Return: 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.
The function returns a list consisting of the ideal given by the blocking the computation of Hilbert bases 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 1.1 [showNuminvs], page 8, Section 1.1 [exportNuminvs], page 8).

Background:

A discrete monomial valuation v on $R = K[X_1, \ldots, X_n]$ is determined by the values $v(X_j)$ of the indeterminates. This function computes the subalgebra $S = \{f \in R : v_i(f) \geq 0,\ i = 1, \ldots, r\}$ for several such valuations v_i, $i = 1, \ldots, r$. It needs the matrix $V = (v_i(X_j))$ as its input.

This function simultaneously determines the S-submodule $M = \{f \in R : v_i(f) \geq w_i,\ i = 1, \ldots, r\}$ for integers $w_1, \ldots w_r$. (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.

Note: The function also gives an error message if the matrix V has the wrong number of columns.

Example:

```
↦ LIB "normaliz.lib";
↦ ring R=0,(x,y,z,w),dp;
↦ intmat V[2][5]=0,1,2,3,4, -1,1,2,1,3;
↦ intersectionValRingIdeals(V);
```

See also: Section 1.1 [diagInvariants], page 6; Section 1.1 [finiteDiagInvariants], page 5; Section 1.1 [intersectionValRings], page 7; Section 1.1 [torusInvariants], page 5.

1.1.0.11 showNuminvs

Procedure from library `normaliz.lib` (see Section 1.1 [normaliz_lib], page 1).

Usage: showNuminvs();

Purpose: prints the numerical invariants

Example:

```
↦ LIB "normaliz.lib";
↦ ring R=0,(x,y,z,t),dp;
↦ ideal I=x3,x2y,y3;
↦ list l=intclMonIdeal(I);
↦ showNuminvs();
```

See also: Section 1.1 [exportNuminvs], page 8.

1.1.0.12 exportNuminvs

Procedure from library `normaliz.lib` (see Section 1.1 [normaliz_lib], page 1).

Usage: exportNuminvs();

Create: Creates top-level variables which contain the numerical invariants. Depending on the options of normaliz different invariants are calculated. Use showNuminvs (Section 1.1 [showNuminvs], page 8) to see which invariants are available.

Example:

```
↦ LIB "normaliz.lib";
↦ ring R=0,(x,y,z,t),dp;
↦ ideal I=x3,x2y,y3;
↦ list l=intclMonIdeal(I);
↦ exportNuminvs();
↦ // for example, now the following variables are set:
↦ nmz_hilbert_basis_elements;
↦ nmz_number_extreme_rays;
↦ nmz_rank;
↦ nmz_number_support_hyperplanes;
↦ nmz_multiplicity;
↦ nmz_primary;
```

See also: Section 1.1 [showNuminvs], page 8.

1.1.0.13 allNmzOptions

Procedure from library `normaliz.lib` (see Section 1.1 [normaliz_lib], page 1).

Usage: proc alNmzOptions();

Purpose: The function prints a list of the Normaliz otions that are available in this library: the string naming the option, the default value, and the option sent to Normaliz.

Example:

```
↦ LIB "normaliz.lib";
↦ allNmzOptions();
```

See also: ⟨undefined⟩ [listNmzOptions], page ⟨undefined⟩; Section 1.1 [setNmzOption], page 9.

1.1.0.14 setNmzOption

Procedure from library `normaliz.lib` (see Section 1.1 [normaliz_lib], page 1).

Usage: setNmzOption(string s, int onoff);

Purpose: If `onoff=1` the option `s` is activated, and if `onoff=0` it is deactivated. The predefined Normaliz options are accessible via the following names:

`-s: supp`
`-t: triang`
`-v: volume`
`-p: hvect` deprecated, replacement:
`-p: hvect_deg1`
`-q: onlyhvect`
`--FVector: fvect`
`-1: height1`
`-G: Gorenst`
`-w: intclosed`
`--WitnessNotIntegrallyClosed: witness`
`-C: classgroup`
`-n: normal` deprecated, replacement:
`-n: hilbbasvol`
`-N: normal_1` deprecated, replacement:
`-N: hilbbas`
`-h: hilb` deprecated, replacement:
`-h: hilbbas_hvect`
`-d: dual`
`-M: genoverori`
`-a: allf`
`--typ: type`
`-c: control` deprecated, replacement:
`-c: verbose`
`-e: errorcheck` allowed, but ignored
`-B: bigint` Use GMP for arbitrary precision integers
`-x=N: threads` In this case the int parameter is used to set the number of threads N, deafult 8, 0 means no explicit limiting.

