Blocks of Character Tables

(Version 0.9.5)

Thomas Breuer

Thomas Breuer Email: sam@math.rwth-aachen.de
Homepage: http://www.math.rwth-aachen.de/~Thomas.Breuer

Copyright

Contents

Chapter 1

Introduction to the CTBlocks Package

The aims of the GAP 4 package CTBlocks are as follows.

- It provides general character theoretic GAP functions for questions about p-blocks, see Chapter 3.
- It provides special GAP functions whose aim is to check conditions that arise in reduction theorems in the context of the Alperin-McKay conjecture and of Alperin's weight conjecture.

If you use this package to solve a problem then please send a short email to sam@math.rwth-aachen.de about it. You can reference the package as follows.

```
@misc{ CTBlocks0.9.5,
  author =       {Breuer, T.},
  title =        {{CTBlocks}, Blocks of Character Tables,
                  {V}ersion 0.9.5},
  month =        {May},
  year =         {2022},
  note =         {\textsf{GAP} package},
  howpublished = {http://www.math.rwth-aachen.de/\~{}Thomas.Breuer/ctblocks}
}
```

For referencing the GAP system in general, use the entry [GAP19] in the bibliography of this manual, see also
http://www.gap-system.org.

1.1 Acknowledgements

The development of the package has been supported by the SFB-TRR 195 "Symbolic Tools in Mathematics and their Applications" (Project-ID 286237555, since 2017).

Several GAP functions, in particular those described in Section 3.8, have been written together with Erzsébet Horváth, and have been used already in the preparation of [BH01].

Thanks to Mohamed Barakat and Gabriel Navarro for discussions and suggestions.

Chapter 2

Tutorial for the CTBlocks Package

This chapter shows a few examples how to use the functionality provided by the CTBlocks package.

2.1 Block invariants of covers of sporadic simple groups

The aim of this section is to produce an overview of the non-cyclic, abelian, faithful blocks of sporadic simple groups and their covers, in a similar format as the tables that are available at `http://www.math.rwth-aachen.de/~Felix.Noeske/tabular.pdf`.

The function `DisplayBlockInvariants` (3.10.4) can be used to show an overview of block invariants.

(One short remark is necessary in advance. Depending on the terminal capabilities, the borders of the table printed by `DisplayBlockInvariants` (3.10.4) may contain non-ASCII characters. However, these characters are not supported by the LaTeX and HTML versions of GAPDoc documents. For the examples in this section, we set the user preference `DisplayFunction` (see Section (**AtlasRep: User preference DisplayFunction**)) to the value `"Print"`, in order to produce output consisting only of ASCII characters.)

Example

```
gap> origpref:= UserPreference( "AtlasRep", "DisplayFunction" );;
gap> SetUserPreference( "AtlasRep", "DisplayFunction", "Print" );
```

The simplest way to use `DisplayBlockInvariants` (3.10.4) is to call it with the identifier of an ordinary character table from GAP's library of character tables, for example the table of the double cover of the alternating group A_6 on six points.

Example

```
gap> DisplayBlockInvariants( "2.A6" );
Block invariants for 2.A6

-----------------------------------------
| p | b | d | k | l | c | a | n | f | r | sr |
-----------------------------------------
| 2 | 1 | 4 | 9 | 3 | - | - | - | + | + |  + |
|   | 2 | 1 | 2 | 1 | + | + | + | + | + |  - |
|   | 3 | 1 | 2 | 1 | + | + | + | + | + |  - |
| 3 | 1 | 2 | 6 | 4 | - | + | - | - | + |  + |
|   | 2 | 0 | 1 | 1 | + | + | + | - | + |  - |
```

```
|   | 3 | 2 | 6 | 4 | - | + | - | + | + |  + |
| 5 | 1 | 1 | 4 | 2 | + | + | - | - | + |  + |
|   | 2 | 0 | 1 | 1 | + | + | + | - | + |  - |
|   | 3 | 0 | 1 | 1 | + | + | + | - | + |  - |
|   | 4 | 0 | 1 | 1 | + | + | + | - | + |  - |
|   | 5 | 1 | 4 | 2 | + | + | - | + | + |  + |
|   | 6 | 0 | 1 | 1 | + | + | + | + | + |  - |
|   | 7 | 0 | 1 | 1 | + | + | + | + | + |  - |
-----------------------------------------------
```

The rows of the table correspond to the p-blocks of the table, for all prime divisors p of the group order. The columns of the table are labelled by

- `p`, `b`, `d`: the characteristic, the block number, and the defect of the block,
- `k` and `l`: the numbers of ordinary and modular irreducibles in the block,
- `c`, `a`, `n`: information whether the defect groups of the block are cyclic, abelian, and normal, respectively,
- `f`, `r`, `sr`: information whether the block is faithful, real, and strongly real, respectively.

We are interested in an overview that is on the one hand more general in the sense that several groups at handled at the same time, and which is on the other hand restricted to faithful blocks with abelian but noncyclic defect groups. As an example, we take the central extensions of the simple group Fi_{22}. (Here we fetch the character tables in the beginning. As we will see below, we will call the relevant functions a second time, and if we use the same character table object in the second round, the values in question are already known and can just be fetched.)

Example

```
gap> tbls:= List( [ "Fi22", "2.Fi22", "3.Fi22", "6.Fi22" ],
>                 CharacterTable );;
gap> DisplayBlockInvariants( tbls, IsFaithful, true,
>      IsBlockWithCyclicDefectGroup, false,
>      IsBlockWithAbelianDefectGroup, [ true, fail ] );
Block invariants
faithful
noncyclic defect group

-------------------------------------------------
| G      | p |   b | d |  k |  l | a | n | r | sr |
-------------------------------------------------
| Fi22   | 5 |   1 | 2 | 20 | 16 | + | - | + |  + |
| 2.Fi22 | 2 |   3 | 2 |  4 |  1 | + | - | + |  + |
|        | 5 |  39 | 2 | 20 | 16 | + | - | + |  + |
| 3.Fi22 | 3 |   2 | 2 |  9 |  2 | + | - | + |  + |
|        |   |   3 | 2 |  9 |  2 | + | - | + |  + |
|        | 5 |  39 | 2 | 20 | 16 | + | - | - |  - |
|        |   |  40 | 2 | 20 | 16 | + | - | - |  - |
| 6.Fi22 | 5 | 107 | 2 | 20 | 16 | + | - | - |  - |
|        |   | 108 | 2 | 20 | 16 | + | - | - |  - |
-------------------------------------------------
```

Note that we can in general not decide from the character table whether the defect groups are abelian, therefore we have to allow for the possible values `true` and `fail` as results of `IsBlockWithAbelianDefectGroup` (3.4.10).

This result is not yet good enough: We want to show a separate table for each prime p that divides the order of one of the groups. (And we do not want to print the header information for each prime.)

Example

```
gap> primes:= Union( List( tbls, t -> Set( Factors( Size( t ) ) ) ) );
[ 2, 3, 5, 7, 11, 13 ]
gap> for p in primes do
> DisplayBlockInvariants( tbls, IsFaithful, true,
>      IsBlockWithCyclicDefectGroup, false,
>      IsBlockWithAbelianDefectGroup, [ true, fail ],
>      UnderlyingCharacteristic, p,
>      rec( header:= [ "", Concatenation( "p = ", String( p ) ) ] ) );
> od;

p = 2

-----------------------------------------
| G      | b | d | k | l | a | n | r | sr |
-----------------------------------------
| 2.Fi22 | 3 | 2 | 4 | 1 | + | - | + |  + |
-----------------------------------------

p = 3

-----------------------------------------
| G      | b | d | k | l | a | n | r | sr |
-----------------------------------------
| 3.Fi22 | 2 | 2 | 9 | 2 | + | - | + |  + |
|        | 3 | 2 | 9 | 2 | + | - | + |  + |
-----------------------------------------

p = 5

-----------------------------------------------
| G      |   b | d |  k |  l | a | n | r | sr |
-----------------------------------------------
| Fi22   |   1 | 2 | 20 | 16 | + | - | + |  + |
| 2.Fi22 |  39 | 2 | 20 | 16 | + | - | + |  + |
| 3.Fi22 |  39 | 2 | 20 | 16 | + | - | - |  - |
|        |  40 | 2 | 20 | 16 | + | - | - |  - |
| 6.Fi22 | 107 | 2 | 20 | 16 | + | - | - |  - |
|        | 108 | 2 | 20 | 16 | + | - | - |  - |
-----------------------------------------------
```

Now we turn to all sporadic simple groups and their central extensions. First we collect the names of the character tables, in the same order as we need for the overview, that is, according to increasing order of the simple groups.

Example

```
gap> names:= [ "M11", "M12", "2.M12", "J1", "M22", "2.M22",
>     "3.M22", "4.M22", "6.M22", "12.M22", "J2", "2.J2",
```

```
>     "M23", "HS", "2.HS", "J3", "3.J3", "M24", "McL", "3.McL",
>     "He", "Ru", "2.Ru", "Suz", "2.Suz", "3.Suz", "6.Suz",
>     "ON", "3.ON", "Co3", "Co2", "Fi22", "2.Fi22", "3.Fi22",
>     "6.Fi22", "HN", "Ly", "Th", "Fi23", "Co1", "2.Co1",
>     "J4", "F3+", "3.F3+", "B", "2.B", "M" ];;
gap> tbls:= List( names, CharacterTable );;
```

The overview is printed in the same way as above.

Example

```
gap> primes:= Union( List( tbls, t -> Set( Factors( Size( t ) ) ) ) );;
gap> for p in primes do
> DisplayBlockInvariants( tbls, IsFaithful, true,
>      IsBlockWithCyclicDefectGroup, false,
>      IsBlockWithAbelianDefectGroup, [ true, fail ],
>      UnderlyingCharacteristic, p,
>      rec( header:= [ "", Concatenation( "p = ", String( p ) ) ] ) );
> od;

p = 2

-----------------------------------------
| G      | b | d | k | l | a | n | r | sr |
-----------------------------------------
| M12    | 2 | 2 | 4 | 3 | + | - | + |  + |
| J1     | 1 | 3 | 8 | 5 | + | - | + |  + |
| J2     | 2 | 2 | 4 | 3 | + | - | + |  + |
| HS     | 2 | 2 | 4 | 3 | + | - | + |  + |
| Ru     | 2 | 2 | 4 | 3 | + | - | + |  + |
| Co3    | 2 | 3 | 8 | 5 | + | - | + |  + |
| 2.Fi22 | 3 | 2 | 4 | 1 | + | - | + |  + |
| F3+    | 2 | 2 | 4 | 3 | + | - | + |  + |
-----------------------------------------

p = 3

-----------------------------------------------
| G      |  b | d |  k |  l | a | n | r | sr |
-----------------------------------------------
| M11    |  1 | 2 |  9 |  7 | + | - | + |  + |
| M22    |  1 | 2 |  6 |  5 | + | - | + |  + |
| 2.M22  |  6 | 2 |  6 |  5 | + | - | + |  + |
| 3.M22  |  2 | 2 |  9 |  2 | + | - | + |  + |
| 4.M22  | 10 | 2 |  6 |  5 | + | - | - |  - |
|        | 11 | 2 |  6 |  5 | + | - | - |  - |
| 6.M22  |  7 | 2 |  9 |  2 | + | - | + |  + |
| M23    |  1 | 2 |  9 |  7 | + | - | + |  + |
| HS     |  1 | 2 |  9 |  7 | + | - | + |  + |
|        |  2 | 2 |  9 |  7 | + | - | + |  + |
| 2.HS   |  7 | 2 |  9 |  5 | + | - | + |  + |
| 3.J3   |  2 | 2 |  9 |  2 | + | - | + |  + |
| He     |  2 | 2 |  9 |  7 | + | - | + |  + |
| Suz    |  2 | 2 |  6 |  5 | + | - | + |  + |
```

3.Suz	3	2	9	2	+	-	+	+
ON	1	4	18	14	+	-	+	+
	2	2	6	5	+	-	+	+
3.Fi22	2	2	9	2	+	-	+	+
	3	2	9	2	+	-	+	+
HN	2	2	9	7	+	-	+	+
Co1	3	2	9	5	+	-	+	+
J4	6	2	9	5	+	-	+	+
F3+	2	2	6	4	+	-	+	+
B	2	2	9	7	+	-	+	+
	3	2	9	7	+	-	+	+
	6	2	9	5	+	-	+	+

p = 5

G	b	d	k	l	a	n	r	sr
J2	1	2	14	6	+	-	+	+
2.J2	6	2	14	6	+	-	+	+
He	1	2	16	14	+	-	+	+
Suz	1	2	16	12	+	-	+	+
2.Suz	19	2	16	12	+	-	+	+
3.Suz	18	2	16	12	+	-	-	-
	19	2	16	12	+	-	-	-
6.Suz	59	2	16	12	+	-	-	-
	60	2	16	12	+	-	-	-
Fi22	1	2	20	16	+	-	+	+
2.Fi22	39	2	20	16	+	-	+	+
3.Fi22	39	2	20	16	+	-	-	-
	40	2	20	16	+	-	-	-
6.Fi22	107	2	20	16	+	-	-	-
	108	2	20	16	+	-	-	-
Fi23	1	2	20	16	+	-	+	+
	2	2	20	16	+	-	+	+
Co1	3	2	16	12	+	-	+	+
F3+	1	2	20	16	+	-	+	+
	2	2	16	14	+	-	+	+
	3	2	20	16	+	-	+	+
3.F3+	45	2	20	16	+	-	-	-
	46	2	20	16	+	-	-	-
	47	2	20	14	+	-	-	-
	48	2	20	14	+	-	-	-
B	2	2	20	16	+	-	+	+
	8	2	20	16	+	-	+	+
M	4	2	20	16	+	-	+	+

p = 7

```
| G     |  b | d |  k |  l | a | n | r | sr |
-----------------------------------------------
| Th    |  1 | 2 | 27 | 24 | + | - | + |  + |
| Co1   |  1 | 2 | 27 | 21 | + | - | + |  + |
| 2.Co1 | 46 | 2 | 27 | 21 | + | - | + |  + |
| B     |  1 | 2 | 27 | 24 | + | - | + |  + |
|       |  2 | 2 | 27 | 24 | + | - | + |  + |
|       |  4 | 2 | 27 | 21 | + | - | + |  + |
| 2.B   | 73 | 2 | 27 | 24 | + | - | + |  + |
| M     |  2 | 2 | 27 | 24 | + | - | + |  + |
-----------------------------------------------

p = 11

---------------------------------------
| G | b | d |  k |  l | a | n | r | sr |
---------------------------------------
| M | 1 | 2 | 50 | 45 | + | - | + |  + |
---------------------------------------
```

Note:

- The two 2-blocks of defect two for $12.M_{22}$ which are listed in `http://www.math.rwth-aachen.de/~Felix.Noeske/tabular.pdf` have a cyclic defect group (the central subgroup of order four) and are therefore not contained in the above output.
- The principal 2-block of J_1 which appears in the above output has been deliberately omitted from the table available in the web.

Finally, we reset the user preference.

Example

```
gap> SetUserPreference( "AtlasRep", "DisplayFunction", origpref );
```

2.2 More about abelian defect groups

We have seen in Section 2.1 that the character tables of the sporadic simple groups and their central extensions determine for all their p-blocks whether the defect groups are abelian: Only - entries occur in the column `a` of the tables shown in that section.

In general this is not the case. Let us look for an example where `IsBlockWithAbelianDefectGroup` (3.4.10) returns `fail`. (This computation takes several minutes.)

