```
#SBATCH --threads-per-core=1
#SBATCH --mem=80000
#SBATCH --array=0-999

# each job will see a different ${SLURM_ARRAY_TASK_ID}
time ../normaliz -c InList --List --Split -X=${SLURM_ARRAY_TASK_ID}
```

A typical file for case (2):

```
#!/bin/sh
#SBATCH --job-name="InList"
#SBATCH --comment="InList"
#SBATCH --time=24:00:00
#SBATCH --ntasks=8
#SBATCH --threads-per-core=1
#SBATCH --mem=80000
#SBATCH --array=0-999

# each job will see a different ${SLURM_ARRAY_TASK_ID}
time ../normaliz -c InList --List -A=${SLURM_ARRAY_TASK_ID} -Z=100
```

The operations started by `DistributedComp` or `CollectLat` can also be performed on an input list.

The time bound set by `normaliz.time` is shared by all files in siuch a way that each file gets the same amount of time.

H. Fusion rings

The computation of fusion rings is a special case of computing lattice points in a polytope that satisfy polynomials equations. We refer the user to the book [24] for the basic theory and to [3] for our computational approach.

H.1. The structure of fusion rings

A *fusion ring* R is an associative free $\mathbb{Z}$-algebra with a fixed $\mathbb{Z}$-basis $b_1,\dots,b_r$. One of the basis elements is the multiplicative unit which is always chosen to be b_1. The bilinear map $R\times R\to R$ given by the multiplication $(x,x')\mapsto xx'$ is the bilinear extension of the product $(b_i,b_j)\mapsto b_ib_j$ and this product can be written uniquely in the form

$$b_ib_j=\sum_{i=1}^{r}N_{ij}^kb_k,\qquad N_{ij}^k\in\mathbb{Z}.$$

One of the distinctive features of fusion rings: $N_{ij}^k\geq 0$ for all i,j,k. The other is the existence of a *duality*, an involution of the set $\{b_1,\dots,b_r\}$, $b_i\mapsto b_{i^*}$ that extends to an antiautomorphism of R, and satisfies the condition $N_{ij}^1=1$ if $i=j*$, and $N_{ij}^1=0$ else.

In total the coefficients N_{ij}^k must satisfy the following conditions (in addition to nonnegativity): for all i,j,k,t

- (Ass) $\sum_s N_{i,j}^s N_{s,k}^t = \sum_s N_{j,k}^s N_{i,s}^t$,
- (Unit) $N_{1,i}^j = N_{i,1}^j = \delta_{i,j}$,
- (Auto) $N_{ij}^{k^*} = N_{j^*i^*}^k$,
- (Dual) $N_{i^*,j}^1 = N_{ji^*}^1 = \delta_{i,j}$.

These conditions imply a very useful identity, called *Frobenius reciprocity*: for all i,j,k one has

$$N_{i,j}^k = N_{i^*,k}^j = N_{j,k^*}^{i^*} = N_{j^*,i^*}^{k^*} = N_{k^*,i}^{j^*} = N_{k,j^*}^i.$$

Together with the fixed values of $N_{i,j}^k$ with $1 \in \{i,j,k\}$ this identity reduced the number of the coefficients that we want to compute to $\sim (r-1)^3/6$. If the commutativity $N_{ij}^k = N_{ji}^k$ is asked for, the 6-term identity extends to a 12-term identity since then $N_{ij}^k = N_{ji}^k$ for all i,j,k.

The fundamental property of fusion rings is given by the *Frobenius–Perron theorem*:

- a square matrix with nonnegative integer entries has a nonnegative real eigenvalue;
- for the maximum real eigenvalues d_i of the left multiplication by b_i, $i = 1,\dots,r$, the assignment $b_i \mapsto d_i$, $i = 1,\dots,r$ extends to a ring homomorphism $R \to \mathbb{R}$;
- it is the only ring homomorphism $R \to \mathbb{R}$ that has nonnegative values on $b_1,\dots,b_r$.

One calls d_i the *Ftobenius–Perron dimension* $\mathrm{FPdim}(b_i)$ *of* b_i, and sets $\mathrm{FPdim}(R) = \sum_i d_i^2$.

