8 easy pieces for "SONATA"

(System of Nearrings and Their Applications)

—

Tutorial

by

the SONATA–team
Institut f. Algebra
Univ. Linz, 4040 Linz, Austria

Contents

The authors

SONATA was written at

Institut f. Algebra
Universität Linz

by the SONATA Team:

Erhard Aichinger
Franz Binder
Jürgen Ecker
Peter Mayr
Christof Nöbauer

1 Groups

SONATA adds some functions for groups. To use the functions provided by SONATA, one has to load it into GAP:

```
gap> LoadPackage( "sonata" );
```

1.1 Thomas' and Wood's catalogue of small groups

Most of the nonabelian groups (even small ones) do not have a popular name (as S_3 or A_4). We like to give unique names to the groups we are working with. The book "Group Tables" by Thomas and Wood classifies all groups up to order 32. In this book every group has a name of the form `m/n`, where `m` is the order of the group and `n` the number of the particular group of order `m`. The cyclic groups have the name `m/1`. Then come the abelian groups, finally the non-abelian ones. To find out the name of a given group in their book we use `IdTWGroup`.

```
gap> G := DihedralGroup( 8 );
<pc group of size 8 with 3 generators>
gap> IdTWGroup( G );
[ 8, 4 ]
```

If we want to refer to the group with the name `8/4` directly we say

```
gap> H := TWGroup( 8, 4 );
8/4
```

Groups which are obtained in this way always come as a group of permutations. We can have a look at the elements of H if we ask for H as a list.

```
gap> AsList( H );
[ (), (2,4), (1,2)(3,4), (1,2,3,4), (1,3), (1,3)(2,4), (1,4,3,2),
  (1,4)(2,3) ]
```

Clearly, G and H are not equal but they are isomorphic. If we want to know what the isomorphism between the two looks like, we use `IsomorphismGroups`. Note, that a homomorphism is determined by the images of the generators.

```
gap> IsomorphismGroups(G,H);
[ f1, f2, f3 ] -> [ (2,4), (1,2,3,4), (1,3)(2,4) ]
```

How many nonisomorphic groups are there of order n? Up to order 1000 the function `NumberSmallGroups` gives the answer. As a shortcut for `TWGroup( 32, 46 )` we may also type `GTW32_46`.

```
gap> NumberSmallGroups( 32 );
51
gap> GTW32_46;
32/46
gap> GTW32_46 = TWGroup( 32, 46 );
true
```

Now we find all nonabelian groups with trivial centre of order at most 32. We use `GroupList`, a list of all groups up to order 32 and filter out the nonabelian ones with trivial center.

```
gap> Filtered( GroupList, g -> not IsAbelian( g ) and
>                             Size(Centre( g ))=1 );
[ 6/2, 10/2, 12/4, 14/2, 18/4, 18/5, 20/5, 21/2, 22/2, 24/12, 26/2,
  30/4 ]
```

This was the first time that we have used a function as an argument. The second argument of the function `Filtered` is a function (`g -> not ...`), which returns for every `g` the boolean value `true` if `g` is not abelian and the size of its centre is 1, and `false` otherwise. This is the easiest way to write a function.

1.2 Subgroups

The function `Subgroups` returns a list of all subgroups of a group. We can use this function and the `Filtered` command to determine all characteristic subgroups of the dihedral group of order 16.

```
gap> D16 := DihedralGroup( 16 );
<pc group of size 16 with 4 generators>
gap> S := Subgroups( D16 );
[ Group([  ]), Group([ f4 ]), Group([ f1 ]), Group([ f1*f3 ]),
  Group([ f1*f4 ]), Group([ f1*f3*f4 ]), Group([ f1*f2 ]),
  Group([ f1*f2*f3 ]), Group([ f1*f2*f4 ]),
  Group([ f1*f2*f3*f4 ]), Group([ f4, f3 ]), Group([ f4, f1 ]),
  Group([ f1*f3, f4 ]), Group([ f4, f1*f2 ]),
  Group([ f1*f2*f3, f4 ]), Group([ f4, f3, f1 ]),
  Group([ f4, f3, f2 ]), Group([ f4, f3, f1*f2 ]),
  Group([ f4, f3, f1, f2 ]) ]
gap> C := Filtered( S, G -> IsCharacteristicInParent( G ) );
[ Group([  ]), Group([ f4 ]), Group([ f4, f3 ]), Group([ f4, f3, f2 ]),
  Group([ f4, f3, f1, f2 ]) ]
```

1.3 Group endomorphisms

Everybody knows that every automorphism of the symmetric group S_3 (= `GTW6_2`) fixes a point (besides the identity of the group). But, are there endomorphisms which fix nothing but the identity? We are going to simply try it out. On our way we will find out that all automorphisms of S_3 are inner automorphisms.

```
gap> G := GTW6_2;
6/2
gap> Automorphisms( G );
[ IdentityMapping( 6/2 ), ^(2,3), ^(1,3), ^(1,3,2), ^(1,2,3), ^(1,2) ]
gap> Endos := Endomorphisms( G );
[ [ (1,2), (1,2,3) ] -> [ (), () ], [ (1,2), (1,2,3) ] -> [ (2,3), () ],
  [ (1,2), (1,2,3) ] -> [ (1,3), () ], [ (1,2), (1,2,3) ] -> [ (1,2), () ],
  [ (1,2), (1,2,3) ] -> [ (2,3), (1,2,3) ],
  [ (1,2), (1,2,3) ] -> [ (2,3), (1,3,2) ],
  [ (1,2), (1,2,3) ] -> [ (1,2), (1,3,2) ],
  [ (1,2), (1,2,3) ] -> [ (1,2), (1,2,3) ],
  [ (1,2), (1,2,3) ] -> [ (1,3), (1,2,3) ],
  [ (1,2), (1,2,3) ] -> [ (1,3), (1,3,2) ] ]
```

Now it is time for real programming, but don't worry, it is all very simple. We write a function which decides whether an endomorphism fixes a point besides the identity or not (in the latter case we call the endomorphism **fixed-point-free**).

```
gap> IsFixedpointfree := function( endo )
>local group;
>  group := Source( endo ); # the domain of endo
>  return ForAll( group, x -> (x <> x^endo) or (x = Identity(group)) );
>  #                          x is not fixed or x is the identity
>end;
function ( endo ) ... end
```

This paragraph says that `IsFixedpointfree` is a function that takes one argument (called `endo`). Now we create a local variable `group` to store the group on which the endomorphism acts (in our example this will always be S_3, but maybe we want to use this function for other groups, too). Local means that GAP may forget this variable as soon as it has computed what we want (and it will forget it instantly afterwards). Now we store the domain of `endo` in the variable `group`. The next line already returns the result. It returns `true` if for all elements `x` of `group` either `x` is not fixed by `endo` or `x` is the identity of the group. This line is a one-to-one translation of the logical statement that `endo` is fixed-point-free.