Example

```
gap> b:= OnePBlock( AllCharacterTableNames(),
>            IsBlockWithAbelianDefectGroup, fail );
Block( CharacterTable( "Isoclinic(2x2.F4(2).2)" ), 2, 2 )
gap> Defect( b );
3
gap> dcl:= ClassPositionsOfDefectGroupOfBlock( b );
```

```
[ 1, 2, 3, 4, 215, 216 ]
gap> t:= UnderlyingCharacterTable( b );;
gap> OrdersClassRepresentatives( t ){ dcl };
[ 1, 4, 2, 4, 2, 4 ]
```

In this example, each defect group in question has order eight and contains at least two involutions and at least three elements of order four. This excludes the two types of nonabelian groups of order eight, hence the defect groups are abelian.

We see that we could easily extend `IsBlockWithAbelianDefectGroup` (3.4.10) such that `true` is returned in the above situation. Similarly, the lattice of normal subgroups can be used in some cases to conclude that the defect groups in question are abelian. However, there are also examples of simple groups for which `IsBlockWithAbelianDefectGroup` (3.4.10) fails.

Example

```
gap> b:= OnePBlock( AllCharacterTableNames( IsSimple, true ),
>            IsBlockWithAbelianDefectGroup, fail );
Block( CharacterTable( "L8(2)" ), 3, 5 )
gap> Defect( b );
3
```

2.3 Examples of block induction

2.3.1 Examples of block induction: Separating examples

Block induction in the sense of Brauer (see `BrauerCorrespondent` (3.8.1)) implies p-regular block induction (see `PRegularCorrespondent` (3.8.2)), which implies extended block induction (see `WheelerCorrespondent` (3.8.4)), and also block induction in the sense of Alperin-Burry (see `AlperinBurryCorrespondent` (3.8.3)) implies extended block induction.

We list those examples from [BH01] that separate these concepts of block induction.

A series of examples where block induction in the sense of Brauer is defined but block induction in the sense of Alperin-Burry is not defined is given in [BH01, Example 2.2]. The smallest member of this series is the dihedral group G of order 24 and H is its 2-core (a cyclic group of order four).

Example

```
gap> g:= DihedralGroup( 24 );;
gap> p:= 2;;
gap> t:= CharacterTable( g );;
gap> h:= PCore( g, 2 );;
gap> Size( h );
4
gap> s:= CharacterTable( h );;
gap> AlperinBurryCorrespondent( s, t, p, 1 );
fail
gap> BrauerCorrespondent( s, t, p, 1 );
1
gap> PRegularCorrespondent( s, t, p, 1 );
1
gap> WheelerCorrespondent( s, t, p, 1 );
1
```

An example where block induction in the sense of Brauer and block induction in the sense of Alperin-Burry are not defined but p-regular block induction is defined is given by $G = S_3$, the symmetric group on three points, and H its trivial subgroup, for $p = 2$.

Example

```
gap> g:= SymmetricGroup( 3 );;
gap> p:= 2;;
gap> t:= CharacterTable( g );;
gap> h:= TrivialSubgroup( g );;
gap> s:= CharacterTable( h );;
gap> AlperinBurryCorrespondent( s, t, p, 1 );
fail
gap> BrauerCorrespondent( s, t, p, 1 );
fail
gap> PRegularCorrespondent( s, t, p, 1 );
1
gap> WheelerCorrespondent( s, t, p, 1 );
1
```

A series of examples where p-regular block induction is not defined but block induction in the sense of Alperin-Burry is defined is given in [BH01, Example 2.3]. One instance of this series is the alternating group G on four points where H has order three and $p = 2$ holds.

Example

```
gap> g:= AlternatingGroup( 4 );;
gap> p:= 2;;
gap> t:= CharacterTable( g );;
gap> h:= SylowSubgroup( g, 3 );;
gap> s:= CharacterTable( h );;
gap> AlperinBurryCorrespondent( s, t, p, 1 );
1
gap> BrauerCorrespondent( s, t, p, 1 );
fail
gap> PRegularCorrespondent( s, t, p, 1 );
fail
gap> WheelerCorrespondent( s, t, p, 1 );
1
```

An example where p-regular block induction and block induction in the sense of Alperin-Burry are defined but block induction in the sense of Brauer is not defined is given by the cyclic group G of order $p = 2$ and H its trivial subgroup.

Example

```
gap> g:= CyclicGroup( 2 );;
gap> p:= 2;;
gap> t:= CharacterTable( g );;
gap> h:= TrivialSubgroup( g );;
gap> s:= CharacterTable( h );;
gap> AlperinBurryCorrespondent( s, t, p, 1 );
1
gap> BrauerCorrespondent( s, t, p, 1 );
fail
gap> PRegularCorrespondent( s, t, p, 1 );
```

```
1
gap> WheelerCorrespondent( s, t, p, 1 );
1
```

An example where extended block induction is defined but neither p-regular block induction nor block induction in the sense of Alperin-Burry are defined is given by [Whe94, Example 2.10]. We have G the simple group of order 168, $p = 3$, and H is a Sylow 2-subgroup of G.

Example

```
gap> g:= PSL(2,7);;
gap> Size( g );  IsSimple( g );
168
true
gap> p:= 3;;
gap> t:= CharacterTable( g );;
gap> h:= SylowSubgroup( g, 2 );;
gap> s:= CharacterTable( h );;
gap> AlperinBurryCorrespondent( s, t, p, 1 );
fail
gap> BrauerCorrespondent( s, t, p, 1 );
fail
gap> PRegularCorrespondent( s, t, p, 1 );
fail
gap> WheelerCorrespondent( s, t, p, 1 );
4
```

(The smallest such example is the symmetric group G on three points, its trivial subgroup H, and $p = 2$.)

2.3.2 Examples of block induction: The Mathieu groups M_{11} and M_{12}, for $p = 2$

We verify the results of [LP10, Example 4.7.8].

First we show that for $p = 2$, block induction in the sense of Brauer (see `BrauerCorrespondent` (3.8.1)) is defined from a p-block b of a proper subgroup H of the Mathieu group M_{11} if and only if H has even order and b is the principal block of H.

Example

```
gap> tom:= TableOfMarks( "M11" );;
gap> g:= UnderlyingGroup( tom );;
gap> p:= 2;;
gap> good:= [];;
gap> bad:= [];;
gap> for i in [ 1 .. Length( OrdersTom( tom ) ) - 1 ] do
>   h:= RepresentativeTom( tom, i );
>   tblh:= CharacterTable( h );
>   for b in [ 1 .. Length( PrimeBlocks( tblh, p ).defect ) ] do
>     if BrauerCorrespondent( tblh, CharacterTable( g ), p, b ) <> fail then
>       AddSet( good, [ Size( h ), b ] );
>     else
>       AddSet( bad, [ Size( h ), b ] );
>     fi;
>   od;
```

```
> od;
gap> good;
[ [ 2, 1 ], [ 4, 1 ], [ 6, 1 ], [ 8, 1 ], [ 10, 1 ], [ 12, 1 ],
  [ 16, 1 ], [ 18, 1 ], [ 20, 1 ], [ 24, 1 ], [ 36, 1 ], [ 48, 1 ],
  [ 60, 1 ], [ 72, 1 ], [ 120, 1 ], [ 144, 1 ], [ 360, 1 ],
  [ 660, 1 ], [ 720, 1 ] ]
gap> bad;
[ [ 1, 1 ], [ 3, 1 ], [ 3, 2 ], [ 3, 3 ], [ 5, 1 ], [ 5, 2 ],
  [ 5, 3 ], [ 5, 4 ], [ 5, 5 ], [ 6, 2 ], [ 6, 3 ], [ 9, 1 ],
  [ 9, 2 ], [ 9, 3 ], [ 9, 4 ], [ 9, 5 ], [ 9, 6 ], [ 9, 7 ],
  [ 9, 8 ], [ 9, 9 ], [ 10, 2 ], [ 10, 3 ], [ 11, 1 ], [ 11, 2 ],
  [ 11, 3 ], [ 11, 4 ], [ 11, 5 ], [ 11, 6 ], [ 11, 7 ], [ 11, 8 ],
  [ 11, 9 ], [ 11, 10 ], [ 11, 11 ], [ 12, 2 ], [ 18, 2 ], [ 18, 3 ],
  [ 18, 4 ], [ 18, 5 ], [ 18, 6 ], [ 20, 2 ], [ 36, 2 ], [ 36, 3 ],
  [ 36, 4 ], [ 55, 1 ], [ 55, 2 ], [ 55, 3 ], [ 55, 4 ], [ 55, 5 ],
  [ 55, 6 ], [ 55, 7 ], [ 60, 2 ], [ 72, 2 ], [ 72, 3 ], [ 120, 2 ],
  [ 144, 2 ], [ 360, 2 ], [ 360, 3 ], [ 660, 2 ], [ 660, 3 ],
  [ 660, 4 ], [ 720, 2 ] ]
```

Next we inspect for which 2-blocks of cyclic subgroups of order six in the Mathieu group M_{12}, block induction in the sense of Brauer is defined.

Example

```
gap> tom:= TableOfMarks( "M12" );;
gap> g:= UnderlyingGroup( tom );;
gap> p:= 2;;
gap> results:= [];;
gap> for i in Positions( OrdersTom( tom ), 6 ) do
>   h:= RepresentativeTom( tom, i );
>   if IsCyclic( h ) then
>     tblh:= CharacterTable( h );
>     Add( results, List( [ 1 .. Length( PrimeBlocks( tblh, p ).defect ) ],
>            b -> BrauerCorrespondent( tblh, CharacterTable( g ), p, b ) ) );
>   fi;
> od;
gap> results;
[ [ 1, fail, fail ], [ fail, 2, 2 ] ]
```

We see that there are two classes of such subgroups in M_{12}, and block induction is defined exactly for the principal block of the groups in the first class, whereas block induction is defined exactly for the two non-principal blocks of the groups in the second class.

2.4 Examples about reality questions

2.4.1 Example: Strongly real classes of sporadic simple groups

We compute which real classes of sporadic simple groups are not strongly real.

Example

```
gap> names:= AllCharacterTableNames( IsSporadicSimple, true,
>                IsDuplicateTable, false );
[ "B", "Co1", "Co2", "Co3", "F3+", "Fi22", "Fi23", "HN", "HS", "He",
```

```
  "J1", "J2", "J3", "J4", "Ly", "M", "M11", "M12", "M22", "M23",
  "M24", "McL", "ON", "Ru", "Suz", "Th" ]
gap> for name in names do
>      t:= CharacterTable( name );
>      test:= Filtered( [ 1 .. NrConjugacyClasses( t ) ],
>                       i -> IsRealClass( t, i )
>                            and not IsStronglyRealClass( t, i ) );
>      if test <> [] then
>        Print( name, ":  ", ClassNames( t, "Atlas" ){ test }, "\n" );
>      fi;
>    od;
Co2:  [ "16B" ]
HN:  [ "8A" ]
M:  [ "8C", "8F", "24F", "24G", "24H", "24J", "32A", "32B", "40A",
  "48A" ]
M22:  [ "8A" ]
M23:  [ "8A" ]
McL:  [ "3A", "5A", "6A", "8A", "10A", "12A" ]
Th:  [ "8B" ]
```

The article [Sul08] claims to list the real classes in sporadic simple groups that are not strongly real. However, the classes of element order 8, 10, and 12 in McL are missing in that list, and the classes 16AB, 22BC, 23AB of Fi_{23} are erroneously claimed to be real.

Let us look at these cases.

The group McL has a unique class of involutions, and only elements of order up to 6 are products of at most two involutions. On the other hand, six more classes of McL are real.

Example

```
gap> t:= CharacterTable( "McL" );
CharacterTable( "McL" )
gap> orders:= OrdersClassRepresentatives( t );
[ 1, 2, 3, 3, 4, 5, 5, 6, 6, 7, 7, 8, 9, 9, 10, 11, 11, 12, 14, 14,
  15, 15, 30, 30 ]
gap> nccl:= Length( orders );
24
gap> Filtered( [ 1 .. nccl ],
>        i -> ClassMultiplicationCoefficient( t, 2, 2, i ) <> 0 );
[ 1, 2, 4, 5, 7, 9 ]
gap> Filtered( [ 1 .. nccl ], i -> PowerMap( t, -1, i ) = i );
[ 1, 2, 3, 4, 5, 6, 7, 8, 9, 12, 15, 18 ]
```

The classes 16AB, 22BC, 23AB of Fi_{23} are not real.

Example

```
gap> t:= CharacterTable( "Fi23" );
CharacterTable( "Fi23" )
gap> orders:= OrdersClassRepresentatives( t );;
gap> filt:= PositionsProperty( orders, x -> x in [ 16, 22, 23 ] );
[ 63, 64, 77, 78, 79, 80, 81 ]
gap> PowerMap( t, -1 ){ filt };
[ 64, 63, 77, 79, 78, 81, 80 ]
gap> ClassNames( t, "Atlas" ){ filt };
[ "16A", "16B", "22A", "22B", "22C", "23A", "23B" ]
```

(Note that the irrational values in the six classes lie in quadratic number fields. For the element orders 22 and 23, it is clear from this fact that the classes cannot be real, because the fields of 11-th and 23-rd roots of unity do not have a real quadratic subfield.)

2.4.2 Example: Strongly real blocks of sporadic simple groups

We compute which real p-blocks of sporadic simple groups are not strongly real.

Example

```
gap> names:= AllCharacterTableNames( IsSporadicSimple, true,
>                IsDuplicateTable, false );
[ "B", "Co1", "Co2", "Co3", "F3+", "Fi22", "Fi23", "HN", "HS", "He",
  "J1", "J2", "J3", "J4", "Ly", "M", "M11", "M12", "M22", "M23",
  "M24", "McL", "ON", "Ru", "Suz", "Th" ]
gap> bl:= AllPBlocks( names, IsRealBlock, true,
>             IsStronglyRealBlock, false );
[ Block( CharacterTable( "McL" ), 2, 4 ) ]
```

We get exactly one example, a 2-block of defect one for the McLaughlin group McL.

Example

```
gap> Defect( bl[1] );
1
gap> dcl:= ClassPositionsOfDefectClasses( bl[1] );
[ 6, 10, 11, 21, 22 ]
gap> tbl:= UnderlyingCharacterTable( bl[1] );;
gap> ClassNames( tbl, "Atlas" ){ dcl };
[ "5A", "7A", "7B", "15A", "15B" ]
gap> Filtered( dcl, c -> IsRealClass( tbl, c ) );
[ 6 ]
gap> Filtered( dcl, c -> IsStronglyRealClass( tbl, c ) );
[  ]
```

This block has exactly one real defect class, which is not strongly real. (The existence of a real defect class for each 2-block is guaranteed by [GM00].)

2.5 Examples about the Loewy length of centres of blocks

2.5.1 Example: Loewy lengths of centres of blocks of sporadic simple groups

For an algebra A, let $J(A)$ denote the Jacobson radical of A, that is, the largest nilpotent ideal of A. We consider the chain $A = J(A)^0 \supset J(A) \supset J(A)^2 \supset \cdots \supset J(A)^{n-1} \supset J(A)^n = \{0\}$ of ideals. Then n is called the *Loewy length* of A. The factors J^{i-1}/J^i, for $1 \leq i \leq n$, are called the *Loewy layers* of A.

The following functions compute, for a given algebra A and its radical $J = J(A)$, the list of dimensions of the radical powers $J(A)^i$ and of the Loewy layers of A, respectively.