By definition $d_1,\dots,d_r$ are algebraic integers. We concentrate on the case in which they belong to $\mathbb{Z}$. The task to be solved is the computation of all fusion rings of a given *type* $(d_1,\dots,d_r)$ and a given duality $(1^*,\dots,r^*)$ (possibly with the additional condition that R is commutative).

Given the type $(d_1,\dots,d_r)$ and the duality, we must find all nonnegative solutions to the linear equations

$$N_{ij}^1 d_1 + \cdots + N_{ij}^r d_r = d_i d_j, \qquad i,j = 1,\dots,r$$

that reflect the homomorphism condition of the assignment $b_i \mapsto d_i$. Furthermore the associativity condition (Ass) must be satisfied that is given by polynomial equations of degree 2.

To set up these equations and to interpret the solutions one must fix coordinates. We do this as follows. The N_{ij}^k with $1 \in \{i,j,k\}$ are inserted into the equations with their fixed values $\in \{0,1\}$. Each tuple (i,j,k) with $1 \notin \{i,j,k\}$ belongs to a set $FR(i,j,k)$ defined by the 6-rem identity (or the 12-term identity). This set is represented by its lexicographic smallest member and the sets are ordered lexicographically by these members.

Examples of input files containing the systems of equations are `pet.in` and `baby.in`. *Normaliz can produce the system of equations itself,* and we explain the necessary input types in Section H.2.

The computation uses the patching variant of project-and-lift (see 7.2.4).The options that control the insertion order of patches can be applied (see 7.2.6).

H.2. Input types and computation gaols

Note that we count types and dualities from 0 *in the following.*

In order to produce the system of equations for fusion rings itself, Normaliz provides the input types

fusion_type – the type of the fusion ring, and
fusion_duality – the duality of the fusion ring.

Both are vectors of length `amb_space`, the fusion rank. If the duality is omitted, Normaliz chooses the identity for it. Example `bracket_4.in`, using the handy `amb_space_auto`:

```
amb_space auto
fusion_type
[1,1,2,3,3,6,6,8,8,8,12,12]
fusion_duality
[0,1,2,3,4,5,6,7,8,9,11,10]
```

The i-th entry of the duality is i^* for $i = 0, \ldots, r-1$. In our case there is only one transposition: $10^* = 11$. The first entry of the duality vector must be 0 unless we want to use it to require certain additional conditions; see Section H.6.

It is of course possible to compute all lattice points satisfying the linear and quadratic equations by asking for `LatticePoints`. But in general the systems has nontrivial automorphisms so that every fusion ring is represented by several isomorphic copies, of which only one is of interest. A further aspect is the distinction between simple and nonsimple fusion rings, where simple means that no proper subset of the basis generates a $\mathbb{Z}$-submodule that is a nontrivial fusion ring (to which the duality restricts). The computation goals for fusion rings are

FusionRings – compute all fusion rings (up to automorphisms),
SimpleFusionRings – compute all simple fusion rings (up to automorphisms),
LatticePoints – compute all fusion rings (allowing isomorphic copies).

The default computation goal is `FusionRings`. For `bracket_4.in` we get the output file

```
148 fusion rings up to isomorphism
0 simple fusion rings up to isomorphism
148 nonsimple fusion rings up to isomorphism

Embedding dimension 231

dehomogenization
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ... 0 0 0 0 0 1

***********************************************************************

0 simple fusion rings up to isomorphism:

148 nonsimple fusion rings up to isomorphism:
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 ... 1 3 3 1 1 1
```

```
...
```

Note that the input for fusion rings is inhomogeneous. So the last one is the homogenizing coordinate.

The computation goals `FusionRings` and `SimpleFusionRings` can also be used for "full" input files, provided they have standard names. Moreover, standard names allow the computation of fusion rings without an input file. See Section H.3.

A further input type is

`candidate_subring` – a 0-1-vector of length `amb_space`.