The result is a function which can be applied to any endomorphism, now. For example we can ask if the fourth endomorphism in the list `E` is fixed-point-free.

```
gap> e := Endos[4];
[ (1,2), (1,2,3) ] -> [ (1,2), () ]
gap> IsFixedpointfree( e );
false
```

Now we filter out the fixed-point-free endomorphisms.

```
gap> Filtered( Endos, IsFixedpointfree );
[ [ (1,2), (1,2,3) ] -> [ (), () ] ]
```

1.4 Finding a set of generators

It is well known that for any finite p-group G the factor $G/\Phi(G)$ modulo the Frattini subgroup $\Phi(G)$ has order $p^{\delta(G)}$, where $\delta(G)$ is the minimal number of generators of G. Moreover the representatives of the residue classes modulo $\Phi(G)$ form a set of generators. So a generating set for a p-group could be obtained in the following way. We choose the group 16/11 (a semidirect product of the cyclic group of order 8 with the cyclic group of order 2).

```
gap> G := GTW16_11;
16/11
gap> F := FrattiniSubgroup( G );
Group([ (1,4,11,14)(2,7,10,16)(3,8,15,9)(5,12,6,13) ])
gap> NontrivialRepresentativesModNormalSubgroup( G, F );
[ (1,16,14,10,11,7,4,2)(3,12,9,5,15,13,8,6),
  (1,3)(2,5)(4,8)(6,10)(7,12)(9,14)(11,15)(13,16),
  (1,13,4,5,11,12,14,6)(2,3,7,8,10,15,16,9) ]
gap> H := Group( last );
Group([ (1,16,14,10,11,7,4,2)(3,12,9,5,15,13,8,6),
  (1,3)(2,5)(4,8)(6,10)(7,12)(9,14)(11,15)(13,16),
  (1,13,4,5,11,12,14,6)(2,3,7,8,10,15,16,9) ])
gap> G = H;  # test
true
```

The variable `last` in the this example refers to the last result, i.e. in this case the list of representatives.

2 Nearrings

A **(left) nearring** is an algebra $(N, +, *)$, where $(N, +)$ is a (not necessarily abelian) group, $(N, *)$ is a semigroup, and the distributive law $x*(y+z) = x*y+x*z$ holds. Such nearrings are called **left nearrings**. A typical example is constructed as follows: take a group $(G, +)$ (not necessarily abelian), and take the set $M(G)$ of all mappings from G to G. Then we define $+$ on $M(G)$ as pointwise addition of mappings, and $*$ by $m * n(\gamma) := n(m(\gamma))$. The multiplication looks more natural if we write functions right of their arguments. Then the definition reads $(\gamma)m * n = ((\gamma)m)n$.

Textbooks on nearrings are [Mel85], [Cla92], [CFF02]. They all use **left nearrings**. The book [Pil83] uses **right nearrings**; these are the algebras that arise if we claim the right distributive law $(x+y)*z = x*z+y*z$ instead of the left distributive law given above.

SONATA uses **left** nearrings throughout.

2.1 Entering nearrings into the system

The problem: Input the nearring given in the example of page 406 of [Pil83] into SONATA.

This nearring is given by an explicit multiplication table. The function `ExplicitMultiplicationNearRing` can be used to do the job. But first, let's get the additive group, which is Klein's four group:

```
gap> G := GTW4_2;
4/2
```

Now we have to establish a correspondence between the elements `0`, `a`, `b`, `c` of the group in the example and GAP's representation of the group elements.

```
gap> AsSortedList( G );
[ (), (3,4), (1,2), (1,2)(3,4) ]
```

Ok, let's map `0` to `()`, `a` to `(3,4)`, `b` to `(1,2)` and `c` to `(1,2)(3,4)`

```
gap> SetSymbols( G, [ "0", "a", "b", "c" ] );
gap> PrintTable( G );
Let:
0 := ()
a := (3,4)
b := (1,2)
c := (1,2)(3,4)

  +   0 a b c
  -----------
  0   0 a b c
  a   a 0 c b
  b   b c 0 a
  c   c b a 0
```

Now for entering the nearring multiplication: We will use the function `NrMultiplicationByOperationTable`. This function requires as one of its arguments a matrix of integers representing the operation table: We choose the entries of `table` according to the positions of the elements of `G` in `AsSortedList( G )`:

```
gap> table := [ [ 1, 1, 1, 1 ],
>               [ 1, 1, 2, 2 ],
>               [ 1, 2, 4, 3 ],
>               [ 1, 2, 3, 4 ] ];
[ [ 1, 1, 1, 1 ], [ 1, 1, 2, 2 ], [ 1, 2, 4, 3 ], [ 1, 2, 3, 4 ] ]
```

Now we are in position to define a nearring multiplication:

```
gap> mul:=NearRingMultiplicationByOperationTable(
>             G, table, AsSortedList(G) );
function( x, y ) ... end
```

And finally, we can define the nearring:

```
gap> N := ExplicitMultiplicationNearRing( G, mul );
ExplicitMultiplicationNearRing ( 4/2 , multiplication )
```

We get no error message, which means that we have indeed defined a nearring multiplication on `G`. Now let's take a look at it:

```
gap> PrintTable( N );
Let:
0 := (())
a := ((3,4))
b := ((1,2))
c := ((1,2)(3,4))

  +   0  a  b  c
  ---------------
  0   0  a  b  c
  a   a  0  c  b
  b   b  c  0  a
  c   c  b  a  0

  *   0  a  b  c
  ---------------
  0   0  0  0  0
  a   0  0  a  a
  b   0  a  c  b
  c   0  a  b  c
```

The symbols used for the elements of the group are also used for the elements of the nearring. Of course, it is still possible to redefine the symbols.