Example

```
gap> DimensionsOfRadicalPowers:= function( A, J )
>     local l, Jpower;
>
>     l:= [ Dimension( J ) ];
>     Jpower:= J;
```

```
>     while Dimension( Jpower ) <> 0 do
>       Jpower:= ProductSpace( Jpower, J );
>       Add( l, Dimension( Jpower ) );
>     od;
>
>     return l;
> end;;
gap> DimensionsOfLoewyLayers:= function( A, J )
>     local dims, l, i;
>
>     if Dimension( A ) = 0 then
>       return [];
>     fi;
>
>     dims:= DimensionsOfRadicalPowers( A, J );
>     l:= [ Dimension( A ) - dims[1] ];
>     for i in [ 2 .. Length( dims ) ] do
>       l[i]:= dims[ i-1 ] - dims[i];
>     od;
>
>     return l;
> end;;
```

We are interested in the dimensions of the Loewy layers of the centres of the principal p-blocks of group algebras FG, where G is a sporadic simple group and F is a field of characteristic p. We are also interested in the dimensions of the Loewy layers of the centre of (the principal block of) $FN_G(P)$, where P is a Sylow p-subgroup of G.

(Note that these dimensions do not depend on the field F, as long as the structure constants of the algebras live in F. In our situation, we may choose F as a prime field, since the block idempotent of a principal block has always rational coefficients.)

Most of the relevant information is available in GAP's library of character tables, thus we will use the character theoretic variant of `SCAlgebraCentreOfBlock` (3.11.3).

Several normalizers of Sylow p-subgroups in sporadic simple groups have hundreds of conjugacy classes, and the computation of the radical of an algebra of that dimension is time consuming. Therefore, we use the fact that the Jacobson radical of the centre $Z(FG)$ of an indecomposable group algebra FG is equal to the augmentation ideal of $Z(FG)$ and thus has an F-basis consisting of the elements $C^+ - |C|1$, where C runs over the nonidentity conjugacy classes of G, and C^+ is the sum of the elements of C. The following function can be used for that.

Example

```
gap> RadicalOfIndecomposableGroupAlgebra:= function( A, tbl )
>     local radgens, gens, classlengths, i;
>
>     radgens:= [];
>     gens:= BasisVectors( CanonicalBasis( A ) );
>     classlengths:= SizesConjugacyClasses( tbl );
>     for i in [ 2 .. Length( gens ) ] do
>       Add( radgens, gens[i] - classlengths[i] * gens[1] );
>     od;
>
```

```
>     return SubalgebraNC( A, radgens, "basis" );
> end;;
```

What we want to compute is done by the following function. It assumes that the (ordinary) character table of the group G is available as `tbl`.

Example

```
gap> DimensionsOfLoewyLayersByTable:= function( tbl, p )
>      local A, J;
>
>      if Length( PrimeBlocks( tbl, p ).defect ) = 1 then
>          # The group algebra is indecomposable.
>          A:= SCAlgebraCentreOfGroupAlgebra( tbl, p );
>          J:= RadicalOfIndecomposableGroupAlgebra( A, tbl );
>        else
>          # Use the simpleminded variant.
>          A:= SCAlgebraCentreOfBlock( tbl, p, 1 );
>          J:= RadicalOfAlgebra( A );
>        fi;
>        return DimensionsOfLoewyLayers( A, J );
> end;;
```

Some of these computations have been carried out in [Sch16], with different methods, and [Sch16, Conjecturere 5.8.1] states that for any sporadic simple group G and any prime p dividing $|G|$, the Loewy length of $Z(B)$ is expected to be an upper bound for the Loewy length of $Z(FN_G(P))$, where B is the principal block of FG, F is a field of characteristic p, and P is a Sylow p-subgroup of G. We check this conjecture as far as the available data admit this.

Example

```
gap> conjecture_is_wrong_for:= [];;
gap> conjecture_is_open_for:= [];;
gap> names:= AllCharacterTableNames( IsSporadicSimple, true,
>                IsDuplicateTable, false : OrderedBy:= Size );
[ "M11", "M12", "J1", "M22", "J2", "M23", "HS", "J3", "M24", "McL",
  "He", "Ru", "Suz", "ON", "Co3", "Co2", "Fi22", "HN", "Ly", "Th",
  "Fi23", "Co1", "J4", "F3+", "B", "M" ]
gap> for name in names do
>      Print( name, ":\n" );
>      t:= CharacterTable( name );
>      for p in Set( Factors( Size( t ) ) ) do
>        dimst:= DimensionsOfLoewyLayersByTable( t, p );
>        Print( "  ", String( p, 2 ), ":  ", String( dimst, -30 ) );
>        nname:= Concatenation( name, "N", String( p ) );
>        n:= CharacterTable( nname );
>        if n = fail then
>          Print( "(no Sylow normalizer table)\n" );
>          Add( conjecture_is_open_for, [ name, p ] );
>        else
>          dimsn:= DimensionsOfLoewyLayersByTable( n, p );
>          if dimsn = dimst then
>            Print( "same dimensions for G and N_G(P)\n" );
>          else
>            Print( dimsn, "\n" );
```

```
>          fi;
>          # Check the conjecture.
>          if Length( dimst ) < Length( dimsn ) then
>            Add( conjecture_is_wrong_for, [ name, p ] );
>          fi;
>        fi;
>      od;
>    od;
M11:
   2:  [ 1, 3, 2, 1, 1 ]            [ 1, 4, 1, 1 ]
   3:  [ 1, 7, 1 ]                  same dimensions for G and N_G(P)
   5:  [ 1, 4 ]                     same dimensions for G and N_G(P)
  11:  [ 1, 5, 1 ]                  same dimensions for G and N_G(P)
M12:
   2:  [ 1, 7, 2, 1 ]               [ 1, 14, 1 ]
   3:  [ 1, 8, 2 ]                  [ 1, 9, 1 ]
   5:  [ 1, 4 ]                     same dimensions for G and N_G(P)
  11:  [ 1, 5, 1 ]                  same dimensions for G and N_G(P)
J1:
   2:  [ 1, 7 ]                     same dimensions for G and N_G(P)
   3:  [ 1, 2 ]                     same dimensions for G and N_G(P)
   5:  [ 1, 2, 1 ]                  same dimensions for G and N_G(P)
   7:  [ 1, 6 ]                     same dimensions for G and N_G(P)
  11:  [ 1, 10 ]                    same dimensions for G and N_G(P)
  19:  [ 1, 6, 1, 1 ]               same dimensions for G and N_G(P)
M22:
   2:  [ 1, 9, 2 ]                  [ 1, 16 ]
   3:  [ 1, 5 ]                     same dimensions for G and N_G(P)
   5:  [ 1, 4 ]                     same dimensions for G and N_G(P)
   7:  [ 1, 3, 1 ]                  same dimensions for G and N_G(P)
  11:  [ 1, 5, 1 ]                  same dimensions for G and N_G(P)
J2:
   2:  [ 1, 12, 3, 1 ]              [ 1, 12, 6 ]
   3:  [ 1, 8, 4 ]                  [ 1, 9, 3 ]
   5:  [ 1, 4, 5, 3, 1 ]            same dimensions for G and N_G(P)
   7:  [ 1, 6 ]                     same dimensions for G and N_G(P)
M23:
   2:  [ 1, 12, 2 ]                 [ 1, 16 ]
   3:  [ 1, 7, 1 ]                  same dimensions for G and N_G(P)
   5:  [ 1, 4 ]                     same dimensions for G and N_G(P)
   7:  [ 1, 3, 1 ]                  same dimensions for G and N_G(P)
  11:  [ 1, 5, 1 ]                  same dimensions for G and N_G(P)
  23:  [ 1, 11, 1 ]                 same dimensions for G and N_G(P)
HS:
   2:  [ 1, 13, 4, 1, 1 ]           [ 1, 30, 4 ]
   3:  [ 1, 7, 1 ]                  same dimensions for G and N_G(P)
   5:  [ 1, 13, 3 ]                 [ 1, 14, 2 ]
   7:  [ 1, 6 ]                     same dimensions for G and N_G(P)
  11:  [ 1, 5, 1 ]                  same dimensions for G and N_G(P)
J3:
   2:  [ 1, 12, 3, 1 ]              [ 1, 12, 6 ]
   3:  [ 1, 15 ]                    same dimensions for G and N_G(P)
```

```
   5:  [ 1, 2, 1 ]                 same dimensions for G and N_G(P)
  17:  [ 1, 8, 1 ]                 same dimensions for G and N_G(P)
  19:  [ 1, 9, 1 ]                 same dimensions for G and N_G(P)
M24:
   2:  [ 1, 18, 5, 1, 1 ]          [ 1, 54, 6 ]
   3:  [ 1, 9, 3 ]                 [ 1, 10, 2 ]
   5:  [ 1, 4 ]                    same dimensions for G and N_G(P)
   7:  [ 1, 3, 1 ]                 same dimensions for G and N_G(P)
  11:  [ 1, 10 ]                   same dimensions for G and N_G(P)
  23:  [ 1, 11, 1 ]                same dimensions for G and N_G(P)
McL:
   2:  [ 1, 12, 4, 1 ]             [ 1, 16 ]
   3:  [ 1, 15, 4, 1 ]             [ 1, 16, 4 ]
   5:  [ 1, 14, 4 ]                [ 1, 15, 3 ]
   7:  [ 1, 3, 1 ]                 same dimensions for G and N_G(P)
  11:  [ 1, 5, 1 ]                 same dimensions for G and N_G(P)
He:
   2:  [ 1, 18, 5, 1, 1 ]          [ 1, 54, 6 ]
   3:  [ 1, 9, 3 ]                 [ 1, 10, 2 ]
   5:  [ 1, 14, 1 ]                same dimensions for G and N_G(P)
   7:  [ 1, 19, 3 ]                [ 1, 21, 1 ]
  17:  [ 1, 8, 1 ]                 same dimensions for G and N_G(P)
Ru:
   2:  [ 1, 30, 1 ]                [ 1, 83, 1 ]
   3:  [ 1, 9, 4 ]                 [ 1, 10, 3 ]
   5:  [ 1, 20, 4 ]                [ 1, 21, 3 ]
   7:  [ 1, 6 ]                    same dimensions for G and N_G(P)
  13:  [ 1, 12 ]                   same dimensions for G and N_G(P)
  29:  [ 1, 14, 1 ]                same dimensions for G and N_G(P)
Suz:
   2:  [ 1, 26, 9, 1, 1 ]          [ 1, 59, 21 ]
   3:  [ 1, 22, 8, 2, 1 ]          [ 1, 24, 11, 3 ]
   5:  [ 1, 10, 5 ]                same dimensions for G and N_G(P)
   7:  [ 1, 6 ]                    same dimensions for G and N_G(P)
  11:  [ 1, 10 ]                   same dimensions for G and N_G(P)
  13:  [ 1, 6, 1 ]                 same dimensions for G and N_G(P)
ON:
   2:  [ 1, 16, 2, 1 ]             [ 1, 27, 4 ]
   3:  [ 1, 14, 3 ]                same dimensions for G and N_G(P)
   5:  [ 1, 4 ]                    same dimensions for G and N_G(P)
   7:  [ 1, 20, 3 ]                [ 1, 21, 2 ]
  11:  [ 1, 10 ]                   same dimensions for G and N_G(P)
  19:  [ 1, 6, 1, 1 ]              same dimensions for G and N_G(P)
  31:  [ 1, 15, 1 ]                same dimensions for G and N_G(P)
Co3:
   2:  [ 1, 16, 11, 3, 1 ]         [ 1, 49, 11 ]
   3:  [ 1, 28, 8, 2 ]             [ 1, 35, 15, 1 ]
   5:  [ 1, 20, 5 ]                [ 1, 21, 4 ]
   7:  [ 1, 6 ]                    same dimensions for G and N_G(P)
  11:  [ 1, 5, 1 ]                 same dimensions for G and N_G(P)
  23:  [ 1, 11, 1 ]                same dimensions for G and N_G(P)
Co2:
```

```
   2:  [ 1, 36, 16, 4, 1, 1 ]        [ 1, 310, 82, 1 ]
   3:  [ 1, 24, 16, 5, 1 ]           [ 1, 30, 12, 2 ]
   5:  [ 1, 21, 5 ]                  [ 1, 22, 4 ]
   7:  [ 1, 6 ]                      same dimensions for G and N_G(P)
  11:  [ 1, 10 ]                     same dimensions for G and N_G(P)
  23:  [ 1, 11, 1 ]                  same dimensions for G and N_G(P)
Fi22:
   2:  [ 1, 38, 14, 6, 1, 1, 1 ]     [ 1, 188, 24, 2 ]
   3:  [ 1, 31, 17, 7, 1, 1 ]        [ 1, 93, 9 ]
   5:  [ 1, 13, 6 ]                  same dimensions for G and N_G(P)
   7:  [ 1, 6 ]                      same dimensions for G and N_G(P)
  11:  [ 1, 5, 1 ]                   same dimensions for G and N_G(P)
  13:  [ 1, 6, 1 ]                   same dimensions for G and N_G(P)
HN:
   2:  [ 1, 36, 7, 1 ]               [ 1, 58, 15 ]
   3:  [ 1, 24, 7, 1 ]               [ 1, 23, 6 ]
   5:  [ 1, 34, 9, 1 ]               [ 1, 64, 3 ]
   7:  [ 1, 6 ]                      same dimensions for G and N_G(P)
  11:  [ 1, 10 ]                     same dimensions for G and N_G(P)
  19:  [ 1, 9, 1 ]                   same dimensions for G and N_G(P)
Ly:
   2:  [ 1, 13, 8, 3 ]               [ 1, 20, 1 ]
   3:  [ 1, 29, 9, 2, 1 ]            [ 1, 30, 16, 4 ]
   5:  [ 1, 39, 7, 1 ]               [ 1, 52, 5 ]
   7:  [ 1, 6 ]                      same dimensions for G and N_G(P)
  11:  [ 1, 5, 1 ]                   same dimensions for G and N_G(P)
  31:  [ 1, 6, 1, 1, 1, 1 ]          same dimensions for G and N_G(P)
  37:  [ 1, 18, 1 ]                  same dimensions for G and N_G(P)
  67:  [ 1, 22, 1, 1 ]               same dimensions for G and N_G(P)
Th:
   2:  [ 1, 41, 3 ]                  [ 1, 88 ]
   3:  [ 1, 38, 2 ]                  [ 1, 69, 3 ]
   5:  [ 1, 21, 5 ]                  [ 1, 22, 4 ]
   7:  [ 1, 23, 3 ]                  same dimensions for G and N_G(P)
  13:  [ 1, 12 ]                     same dimensions for G and N_G(P)
  19:  [ 1, 18 ]                     same dimensions for G and N_G(P)
  31:  [ 1, 15, 1 ]                  same dimensions for G and N_G(P)
Fi23:
   2:  [ 1, 48, 25, 7, 4, 2, 1, 1 ]  [ 1, 152, 141, 23, 2 ]
   3:  [ 1, 51, 28, 9, 3, 1, 1 ]     [ 1, 230, 58, 3 ]
   5:  [ 1, 13, 6 ]                  same dimensions for G and N_G(P)
   7:  [ 1, 6 ]                      same dimensions for G and N_G(P)
  11:  [ 1, 10 ]                     same dimensions for G and N_G(P)
  13:  [ 1, 6, 1 ]                   same dimensions for G and N_G(P)
  17:  [ 1, 16 ]                     same dimensions for G and N_G(P)
  23:  [ 1, 11, 1 ]                  same dimensions for G and N_G(P)
Co1:
   2:  [ 1, 64, 27, 2, 1, 1 ]        (no Sylow normalizer table)
   3:  [ 1, 36, 25, 10, 1, 1 ]       [ 1, 73, 35, 19 ]
   5:  [ 1, 23, 14, 7 ]              [ 1, 26, 9, 1 ]
   7:  [ 1, 12, 14 ]                 same dimensions for G and N_G(P)
  11:  [ 1, 10 ]                     same dimensions for G and N_G(P)
```

```
  13:  [ 1, 12 ]                    same dimensions for G and N_G(P)
  23:  [ 1, 11, 1 ]                 same dimensions for G and N_G(P)
J4:
   2:  [ 1, 51, 6, 1 ]              (no Sylow normalizer table)
   3:  [ 1, 9, 4 ]                  [ 1, 10, 3 ]
   5:  [ 1, 4 ]                     same dimensions for G and N_G(P)
   7:  [ 1, 3, 1 ]                  same dimensions for G and N_G(P)
  11:  [ 1, 43, 5 ]                 [ 1, 44, 4 ]
  23:  [ 1, 22 ]                    same dimensions for G and N_G(P)
  29:  [ 1, 28 ]                    same dimensions for G and N_G(P)
  31:  [ 1, 10, 1, 1 ]              same dimensions for G and N_G(P)
  37:  [ 1, 12, 1, 1 ]              same dimensions for G and N_G(P)
  43:  [ 1, 14, 1, 1 ]              same dimensions for G and N_G(P)
F3+:
   2:  [ 1, 71, 20, 4, 1 ]          (no Sylow normalizer table)
   3:  [ 1, 72, 24, 4 ]             (no Sylow normalizer table)
   5:  [ 1, 13, 6 ]                 same dimensions for G and N_G(P)
   7:  [ 1, 26, 4 ]                 [ 1, 27, 3 ]
  11:  [ 1, 10 ]                    same dimensions for G and N_G(P)
  13:  [ 1, 12 ]                    same dimensions for G and N_G(P)
  17:  [ 1, 16 ]                    same dimensions for G and N_G(P)
  23:  [ 1, 11, 1 ]                 same dimensions for G and N_G(P)
  29:  [ 1, 14, 1 ]                 same dimensions for G and N_G(P)
B:
   2:  [ 1, 140, 30, 7, 1 ]         (no Sylow normalizer table)
   3:  [ 1, 81, 41, 13, 4, 1, 1 ]   [ 1, 230, 58, 3 ]
   5:  [ 1, 56, 18, 4 ]             [ 1, 52, 5 ]
   7:  [ 1, 23, 3 ]                 same dimensions for G and N_G(P)
  11:  [ 1, 10 ]                    (no Sylow normalizer table)
  13:  [ 1, 12 ]                    (no Sylow normalizer table)
  17:  [ 1, 16 ]                    (no Sylow normalizer table)
  19:  [ 1, 18 ]                    (no Sylow normalizer table)
  23:  [ 1, 11, 1 ]                 (no Sylow normalizer table)
  31:  [ 1, 15, 1 ]                 same dimensions for G and N_G(P)
  47:  [ 1, 23, 1 ]                 (no Sylow normalizer table)
M:
   2:  [ 1, 169, 12, 1 ]            (no Sylow normalizer table)
   3:  [ 1, 147, 14, 2, 1 ]         (no Sylow normalizer table)
   5:  [ 1, 106, 17, 4, 1 ]         [ 1, 159, 37, 4 ]
   7:  [ 1, 81, 9, 1 ]              [ 1, 82, 17 ]
  11:  [ 1, 39, 10 ]                same dimensions for G and N_G(P)
  13:  [ 1, 56, 5 ]                 [ 1, 57, 4 ]
  17:  [ 1, 16 ]                    (no Sylow normalizer table)
  19:  [ 1, 18 ]                    (no Sylow normalizer table)
  23:  [ 1, 11, 1 ]                 (no Sylow normalizer table)
  29:  [ 1, 28 ]                    (no Sylow normalizer table)
  31:  [ 1, 15, 1 ]                 (no Sylow normalizer table)
  41:  [ 1, 40 ]                    (no Sylow normalizer table)
  47:  [ 1, 23, 1 ]                 same dimensions for G and N_G(P)
  59:  [ 1, 29, 1 ]                 (no Sylow normalizer table)
  71:  [ 1, 35, 1 ]                 (no Sylow normalizer table)
```