It specifies a subset of the basis of the fusion ring that is used for testing simplicity: The entries 1 mark the basis vectors selected for the candidate subring. This can be useful for very hard computations. However 'simple "must then be understood as "not containing the candidate" and "nonsimple" is the opposite. Example `bracket_3_cand.in`:

```
amb_space auto

fusion_type
[1,1,2,3,3,6,6,8,8,8,12,12]

candidate_subring
[1,1,0,0,0,0,0,0,0,0,0,0]
```

If a `candidate_subring` is given, the default computation goal is changed to `SingleLatiicePoint`. If another computation goal is set explicitly, then the `candidate_subring` is disregarded.

If you only want to find out whether there is a fusion ring for your type and duality, you can use the option

`SingleFusionRing`

If there exist simple and nonsimple fusion rings for your data, then it is impossible to predict whether the single fusion ring will be simple or nonsimple. However, the output files will test the single fusion ring for these properties. Example `bracket_4_single.in`:

```
amb_space auto
fusion_type
[1,1,2,3,3,6,6,8,8,8,12,12]
fusion_duality
[0,1,2,3,4,5,6,7,8,9,11,10]
SingleFusionRing
```

It yields the output

```
1 fusion rings up to isomorphism (only single fusion ring  asked for)
0 simple fusion rings up to isomorphism
1 nonsimple fusion rings up to isomorphism

Embedding dimension = 276
```

```
dehomogenization
0 0 0 0 0 0 0 0 0 0 0 0 0 0  ...0 0 0 0 0 1

************************************************************************

0 simple fusion rings up to isomorphism:

1 nonsimple fusion rings up to isomorphism:
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0...  3 3 1 1 1
```

Note that the single fusion ring found is not uniquely determined. The option can save considerable computation time, but only if there exists a fusion ring.

Finally one can also generate an input file with a system of linear equations that constitutes a necessary condition of fusion rings for the given type: if the "partition system" has no solution in nonnegative integers, then there are no fusion rings for the given type, regardless of the duality). See [3]. The input type is

`fusion_type_for_partition` – the type to be tested.

The default computation goal is `SingleLatticePoint` since (at present) we are only interested in the solubility. Also `LatticePoints` is allowed, but be aware of potentially very large numbers of solutions. Example `bracket_3_part.in`:

```
amb_space auto
fusion_type_for_partition
[1,1,2,3,3,6,6,8,8,8,12,12]
```

which yields

```
1 module generators (only single lattice point asked for)

embedding dimension = 57

dehomogenization:
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...

Lattice point:
0 0 0 0 0 0 1 0 0 0 0 2 0 0 0 2 0 0 3 0 2 1 ...

************************************************************************
```

Note that the single lattice point is not uniquely determined. There are 300 lattice points in the polytope.

The system of degree 2 polynomial equations resulting from the associativity condition is highly overdetermined. For example, `pet_new.in` generates 240 equations, but a run with `MinimizePolyEquations` reduces them to 50. Using `MinimizePpolyEquations` is usually not a good idea since the formal minimization takes much more time than a run without it. Nor-

maliz uses *heuristic minimization* by counting how often a vector that has passed all preceding equations is not a solution. If an equation has "never" failed after a certain number n of vectors passing it, it is declared ineffective and skipped latter on. The number n depends on the FPdim of the fusion ring. However, a vector is only declared a final solution after checking all equations on it. So there is no danger of false results. But if an equation gets discarded prematurely, the computation time can explode. As a prevention, Normaliz offers the option

NoHeuristicMinimization, --NHM

Unfortunately, this option often doubles the computation time.

H.3. Standard names and virtual input files

Fusion rings can be computed without an input file – the input file exists "virtually". For this variant the project name must contain the fusion type and duality. Such "standard names" have the structure

```
[<t>][<d>]
```

where `<t>` is the type and `<d>` is the duality. Both are comma separated integer vectors of the same length (the fusion rank). Example:

```
[1,1,2,3,3,6,6,8,8,8,12,12][0,1,2,3,4,5,6,7,8,9,11,10]
```

This standard name can be prefixed by a path which defines the directory where the output file is placed.

Note: if there exists a file `[<t>][<d>].in` , it is read and evaluated. It is not allowed to be empty. So, if you want to have a real input file, it must contain the same data as an input file whose name is not standard. Commutativity can be forced by starting the duality with `-1`.