2.2 Some simple questions about the nearring

Now, that the nearring is in the system, let's ask some questions about it. A nearring is a nearfield if it has more than one element and its nonzero elements are a group with respect to multiplication. A textbook on nearfields is [Wäh87]. They are interesting structures, closely connected to sharply 2-transitive permutation groups and fixedpointfree automorphism groups of groups.

```
gap> IsNearField( N );
false
gap> IsIntegralNearRing( N );
false
gap> IsNilpotentNearRing( N );
false
```

[Pil83] is correct ... Well at least in this case.;-))

2.3 Entering the nearring with less typing

Certainly, everybody has immediately seen, that this nearring is a transformation nearring on GTW4_2 which is generated by the transformations 0 to 0, a to a, b to c, c to b, and the identity transformation, so

```
gap> t := GroupGeneralMappingByImages(
>           G, G, AsSortedList(G), AsSortedList(G){[1,2,4,3]} );
[ (), (3,4), (1,2), (1,2)(3,4) ] -> [ (), (3,4), (1,2)(3,4), (1,2) ]
gap> id := IdentityMapping( G );
IdentityMapping( 4/2 )
gap> T := TransformationNearRingByGenerators( G, [t,id] );
TransformationNearRingByGenerators(
[ [ (), (3,4), (1,2), (1,2)(3,4) ] -> [ (), (3,4), (1,2)(3,4), (1,2) ],
  IdentityMapping( 4/2 ) ])
```

Let's see what we've got:

```
gap> PrintTable(T);
Let:
n0 := <mapping: 4/2 -> 4/2 >
n1 := <mapping: 4/2 -> 4/2 >
n2 := <mapping: 4/2 -> 4/2 >
n3 := <mapping: 4/2 -> 4/2 >

  +   n0  n1  n2  n3
 --------------------
 n0   n0  n1  n2  n3
 n1   n1  n0  n3  n2
 n2   n2  n3  n0  n1
 n3   n3  n2  n1  n0

  *   n0  n1  n2  n3
 --------------------
 n0   n0  n0  n0  n0
 n1   n0  n0  n1  n1
 n2   n0  n1  n2  n3
 n3   n0  n1  n3  n2
```

Obviously, we've got the correct nearring. Let's make for sure:

```
gap> IsIsomorphicNearRing( N, T );
true
```

However, N and T are certainly not equal:

```
gap> N = T;
false
```

3 The nearring library

There are many non-isomorphic nearrings, even of small order. All non-isomorphic nearrings of orders 2 to 15 and all non-isomorphic nearrings with identity up to order 31 with exception of those on the elementary abelian groups of orders 16 and 27 are collected in the SONATA nearring library.

3.1 An application of the nearring library

The number of nearrings in the library is big. For example, try

```
gap> NumberLibraryNearRings( GTW12_3 );
48137
```

Try your favorite small groups with this function to get an impression of these numbers.

Of course, no one can know all these nearrings personally. Therefore, the main purpose of the nearring library is to filter out the nearrings of interest.

Consider for example the following

Problem: How many non-rings with identity of order 4 are there and what do they look like? If you cannot answer this question adhoc, stay tuned.

Let's start with the groups of order 4. Of course you know, there are 2 groups of order 4: `GTW4_1`, the cyclic group and `GTW4_2`, Klein's four group.

Let's go for `GTW4_1` first:

```
gap> NumberLibraryNearRingsWithOne( GTW4_1 );
1
gap> Filtered( AllLibraryNearRingsWithOne( GTW4_1 ),
>              n -> not IsDistributiveNearRing( n ) );
[  ]
```

So, the only nearring with identity there is on `GTW4_1` is the ring. Well... you knew that before, didn't you?

Now for `GTW4_2`:

```
gap> NumberLibraryNearRingsWithOne( GTW4_2 );
5
gap> Filtered( AllLibraryNearRingsWithOne( GTW4_2 ),
>              n -> not IsDistributiveNearRing( n ) );
[ LibraryNearRing(4/2, 12), LibraryNearRing(4/2, 22) ]
```

Here we go:

```
gap> PrintTable( LibraryNearRing( GTW4_2, 12 ) );
Let:
n0 := (())
n1 := ((3,4))
n2 := ((1,2))
n3 := ((1,2)(3,4))
```

```
 +   n0  n1  n2  n3
-------------------
n0   n0  n1  n2  n3
n1   n1  n0  n3  n2
n2   n2  n3  n0  n1
n3   n3  n2  n1  n0

 *   n0  n1  n2  n3
-------------------
n0   n0  n0  n0  n0
n1   n0  n0  n1  n1
n2   n0  n0  n2  n2
n3   n0  n1  n2  n3
gap> PrintTable( LibraryNearRing( GTW4_2, 22 ) );
Let:
n0 := (())
n1 := ((3,4))
n2 := ((1,2))
n3 := ((1,2)(3,4))

 +   n0  n1  n2  n3
-------------------
n0   n0  n1  n2  n3
n1   n1  n0  n3  n2
n2   n2  n3  n0  n1
n3   n3  n2  n1  n0

 *   n0  n1  n2  n3
-------------------
n0   n0  n0  n2  n2
n1   n0  n1  n2  n3
n2   n0  n2  n2  n0
n3   n0  n3  n2  n1
```

3.2 Appendix K revisited

An alternative to filtering the nearring library is to use a `for ... do ... od` construction.

We shall demonstrate this by recomputing the list of nearrings given in appendix K of [Pil83], i.e. a list of all nearrings on the dihedral group of order 8 (`GTW8_4`) which have an identity, are non-zerosymmetric or are integral.

First, we initialize the variable `nr_list` as the empty list:

```
gap> nr_list := [ ];
[ ]
```

Now, we write ourselves a `for` loop and add those nearrings we want:

```
gap> for i in [1..NumberLibraryNearRings( GTW8_4 )] do
>       n := LibraryNearRing( GTW8_4, i );
>       if ( not IsZeroSymmetricNearRing( n ) or
>            IsIntegralNearRing( n ) or
>            Identity( n ) <> fail
>          ) then
>         Add( nr_list, n );
>       fi;
>    od;
gap> Length( nr_list );
141
```

How many boolean nearrings are amongst these? We call a nearring **boolean** if $x * x = x$ for all $x \in N$.

```
gap> Filtered( nr_list, IsBooleanNearRing );
[ LibraryNearRing(8/4, 1314), LibraryNearRing(8/4, 1380),
  LibraryNearRing(8/4, 1446), LibraryNearRing(8/4, 1447) ]
```

Which correspond to the numbers 140, 86, 99, and 141 in [Pil83], appendix K, accordingly.