Further Normaliz otions can be adeded to tzhe nlist by addNmzOption.I

Example:

```
↦ LIB "normaliz.lib";
↦ setNmzOption("only_hvect",1);
↦ showNmzOptions();
```

See also: Section 1.1 [addNmzOption], page 10; Section 1.1 [allNmzOptions], page 9; Section 1.1 [resetNmzOptions], page 10; Section 1.1 [showNmzOptions], page 10.

1.1.0.15 addNmzOption

Procedure from library `normaliz.lib` (see Section 1.1 [normaliz_lib], page 1).

Usage: addNmzOption(string short_cut, string nmz_option)

Note: This function allows the addition of Normaliz options
beyond the predefined ones.After addition the option is activated can be (de/re)activated via setNmzOption.
Note: The function prefixes a single letter option by - and multiletter options by –.

Example:

```
↦ LIB "normaliz.lib";
↦ addNmzOption("FL", "FaceLattice");
↦ showNmzOptions();
↦ addNmzOption("SC", "y");
```

See also: Section 1.1 [allNmzOptions], page 9; Section 1.1 [resetNmzOptions], page 10; Section 1.1 [setNmzOption], page 9; Section 1.1 [showNmzOptions], page 10.

1.1.0.16 showNmzOptions

Procedure from library `normaliz.lib` (see Section 1.1 [normaliz_lib], page 1).

Usage: showNmzOptions();

Return: Returns the string of activated options.

Note: This string is used as parameter when calling Normaliz.

Example:

```
↦ LIB "normaliz.lib";
↦ setNmzOption("hilb",1);
↦ showNmzOptions();
```

See also: Section 1.1 [setNmzOption], page 9.

1.1.0.17 resetNmzOptions

Procedure from library `normaliz.lib` (see Section 1.1 [normaliz_lib], page 1).

Usage: setNmzOption(string s, int onoff);

Purpose: Resets the options to the dafault value.

Example:

```
↦ LIB "normaliz.lib";
↦ setNmzOption("only_hvect",1);
↦ showNmzOptions();
↦ resetNmzOptions();
↦ showNmzOptions();
```

See also: Section 1.1 [addNmzOption], page 10; Section 1.1 [allNmzOptions], page 9; Section 1.1 [setNmzOption], page 9; Section 1.1 [showNmzOptions], page 10.

1.1.0.18 normaliz

Procedure from library `normaliz.lib` (see Section 1.1 [normaliz_lib], page 1).

Usage: normaliz(intmat sgr,string nmz_type);
normaliz(intmat sgr, string nmz_type, intmat sgr2, string nmz_type2, ...);

Return: The function applies Normaliz to the parameter sgr in the type set by nmz_type. The function returns the intmat defined by the file with suffix gen.

It is also possible to give more than one pair of matrix and type. In this case all matrices and types are used.

Note: You will find procedures for many applications of Normaliz in this library, so the explicit call of this procedure may not be necessary.

Example:

```
↦ LIB "normaliz.lib";
↦ ring R=0,(x,y,z),dp;
↦ intmat M[3][2]=3,1,
↦ 3,2,
↦ 1,3;
↦ normaliz(M,1);
↦ intmat Inequalities[2][3] = 2,-1,0, // 2x-y >= 0
↦ 1, 1,0; //  x+y >= 0
↦ intmat Equation[1][3] = 0,1,-1;    // y = z
↦ intmat Congruence[1][4] = 1,0,0,3;  // x = 0 (3)
↦ normaliz(Inequalities,"inequalities",Equation,"equations",Congruence,"con\
  gruences");
```

See also: Section 1.1 [diagInvariants], page 6; Section 1.1 [ehrhartRing], page 3; Section 1.1 [finiteDiagInvariants], page 5; Section 1.1 [intclMonIdeal], page 4; Section 1.1 [intclToricRing], page 1; Section 1.1 [intersectionValRingIdeals], page 7; Section 1.1 [intersectionValRings], page 7; Section 1.1 [normalToricRing], page 2; Section 1.1 [torusInvariants], page 5.

1.1.0.19 setNmzExecPath

Procedure from library `normaliz.lib` (see Section 1.1 [normaliz_lib], page 1).

Usage: setNmzExecPath(string s); `s` path to the Normaliz executable

Create: `Normaliz::nmz_exec_path` to save the given path `s`

Note: It is not necessary to use this function if the Normaliz executable is in the search path of the system.