We see for example that for the principal block B of the group algebra of McL over a field F of characteristic 2, the Loewy length of $Z(B)$ is 4, and the dimensions of $J^i(Z(B))$, for $0 \leq i \leq 4$, are $18, 17, 5, 1, 0$, respectively. In [Sch16, p. 113], it is erroneously stated that the Loewy length is larger than 5, and that $J^2(Z(B))$ has dimension 11.

The largest Loewy length in the above list occurs for the group Fi_{23} in characteristic 2.

Now let us look what we know about the abovementioned conjecture.

Example

```
gap> conjecture_is_wrong_for;
[  ]
gap> conjecture_is_open_for;
[ [ "Co1", 2 ], [ "J4", 2 ], [ "F3+", 2 ], [ "F3+", 3 ], [ "B", 2 ],
  [ "B", 11 ], [ "B", 13 ], [ "B", 17 ], [ "B", 19 ], [ "B", 23 ],
  [ "B", 47 ], [ "M", 2 ], [ "M", 3 ], [ "M", 17 ], [ "M", 19 ],
  [ "M", 23 ], [ "M", 29 ], [ "M", 31 ], [ "M", 41 ], [ "M", 59 ],
  [ "M", 71 ] ]
```

Most of the open cases belong to primes p that divide the group orders only once, but for which the character tables of the corresponding Sylow p-normalizers are not (yet) contained in GAP's library.

Example

```
gap> defectone:= Filtered( conjecture_is_open_for,
>        pair -> Size( CharacterTable( pair[1] ) ) mod pair[2]^2 <> 0 );
[ [ "B", 11 ], [ "B", 13 ], [ "B", 17 ], [ "B", 19 ], [ "B", 23 ],
  [ "B", 47 ], [ "M", 17 ], [ "M", 19 ], [ "M", 23 ], [ "M", 29 ],
  [ "M", 31 ], [ "M", 41 ], [ "M", 59 ], [ "M", 71 ] ]
```

The following table lists the structures of the groups in question, as they are stated in [CCN+85, pp. 217, 234].

G	p	Structure
B	11	$11:10\times S_5$
	13	$13:12\times S_4$
	17	$(17:8\times 2^2).2$
	19	$19:18\times 2$
	23	$23:11\times 2$
	47	$47:23$
M	17	$(17:8\times L_3(2)).2$
	19	$(19:9\times A_5):2$
	23	$23:11\times S_4$
	29	$(29:14\times 3).2$
	31	$31:15\times S_3$
	41	$41:40$
	59	$59:29$
	71	$71:35$

Table: Some normalizers of cyclic Sylow p-subgroups

We compute the dimensions of the layers in these cases.

Example

```
gap> # G = B, p = 11
gap> DimensionsOfLoewyLayersByTable(
>        CharacterTable( "P:Q", [ 11, 10 ] ) *
>        CharacterTable( "S5" ), 11 );
[ 1, 10 ]
gap> # G = B, p = 13
gap> DimensionsOfLoewyLayersByTable(
>        CharacterTable( "P:Q", [ 13, 12 ] ) *
>        CharacterTable( "S4" ), 13 );
[ 1, 12 ]
gap> # G = B, p = 17
gap> tblH1:= CharacterTable( "17:8" );;
gap> tblG1:= CharacterTable( "17:16" );;
gap> tblH2:= CharacterTable( "2^2" );;
gap> tblG2:= CharacterTable( "D8" );;
gap> DimensionsOfLoewyLayersByTable(
>        CharacterTableOfIndexTwoSubdirectProduct(
>            tblH1, tblG1, tblH2, tblG2, "(17:8x2^2).2" ).table, 17 );
[ 1, 16 ]
gap> # G = B, p = 19
gap> DimensionsOfLoewyLayersByTable(
>        CharacterTable( "P:Q", [ 19, 18 ] ) *
>        CharacterTable( "C2" ), 19 );
[ 1, 18 ]
gap> # G = B, p = 23
gap> DimensionsOfLoewyLayersByTable(
>        CharacterTable( "P:Q", [ 23, 11 ] ) *
>        CharacterTable( "C2" ), 23 );
[ 1, 11, 1 ]
gap> # G = B, p = 47
gap> DimensionsOfLoewyLayersByTable(
>        CharacterTable( "P:Q", [ 47, 23 ] ), 47 );
[ 1, 23, 1 ]
gap> # G = M, p = 17
gap> tblH1:= CharacterTable( "17:8" );;
gap> tblG1:= CharacterTable( "17:16" );;
gap> tblH2:= CharacterTable( "L3(2)" );;
gap> tblG2:= CharacterTable( "L3(2).2" );;
gap> DimensionsOfLoewyLayersByTable(
>        CharacterTableOfIndexTwoSubdirectProduct(
>            tblH1, tblG1, tblH2, tblG2, "(17:8xL3(2)).2" ).table, 17 );
[ 1, 16 ]
gap> # G = M, p = 19
gap> tblH1:= CharacterTable( "19:9" );;
gap> tblG1:= CharacterTable( "19:18" );;
gap> tblH2:= CharacterTable( "A5" );;
gap> tblG2:= CharacterTable( "S5" );;
gap> DimensionsOfLoewyLayersByTable(
>        CharacterTableOfIndexTwoSubdirectProduct(
>            tblH1, tblG1, tblH2, tblG2, "(19:9xA5).2" ).table, 19 );
[ 1, 18 ]
gap> # G = M, p = 23
```

```
gap> DimensionsOfLoewyLayersByTable(
>        CharacterTable( "P:Q", [ 23, 11 ] ) *
>        CharacterTable( "S4" ), 23 );
[ 1, 11, 1 ]
gap> # G = M, p = 29
gap> tblH1:= CharacterTable( "29:14" );;
gap> tblG1:= CharacterTable( "29:28" );;
gap> tblH2:= CharacterTable( "C3" );;
gap> tblG2:= CharacterTable( "S3" );;
gap> DimensionsOfLoewyLayersByTable(
>        CharacterTableOfIndexTwoSubdirectProduct(
>            tblH1, tblG1, tblH2, tblG2, "(29:14x3).2" ).table, 29 );
[ 1, 28 ]
gap> # G = M, p = 31
gap> DimensionsOfLoewyLayersByTable(
>        CharacterTable( "P:Q", [ 31, 15 ] ) *
>        CharacterTable( "S3" ), 31 );
[ 1, 15, 1 ]
gap> # G = M, p = 41
gap> DimensionsOfLoewyLayersByTable(
>        CharacterTable( "P:Q", [ 41, 40 ] ), 41 );
[ 1, 40 ]
gap> # G = M, p = 59
gap> DimensionsOfLoewyLayersByTable(
>        CharacterTable( "P:Q", [ 59, 29 ] ), 59 );
[ 1, 29, 1 ]
gap> # G = M, p = 71
gap> DimensionsOfLoewyLayersByTable(
>        CharacterTable( "P:Q", [ 71, 35 ] ), 71 );
[ 1, 35, 1 ]
```

For those normalizers N in the above computations that are not direct products, the subdirect product structure follows from the containment of N in known subgroups of B or M, respectively:

- $(17:8\times 2^2).2 < (2^2\times F_4(2)).2 < B$,
- $(17:8\times L_3(2)).2 < (S_4(4).2\times L_3(2)).2 < M$,
- $(19:9\times A_5).2 < (U_3(8).3\times A_5).2 < M$, and
- $(29:14\times 3).2 < 3.Fi_{24} < M$.

Seven cases remain to be checked. From [Wil98], we know that the Sylow 2-subgroups of Co_1, J_4, F_{3+}, B, and M are self-normalizing in the simple group, and that the Sylow 3-normalizers of F_{3+} and M have the $3'$-parts 2^3 and 2^6, respectively.

Example

```
gap> hard:= Filtered( conjecture_is_open_for,
>        pair -> Size( CharacterTable( pair[1] ) ) mod pair[2]^2 = 0 );
[ [ "Co1", 2 ], [ "J4", 2 ], [ "F3+", 2 ], [ "F3+", 3 ], [ "B", 2 ],
  [ "M", 2 ], [ "M", 3 ] ]
```

The first four of these cases and the last one are small enough for our appproach, GAP needs several hours for each of the computations. The results are as follows.

- The Sylow 2-subgroup of Co_1 has 782 conjugacy classes, the dimensions of the Loewy layers of the centre of its group algebra over the field with two elements are $1, 766, 15$.
- The Sylow 2-subgroup of J_4 has 581 conjugacy classes, the dimensions of the Loewy layers of the centre of its group algebra over the field with two elements are $1, 553, 27$.
- The Sylow 2-subgroup of F_{3+} has 581 conjugacy classes, the dimensions of the Loewy layers of the centre of its group algebra over the field with two elements are $1, 553, 27$.
- The Sylow 3-normalizer of F_{3+} is supersolvable and has 701 conjugacy classes, the dimensions of the Loewy layers of the centre of its group algebra over the field with three elements are $1, 620, 78, 2$.
- The Sylow 3-normalizer of M has 810 conjugacy classes, the dimensions of the Loewy layers of the centre of its group algebra over the field with three elements are $1, 698, 108, 3$.

The other two open cases are more involved. For example, the Sylow 2-subgroup of the Monster has 26752 conjugacy classes. We should better think about another strategy.

Remark:

The Sylow 2-subgroups of the groups J_4 and F_{3+} have the same number of conjugacy classes and the same Loewy layer dimensions. In fact, they have the same irreducible characters. This implies that the centres of their group algebras are isomorphic, because the structure constants that define the multiplication in these algebras depend only on the character values, not on the element orders (see `ClassMultiplicationCoefficient` (**Reference: ClassMultiplicationCoefficient for character tables**)).

The two groups are *not isomorphic*, since the numbers of conjugacy classes of elements of given orders are different: The Sylow 2-subgroups of J_4 have 106, 398, and 75 conjugacy classes of elements of the orders 2, 4, and 8, respectively, whereas these numbers are 98, 394, and 87 in the case of the Sylow 2-subgroups of F_{3+}.

The similarity of the two Sylow 2-subgroups can be explained by the fact that the groups are contained in (maximal) subgroups of the structure $2^{11}.M_{24}$ in J_4 and F_{3+}, where the involved 11-dimensional modules for M_{24} are isomorphic; also these maximal subgroups have the same irreducible characters. (But note that also the Sylow 2-subgroup of Co_1 is contained in a (maximal) subgroup of the structure $2^{11}.M_{24}$.)

A GAP session that shows these facts is listed below. (A direct call of `IsomorphismGroups` (**Reference: IsomorphismGroups**) for the two groups did not finish after several days.) Note that the Sylow 2-subgroup of J_4 is contained in the maximal subgroups of the type $2^{11} : M_{24}$, for which a permutation representation on 2^{11} points is known.

First we construct the two Sylow 2-subgroups, represented as PC groups (see Chapter (**Reference: Pc Groups**)).