For those who got interested in boolean nearrings: many results about them have been collected in [Pil83], 9.31.

4 Nearrings of transformations on groups

We are going to study transformations on the alternating group on four elements A_4.

The problem: Let T be the nearring of mappings from A_4 to A_4 generated by the single mapping t which maps (2,3,4) to (2,4,3), (2,4,3) to (1,2)(3,4), (1,2)(3,4) to (1,2,3), (1,2,3) back to (2,3,4) and all other elements of A_4 to the neutral element (). Then, how many mappings are there in T that have (1,2,3) as a fixed point? If there are only a few we would be interested in a list of all of these.

The solution:
The first thing to do is create the nearring T. So we start with the group A_4, which can easily be constructed with the command

```
gap> A4 := AlternatingGroup( 4 );
Alt( [ 1 .. 4 ] )
```

The result is an object which represents the group A_4. If we want to see its elements we have to ask GAP to make a list of elements out of the group.

```
gap> AsSortedList( A4 );
[ (), (2,3,4), (2,4,3), (1,2)(3,4), (1,2,3), (1,2,4), (1,3,2),
 (1,3,4), (1,3)(2,4), (1,4,2), (1,4,3), (1,4)(2,3) ]
```

Now we create the mapping t. We use the function `MappingByPositionList` to enter it.

```
t := EndoMappingByPositionList( A4, [1,3,4,5,2,1,1,1,1,1,1,1] );
<mapping: AlternatingGroup( [ 1 .. 4 ] ) -> AlternatingGroup(
[ 1 .. 4 ] ) >
```

For `Mappings` the usual operations `+` and `*` can be used to add and multiply them.

```
gap> t+t;
<mapping: AlternatingGroup( [ 1 .. 4 ] ) -> AlternatingGroup(
[ 1 .. 4 ] ) >
gap> last * t;
<mapping: AlternatingGroup( [ 1 .. 4 ] ) -> AlternatingGroup(
[ 1 .. 4 ] ) >
```

(Recall that `last` stands for the result of the last computation, in this case this is `t + t`). Now we can construct the nearring. We use the function `TransformationNearRingByGenerators` which asks for the group (A_4) and a list of generating elements (the list with t as the only entry) as arguments.

```
gap> T := TransformationNearRingByGenerators( A4, [ t ] );;
```

Nearrings, allthough generated by a single element can become rather big. Before we print out all elements we ask for the size of T.

```
gap> Size( T );
20736
```

It seems reasonable not to print all elements. **Note** that they are not even computed, yet. All we wanted to know was the size of T and this can be computed without generating all elements. But, yes, we could generate them with `AsList` or `AsSortedList`. At last we want to find out how many of these 20736 `GroupTransformations` have (1,2,3) as a fixed point. We filter them out, but we use a second semicolon at the end to suppress printing, because there might be a lot of them. Then we ask for the length of the resulting list F of mappings.

```
gap> F := Filtered( T, tfm -> Image( tfm, (1,2,3) ) = (1,2,3) );;
gap> Length( F );
1728
```

It seems not to be worth printing the whole list. But we could for example choose a random transformation from this list F for testing purposes.

```
gap> Random( F );;
```

There are of course other properties of the nearring T which might be interesting. It is clear that a nearring which is generated by a single element is not necessarily abelian. T is a counterexample. As for finding counterexamples, SONATA can be used as a research tool.

```
gap> IsCommutative( T );
false
```

Finally, we try to disprove the conjecture that every transformation nearring on an abelian group that is generated by a single element must be commutative.

```
gap> g := CyclicGroup(2);;
gap> m := MapNearRing(g);;
gap> Filtered( m, n -> not( IsCommutative(
>        TransformationNearRingByGenerators( g, [n] ) ) ) );
gap> [ <mapping: Group( [ f1 ] ) -> Group( [ f1 ] ) >,
       <mapping: Group( [ f1 ] ) -> Group( [ f1 ] ) > ]
gap> GraphOfMapping(last[1]);
[ [ <identity> of ..., f1 ], [ f1, <identity> of ... ] ]
```

5 Some interesting nearrings

One motivation for creating SONATA was to study particular near-rings associated with a given group G: the **inner automorphism nearring** $I(G)$, the **automorphism nearring** $A(G)$, and the **endomorphism nearring** $E(G)$. The nearring $I(G)$ is the smallest subnearring of the nearring $M(G)$ of all mappings from G into G that contains all inner automorphisms; similarly $A(G)$ and $E(G)$ are defined. [Mel85] contains a lot of information on these near-rings.

5.1 Nearrings generated by endomorphisms on a group

Let us compute the nearring $I(A_4)$, which is the nearring of all zero-symmetric polynomial functions on the group A_4.

```
gap> I := InnerAutomorphismNearRing ( AlternatingGroup ( 4 ) );
InnerAutomorphismNearRing( Alt( [ 1 .. 4 ] ) )
gap> Size (I);
3072
```

For a polynomial function, we can ask for a polynomial that induces it.

```
gap> p := Random( I );
<mapping: AlternatingGroup( [ 1 .. 4 ] ) -> AlternatingGroup( [ 1 .. 4 ] ) >
gap> PrintAsTerm( p );
 - g1 + g2 - x - g2 + g1 + g2 + g1 - x + g2 - x + 2 * g1 -
3 * x - g1 + x + g2 - x - g2 + g1 + x - g1 + x - g1 + x +
g1 + x - g2 - x + g2 - g1 - x + g1 + x
gap> GeneratorsOfGroup( AlternatingGroup( 4 ) );
[ (1,2,3), (2,3,4) ]
```

We get a polynomial (not necessarily the shortest possible polynomial) that induces the polynomial function. The expressions `g1` and `g2` stand for the first and second generator of the group respectively.