Try

```
/path/to/normaliz -c [1,1,2,3,3,6,6,8,8,8,12,12][0,1,2,3,4,5,6,7,8,9,11,10]
```

to see the computation with virtual input file.

This trick can also be used for partition files where the standard name is only the type. Example:

```
[1,1,2,3,3,6,6,8,8,8,12,12]
```

Try

```
/path/to/normaliz -c [1,1,2,3,3,6,6,8,8,8,12,12]
```

If you still have 'full" input files with linear and polynomial equations for fusion rings, you can run them with the computation goals `FusionRings` or `SimpleFusionRings`. However, the fusion data must be transported by a standard name.

Normaliz can produce a real input file for a standard name by

--MakeFusionInput, --MFI

For example,

```
/path/to/normaliz -c [1,1,2,3,3,6,6,8,8,8,12,12] --MFI
```

will produce `[1,1,2,3,3,6,6,8,8,8,12,12].in`.

Virtual input files can be used in lists.

Note: an input files is only generated if it does not exist yet.

H.4. Nonintegral fusion rings

It is possible to compute nonintegral fusion rings. For them the type must be specified by elements from an algebraic number field. At present it must be embedded into $\mathbb{R}$. In order to define the number field, one needs a real input file. Example `EH1.in`:

```
amb_space auto
number_field min_poly (5+4*a-5*a^2+a^3) embedding [3 +/- 0.5]

fusion_type
[1, (a) (a^2 - a - 1) (2*a^2 - 3*a - 4) (3*a^2 - 5*a - 4) (4*a^2 - 7*a - 6)]
```

it gives the output

```
1 fusion rings up to isomorphism
1 simple fusion rings up to isomorphism
0 nonsimple fusion rings up to isomorphism

Embedding dimension 36

dehomogenization
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

**********************************************************************

1 simple fusion rings up to isomorphism:
1 1 0 0 0 1 0 1 0 0 1 1 1 1 2 1 1 1 1 1 1 2 2 3 3 1 2 2 3 3 4 4 5 6 7 1

0 nonsimple fusion rings up to isomorphism:
```

H.5. Full fusion data

The output for fusion rings shown in the previous section is the short form that uses Frobenius reciprocity, (Dual) and (Unit). However, Normaliz can also provide the fusion data (N_{ij}^k) in full form. For a single fusion ring it is a list of Matrices M_i. The matrix M_i contains the numbers N_{ij}^k where j is the row index and k is the column index. (The transpose is the matrix of left multiplication by the basis element b_i.) The option

FusionData

asks for the additional output file

<project>.fus

It contains a list of lists, one inner list for every computed fusion ring. As an example we take `pet_new.in` that produces 2 fusion rings. The usual output file is `pet_new.out`:

```
...
2 simple fusion rings up to isomorphism:
1 1 0 0 1 1 0 0 1 1 1 1 1 1 1 1 1 1  ...  1 1 2 1 2 1 2 2 1 1
1 1 0 0 1 1 0 0 1 1 1 1 1 1 1 1 1 1 ... 1 1 1 2 1 2 2 1 3 0 1
...
```

With the option `FusionData` in `pet_new.in` we run

```
./normaliz -c example/pet_new
```

and get the additional file `pet_new.fus` (with comments `<--- ...`):

```
[      <--------------  start of list of fusion rings
[    <--------------  fusion ring 1
[  <--------------  matrix 1
[1,0,0,0,0,0,0],  <---- row 1
[0,1,0,0,0,0,0],
...
[0,0,0,0,0,0,1]  <---- last row
], <--------------  end of matrix 1
...
],   <------------- end of fusion ruing 1
...
]     <------------- end of outer list
```

H.6. Necessary conditions for modular categorification

Fusion rings that allow modular categorification (see [3] and [24]) must satisfy certain conditions. Normaliz can be asked to compute only fusion rings that satisfy them.