Example

```
gap> g1:= AtlasGroup( "Fi24'" );;
gap> s1:= SylowSubgroup( g1, 2 );;
gap> pc1:= Image( IsomorphismPcGroup( s1 ) );
gap> j4:= CharacterTable( "J4" );;
gap> j4m1:= CharacterTable( Maxes( j4 )[1] );;
```

```
gap> ( Size( j4 ) / Size( j4m1 ) ) mod 2;
1
gap> GroupInfoForCharacterTable( j4m1 );
[ [ "AtlasSubgroup", [ "J4", 1 ] ],
  [ "PrimitiveGroup", [ 2048, 11 ] ] ]
gap> g2:= PrimitiveGroup( 2048, 11 );;
gap> s2:= SylowSubgroup( g2, 2 );;
gap> pc2:= Image( IsomorphismPcGroup( s2 ) );;
gap> Collected( Factors( Size( pc1 ) ) );
[ [ 2, 21 ] ]
gap> Size( pc1 ) = Size( pc2 );
true
```

Next we compute the two character tables and check the above claims. (Currently GAP does not recognize automatically that the PC groups are in fact p-groups, but we have to set the supersolvability flag in order to use an efficient algorithm for the computations.)

Example

```
gap> IsPGroup( pc1 );;  IsPGroup( pc2 );;
gap> t1:= CharacterTable( pc1 );;
gap> irr1:= Irr( t1 );;
gap> t2:= CharacterTable( pc2 );;
gap> irr2:= Irr( t2 );;
gap> deg:= CharacterDegrees( t1 );
[ [ 1, 32 ], [ 2, 56 ], [ 4, 80 ], [ 8, 84 ], [ 16, 109 ], [ 32, 70 ],
  [ 64, 94 ], [ 128, 42 ], [ 256, 14 ] ]
gap> deg = CharacterDegrees( t2 );
true
gap> IsRecord( TransformingPermutations( irr1, irr2 ) );
true
gap> Collected( OrdersClassRepresentatives( t1 ) );
[ [ 1, 1 ], [ 2, 98 ], [ 4, 394 ], [ 8, 87 ], [ 16, 1 ] ]
gap> Collected( OrdersClassRepresentatives( t2 ) );
[ [ 1, 1 ], [ 2, 106 ], [ 4, 398 ], [ 8, 75 ], [ 16, 1 ] ]
```

Chapter 3

Character theoretic functions for p-blocks

3.1 The character theoretic setup

The aim of this GAP package is to provide functions to compute information about p-blocks of groups only from character tables. This means that we know the ordinary character table t of the finite group G, say, and that we know the ordinary character table of some subgroup H of G, together with the class fusion that describes the embedding of H in G, but we cannot assume that G is given or that the p-modular character table of G is known. If such information is actually available then more operations may return non-`fail` results: If G is known via the attribute `UnderlyingGroup` (**Reference: UnderlyingGroup for character tables**) of the character table in question then calling `DefectGroup` (3.4.2) makes sense, and if the p-modular character table of G is known then calling `IBr` (3.3.6) for the block object in question makes sense.

For example, the distribution of the irreducible characters of t to p-blocks can be computed with `PrimeBlocks` (**Reference: PrimeBlocks**). Note that this information depends on t: When we talk about "the b-th p-block of the character table t" then this means that the list `Irr(`t`)` contains characters of exactly $b-1$ other p-blocks before the first character of the block in question appears.

It is clear that the character theoretic setup does not allow us to perform block theoretic computations with (ideals of) group rings. Only questions about the centre of a group ring may make sense, for example a block idempotent can be represented by a class function of the given character table (see `CoefficientsOfOsimaIdempotent` (3.3.13)), which is interpreted as the coefficient vector w.r.t. the basis of class sums, that is, the i-th entry is the coefficient of the the sum over the i-th conjugacy class.

Some definitions can be motivated by the identification of p-blocks with ideals of group rings, in this context it is natural to talk about the dimension (see `Dimension` (3.3.9)) or the kernel (see `ClassPositionsOfKernel` (3.3.11)) of a block. In this sense, we take the pragmatic viewpoint stated in [Nav98, p. 57]: "Most authors write $B = e_B FG$. We will also use this notation when we find it convenient." (Here B is a p-block of the group G, F is a field of characteristic p specified in the given p-modular system (see Section 3.2), and e_B is the block idempotent (see `CoefficientsOfOsimaIdempotent` (3.3.13)) of B.)

3.2 Theoretical background of blocks of character tables

"Let (K,R,F) be a p-modular system for the group G." Many statements in modular representation theory start with this sentence or a similar one, where p is a prime integer, R is a complete discrete valuation ring with quotient field K of characteristic zero and $F = R/M$ is a field of characteristic p, for a fixed maximal ideal M, say, of R.

For computational purposes, it is useful to choose the p-modular system in a specific way, as is described in [LP10, Section 4.2] and to some extent also in [JLPW95, Sections 2-5]. The idea is as follows. For positive integers m, we set $\zeta_m = \exp(2\pi i/m)$. The set of complex roots of unity of finite order coprime to p consists exactly of the powers of all ζ_m, where m is coprime to p. We fix a bijection from this set to the nonzero elements in finite extensions of the field with p elements, such that the restriction of this map to the m-th roots of unity (for any integer m that is coprime to p) defines a ring homomorphism $*$ from $\mathbb{Z}[\zeta_m]$ into those finite fields of characteristic p that contain elements of multiplicative order m. More generally, we can take the union of the rings $\mathbb{Z}_p[\zeta_m]$ as the domain of $*$, where $\mathbb{Z}_p$ is the ring of of p-local numbers in $\mathbb{Q}$.

For a group G whose exponent has p'-part m, this setup allows us to lift eigenvalues of p-modular representations of G to $\mathbb{Z}[\zeta_m]$ and thus to compute Brauer character values in this ring from matrices over finite fields (see `BrauerCharacterValue` (**Reference: BrauerCharacterValue**)). Moreover, this setup holds also for subgroups of G, so we can restrict and induce Brauer characters via the class fusions in question.

We do not need the p-modular system (K,R,F) explicitly for actual GAP computations. We can think of R as the completion of a valuation ring in $\mathbb{Q}(\zeta_m)$, and the kernel of the map $*$ is the maximal ideal M of R. The map $*$ is implemented via the function `FrobeniusCharacterValueExt` (4.2.3) (which generalizes `FrobeniusCharacterValue` (**Reference: FrobeniusCharacterValue**)), the images are elements of finite fields in the sense of `IsFFE` (**Reference: IsFFE**). If we are interested in the interpretation of finite field elements in terms of polynomials then `ReductionToFiniteField` (4.2.4) can be used. See also Chapter (**Reference: Cyclotomic Numbers**) for more details how elements of cyclotomic fields are handled by GAP.

Note that the Brauer characters of G, which depend on the choice of the ideal M, are uniquely determined in our setup. Also the question whether a given conjugacy class is a defect class of some p-block can be decided, but may require explicit computations of images under the map $*$, and this can be a hard problem. On the other hand, questions such as the distribution of the ordinary irreducible characters of G into p-blocks can be decided without computations in finite fields, and these questions are not computationally hard (see `PrimeBlocks` (**Reference: PrimeBlocks**)); the same holds for the question whether some algebraic conjugate of a given conjugacy class is a defect class of some p-block (see Section 3.4).

3.2.1 A number theoretic lemma

Lemma: Let R be the ring of algebraic integers in $\mathbb{C}$, p be a prime integer, and M be a maximal ideal in R that contains pR. For positive integers m, set $\zeta_m = \exp(2\pi i/m)$. Consider $\alpha \in R \cap \mathbb{Q}(\zeta_m)$ such that all algebraic conjugates of α lie in M. Then α lies in every maximal ideal of R that contains pR, and we have $\alpha^k \in pR$, where k is the degree of the minimal polynomial of α over $\mathbb{Q}$. Moreover, if p does not divide m then $\alpha \in pR$.

Proof: (see [LP10, proof of Theorem 4.4.8], [Nav98, proof of Theorem 3.2], or [Isa76, Theorem (15.18)]) Let A denote the set of algebraic conjugates of α (thus $|A| = k$ holds), and set $f = \prod_{a \in A}(X - a) \in \mathbb{Q}[X]$. All coefficients except the leading one lie in $M \cap \mathbb{Q} = M \cap R \cap \mathbb{Q} = M \cap \mathbb{Z} = p\mathbb{Z}$, thus $0 =$

$f(\alpha) \equiv \alpha^k \pmod{pR}$. Since maximal ideals are prime, α lies in any maximal ideal of R containing pR. If p does not divide m then p does not ramify in $\mathbb{Z}[\zeta_m] = R \cap \mathbb{Q}(\zeta_m)$, see [Was97, Proposition 2.3]. Thus pR has a unique factorization of the form $P_1 P_2 \cdots P_l$, for prime ideals P_i in R which are pairwise different. Since the ideal $\alpha^k R = (\alpha R)^k$ is contained in pR, each P_i divides $(\alpha R)^k$ and hence αR, which means $\alpha \in pR$.

A well-known application of this lemma is the decision whether two given ordinary irreducible characters χ, ψ of a group G, say, belong to the same p-block, that is, whether the corresponding central characters ω_χ, ω_ψ are congruent modulo the fixed maximal ideal M of R. It is sufficient to test this property for p-regular elements of G (see [LP10, proof of Theorem 4.4.8]). In other words, if ω_χ and ω_ψ are *not* congruent modulo M then there is a class C of p-regular elements in G such that $(\omega_\chi - \omega_\psi)(C^+) \notin M$, where $C^+ \in RG$ denotes the sum of elements in C. Hence all we have to check is whether $(\omega_\chi - \omega_\psi)(C^+)/p$ is an algebraic integer, for representatives C of families of algebraic conjugate classes of p-regular elements in G. See `PrimeBlocks` (**Reference: PrimeBlocks**) for more information about the GAP implementation, and the statements cited in the above proof for variants of this theorem.

Similarly, induced blocks can be computed by inducing central characters. Note that in this case, we *cannot* restrict the comparison to p-regular classes, because the values at p-singular classes may yield that no induced block is defined, see [LP10, Theorem 4.7.7]. See also the examples in Section 3.8 for details.

Another application of the lemma is the computation of defect classes up to Galois conjugacy (see Section 3.4).

3.3 Block objects for character tables

This section introduces GAP objects that represent p-blocks of character tables. One may argue that such objects are not necessary because one can simply replace the object that represents the b-th p-block of the character table t by the triple $[t, p, b]$. Another argument against these objects is that the documentation of functions for them is longer than the GAP code that implements them. On the other hand, such objects are convenient as inputs and outputs of operations, and for caching results of computations as values of attributes, such as `Irr` (3.3.5) and `IBr` (3.3.6).

Thus we offer such GAP objects, but we offer also variants of the relevant functions that accept the arguments t, p, b instead.

See Section 3.3.14 for technical details.

3.3.1 Block

▷ `Block(tbl, p, b)` (operation)

For an ordinary character table `tbl`, a prime integer `p`, and a positive integer `b`, `Block` returns the block object that represents the `b`-th `p`-block of `tbl` if this block exists, otherwise `fail` is returned.

Example

```
gap> Block( CharacterTable( "A5" ), 2, 1 );
Block( CharacterTable( "A5" ), 2, 1 )
gap> Block( CharacterTable( "A5" ), 2, 3 );
fail
```

3.3.2 Defining attributes of block objects

▷ UnderlyingCharacterTable(*B*) (attribute)
▷ UnderlyingCharacteristic(*B*) (attribute)
▷ NumberOfBlock(*B*) (attribute)

These are the defining attributes of the block object *B*. There are no methods for computing their values, the values must be set upon creation of the block object.

For the *b*-th *p*-block of the character table *t*, the values of UnderlyingCharacterTable, UnderlyingCharacteristic, and NumberOfBlock are *t*, *p*, and *b*, respectively.

Example

```
gap> b:= Block( CharacterTable( "A5" ), 2, 1 );
Block( CharacterTable( "A5" ), 2, 1 )
gap> UnderlyingCharacterTable( b );
CharacterTable( "A5" )
gap> UnderlyingCharacteristic( b );
2
gap> NumberOfBlock( b );
1
```

3.3.3 PrincipalBlock

▷ PrincipalBlock(*tbl*, *p*) (operation)

For an ordinary character table *tbl* and a prime integer PrincipalBlock returns the block object for the *p*-block of *tbl* that contains the trivial character of *tbl*.

Note that the trivial character need not be the first character in the list of irreducibles of *tbl*, so the principal block need not have number 1.

Example

```
gap> PrincipalBlock( CharacterTable( "A5" ), 2 );
Block( CharacterTable( "A5" ), 2, 1 )
```

3.3.4 PBlocks

▷ PBlocks(*tbl*, *p*) (function)

For an ordinary character table *tbl* and a prime integer *p*, PBlocks returns the list of block objects that represent the *p*-blocks of *tbl*.

Example

```
gap> PBlocks( CharacterTable( "A5" ), 2 );
[ Block( CharacterTable( "A5" ), 2, 1 ),
  Block( CharacterTable( "A5" ), 2, 2 ) ]
```

3.3.5 Irr (for a block object)

▷ Irr(*B*) (attribute)
▷ Irr(*ordtbl*, *p*, *b*) (operation)

The arguments can be either a block object *B* which represents the *b*-th *p*-block of the ordinary character table *ordtbl*, or the three values *ordtbl*, *p*, and *b*.

Irr returns the sublist of those irreducible characters of *ordtbl* that belong to the block, in the same ordering as in the Irr (**Reference: Irr**) list of *ordtbl*.

Example

```
gap> irr:= Irr( Block( CharacterTable( "A5" ), 2, 1 ) );
[ Character( CharacterTable( "A5" ), [ 1, 1, 1, 1, 1 ] ),
  Character( CharacterTable( "A5" ),
  [ 3, -1, 0, -E(5)-E(5)^4, -E(5)^2-E(5)^3 ] ),
  Character( CharacterTable( "A5" ),
  [ 3, -1, 0, -E(5)^2-E(5)^3, -E(5)-E(5)^4 ] ),
  Character( CharacterTable( "A5" ), [ 5, 1, -1, 0, 0 ] ) ]
gap> irr = Irr( CharacterTable( "A5" ), 2, 1 );
true
```

3.3.6 IBr (for a block object)

▷ IBr(*B*) (attribute)

▷ IBr(*modtbl, b*) (operation)

The arguments can be either a block object *B* which represents the *b*-th *p*-block of the ordinary character table whose *p*-modular Brauer table is *modtbl*, or the two values *modtbl* and *b*.

IBr returns the sublist of those irreducible characters of *modtbl* that belong to the block, in the same ordering as in the Irr (**Reference: Irr**) list of *modtbl*.

If *B* is given, it may happen that the irreducible *p*-modular Brauer characters for the character table that defines *B* are not available. In this case, IBr returns fail.

Example

```
gap> ibr:= IBr( Block( CharacterTable( "A5" ), 2, 1 ) );
[ Character( BrauerTable( "A5", 2 ), [ 1, 1, 1, 1 ] ),
  Character( BrauerTable( "A5", 2 ),
  [ 2, -1, E(5)+E(5)^4, E(5)^2+E(5)^3 ] ),
  Character( BrauerTable( "A5", 2 ),
  [ 2, -1, E(5)^2+E(5)^3, E(5)+E(5)^4 ] ) ]
gap> ibr = IBr( CharacterTable( "A5" ) mod 2, 1 );
true
gap> IBr( Block( CharacterTable( "M" ), 2, 1 ) );
fail
```

3.3.7 DecompositionMatrix (for a block object)

▷ DecompositionMatrix(*B*) (attribute)

For a *p*-block *B* of a character table *t*, say. DecompositionMatrix returns the decomposition matrix of *B*, that is, the matrix whose rows and columns are indexed by the ordinary irreducible characters and the *p*-modular Brauer characters, respectively, that belong to *B*.