Now we compute the nearring that is additively generated by the automorphisms of the dihedral group of order 8. This nearring is usually called $A(D_8)$.

```
gap> A := AutomorphismNearRing ( DihedralGroup ( 8 ) );
AutomorphismNearRing( <pc group of size 8 with 3 generators> )
gap> Size (A);
32
```

Much attention has been devoted to the nearring $E(S_4)$, which is the nearring additively generated by the endomorphisms on the symmetric group on four letters.

```
gap> EndS4 := EndomorphismNearRing ( SymmetricGroup ( 4 ) );
EndomorphismNearRing( Sym( [ 1 .. 4 ] ) )
gap> Size ( EndS4 );
927712935936
gap> F1 := last;;
gap> Collected ( Factors( F1 ));
[ [ 2, 35 ], [ 3, 3 ] ]
```

In the last example, we have computed the size of $E(S_4)$ as $2^{35} \cdot 3^3$.

We have also included some less popular examples of nearrings. One of those is the nearring $H(G, U)$. This is the nearring that is generated by all endomorphisms on G whose range lies in the subgroup U of G. We do an example on the group 16/8 in the classification of Thomas and Wood. It is a subdirectly irreducible group of order 16, and the factor modulo the monolith is isomorphic to the elementary abelian group of order 8.

```
gap> G := GTW16_8;
16/8
gap> U := First ( NormalSubgroups( G ), N -> Size(N) = 2 );
Group([ ( 1, 5)( 2,10)( 3,11)( 4,12)( 6,15)( 7,16)( 8, 9)(13,14) ])
gap> HGU := RestrictedEndomorphismNearRing (G, U);
RestrictedEndomorphismNearRing( 16/8, Group(
[ ( 1, 5)( 2,10)( 3,11)( 4,12)( 6,15)( 7,16)( 8, 9)(13,14) ]) )
gap> Size (HGU);
8
```

It is interesting to compare this nearring to the nearring of all functions e in the endomorphism nearring $E(G)$ with the property $e(G) \subseteq U$.

```
gap> EofG := EndomorphismNearRing ( G );
EndomorphismNearRing( 16/8 )
gap> EGU := NoetherianQuotient ( EofG, U, G );
NoetherianQuotient( Group(
[ ( 1, 5)( 2,10)( 3,11)( 4,12)( 6,15)( 7,16)( 8, 9)(13,14) ]) ,16/8 )
gap> Size ( EGU );
128
```

If N is a transformation nearring on G, and U, V are subsets of G then `NoetherianQuotient (N,U,V)` returns the collection of all mappings $f \in N$ such that $f(V) \subseteq U$.

5.2 More information than just the size

In this section, we use SONATA to produce some interesting information about the nearring $I(S_3)$, which is the nearring of all zero-symmetric polynomial functions on the group S_3.

```
gap> G := SymmetricGroup ( 3 );
Sym( [ 1 .. 3 ] )
gap> I := InnerAutomorphismNearRing ( G );
InnerAutomorphismNearRing( Sym( [ 1 .. 3 ] ) )
gap> Size( I );
54
```

Now we would like to see how many of these 54 functions are idempotent. First a complicated version.

```
gap> Filtered ( I,
>       t -> ForAll( G, g -> Image(t, g) = Image(t, Image(t, g)) ) );;
gap> Length( last );
18
```

Now a simpler version.

```
gap> Filtered ( I, i -> i^2 = i );;
gap> Length( last );
18
```

5.3 Centralizer nearrings

Let Φ be a subset of the endomorphisms of a group G. Then we define $M_\Phi(G)$ as the set of all mappings $m : G \to G$ that satisfy $m \circ \varphi = \varphi \circ m$ for all $\varphi \in \Phi$. This set is closed under addition and composition of mappings, and hence a subnearring of $M(G)$. The set $M_\Phi(G)$ is called the centralizer nearring of G determined by Φ. It need not necessarily be zero-symmetric.

In the following examples, we compute the centralizer nearring $M_{End(S_3)}(S_3)$.

```
gap> G := SymmetricGroup( 3 );
Sym( [ 1 .. 3 ] )
gap> endos := Endomorphisms( G );
[ [ (1,2,3), (1,2) ] -> [ (), () ], [ (1,2,3), (1,2) ] -> [ (), (1,3) ],
  [ (1,2,3), (1,2) ] -> [ (), (2,3) ], [ (1,2,3), (1,2) ] -> [ (), (1,2) ],
  [ (1,2,3), (1,2) ] -> [ (1,2,3), (1,3) ],
  [ (1,2,3), (1,2) ] -> [ (1,3,2), (1,2) ],
  [ (1,2,3), (1,2) ] -> [ (1,3,2), (1,3) ],
  [ (1,2,3), (1,2) ] -> [ (1,2,3), (2,3) ],
  [ (1,2,3), (1,2) ] -> [ (1,2,3), (1,2) ],
  [ (1,2,3), (1,2) ] -> [ (1,3,2), (2,3) ] ]
gap> C := CentralizerNearRing( G, endos );
CentralizerNearRing( Sym( [ 1 .. 3 ] ), ... )
gap> Size ( C );
6
```

An **ideal** of a nearring $(N, +, *)$ is a subset I such that I is a normal subgroup of $(N, +)$, and for all $i \in I$, $n, m \in N$, we have $(m + i) * n - m * n \in I$ and $n * i \in I$. Ideals are in one-to-one correspondence to the congruence relations on $(N, +, *)$.

Do you think that this nearring is simple? Alan Cannon does not think so, and, in fact, SONATA tells us:

```
gap> I := NearRingIdeals( C );
[ < nearring ideal >, < nearring ideal >, < nearring ideal >,
  < nearring ideal > ]
gap> List( I, Size );
[ 1, 2, 3, 6 ]
```

So, we have ideals of size 1,2,3 and 6.