H.6.1. Commutativity

The first condition is commutativity. Then the first entry of the duality is -1. Example `bracket_4_comm.in`:

```
amb_space auto
fusion_type
[1,1,2,3,3,6,6,8,8,8,12,12]
fusion_duality
```

```
[-1,1,2,3,4,5,6,7,8,9,11,10]
```

This somewhat strange way to communicate commutativity is necessary because Normaliz must know it already at the time of construction. It would come too late as an algorithmic variant.

Note that forcing commutativity is superfluous if the duality is the identity. However, in other cases it can speed up computations significantly.

H.6.2. Graded structure

Normaliz requires that fusion rings to which the options in this section are to be applied are commutative, as explained in the preceding section.

The base elements b_i with $d_i = 1$ form a group under multiplication. Let m be their number. For modular categorification the ring must be graded with respect to this group. The homogeneous components are modules over the neutral component and generated by subsets B_i, $i = 1, \dots, m$, of $\{b_0, \dots, b_{r-1}\}$ as $\mathbb{Z}$-modules. (We are counting basis elements from 0.) Set $f_i = \sum_{j \in B_i} d_j^2$. Then $f_1 = \cdots = f_m$.

See [24, Section 3.5] for basic facts about gradings of fusion rings and [24, 8.22.9(iii), 4.14.3] for the existence in the case of modular categorification.

At present Normaliz allows only $m \leq 4$. The reason for this restriction is that the group table that underlies the grading is uniquely determined by the duality. For groups of higher order further input data would be necessary, at least if one wants to use the grading already in the computation for which it often has a significant effect.

There is another ambiguity that must be taken care of: the type and the duality may allow different partitions of the set of base vectors that are compatible to the group structure. To be on the safe side one first runs the input file with the option

`ModularGradings`

Example `find_mod_grad.in`:

```
amb_space 14
fusion_type
1 1 1 1 2 2 2 2 2 2 2 4 4 4
fusion_duality
-1 1 2 3 4 5 6 7 8 9 10 11 12 13
ModularGradings
```

Result:

```
2 modular gradings


**********************************************************************

2 modular gradings:
modular grading 1
```

```
0 1 2 3 4 5 6 7
8 11
9 12
10 13
--------------------
modular grading 2

0 1 2 3 11
4 5 6 7 8
9 12
10 13
--------------------
```

All other modular gradings differ from the two above by automorphisms of the system, and it is enough to consider only one representative in each orbit.

For the computation one must fix one of the gradings as in the input by

`modular_grading <g>`

where `<g>` is the index of the grading, as in `mod_grad.-in`:

```
amb_space 14
fusion_type
1 1 1 1 2 2 2 2 2 2 2 4 4 4
fusion_duality
-1 1 2 3 4 5 6 7 8 9 10 11 12 13
modular_grading 2
UseModularGrading
```

The option

`UseModularGrading`

has told Normaliz to use the chosen modular grading. If there is only one modular grading, the choice is superfluous.

H.6.3. Induction to the center

Let R be a fusion ring with fusion data $(N_{i,j}^k)$ and basis $\{a_1,\dots,a_r\}$, where a_1 is the unit. For simplicity, we restrict ourselves to integral and commutative R. Both these properties are essential for the discussion below.

Assume that R admits a categorification into a fusion category $\mathscr{C}$ over the complex field. ($\mathscr{C}$ is not necessarily uniquely determined.) Then the Drinfeld center $Z(\mathscr{C})$ of $\mathscr{C}$ is an integral modular fusion category see [24, Section 9.2] for the mathematics). Let ZR be the Grothendieck ring of $Z(\mathscr{C})$ Let $\{b_1,\dots,b_n\}$ be the basis of ZR, where $n \geq r$. By the properties of the Drionfeld center, ZR is an integral commutative $1/2$-Frobenius fusion ring: this means that $\mathrm{FPdim}(b_i)^2$ divides $\mathrm{FPdim}(ZR)$ in $\mathbb{Z}$ (FPdim was introduced on p. 270).