If the *p*-modular character table of *t* is not known then fail is returned.

A variant without block objects is DecompositionMatrix (**Reference: DecompositionMatrix**).

Example

```
gap> DecompositionMatrix( Block( CharacterTable( "A5" ), 2, 1 ) );
[ [ 1, 0, 0 ], [ 1, 0, 1 ], [ 1, 1, 0 ], [ 1, 1, 1 ] ]
gap> DecompositionMatrix( Block( CharacterTable( "A5" ), 2, 2 ) );
[ [ 1 ] ]
gap> DecompositionMatrix( Block( CharacterTable( "M" ), 2, 1 ) );
fail
```

3.3.8 DimensionsOfDecompositionMatrix (for a block object)

▷ `DimensionsOfDecompositionMatrix(B)` (attribute)

For a p-block B of a character table t, say. `DimensionsOfDecompositionMatrix` returns the list $[k(B),l(B)]$ of the numbers of rows and columns of the decomposition matrix of B. That is, $k(B)$ is the number of ordinary irreducible characters in B, and $l(B)$ is the number of irreducible Brauer characters in B.

Note that these numbers can be computed from the ordinary character table t, whereas `DecompositionMatrix` (3.3.7) requires the irreducible p-modular Brauer characters of t.

Example

```
gap> b:= Block( CharacterTable( "M" ), 2, 1 );;
gap> DimensionsOfDecompositionMatrix( b );
[ 183, 55 ]
```

3.3.9 Dimension (for a block object)

▷ `Dimension(B)` (attribute)

The dimension of the p-block B is $\sum_\chi \chi(1)^2$, where the summation runs over the ordinary irreducible characters χ of B.

This is the dimension of the algebra eFG, where e is the block idempotent of B (see `CoefficientsOfOsimaIdempotent` (3.3.13)) and G is the underlying group of B.

(For details, see [Nav98, Thm. 3.14].)

Example

```
gap> List( PBlocks( CharacterTable( "2.A5" ), 3 ), Dimension );
[ 42, 9, 9, 24, 36 ]
```

3.3.10 CentralCharacter (for a block object)

▷ `CentralCharacter(B)` (attribute)

▷ `CentralCharacter(tbl, p, b)` (operation)

The *central character* of the p-block B of the group G is the algebra homomorphism $\lambda_B : \mathrm{Z}(FG) \to F$ that is defined by $\lambda_B(C^+) = \omega_\chi(C^+)^*$, for $\chi \in \mathrm{Irr}(B)$ and class sums C^+ of G, see [LP10, p. 324] or [Nav98, p. 48]. Here ω_χ is the central character of χ as defined in `CentralCharacter` (**Reference: CentralCharacter**), that is, $\omega_\chi((g^G)^+) = |g^G|\chi(g)/\chi(1)$ for $g \in G$. (Note that the values of ω_χ are complex numbers, not elements of a finite field.)

We represent the central character of the `b`-th `p`-block of the character table `tbl` by the vector of reductions of the list returned by `CentralCharacter` (**Reference: CentralCharacter**) with argument an irreducible character in the given block. Note that the central character of the block depends on the choice of the `p`-modular system, see Section 3.2.

`CentralCharacter` is based on `FrobeniusCharacterValueExt` (4.2.3), thus the returned list contains `fail` entries exactly when a Conway polynomial would be needed that is not available.

Example

```
gap> CentralCharacter( CharacterTable( "A5" ), 2, 1 );
[ Z(2)^0, Z(2)^0, 0*Z(2), 0*Z(2), 0*Z(2) ]
gap> CentralCharacter( Block( CharacterTable( "A5" ), 2, 1 ) );
[ Z(2)^0, Z(2)^0, 0*Z(2), 0*Z(2), 0*Z(2) ]
```

3.3.11 ClassPositionsOfKernel (for a block object)

▷ `ClassPositionsOfKernel(B)` (attribute)

Let `B` be a p-block of the group G, say. We define the *kernel* of `B` as the intersection of the kernels of the ordinary irreducible characters in `B`. and `ClassPositionsOfKernel` returns the list of positions of those conjugacy classes of G (w. r. t. the underlying character table of `B`) that lie inside the kernel.

The kernel of `B` is the kernel of the natural action of G on the algebra eFG, where e is the block idempotent of `B` (see `CoefficientsOfOsimaIdempotent` (3.3.13)).

The kernel of `B` is equal to the intersection of the kernel of *any* character in `B` with the largest normal p'-subgroup of G, which is usually denoted by $O_{p'}(G)$, see [LP10, Theorem 4.11.13], [Nav98, Theorem 6.10], or [NT89, Ch. 5, Sect. 8]. In particular, the kernel of the principal p block of G is equal to $O_{p'}(G)$.

Example

```
gap> t:= CharacterTable( "6.A6" );;
gap> List( PBlocks( t, 2 ), ClassPositionsOfKernel );
[ [ 1, 3, 5 ], [ 1, 3, 5 ], [ 1, 3, 5 ], [ 1 ], [ 1 ] ]
gap> List( PBlocks( t, 3 ), ClassPositionsOfKernel );
[ [ 1, 4 ], [ 1, 4 ], [ 1 ] ]
gap> List( PBlocks( t, 5 ), ClassPositionsOfKernel );
[ [ 1, 2, 3, 4, 5, 6 ], [ 1, 2, 3, 4, 5, 6 ], [ 1, 2, 3, 4, 5, 6 ],
  [ 1, 2, 3, 4, 5, 6 ], [ 1, 3, 5 ], [ 1, 3, 5 ], [ 1, 3, 5 ],
  [ 1, 4 ], [ 1, 4 ], [ 1, 4 ], [ 1, 4 ], [ 1 ], [ 1 ] ]
```

3.3.12 IsFaithful (for a block object)

▷ `IsFaithful(B)` (property)

The block `B` is called *faithful* if its kernel is trivial, see `ClassPositionsOfKernel` (3.3.11).

Example

```
gap> t:= CharacterTable( "6.A6" );;
gap> List( PBlocks( t, 2 ), IsFaithful );
[ false, false, false, true, true ]
```

3.3.13 CoefficientsOfOsimaIdempotent (for a block object)

▷ `CoefficientsOfOsimaIdempotent(`*B*`)` (attribute)
▷ `CoefficientsOfOsimaIdempotent(`*tbl, p, b*`)` (operation)

Let B be a p-block of the group G. The *primitive idempotent* corresponding to $\chi \in \mathrm{Irr}(G)$ is

$$e_\chi = (\chi(1)/|G|)\sum_g \chi(g^{-1})g \in \mathrm{Z}(\mathbb{C}G),$$

where the summation runs over the elements of G.

The *Osima idempotent* of B is defined as $f_B = \sum_\chi e_\chi$, where the summation runs over $\chi \in \mathrm{Irr}(B)$. The coefficient of $g \in G$ in f_B is

$$a_g = (1/|G|)\sum_\chi \chi(1)\chi(g^{-1}),$$

where the summation runs over $\chi \in \mathrm{Irr}(B)$.

`CoefficientsOfOsimaIdempotent` returns the vector $[a_{g_1}, a_{g_2}, \ldots]$, where g_i lies in the i-th conjugacy class of the character table *tbl*.

The name "Osima idempotent" appears in [Isa76, p. 277]. Both f_B and its reduction $(f_B)^*$ are called *block idempotent* in [LP10, p. 324], whereas this term is used only for $e_B = (f_B)^*$ in [Nav98, p. 55].

Example

```
gap> CoefficientsOfOsimaIdempotent( CharacterTable( "A5" ), 2, 1 );
[ 11/15, 0, -1/15, 1/15, 1/15 ]
gap> CoefficientsOfOsimaIdempotent( Block( CharacterTable( "A5" ), 2, 1 ) );
[ 11/15, 0, -1/15, 1/15, 1/15 ]
```

3.3.14 Technicalities of block objects for character tables

▷ `IsBlockOfCharacterTable(`*B*`)` (category)

A p-block of a character table can be represented by a GAP object in the filter `IsBlockOfCharacterTable`. The idea is that such an object is created by `Block` (3.3.1) from the defining data, and that there are various operations that can take a block object as arguments (for example attributes such as `Defect` (3.4.1) and `Irr` (3.3.5)) or can return block objects (for example `PrincipalBlock` (3.3.3)).

Block objects aren't domains in the sense of (**Reference: Domains**), in particular they do not have elements; one can use `Irr` (3.3.5) and `IBr` (3.3.6) for accessing the ordinary or modular irreducible characters that "belong to" the given block.

Two objects that represent blocks of character tables are compared w.r.t. the triples `[ id, p, b ]`, where `id` is the identifier of the underlying character table, `p` is the underlying characteristic, and `b` is the number of the block.

The declarations of operations and the installations of methods for these block objects should not cause conflicts to situations where blocks of algebras, in the sense of direct summands, are considered. (At the moment, there seems to be no GAP package with such functionality.)

Example

```
gap> tbl:= CharacterTable( "D8" );;
gap> tbl2:= CharacterTable( "A4" );;
gap> IsBlockOfCharacterTable( tbl );
false
gap> IsBlockOfCharacterTable( Block( tbl, 2, 1 ) );
true
gap> Block( tbl, 2, 1 ) = Block( tbl2, 2, 1 );
false
gap> Block( tbl, 2, 1 ) < Block( tbl2, 2, 1 );
true
```

3.4 Defect, defect classes, and defect groups

Let G be a group and p be a prime integer. For any positive integer m, let m_p denote the highest power of p that divides m.

The *p-defect groups* of a conjugacy class C of G are the Sylow p-subgroups of the centralizers in G of the elements in C.

Let t be the character table of the group G, p be a prime integer, and B be a p-block of t. The *defect* of B is defined as the integer

$$\max\{\log_p(|G|_p/\chi(1)_p); \chi \in \mathrm{Irr}(B)\},$$

see [Nav98, Def. 3.15] and [LP10, Def. 4.4.12].

A class g^G of p-regular elements in G is called a *defect class* of B if both $\lambda_B((g^G)^+)$ and $(a_g)^*$ are nonzero, where λ_B is the central character of B (see `CentralCharacter` (3.3.10)) and a_g is the coefficient of the Osima idempotent of B at g (see `CoefficientsOfOsimaIdempotent` (3.3.13)). Note that the set of defect classes of a block depends on the choice of the p-modular system, see Section 3.2.

The *p-defect groups* of a B are the defect groups of the defect classes of B. Equivalently, the defect groups of B are the defect groups of arbitrary algebraic conjugates of defect classes of B. ([Isa76, p. 280] defines the defect of a block via the order of its defect group –if the defect group has order p^d then the defect is d. Then it is shown in [Isa76, Theorem 15.41] that this definition coincides with the above one.)

By the lemma from Section 3.2.1, the set of algebraic conjugates of the defect classes of a p-block does *not* depend on the choice of the p-modular system and can be computed without computing p-modular reductions of cyclotomic numbers. In order to see this, we fix a class $C = g^G$ of p-regular elements of order m in G, and ask whether at least one algebraic conjugate of C is a defect class of a given p-block B. This is *not* the case if and only if $\lambda_B((h^G)^+) \cdot (a_h)^* = 0$ holds for all algebraic conjugates h of g, that is, for all generators h of the cyclic group $\langle g \rangle$. Choose $\chi \in \mathrm{Irr}(B)$ and set $\beta = \omega_\chi((g^G)^+) \cdot a_g$. Then there is an integer n coprime to p such that $\alpha = n \cdot \beta$ is an algebraic integer, and $\alpha^* = 0$ holds if and only if $\beta^* = \lambda_B((g^G)^+) \cdot (a_g)^* = 0$ holds. Thus no algebraic conjugate of C is a defect class of B if and only if $\alpha/p \in \mathbb{Z}[\zeta_m]$ holds.

3.4.1 Defect

▷ `Defect(B)` (attribute)

▷ `Defect(tbl, p, b)` (operation)

For a block that is either given by a block object *B* or as the *b*-th *p*-block of the character table *tbl*, Defect returns the defect of this block, see above.

Example

```
gap> Defect( Block( CharacterTable( "A5" ), 2, 1 ) );
2
gap> Defect( CharacterTable( "A5" ), 2, 1 );
2
```

3.4.2 DefectGroup

▷ DefectGroup(*B*) (attribute)
▷ DefectGroup(*tbl, p, b*) (operation)

Let *tbl* be an ordinary character table with underlying group (see UnderlyingGroup (**Reference: UnderlyingGroup for character tables**)) *G*, say, *p* be a prime integer, and *b* be a positive integer denoting the number of a *p*-block of *tbl*. Alternatively, let *B* be the *b*-th *p*-block of *tbl*.

DefectGroup returns a defect group of the *b*-th *p*-block of *tbl*.

If the UnderlyingGroup (**Reference: UnderlyingGroup for character tables**) is not known then fail is returned.

Example

```
gap> tbl:= CharacterTable( AlternatingGroup( 5 ) );;
gap> DefectGroup( tbl, 2, 1 );
Group([ (2,3)(4,5), (2,4)(3,5) ])
gap> DefectGroup( Block( CharacterTable( "A5" ), 2, 1 ) );
fail
```

3.4.3 NormalizerOfDefectGroup

▷ NormalizerOfDefectGroup(*B*) (attribute)
▷ NormalizerOfDefectGroup(*tbl, p, b*) (operation)

Let *tbl* be an ordinary character table with underlying group (see UnderlyingGroup (**Reference: UnderlyingGroup for character tables**)) *G*, say, *p* be a prime integer, and *b* be a positive integer denoting the number of a *p*-block of *tbl*. Alternatively, let *B* be the *b*-th *p*-block of *tbl*.

NormalizerOfDefectGroup returns the normalizer in *G* of a defect group of the *b*-th *p*-block of *tbl*.

If the UnderlyingGroup (**Reference: UnderlyingGroup for character tables**) value is not set in *tbl* then fail is returned.

Note that there are methods for fetching defect normalizer information from the CTBlocks database, see NormalizerOfDefectGroup.

Example

```
gap> tbl:= CharacterTable( AlternatingGroup( 5 ) );;
gap> NormalizerOfDefectGroup( tbl, 2, 1 );
Group([ (2,3)(4,5), (2,4)(3,5), (3,4,5) ])
gap> NormalizerOfDefectGroup( Block( CharacterTable( "A5" ), 2, 1 ) );
fail
```

3.4.4 ClassPositionsOfDefectGroupOfClass

▷ ClassPositionsOfDefectGroupOfClass(*tbl, p, c*) (operation)

Let *p* be a prime integer and *c* be the position of a *p*-regular class in the ordinary character table *tbl*. ClassPositionsOfDefectGroupOfClass returns the list of positions of those classes that contain elements of the defect groups of the *c*-th class.

Note that for a *p*-regular element *g* in a group *G*, the classes of those *p*-elements in *G* that contain elements in the centralizer of *g* in *G* are the *p*-parts of those classes of *G* that contain the roots of *g*.