5.4 Finding affine complete groups

We shall now construct all compatible (= congruence preserving) functions on the group 16/6 (Thomas-Wood-notation); this is the 6^{th} group of order 16 in [TW80]. It is the direct product of D_8 and C_2. Let G be this group. We first construct the nearring $P(G)$ of all polynomial functions. Then we construct all those functions that can be interpolated at every subset of G with at most two elements by a function in $P(G)$ by using the function `LocalInterpolationNearRing`: these are the compatible functions on G (see [Pil83]).

```
gap> P := PolynomialNearRing( GTW16_6 );
PolynomialNearRing( 16/6 )
gap> Size( P );
256
gap> C := LocalInterpolationNearRing(P, 2);
LocalInterpolationNearRing( PolynomialNearRing( 16/6 ), 2 )
gap> Size (C);
256
```

Hence the group 16/6 is 1-affine complete. A much faster algorithm for computing the nearring of compatible functions can be used.

```
gap> C := CompatibleFunctionNearRing( GTW16_6 );
< transformation nearring with 7 generators >
gap> Size(C);
256;
```

Finally, the fastest way to decide 1-affine completeness is to use the function `Is1AffineComplete`.

```
gap> Is1AffineComplete( GTW16_6 );
true
```

When studying polynomial functions on direct products of groups, it is important to know the smallest positive number l such that the zero-function can be expressed by a term $a_1 + e_1 \cdot x + a_2 + \cdots + e_n \cdot x + a_{n+1}$ with $\sum e_i = l$. This l has been called the **length** of the group by S.D.Scott.

```
gap> ScottLength( SymmetricGroup( 3 ) );
2
```

6 Ideals, factors, and direct products of nearrings

An **ideal** of a nearring $(N, +, *)$ is a subset I such that I is a normal subgroup of $(N, +)$, and for all $i \in I$, $n, m \in N$, we have $(m + i) * n - m * n \in I$ and $n * i \in I$. Ideals are in one-to-one correspondence to the congruence relations on $(N, +, *)$.

A **right ideal** of a nearring $(N, +, *)$ is a subset I such that I is a normal subgroup of $(N, +)$, and for all $i \in I$, $n, m \in N$, we have $(m + i) * n - m * n \in I$. Right ideals are in one-to-one correspondence to the congruence relations on $(N, +, \{\lambda_m | m \in M\})$, where $\lambda_m(n) := n * m$. Hence, right ideals describe the congruences of the N-group N_N.

A **left ideal** of a nearring $(N, +, *)$ is a subset I such that I is a normal subgroup of $(N, +)$, and for all $i \in I$, $n \in N$, we have $n * i \in I$.

6.1 Direct products

For all sorts of nearrings direct products $A \times B$ can be constructed. The result is again a nearring. In the case that both A and B are `TransformationNearRings`, the result will be a `TransformationNearRing` acting on the direct product of the groups A and B act on. In any other case the result is an `ExplicitMultiplicationNearRing`, even if one of the factors is a `TransformationNearRing`. In any case, the elements of a direct product are **not** pairs or tuples.

```
gap> A := LibraryNearRing( GTW8_2, 12 );
LibraryNearRing(8/2, 12)
gap> B := LibraryNearRing( GTW12_4, 13 );
LibraryNearRing(12/4, 13)
gap> D := DirectProductNearRing( A, B );
DirectProductNearRing( LibraryNearRing(8/2, 12),
  LibraryNearRing(12/4, 13) )
gap> SetName( D, "A x B" );
gap> D;
A x B
```

In this case the result is an `ExplicitMultiplicationNearRing`. It is a good idea to give a shorter name to the nearring D, because we will investigate one of its ideals in the next section.

6.2 Ideals and factors

We go on with the last example of the previous section and try to compute a left ideal which is generated by two elements, namely the second and the twenty-fifth in the sorted list of elements. The GAP function `list{[ poss ]}` constructs a list of those elements of the list `list` the position in the list `list` of which is in the list `poss`. For short, `elms{[2,25]}` is a list which contains the second and the twenty-fifth element of the list `elms`.

```
gap> elms := AsSortedList( D );;
gap> gens := elms{[2,25]};
[ (( 8, 9,10)), ((3,5)(4,6)) ]
gap> L := NearRingLeftIdealByGenerators( D, gens );
< nearring left ideal >
```

Now we can start investigating I. We can compute its size and test if it is an ideal.

```
gap> Size( L );
24
gap> IsNearRingRightIdeal( L );
true
gap> L;
< nearring ideal of size 24 >
```

So L is a two-sided ideal with 24 elements. Now we are getting interested in L. Is it a maximal ideal, what is the factor D/L?

```
gap> IsMaximalNearRingIdeal( L );
false
gap> F := D/L;
FactorNearRing( A x B, < nearring ideal of size 24 > )
gap> PrintTable( F, "am" );

  +   n0  n1  n2  n3
 --------------------
 n0   n0  n1  n2  n3
 n1   n1  n0  n3  n2
 n2   n2  n3  n0  n1
 n3   n3  n2  n1  n0

  *   n0  n1  n2  n3
 --------------------
 n0   n0  n0  n0  n0
 n1   n0  n0  n0  n0
 n2   n0  n0  n0  n0
 n3   n0  n0  n0  n0
```

Here, we use `PrintTable` with a second argument, because we do not want to see all the information. Here `a` stands for addition and `m` stands for multiplication table. For more options see the reference manual. Obviously, F is a constant nearring on a group of order 4. The additive group of the nearring is $\mathbb{Z}_2 \times \mathbb{Z}_2$. To make this fact more obvious, we choose other names (symbols) for the elements of the nearring and print the addition table again.

```
gap> IsElementaryAbelian( GroupReduct( F ) );
true
gap> # this would also convince us
gap> IsCyclic( GroupReduct( F ) );
false
gap> SetSymbols( F, ["(0,0)","(0,1)","(1,0)","(1,1)"] );
gap> PrintTable( F, "m" );

     *   (0,0)  (0,1)  (1,0)  (1,1)
  ----------------------------------
  (0,0)   (0,0)  (0,0)  (0,0)  (0,0)
  (0,1)   (0,0)  (0,0)  (0,0)  (0,0)
  (1,0)   (0,0)  (0,0)  (0,0)  (0,0)
  (1,1)   (0,0)  (0,0)  (0,0)  (0,0)
```

So F is the zero-ring on $\mathbb{Z}_2 \times \mathbb{Z}_2$, which is not simple, but we knew that before.

Of course all this operations can be applied to all nearrings.

7 Planar nearrings

We recall the definition of planar nearrings and basic results (see [Cla92]). Let $(N, +, \cdot)$ be a left nearring. For $a, b \in N$ we define $a \equiv b$ iff $a \cdot n = b \cdot n$ for all $n \in N$. If $a \equiv b$, then a and b are called **equivalent multipliers**. A nearring N is called **planar** if $|N/_{\equiv}| \geq 3$ and if for any two non-equivalent multipliers a and b in N, for any $c \in N$, the equation $a \cdot x = b \cdot x + c$ has a unique solution.

A **Ferrero pair** is a pair of finite groups (N, Φ) such that Φ is a fixed-point-free automorphism group of $(N, +)$.