Let $d_i = \mathrm{FPdim}(a_i)$ and $m_i = \mathrm{FPdim}(b_i)$. Then, $\mathrm{FPdim}(R) = \sum_i d_i^2$ and $\mathrm{FPdim}(ZR) = \sum_i m_i^2$. There is a theorem stating that $\mathrm{FPdim}(ZR) = \mathrm{FPdim}(R)^2$ [24, Thm. 7.16.6]. By the $1/2$-Frobenius property, m_i^2 divides $\mathrm{FPdim}(ZR)$, so m_i divides $\mathrm{FPdim}(R)$.

There is a ring morphism $F : ZR \to R$ preserving FPdim, induced by the (so-called) forgetful functor $Z(\mathscr{C}) \to \mathscr{C}$. Thus

$$F(b_i) = \sum_j F_{i,j} a_j,$$

where $F_{i,j}$ are nonnegative integers and

$$m_i = \mathrm{FPdim}(\sum F_{i,j} a_j) = \sum_j F_{i,j} d_j.$$

There is an additive morphism $I : R \to ZR$ (not preserving FPdim, so not multiplicative) induced by the adjoint of the forgetful functor. As a matrix, I is just the transpose of F, i.e.,

$$I(a_j) = \sum_i F_{i,j} b_i = \sum_i F_{i,j} b_i.$$

The $r \times n$ matrix of I is usually called the *induction matrix*. It satisfies a list of properties implying that for a given fusion ring, there are only finitely many possible induction matrices. Normaliz can compute them—at least in principle since the computation may need an astronomical time.

There can be zero, one or several possible induction matrices. If none, then the fusion ring R is excluded from categorification, which is very useful. If there are induction matrices but no ZR compatible with them, then R is excluded as well from categorification. Idem if there are compatible ZR but no modular data.

In general, for a given fusion ring R, several ZR are possible, and several n (= rank(ZR)) are possible. Hence, the rank n of ZR is also a variable.

A theorem [24, Prop. 9.2.2] states that, for all j,

$$F(I(a_j)) = \sum_t a_t a_j a_{t^*}.$$

But

$$F(I(a_j)) = \sum_k \left(\sum_i F_{i,j} F_{i,k} \right) a_k,$$

and

$$\sum_t a_t a_j a_{t^*} = \sum_k \left(\sum_{s,t} N_{t,j}^s N_{s,t^*}^k \right) a_k.$$

We get the following equation:

$$\sum_i F_{i,j} F_{i,k} = \sum_{s,t} N_{t,j}^s N_{s,t^*}^k \quad \text{for all} \quad j,k = 1,\ldots,n. \tag{3}$$

The left multiplication matrix for $F(I(a_1)) = \sum_t a_t a_{t^*}$ admits eigenvalues $(f_i)_{i=1,..,r}$ called formal codegrees, which by a theorem [33, Cor. 2.14], must be integers dividing $\mathrm{FPdim}(R)$. Moreover, for $i \in \{1,..,r\}$, we can choose $m_i = \mathrm{FPdim}(R)/f_i$. Note that this is a negative criterion: if the sum of the multiplicities of these eigenvalues is $< r$, then there is no induction matrix.

Note that

$$F(I(a_1)) = \sum_t a_t a_{t^*} = \sum_k \left(\sum_t N_{t,t^*}^k \right) a_k,$$

so the left multiplication matrix for $F(I(a_1))$ is

$$\left(\sum_{t,k} N_{t,t^*}^k N_{k,l}^s \right)_{s,l}.$$

Finally:

$$\begin{gathered} F(b_1) = a_1, \text{ so } F_{1,j} = \delta_{1,j}, \\ F_{i,1} = 1, \text{ for all } i \in \{1,..,r\}, \\ F_{i,1} = 0 \text{ for all } i \in \{r+1,\ldots,n\}. \end{gathered}$$

The rows of the matrix $F = (F_{ij})$ must satisfy the condition

$$t = \sum_j F_{ij} d_j, \qquad t \mid \mathrm{FPdim}(R).$$

The starting point of the computation therefore is to find all solutions to the equation $t = \sum_j F_{ij} d_j$ for the divisors t of $\mathrm{FPdim}(R)$ that satisfy the additional conditions just mentioned. Then we must assemble F from these rows so that Equation (3) is satisfied as well as $\sum_{i=1}^n m_i^2 = \mathrm{FPdim}(R)^2$.