Example

```
gap> tbl:= CharacterTable( "J2" );;
gap> OrdersClassRepresentatives( tbl );
[ 1, 2, 2, 3, 3, 4, 5, 5, 5, 5, 6, 6, 7, 8, 10, 10, 10, 10, 12, 15,
  15 ]
gap> ClassPositionsOfDefectGroupOfClass( tbl, 2, 4 );
[ 1, 2, 6 ]
```

3.4.5 ClassPositionsOfDefectGroupOfBlock

▷ ClassPositionsOfDefectGroupOfBlock(*B*) (attribute)
▷ ClassPositionsOfDefectGroupOfBlock(*tbl, p, b*) (operation)

Let *tbl* be an ordinary character table, *p* be a prime integer, and *b* be a positive integer denoting the number of a *p*-block of *tbl*. Alternatively, let *B* be the *b*-th *p*-block of *tbl*.

ClassPositionsOfDefectGroupOfBlock returns the list of positions of those classes that contain elements of the defect groups of the *b*-th *p*-block of *tbl*.

Example

```
gap> tbl:= CharacterTable( "J2" );;
gap> ClassPositionsOfDefectGroupOfBlock( Block( tbl, 2, 1 ) );
[ 1, 2, 3, 6, 14 ]
gap> ClassPositionsOfDefectGroupOfBlock( tbl, 2, 1 );
[ 1, 2, 3, 6, 14 ]
```

3.4.6 IsClassWithNormalDefectGroup

▷ IsClassWithNormalDefectGroup(*tbl, p, c*) (operation)

returns true if the *p*-defect groups of the *c*-th class of the ordinary character table *tbl* are normal subgroups of the group of *tbl*, and false otherwise.

The *c*-th class of *tbl* must be *p*-regular.

Example

```
gap> IsClassWithNormalDefectGroup( CharacterTable( "A5" ), 2, 1 );
false
gap> IsClassWithNormalDefectGroup( CharacterTable( "A5" ), 2, 3 );
true
```

3.4.7 IsBlockWithNormalDefectGroup

▷ IsBlockWithNormalDefectGroup(*B*) (property)
▷ IsBlockWithNormalDefectGroup(*tbl*, *p*, *b*) (operation)

returns true if the defect groups of the *b*-th *p*-block of the ordinary character table *tbl* are normal subgroups of the group of *tbl*, and false otherwise.

The block can also be entered as a block object *B*.

Example

```
gap> IsBlockWithNormalDefectGroup( CharacterTable( "A5" ), 2, 1 );
false
gap> IsBlockWithNormalDefectGroup( Block( CharacterTable( "A5" ), 2, 2 ) );
true
```

3.4.8 IsClassWithCyclicDefectGroup

▷ IsClassWithCyclicDefectGroup(*tbl*, *p*, *c*) (operation)

returns true if the *p*-defect groups of the *c*-th class of the ordinary character table *tbl* are cyclic, and false otherwise.

The *c*-th class of *tbl* must be *p*-regular.

Example

```
gap> IsClassWithCyclicDefectGroup( CharacterTable( "A5" ), 2, 1 );
false
gap> IsClassWithCyclicDefectGroup( CharacterTable( "A5" ), 2, 3 );
true
```

3.4.9 IsBlockWithCyclicDefectGroup

▷ IsBlockWithCyclicDefectGroup(*B*) (property)
▷ IsBlockWithCyclicDefectGroup(*tbl*, *p*, *b*) (operation)

returns true if the defect groups of the *b*-th *p*-block of the ordinary character table *tbl* are cyclic, and false otherwise.

The block can also be entered as a block object *B*.

Example

```
gap> IsBlockWithCyclicDefectGroup( CharacterTable( "A5" ), 2, 1 );
false
gap> IsBlockWithCyclicDefectGroup( Block( CharacterTable( "A5" ), 2, 2 ) );
true
```

3.4.10 TestBlockWithAbelianDefectGroup

▷ TestBlockWithAbelianDefectGroup(*B*) (attribute)
▷ TestBlockWithAbelianDefectGroup(*tbl*, *p*, *b*) (operation)
▷ IsBlockWithAbelianDefectGroup(*B*) (attribute)
▷ IsBlockWithAbelianDefectGroup(*tbl*, *p*, *b*) (operation)

Brauer's height zero conjecture states that the defect groups of a p-block of G are abelian if and only if all ordinary irreducible characters in the block have height zero (cf. [LP10, Conjecture 4.14.6] or [Nav98, p. 212]). If this conjecture would be true then it would allow us to compute from the character table of a group whether the defect groups of a block are abelian.

Currently it is known that the existence of an irreducible character of positive defect implies that the defect groups of its block are nonabelian (see [KM13]). The other direction of the conjecture is still open, and it is in general not known whether the character table of a group determines which defect groups of its p-blocks are abelian. Therefore, there is no GAP function `IsBlockWithAbelianDefectGroup` with return values only `true` and `false`. Instead, `TestBlockWithAbelianDefectGroup` tests some necessary conditions and allows for a `fail` result. The return value is a record as described below.

Let *`tbl`* be an ordinary character table, `p` be a prime integer, and `b` be a positive integer denoting the number of a `p`-block of *`tbl`*. Alternatively, let *`B`* be the `b`-th `p`-block of *`tbl`*.

`TestBlockWithAbelianDefectGroup` returns a record with the following components.

`resultHZC`
: `true` if all ordinary irreducible characters in the block have height zero, and `false` otherwise; if Brauer's height zero conjecture is true then this holds if and only if the defect groups of the block are abelian,

`result`
: `true` or `false` if the defect groups of the block can be shown to be abelian or nonabelian, respectively, and `fail` if the criteria listed below do not suffice for a decision.

`comment`
: a string that explains the reason for the value of the `result` component.

The following criteria are used.

- If not all ordinary irreducible characters in the block have height zero then the defect groups of the block are nonabelian, by [KM13].
- If the defect is at most 2 or if the exponent of the defect groups is 2 or if the defect groups are cyclic then the defect groups are abelian.
- If *`tbl`* is `p`-solvable then the height zero conjecture holds for *`tbl`*, by [GW84].
- If the defect groups contain elements whose centralizer order is not divisible by the order of the defect groups then the defect groups are nonabelian.
- For a block of maximal defect, [NST15] yields that the defect groups (the Sylow subgroups) are abelian if and only if all ordinary irreducibles in the principal `p`-block have height zero and the class lengths of all `p`-elements are not divisible by `p`.
- For a block whose defect group is normal in the group of *`tbl`*, the height zero conjecture is true by a theorem of Reynolds, see [Nav98, Thm. 9.23].
- If the Sylow `p`-subgroups the group of *`tbl`* are abelian (which can be computed by [NST15]) then the defect groups of any `p`-block are abelian.

Example

```
gap> TestBlockWithAbelianDefectGroup( CharacterTable( "A5" ), 2, 1 );
rec( comment := "defect at most 2", result := true, resultHZC := true
 )
gap> TestBlockWithAbelianDefectGroup( CharacterTable( "A6" ), 2, 1 );
rec( comment := "not all irreducibles have height zero",
  result := false, resultHZC := false )
gap> TestBlockWithAbelianDefectGroup( CharacterTable( "J1" ), 2, 1 );
rec( comment := "defect group is an el. ab. 2-group", result := true,
  resultHZC := true )
gap> TestBlockWithAbelianDefectGroup(
>      CharacterTable( "2.L2(23).2" ), 2, 2 );
rec( comment := "cyclic defect group", result := true,
  resultHZC := true )
gap> TestBlockWithAbelianDefectGroup(
>      CharacterTable( "U3(11)" ), 2, 2 );
rec( comment := "no criterion yields a decision", result := fail,
  resultHZC := true )
```

More examples are shown in Section 2.2.

`IsBlockWithAbelianDefectGroup` returns the value of the `result` component of the result of `TestBlockWithAbelianDefectGroup`, called with the same arguments.

Example

```
gap> IsBlockWithAbelianDefectGroup( CharacterTable( "A5" ), 2, 1 );
true
gap> IsBlockWithAbelianDefectGroup( CharacterTable( "A6" ), 2, 1 );
false
gap> IsBlockWithAbelianDefectGroup( CharacterTable( "J1" ), 2, 1 );
true
gap> IsBlockWithAbelianDefectGroup( CharacterTable( "2.L2(23).2" ),
>        2, 2 );
true
gap> IsBlockWithAbelianDefectGroup( CharacterTable( "U3(11)" ),
>        2, 2 );
fail
```

(Note that technically, `IsBlockWithAbelianDefectGroup` cannot be implemented via a property in the sense of GAP's type system, see (**Reference: Properties**). It is implemented via an attribute.)

3.4.11 ClassPositionsOfDefectClasses

▷ `ClassPositionsOfDefectClasses(`*B*`)` (attribute)

▷ `ClassPositionsOfDefectClasses(`*tbl*`, `*p*`, `*b*`)` (operation)

Let *tbl* be an ordinary character table, *p* be a prime integer, and *b* be a positive integer denoting the number of a *p*-block of *tbl*. Alternatively, let *B* be the *b*-th *p*-block of *tbl*.

`ClassPositionsOfDefectClasses` returns the list of positions of those conjugacy classes of *tbl* that are defect classes of its *b*-th *p*-block if GAP can decide this with reasonable effort, and `fail` otherwise. The result `fail` occurs if and only if either *tbl* has no *b*-th *p*-block or if the known Conway polynomials (see `ConwayPolynomial` (**Reference: ConwayPolynomial**)) do not suffice.

Example

```
gap> ClassPositionsOfDefectClasses( CharacterTable( "A5" ), 2, 1 );
[ 1 ]
gap> b:= Block( CharacterTable( "A5" ), 2, 2 );;
gap> ClassPositionsOfDefectClasses( b );
[ 3, 4, 5 ]
gap> b:= Block( CharacterTable( "J4" ), 23, 32 );;
gap> ClassPositionsOfDefectClasses( b );
#I  the Conway polynomial of degree 21 for p = 23 is not known
fail
```

3.4.12 ClassPositionsOfDefectClassesUpToGaloisConjugacy

▷ `ClassPositionsOfDefectClassesUpToGaloisConjugacy(`*B*`)` (attribute)
▷ `ClassPositionsOfDefectClassesUpToGaloisConjugacy(`*tbl*`, `*p*`, `*b*`)` (operation)

`ClassPositionsOfDefectClassesUpToGaloisConjugacy` is similar to `ClassPositionsOfDefectClasses` (3.4.11). The result is either `fail` or the list of those class positions such that at least one algebraic conjugate class is a defect class of the *b*-th *p*-block of *tbl*.

By the lemma from Section 3.2.1, this information can be computed without using Conway polynomials, thus `fail` is returned only if *tbl* has no *b*-th *p*-block. (Note that `ClassPositionsOfDefectClasses` (3.4.11) returns `fail` also if not all defect classes can be determined.)

Example

```
gap> b:= Block( CharacterTable( "A5" ), 2, 2 );
Block( CharacterTable( "A5" ), 2, 2 )
gap> ClassPositionsOfDefectClassesUpToGaloisConjugacy( b );
[ 3, 4, 5 ]
gap> ClassPositionsOfDefectClassesUpToGaloisConjugacy(
>        CharacterTable( "J4" ), 23, 32 );
[ 2, 3, 5, 6, 7, 14, 15, 16, 19, 20, 34, 57, 58, 59, 60 ]
```

3.5 Radical *p*-subgroups

Let *D* be a *p*-subgroup of the group *G*, where *p* is a prime integer. Then *D* is called a *radical p-subgroup of G* if *D* is equal to the *p*-core (see `PCore` (**Reference: PCore**)) of the normalizer of *D* in *G*, see [LP10, Def. 4.7.25] and [Nav98, p. 84].

For example, the *p*-core of *G* (see `PCore` (**Reference: PCore**)) is radical, and every radical *p*-subgroup of *G* contains the *p*-core of *G*. Moreover, the defect groups of *p*-blocks of *G* (see Section 3.4) are radical. In particular the Sylow *p*-subgroups of *G* are radical.

Radical *p*-subgroups are important for example if one deals with *p*-weights, see Section 3.7. See Section 3.6 for computing chains of radical *p*-subgroups.

3.5.1 IsRadicalPSubgroup

▷ `IsRadicalPSubgroup(`*G*`, `*D*`, `*p*`)` (operation)

For a subgroup *D* of the group *G* (this relation is *not* checked), IsRadicalPSubgroup returns true if *D* is a radical *p*-subgroup of *G* (see above), and false otherwise.

Example

```
gap> g:= SmallGroup( 48, 29 );;  # GL(2,3)
gap> syl2:= SylowSubgroup( g, 2 );;
gap> ccl:= ConjugacyClassesSubgroups( syl2 );;
gap> reps:= List( ccl, Representative );;
gap> List( reps, Size );
[ 1, 2, 2, 4, 4, 4, 8, 8, 8, 16 ]
gap> israd:= List( reps, r -> IsRadicalPSubgroup( g, r, 2 ) );
[ false, false, false, false, false, false, true, false, false, true ]
```

3.5.2 NormalizerOfRadicalPSubgroup

▷ NormalizerOfRadicalPSubgroup(*G, D, p*) (operation)

Let *D* be a *p*-subgroup of the group *G*, where *p* is a prime integer. (It is *not* checked whether *D* is actually a subgroup of *G*.) NormalizerOfRadicalPSubgroup returns fail if *D* is not a radical *p*-subgroup of *G*, see IsRadicalPSubgroup (3.5.1), and the normalizer of *D* in *G* otherwise.

The advantage of using this function instead of IsRadicalPSubgroup (3.5.1) is that the normalizer need not be computed twice if one is interested in it.

Example

```
gap> # (continuing the example for IsRadicalPSubgroup)
gap> norm:= List( reps, r -> NormalizerOfRadicalPSubgroup( g, r, 2 ) );
[ fail, fail, fail, fail, fail, fail, Group([ f1, f2, f3, f4, f5 ]),
  fail, fail, Group([ f1, f3, f4, f5 ]) ]
gap> Positions( israd, false ) = Positions( norm, fail );
true
```

3.5.3 RepresentativesRadicalPSubgroupsAndNormalizers

▷ RepresentativesRadicalPSubgroupsAndNormalizers(*G, p[, arec]*) (function)
▷ RepresentativesRadicalPSubgroupsAndNormalizers(*tom, p*) (function)

Let the first argument be either a group *G* or the table of marks *tom* of the group *G* such that the filter IsTableOfMarksWithGens (**Reference: IsTableOfMarksWithGens**) is set in *tom*, and let *p* be a prime integer. RepresentativesRadicalPSubgroupsAndNormalizers returns a list of records with the components subgroup and normalizer, with the values U and N, respectively, such that U runs over representatives of conjugacy classes of radical p-subgroups of *G* and N is the normalizer of U in *G*.

Calling PCore (**Reference: PCore**) with the arguments N and *p* yields U, this value is cached inside N.

If the first argument is a table of marks *tom* then also the components subgroupnr and normalizernr are bound in the records that are returned, their values are the positions of the conjugacy classes of U and N in the list of classes of subgroups of *tom*. Thus these numbers can be used for fetching a conjugate of U and N with RepresentativeTom (**Reference: RepresentativeTom**).

The optional argument `arec`, if given, must be a record. It can be used to submit data for accelerating the computations in the case that a group `G` is given as the first argument. The following components of `arec` are supported.

`Maxes`
: a list l of subgroups of `G` such that any p-local subgroup of `G` is conjugate to a subgroup of one of the groups in l; this holds if l contains representatives of all classes of maximal subgroups of `G`, but often a smaller list suffices,

`CharacterTable`
: the ordinary character table of `G`.