Example:

```
↦ LIB "normaliz.lib";
↦ setNmzExecPath("../Normaliz/");
```

See also: Section 1.1 [setNmzOption], page 9.

1.1.0.20 writeNmzData

Procedure from library `normaliz.lib` (see Section 1.1 [normaliz_lib], page 1).

Usage: writeNmzData(intmat M, string nmz_type);
writeNmzData(intmat M, string nmz_type, intmat M2, string nmz_type2, ...);

Create: Creates an input file for Normaliz from the matrix M. The second parameter sets the type. How the matrix is interpreted depends on the type. See the Normaliz documentation for more information.

It is also possible to give more than one pair of matrix and type. In

Note: Needs an explicit filename set. The filename is created from the current filename.
Note that all high level functions in normaliz.lib write and read their data automatically to and from the hard disk so that writeNmzData will hardly ever be used explicitly.

Example:

```
↦ LIB "normaliz.lib";
↦ setNmzFilename("VeryInteresting");
↦ intmat sgr[3][3]=1,2,3,4,5,6,7,8,10;
↦ writeNmzData(sgr,"cone_and_lattice");
↦ int dummy=system("sh","cat VeryInteresting.in");
↦ intmat Inequalities[2][3] = 2,-1,0, // 2x-y >= 0
↦ 1, 1,0; //  x+y >= 0
↦ intmat Equation[1][3] = 0,1,-1;    // y = z
↦ intmat Congruence[1][4] = 1,0,0,3;  // x = 0 (3)
↦ writeNmzData(Inequalities,"inequalities",Equation,"equations",Congruence,\
  "congruences");
↦ dummy=system("sh","cat VeryInteresting.in");
```

See also: Section 1.1 [readNmzData], page 12; Section 1.1 [rmNmzFiles], page 14; Section 1.1 [setNmzDataPath], page 13; Section 1.1 [setNmzFilename], page 13.

1.1.0.21 readNmzData

Procedure from library `normaliz.lib` (see Section 1.1 [normaliz_lib], page 1).

Usage: readNmzData(string suffix);

Return: 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. In addition to file suffixes, also sup, equ and cgr are allowed. They extract the support hyperplanes, equations and congruences, respectively, from the cst file.

Note: Needs an explicit filename set by setNmzFilename.
Note that all functions in normaliz.lib write and read their data automatically so that readNmzData will usually not be used explicitly.
This function reads only the first matrix in a file! But see su, equ, cgr above.)
It is the responsability of the user to make sure that an output file read by this function has been created for the current input file of Normaliz. Files are not automatically removed before a new computation starts.
Not every output file can be read by this function.

Example:

```
↦ LIB "normaliz.lib";
↦ setNmzFilename("VeryInteresting");
↦ intmat sgr[3][3]=1,2,3,4,5,6,7,8,9;
↦ intmat sgrnormal=normaliz(sgr,"cone");
↦ readNmzData("sup");
↦ readNmzData("equ");
```

See also: Section 1.1 [rmNmzFiles], page 14; Section 1.1 [setNmzDataPath], page 13; Section 1.1 [setNmzFilename], page 13; Section 1.1 [writeNmzData], page 12.

1.1.0.22 setNmzFilename

Procedure from library `normaliz.lib` (see Section 1.1 [normaliz_lib], page 1).

Usage: setNmzFilename(string s);

Create: `Normaliz::nmz_filename` to save the given filename `s`

Note: The function sets the filename for the exchange of data. Unless a path is set by setNmzDataPath, files will be created in the current directory.
If a non-empty filename is set, the files created for and by Normaliz are kept. This is mandatory for the data access functions (see Section 1.1 [writeNmzData], page 12 and Section 1.1 [readNmzData], page 12).
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.

Example:

```
↦ LIB "normaliz.lib";
↦ setNmzDataPath("examples/");
↦ setNmzFilename("example1");
↦ //now the files for the exchange with Normaliz are examples/example1.SUFF\
  IX
```

See also: Section 1.1 [readNmzData], page 12; Section 1.1 [rmNmzFiles], page 14; Section 1.1 [setNmzDataPath], page 13; Section 1.1 [writeNmzData], page 12.

1.1.0.23 setNmzDataPath

Procedure from library `normaliz.lib` (see Section 1.1 [normaliz_lib], page 1).

Usage: setNmzDataPath(string s);

Create: `Normaliz::nmz_data_path` to save the given path `s`

Note: The function sets the path for the exchange of data. By default the files will be created in the current directory.
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.