Our computation goal is

`InductionMatrices`

As an example we take `[1,1][0,1].in`:

```
amb_space 2
fusion_type
1 1
fusion_duality
0 1
InductionMatrices
```

The induction matrices are contained in

`<project>.ind`

They are printed with the conventions for F above. For our example it is `[1,1][0,1].ind`:

```
[     <----- begin outer list over the fusion rings computed
  [   <----- inner list of vdata for the current fusion ring
    [
      [0,1] <----- the fusion rfing in the format of <project>.out
    ],
    [  < first matrix F for the current fusion rfing
      [1,0],
      [1,0],
      [0,1],
      [0,1]
    ],
    [
      [1,1,1,1] <---- type of the potential ZR
    ],
    [ <---- list of pairs (i,i*) that are possible
      [1,1],
      [2,2],
      [2,3],
      [3,3]
    ]
  ]
]
```

The rows of F are ordered by ascending m_i. The additional data are meant as a help for setting up the input file for the next step that we discuss below. Note that the matrix F does not define the duality of ZR. The list of pairs (i, i^*) is meant as a help for finding suitable dualities.

Our example defines only a single fusion ring. In general there are more than one. In this case one can either produce induction matrices for all fusion rings or pick one by

`chosen_fusion_ring <s>`

where `<s>` is a number between 1 and the number of fusion rings computed. Example `chosen_2,in`:

```
amb_space 5
fusion_type
1 1 2 3 3
fusion_duality
0 1 2 3 4
InductionMatrices
chosen_fusion_ring 2
```

You can vary the file by choosing fusion ring 1 (no induction matrix) or omit the choice completely.

The second step is computing the potential centers defined by the matrices F. Example `mini_ind.in` is

```
amb_space auto
```

```
fusion_type
[1,1,1,1]
fusion_ring_map
[
[1,0],
[1,0],
[0,1],
[0,1]
]
fusion_image_type
[1, 1]
fusion_image_duality
[0, 1]
fusion_image_ring
[0,1]
```

Important: The input types

fusion_image_type

fusion_image_duality

fusion_image_ring

fusion_ring_map

must be *formatted* matrices.

`mini_ind.out` is

```
1 fusion rings up to isomorphism
0 simple fusion rings up to isomorphism
1 nonsimple fusion rings up to isomorphism

Embedding dimension = 11

dehomogenization
0 0 0 0 0 0 0 0 0 0 1

**********************************************************************

0 simple fusion rings up to isomorphism:

1 nonsimple fusion rings up to isomorphism:
0 0 0 0 1 0 0 0 0 0 1
```

The input as in `mini_ind.in` makes Normaliz compute only those fusion rings for the given `fusion_type` and `fusion_duality` for which the matrix F defines a homomorphism to the image defined by `fusion_image_type`, `fusion_image_duality` and fixed in `fusion_image_ring`.

A necessary condition is $F_{ij^*} = F_{i*j}$ for $i = 1,\dots,n$ and $j = 1,\dots,r$ where $j*$ is defined by the duality on R and i^* by the duality on ZR. It is easy to check that F defines a homomorphism

if and only if

$$\sum_{k=1}^{n} M_{i,j}^{k} F_{k,t} = \sum_{l,s=1}^{r} F_{i,l} F_{j,s} N_{l,s}^{t}, \qquad i,j = 1,\ldots,n,\ t = 1,\ldots,r.$$

This system of linear equations for $M_{i,j}^{k}$ is added to the constraints defining ZR.

Remark. All computations above only verify necessary conditions for a Drinfeld center. The fusion rings ZR that are defined by an induction matrix and a compatible duality and for which F is a homomorphism to R are not necessarily Grothrnfieck rings of Drinfeld centers of the categorifications $\mathscr{C}$ of R. A simple example: if one changes the `fusion_image_duality` in `mini_ind.in` to $[0,1,3,2]$, one obtains the group ring of the cyclic group C_4 as ZR. But the Drinfeld centers of the categorifications of C_2 all have the group ring of $C_2 \times C_2$ as their Grothendieck ring (as we got it for the duality $[0,1,2,3]$).