Starting with a Ferrero pair (N, Φ) we can construct a planar nearring in the following way: Select representatives, say $e_1, \ldots, e_t$, for some or all of the non-trivial orbits of N under Φ. Let $C = \Phi(e_1) \cup \ldots \cup \Phi(e_t)$. For each $x \in N$ we define $a \cdot x = 0$ for $a \in N \setminus C$, and $a \cdot x = \phi_a(x)$ for $a \in \Phi(e_i) \subset C$ and $\phi_a(e_i) = a$. Then $(N, +, \cdot)$ is a (left) planar nearring with $|N/_{\equiv}| = |\Phi| + 1$.

Every finite planar nearring can be constructed from some Ferrero pair together with a set of orbit representatives in this way.

The problem: Find a planar nearring with 25 elements and 9 pairwise non-equivalent multipliers.

The solution: We follow the Ferrero method described above for defining a nearring multiplication on an additive group. First we have to find a fixed-point-free (fpf) automorphism group of order 8 on a group of order 25.

We start with the cyclic group of order 25: First of all we ask for the existence of an fpf automorphism group on `CyclicGroup(25)` by computing an upper bound for its order.

```
gap> FpfAutomorphismGroupsMaxSize( CyclicGroup(25) );
[ 4, 1 ]
```

This function returns a list with two integers, 4 and 1. The first number is an upper bound for the size of an fpf automorphism group; if there is a metacyclic fpf automorphism group, then it has a cyclic normal subgroup of index dividing the second number. These bounds are not sharp. If the upper bound for the size of an fpf automorphism group on some group is 1, we know that there is no nontrivial fpf automorphism group, no Ferrero pair, and no planar nearring on this group at all.

Here, SONATA does not exclude the possibility that the cyclic group of order 25 has an fpf automorphism group of order 4. However, we can be sure that all fpf automorphism groups are cyclic and that none of them has size 8.

Thus we have to consider the elementary abelian group of order 25 instead.

```
gap> FpfAutomorphismGroupsMaxSize( ElementaryAbelianGroup(25) );
[ 24, 2 ]
```

There might even exist an fpf automorphism group of order 24. (In fact there is more than one. The reference manual explains how to obtain all nearfields of size 25.) For our example, we could compute either a cyclic automorphism group or one isomorphic to the quaternion group with 8 elements. Let's try the latter.

```
gap> aux := FpfAutomorphismGroupsMetacyclic( [5,5], 4, -1 );
[ [ [ [ f1, f2 ] -> [ f1^2, f2^3 ], [ f1, f2 ] -> [ f2^4, f1 ] ] ],
  <pc group of size 25 with 2 generators> ]
```

Here, the function `FpfAutomorphismGroupsMetacyclic` determines the metacyclic fpf automorphism groups on `AbelianGroup([5,5])` with generators p, q satisfying $p^4 = 1, p^q = p^{-1}$, and $q^2 = p^2$. For each conjugacy class of such groups one representative is given. Conjugacy is determined within the whole automorphism group of `AbelianGroup([5,5])`. The actual output of the function is a list with 2 elements. The first is not the list of fpf groups up to conjugacy but the list of automorphisms p, q generating those groups. The second element is simply the group `AbelianGroup([5,5])`, on which the automorphisms act.

Since there is only one pair of generators p, q, all fpf automorphism groups isomorphic to the quaternion group are conjugate. Now, we have our Ferrero pair (G, Φ).

```
gap> phi := Group( aux[1][1] );
<group with 2 generators>
gap> G := aux[2];
<pc group of size 25 with 2 generators>
```

Next we have to pick some orbit representatives. We note that for a fixed Ferrero pair distinct choices of representatives may yield isomorphic nearrings. The function `OrbitRepresentativesForPlanarNearRing` returns exactly one set of representatives of given cardinality for each isomorphism class of planar nearrings which can be generated from (G, Φ).

```
gap> OrbitRepresentativesForPlanarNearRing( G, phi, 1 );
[ [ f1 ] ]
```

This tells us that all planar nearrings obtained from (G, Φ) with one orbit representative are in fact isomorphic. What happens if we choose 2 representatives?

```
gap> reps := OrbitRepresentativesForPlanarNearRing( G, phi, 2 );
[ [ f1, f1*f2 ], [ f1, f1^2*f2^2 ] ]
```

We obtain 2 non-isomorphic planar near-rings. Let's just construct one of them. The result will be an `ExplicitMultiplicationNearRing`.

```
gap> n := PlanarNearRing( G, phi, reps[1] );
ExplicitMultiplicationNearRing ( <pc group of size 25 with
2 generators> , multiplication )
```

How many non-isomorphic planar nearrings can be defined from our Ferrero pair (G, Φ) in total? Since there are 3 non-trivial orbits of Φ on G, we may choose up to 3 representatives.

```
gap> Length(OrbitRepresentativesForPlanarNearRing( G, phi, 3 ));
6
```

Summing all up, we get exactly 9 non-isomorphic planar nearrings with elementary abelian additive group of order 25 whose multiplication is defined using a quaternion group of fpf automorphisms.

8 Designs

Various designs can be obtained from nearrings. The design structure and basic functions for the manipulation of designs have been implemented within SONATA, as well as the means to construct the most popular nearring designs. Please see the reference manual for other ways to generate designs, e.g. by a set of blocks or by an incidence matrix.

8.1 Designs from planar nearrings

We are going to generate a design with a feasible parameter set from a planar nearring, respectively from a Ferrero pair. For notations and definitions we refer to [Cla92].

Let N be a (left) planar nearring. Then we can define a design $(N, B^*, \in)$ with N as the set of points and $\{N^* \cdot a + b | a, b \in N, a \neq 0\}$ as set of blocks. Here $N^* = \{x \in N | x \cdot N = N\}$. We note that such a design is always a BIB-design. Suppose that the planar nearring N is obtained from the Ferrero pair (G, Φ) as described in the previous chapter with $v := |G|$, $k := |\Phi|$. We can identify G and the additive group of the nearring N. Then $N^* \cdot a = \Phi(a)$ for all $a \in N$. The number of points of $(N, B^*, \in)$ is v, each block has size k, and any 2 distinct points are together incident with precisely $k - 1$ blocks. We say that $(N, B^*, \in)$ is a $2 - (v, k, k - 1)$ design. Note that k divides $v - 1$.