Example:

```
↦ LIB "normaliz.lib";
↦ setNmzDataPath("examples/");
↦ setNmzFilename("example1");
```

```
↦ //now the files for the exchange with Normalize are examples/example1.SUF\
  FIX
```

See also: Section 1.1 [readNmzData], page 12; Section 1.1 [rmNmzFiles], page 14; Section 1.1 [setNmzFilename], page 13; Section 1.1 [writeNmzData], page 12.

1.1.0.24 writeNmzPaths

Procedure from library `normaliz.lib` (see Section 1.1 [normaliz_lib], page 1).

Create: the file nmz_sing_exec.path where the path to the Normaliz executable is saved
the file nmz_sing_data.path where the directory for the exchange of data is saved

Note: Both files are saved in the current directory. If one of the names has not been defined, the corresponding file is created, but contains nothing.

Example:

```
↦ LIB "normaliz.lib";
↦ setNmzExecPath("../Normaliz/");
↦ writeNmzPaths();
↦ int dummy=system("sh","cat nmz_sing_exec.path");
↦ dummy=system("sh","cat nmz_sing_data.path");
```

See also: Section 1.1 [setNmzDataPath], page 13; Section 1.1 [setNmzExecPath], page 11; Section 1.1 [startNmz], page 14.

1.1.0.25 startNmz

Procedure from library `normaliz.lib` (see Section 1.1 [normaliz_lib], page 1).

Usage: startNmz();

Purpose: 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.

Example:

```
↦ LIB "normaliz.lib";
↦ writeNmzPaths();
↦ startNmz();
```

See also: Section 1.1 [setNmzDataPath], page 13; Section 1.1 [setNmzExecPath], page 11; Section 1.1 [writeNmzPaths], page 14.

1.1.0.26 rmNmzFiles

Procedure from library `normaliz.lib` (see Section 1.1 [normaliz_lib], page 1).

Usage: rmNmzFiles();

Purpose: This function removes the files created for and by Normaliz, using the last filename specified. It needs an explicit filename set (see Section 1.1 [setNmz-Filename], page 13).

Example:

```
↦ LIB "normaliz.lib";
↦ setNmzFilename("VeryInteresting");
↦ rmNmzFiles();
```

See also: Section 1.1 [readNmzData], page 12; Section 1.1 [setNmzDataPath], page 13; Section 1.1 [setNmzFilename], page 13; Section 1.1 [writeNmzData], page 12.

1.1.0.27 mons2intmat

Procedure from library `normaliz.lib` (see Section 1.1 [normaliz_lib], page 1).

Usage: mons2intmat(ideal I);

Return: Returns the intmat whose rows represent the leading exponents of the (non-zero) elements of I. The length of each row is nvars(basering).

Example:

```
↦ LIB "normaliz.lib";
↦ ring R=0,(x,y,z),dp;
↦ ideal I=x2,y2,x2yz3;
↦ mons2intmat(I);
```

See also: Section 1.1 [intmat2mons], page 15.

1.1.0.28 intmat2mons

Procedure from library `normaliz.lib` (see Section 1.1 [normaliz_lib], page 1).

Usage: intmat2mons(intmat M);

Return: an ideal generated by the monomials which correspond to the exponent vectors given by the rows of `M`

Note: The number of variables in the basering `nvars(basering)` has to be at least the number of columns `ncols(M)`, otherwise the function exits with an error. is thrown (see ⟨undefined⟩ [ERROR], page ⟨undefined⟩).

Example:

```
↦ LIB "normaliz.lib";
↦ ring R=0,(x,y,z),dp;
↦ intmat expo_vecs[3][3] =
↦ 2,0,0,
↦ 0,2,0,
↦ 2,1,3;
↦ intmat2mons(expo_vecs);
```

See also: Section 1.1 [mons2intmat], page 15.

1.1.0.29 binomials2intmat

Procedure from library `normaliz.lib` (see Section 1.1 [normaliz_lib], page 1).

Usage: binomials2intmat(ideal I);

Return: Returns the intmat whose rows represent the exponents of the (non-zero) elements of I which have to be binomials.
The length of each row is nvars(basering).

Example:

```
↦ LIB "normaliz.lib";
↦ ring S = 37,(u,v,w,x,y,z),dp;
↦ ideal I = u2v-xyz, ux2-vyz, uvw-y2z;
↦ binomials2intmat(I);
```

See also: Section 1.1 [intmat2mons], page 15; Section 1.1 [mons2intmat], page 15.

2 Index