The problem: Find a $2 - (16, 5, 4)$ design $(N, B^*, \in)$ if possible.

The solution: Since our design should have 16 points, we need a planar nearring of size 16. Furthermore, this planar nearring has an underlying Ferrero pair with a fixed-point-free (fpf) automorphism group of order 5, since the blocks should be of that size.

Let (G, Φ) be such a Ferrero pair. We note that Φ is fpf on all characteristic subgroups of G, in particular, on the center of G. For our example the size of the center modulo 5 has to be congruent to 1. Therefore the center has order 16, and G is abelian. Since Φ is also fpf on any factor of G by a characteristic subgroup, Φ is fpf on the factor of G by its Frattini subgroup. Thus G is elementary abelian.

Now we compute an fpf automorphism group of order 5 on the elementary abelian group of order 16:

```
gap> aux := FpfAutomorphismGroupsCyclic( [2,2,2,2], 5 );
[ [ [ f1, f2, f3, f4 ] -> [ f4, f1*f2, f2*f3, f3*f4 ] ],
  <pc group of size 16 with 4 generators> ]
gap> a := aux[1][1];
[ f1, f2, f3, f4 ] -> [ f4, f1*f2, f2*f3, f3*f4 ]
gap> phi := Group( a );;
gap> G := aux[2];;
```

Up to conjugacy there is only one fpf automorphism group of order 5. We note that all planar nearrings obtained from a fixed Ferrero pair (G, Φ) yield the same design. We build the blocks $\Phi(a) + b$ for $a, b \in G, a \neq 0$, of $(N, B^*, \in)$ directly from (G, Φ) without actually generating a nearring.

```
gap> D := DesignFromFerreroPair( G, phi, "*" );
<a 2 - ( 16, 5, 4 ) nearring generated design>
```

The reference manual describes other options for `DesignFromFerreroPair` besides `"*"` and the correponding designs as well.

8.2 Information on designs

We investigate the design `D`, which we have generated from a planar nearring in the last section.

```
gap> D;
<a 2 - ( 16, 5, 4 ) nearring generated design>
gap> DesignParameter( D );
[ 2, 16, 48, 15, 5, 4 ]
```

`DesignParameter( D )` returns the set of parameters t, v, b, r, k, λ of the design `D`. Here there are 16 points, 48 blocks, every point is incident with precisely 15 blocks, every block is incident with precisely 5 points, every 2 distinct points are together incident with precisely 4 blocks. The design can be visualized by printing the incidence matrix. The rows are labelled by the points, the columns by the blocks. The point of number i is incident with the block of number j if and only if the entry in the i-th row, j-th column is 1.

```
gap> PrintIncidenceMat( D );
...1...1...11...1...1..1..11..1...1..1..1...11..
1......1...1..1.1...11...1..1..1...11..1..1....1
.1..1......11..1...1..1...11....1..1..11..1...1.
..1..1..1...1...1.1...1..1...11...1.1....11...1.
1....11..1......1..1.1....1.1...11...1...1..1.1.
.1..1.1...1..1......11....1..11....11....1.1.1..
..1..1.1.1...1...1...1...1.1....1.1...11...1.1..
..11...1..1.1..1..1.....1...1...1..1.1...1.1...1
..1.1...1.1...1..1.1.1.....1..1..1...1..1.1....1
1...1.1....1.1...11...1.1.......1.1.1...1...1..1
1...1...11....11....1..11....1....1..1.1...1..1.
.1...1..1.1.1....1.1...11...1..1....1..1....11..
.1.1....11...1.1....1.1..1.1...1.1.......1..1..1
.1.1..1....1..1..11....1.1..1.1..1....1....1..1.
..11..1..1....1.1..1..1.1....1.1...1..1.1....1..
1....1.1..1..1.1..1....1..1..1.1.1....1.1.1.....
```

By checking the entries in the incidence matrix we realize this design is circular, that is, any 2 distinct blocks have at most 2 points in common. Actually, this is not so easy to see. We prefer to do it like this:

```
gap> IsCircularDesign( D );
true
```

There are convenient functions to check which points are incident with a given set of blocks and vice versa.

```
gap> PointsIncidentBlocks( D, [2,7] );
[ 6, 14 ]
gap> BlocksIncidentPoints( D, [6,14] );
[ 2, 7, 31, 44 ]
```

Here the 6-th and the 14-th point are incident with the given blocks with numbers 2 and 7. The numbering corresponds to that of the columns and rows of the incidence matrix. Blocks 2, 7, 31, and 44 are incident with points 6 and 14.

The cardinalities of the intersections of one block with all the others, e.g. of the 4-th block with all 48 blocks are given like this.

```
gap> BlockIntersectionNumbers( D, 4 );
[ 0, 2, 2, 5, 0, 0, 2, 2, 1, 2, 1, 2, 2, 1, 2, 2, 2, 1, 2, 1, 2, 0,
  2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 2, 1, 2, 0, 2, 2, 0, 2, 2, 0, 2,
  2, 2, 1, 2 ]
```

Of course, all these functions can be applied to all kinds of designs no matter how they have been generated.

Bibliography

[CFF02] Celestina Cotti Ferrero and Giovanni Ferrero. *Nearrings. Some Developments Linked to Semigroups and Groups*. Kluwer Academic Publishers, Dordrecht, Boston, London, 2002.

[Cla92] James R. Clay. *Nearrings. Geneses and Applications*. Oxford University Press, Oxford, New York, Tokyo, 1992.

[Mel85] J. D. P. Meldrum. *Near-rings and their links with groups*. Pitman (Advanced Publishing Program), Boston, Mass., 1985.

[Pil83] Günter Pilz. *Near-Rings. The Theory and its Applications*, volume 23 of *North-Holland Mathematics Studies*. North-Holland Publishing Company, Amsterdam, New York, Oxford, revised edition, 1983.

[TW80] A. D. Thomas and G. V. Wood. *Group tables*. Shiva Publishing Ltd., Nantwich, 1980.

[Wäh87] Heinz Wähling. *Theorie der Fastkörper*. Thales Verlag, Essen, 1987.

Index

This index covers only this manual. A page number in *italics* refers to a whole section which is devoted to the indexed subject. Keywords are sorted with case and spaces ignored, e.g., "`PermutationCharacter`" comes before "permutation group".