The idea behind the `Maxes` component is that the union L, say, of class representatives of p-radical subgroups in the subgroups in the list l covers the classes of non-normal radical p-subgroups of `G`: If U is p-radical in `G` and not normal in G then U is p-radical in some maximal subgroup of `G` that contains U. (If U is not normal in `G` then the normalizer N of U in `G` is contained in some maximal subgroup M of `G`, thus N is also the normalizer of U in M, and hence U is p-radical in M.) Thus we can find candidates for all classes of radical p-subgroups of `G` by omitting those subgroups from L that are not p-radical in `G`, omitting also duplicates in the sense that the subgroups are conjugate in `G`, and adding the p-core of `G`.

Example

```
gap> g:= DihedralGroup( 8 );;
gap> RepresentativesRadicalPSubgroupsAndNormalizers( g, 2 );
[ rec( normalizer := Group([ f1, f2, f3 ]),
      subgroup := Group([ f1, f2, f3 ]) ) ]
gap> g:= SymmetricGroup( 5 );;
gap> RepresentativesRadicalPSubgroupsAndNormalizers( g, 2 );
[ rec( normalizer := Sym( [ 1 .. 5 ] ), subgroup := Group(()) ),
  rec( normalizer := Group([ (3,4,5), (3,4), (1,2) ]),
      subgroup := Group([ (1,2) ]) ),
  rec( normalizer := Group([ (1,2)(3,4), (1,3)(2,4), (2,4), (2,3) ]),
      subgroup := Group([ (1,3)(2,4), (1,2)(3,4) ]) ),
  rec( normalizer := Group([ (1,4)(2,3), (3,4), (1,2) ]),
      subgroup := Group([ (3,4), (1,2), (1,3)(2,4) ]) ) ]
gap> tom:= TableOfMarks( "A5" );;
gap> RepresentativesRadicalPSubgroupsAndNormalizers( tom, 2 );
[ rec( normalizer := Alt( [ 1 .. 5 ] ), normalizernr := 9,
      subgroup := Group(()), subgroupnr := 1 ),
  rec( normalizer := Group([ (2,4)(3,5), (2,3)(4,5), (3,4,5) ]),
      normalizernr := 8, subgroup := Group([ (2,3)(4,5), (2,4)(3,5) ])
        , subgroupnr := 4 ) ]
```

The above computation shows that the alternating group on five points has exactly two classes of radical 2-subgroups, the trivial subgroup and a Klein four group contained in the stabilizer of a point. Thus we can restrict the computations to this point stabilizer and get essentially the same result.

Example

```
gap> g:= AlternatingGroup( 5 );;
gap> r:= rec( Maxes:= [ Stabilizer( g, 5 ) ],
>             CharacterTable:= CharacterTable( "A5" ) );;
gap> RepresentativesRadicalPSubgroupsAndNormalizers( g, 2, r );
```

```
[ rec( normalizer := Alt( [ 1 .. 5 ] ), subgroup := Group(()) ),
  rec( normalizer := Group([ (1,4)(2,3), (1,3)(2,4), (2,3,4) ]),
      subgroup := Group([ (1,4)(2,3), (1,3)(2,4) ]) ) ]
```

The following methods are available.

- If a table of marks *tom* is given then we run over the classes of p-subgroups.
- If a group *G* but no record *arec* with Maxes component is given then first we run over the classes of subgroups of a Sylow p-subgroup of G/N, where N is the p-core of *G*, then fuse G/N-conjugate classes, and finally compute preimages under the natural epimorphism from *G* to G/N.
- If a group *G* and a record *arec* with Maxes component l are given then we run over the subgroups in l and compute, for each such subgroup U, its p-radical subgroups with the above method. Finally, representatives under *G*-conjugacy are computed.

Once RepresentativesRadicalPSubgroupsAndNormalizers has computed the result, it is stored via the attribute ComputedRepresentativesRadicalPSubgroupsAndNormalizers.

3.6 Chains of radical p-subgroups

A *p-chain of length n* of the group G is any strictly increasing chain $P_0 < P_1 < \cdots < P_n$ of p-subgroups of G, see [LP10, p. 433]. The *normalizer* of this chain is defined as $N_0 \cap N_1 \cap \cdots \cap N_n$, where N_i is the normalizer of P_i in G.

3.6.1 RepresentativesChainsOfRadicalPSubgroupsAndNormalizers

▷ RepresentativesChainsOfRadicalPSubgroupsAndNormalizers(*G, p[, arec]*) (function)
▷ RepresentativesChainsOfRadicalPSubgroupsAndNormalizers(*tom, p*) (function)

Called with the same arguments as RepresentativesRadicalPSubgroupsAndNormalizers (3.5.3), this function returns a list l, say, such that $l[k]$ describes those ascending p-chains of length k of the group in question (that is, either *G* or the underlying group of *tom*), up to conjugation in this group, that start at the p-core P, say, of this group and that consist of radical p-subgroups. Each entry in $l[k]$ is a record with the following components.

subgroups
: the list $[P_1, P_2, \ldots, P_k]$ such that $P < P_1 < P_2 < \cdots < P_k$ is the p-chain in question (note that the p-core itself does not appear in the list) and

normalizer
: the normalizer of the chain.

The function calls the function RepresentativesRadicalPSubgroupsAndNormalizers (3.5.3) for computing class representatives of radical p-subgroups. The optional argument *arec*, if present, is used for this purpose.

Example

```
gap> g:= DihedralGroup( 8 );;
gap> RepresentativesChainsOfRadicalPSubgroupsAndNormalizers( g, 2 );
[ ]
gap> g:= SymmetricGroup( 5 );;
gap> RepresentativesChainsOfRadicalPSubgroupsAndNormalizers( g, 2 );
[ [ rec( normalizer := Group([ (3,4,5), (3,4), (1,2) ]),
          subgroups := [ Group([ (1,2) ]) ] ),
      rec( normalizer := Group([ (1,2)(3,4), (1,3)(2,4), (2,4), (2,3)
             ]), subgroups := [ Group([ (1,3)(2,4), (1,2)(3,4) ]) ] ),
      rec( normalizer := Group([ (1,4)(2,3), (3,4), (1,2) ]),
          subgroups := [ Group([ (3,4), (1,2), (1,3)(2,4) ]) ] ) ],
  [
      rec( normalizer := Group([ (3,4), (1,2) ]),
          subgroups :=
            [ Group([ (1,2) ]), Group([ (3,4), (1,2), (1,3)(2,4) ]) ]
         ),
      rec( normalizer := Group([ (1,4)(2,3), (3,4), (1,2) ]),
          subgroups := [ Group([ (1,3)(2,4), (1,2)(3,4) ]),
              Group([ (3,4), (1,2), (1,3)(2,4) ]) ] ) ] ]
gap> tom:= TableOfMarks( "A5" );;
gap> RepresentativesChainsOfRadicalPSubgroupsAndNormalizers( tom, 2 );
[ [ rec( normalizer := Group([ (2,4)(3,5), (2,3)(4,5), (3,4,5) ]),
          subgroups := [ Group([ (2,3)(4,5), (2,4)(3,5) ]) ] ) ] ]
```

`RepresentativesChainsOfRadicalPSubgroupsAndNormalizers` stores the computed result via the attribute `ComputedRepresentativesChainsOfRadicalPSubgroupsAndNormalizers`.

3.7 p-weights

Let G be a group, p be a prime integer, P be a p-subgroup of G, and N be the normalizer of P in G. A *p-weight* of G is defined as a pair (P,ψ) where ψ is an irreducible defect zero character of N/P, that is, the p-parts of $\psi(1)$ and $|N/P|$ are equal (see [Nav98, p. 90]).

The p-weight (P,ψ) is said to *belong* to the p-block B of G if the p-block of ψ induces B, see Section 3.8. Note that each p-weight belongs to a block, because block induction (in the sense of Brauer) from N to G is always defined, see for example [LP10, Thm. 4.7.19].

A more suitable way of representing p-weights in our context –we want to deal with character tables instead of groups– is the following.

Let t be the ordinary character table of G and n be the ordinary character table of N, and if n is different from t then assume that the class fusion of N in G is stored on n. Then the p-weight (P,ψ) can be represented by the pair (n,ψ'), where ψ' is the irreducible character of N that is obtained from ψ by inflation.

Note that P is p-radical in G (see Section 3.5) if N/P has a p-block of defect zero, thus P is uniquely determined by N as its p-core. (To see this, note that the p-core of a group G is contained in the defect groups of all p-blocks of G, see [Nav98, Thm. 4.8] or [LP10, Thm. 4.6.10].)

Note also that the character table of N determines which of its conjugacy classes lie in P, see `ClassPositionsOfPCore` (**Reference: ClassPositionsOfPCore**).

In order to deal also with cases where the ordinary character table of N is not known, we provide the following alternative description of p-weights. Suppose that there is a chain $1 < K \leq P < N \leq$

$M < G$ of subgroups of G such that K is normal in M, and such that the character tables m, m', and n' of M, M/K, and N/K, respectively, are known. Again we have to assume that the relevant class fusions are stored on the character tables: the fusion that describes the embedding of N/K in M/K and the fusion that describes the projection from M to M/K. In this situation, we can represent (P, ψ) by the pair $((n', m', m), \psi')$, where ψ' is the irreducible character of N/K that is obtained from ψ by inflation.

Note that we can compute, for a p-weight described by $((n', m', m), \psi')$, to which p-block of G it belongs: Let ψ be the irreducible character of N that is obtained from ψ' by inflation, and let B denote the p-block of G that is induced by the p-block of ψ. We compute B by first inducing the p-block of ψ' from N/K to M/K, then taking the corresponding block of M, and then inducing this block to G. (Note that block induction is transitive, see for example [LP10, Exercise 4.7.3]).

3.7.1 DescriptionOfPWeights

▷ `DescriptionOfPWeights(B)` (function)
▷ `DescriptionOfPWeights(tbl, p[, b])` (function)

First consider the case of two arguments. Let `tbl` be the ordinary character table of a group G, say, and `p` be a prime integer. `DescriptionOfPWeights` returns either `fail` or a record r that describes the p-weights of G, up to G-conjugacy, as follows.

Suppose that G has k conjugacy classes of radical p-subgroups, with representatives $P_1, P_2, \ldots, P_k$.

The `normalizers` component of r is a dense list $[n_1, n_2, \ldots, n_k]$, where n_i is either the character table of the normalizer N_i of P_i in G, or n_i is a triple $((n_i)', (m_i)', m_i)$ of character tables as defined above in this section, which describes this normalizer.

The `characters` component of r is a dense list of the same length k, its i-th entry is the list of those irreducible characters ψ of n_i or of $(n_i)'$, respectively, with the property that P_i is contained in the kernel of ψ and that ψ has defect zero w.r.t. the factor group N_i/P_i.

The `blocks` component of r is a dense list of the same length k, its i-th entry is the list of numbers of the blocks in `tbl` to which the given characters ψ of n_i belong.

Now consider the case of three arguments, where `b` is a positive integer, and the case of one argument `B` that stands for the `b`-th `p`-block of `tbl`. In these cases, the format of the result is the same, but it is restricted to those classes of radical p-subgroups and to those irreducible characters for which the induced block is `B`.

Example

```
gap> DescriptionOfPWeights( CharacterTable( AlternatingGroup( 5 ) ), 2 );
rec( blocks := [ [ 2 ], [ 1, 1, 1 ] ],
  characters :=
    [ [ Character( CharacterTable( Alt( [ 1 .. 5 ] ) ),
          [ 4, 0, 1, -1, -1 ] ) ],
      [ Character( CharacterTable( Alt( [ 1 .. 4 ] ) ),
          [ 1, 1, 1, 1 ] ),
          Character( CharacterTable( Alt( [ 1 .. 4 ] ) ),
          [ 1, E(3)^2, E(3), 1 ] ),
          Character( CharacterTable( Alt( [ 1 .. 4 ] ) ),
          [ 1, E(3), E(3)^2, 1 ] ) ] ],
  normalizers := [ CharacterTable( Alt( [ 1 .. 5 ] ) ),
      CharacterTable( Alt( [ 1 .. 4 ] ) ) ] )
gap> DescriptionOfPWeights( CharacterTable( AlternatingGroup( 5 ) ), 2, 1 );
rec(
```

```
characters :=
  [ [  ], [ Character( CharacterTable( Alt( [ 1 .. 4 ] ) ),
        [ 1, 1, 1, 1 ] ),
        Character( CharacterTable( Alt( [ 1 .. 4 ] ) ),
        [ 1, E(3)^2, E(3), 1 ] ),
        Character( CharacterTable( Alt( [ 1 .. 4 ] ) ),
        [ 1, E(3), E(3)^2, 1 ] ) ] ],
normalizers := [ CharacterTable( Alt( [ 1 .. 5 ] ) ),
    CharacterTable( Alt( [ 1 .. 4 ] ) ) ] )
```

If the attribute `UnderlyingGroup` (**Reference: UnderlyingGroup for character tables**) is not set in *tbl* and if the *p*-weights of *tbl* are not yet known then `fail` is returned.

3.8 Block induction

Block induction relates certain p-blocks of subgroups of a group G to p-blocks of G. From the various definitions of induced blocks that appear in the literature, each two of the following ones coincide when both of them are defined.

In all cases, G is a finite group, p is a prime integer, H is a subgroup of G, and we are given a p-block b of H. Recall that the central character ω_b^* is defined as the p-modular reduction of the class function ω_χ, for any irreducible character χ in b.

Block induction in the sense of Brauer

Let $s_H : \mathrm{Z}(FG) \to \mathrm{Z}(FH)$ be the F-linear map that is defined by $C^+ \mapsto (C \cap H)^+$, for each conjugacy class C of G. If $\lambda_b \circ s_H = \lambda_B$ (see `CentralCharacter` (3.3.10)) holds for some p-block B of G then we call B the block induced by b, and write $B = b^G$.

The function `BrauerCorrespondent` (3.8.1) can be used to compute the number of the block B. See [LP10, Section 4.7] or [Nav98, p. 86 ff.] for details. Note that $\lambda_b \circ s_H$ can be computed as $(\omega_{\chi^G})^*$, if we extend the definition of `CentralCharacter` (**Reference: CentralCharacter**) to arbitrary characters, see [LP10, Lemma 4.7.5].

p-regular block induction

For any two different p-blocks B, C of G, the class functions λ_B, λ_C differ at some p-regular class, see [LP10, Proof of Thm. 4.4.8]. Thus it makes sense to define the induced block B by the condition that $\lambda_b \circ s_H$ and λ_B coincide at p-regular classes –if such a block B exists then it is unique. If b^G is defined in the sense of Brauer then of course $B = b^G$ holds.

The function `PRegularCorrespondent` (3.8.2) can be used to compute the number of the block B. Note that this computation is cheaper than that of b^G. Thus it is advisable to call `PRegularCorrespondent` (3.8.2) instead of `BrauerCorrespondent` (3.8.1) if one knows in advance that b^G is defined.

Block induction in the sense of Alperin-Burry

This induction concept is defined not character theoretically but using the interpretation of the p-blocks as the indecomposable direct summands of the group algebra RG. Namely, RG is a $R[G \times G]$-module, w. r. t. the action $x \cdot (g_1, g_2) = g_1^{-1} x g_2$, for $x \in RG$ and $g_1, g_2 \in G$. Restricting this action on a block B of RG to $R[H \times H]$ yields a direct sum of blocks of RH. If B is the unique block of RG with the property that the block b of RH occurs as a direct summand of the

restriction then block induction in the sense of Alperin-Burry is defined, and B is the induced block (see [AB80]).

The function `AlperinBurryCorrespondent` (3.8.3) can be used to compute the number of the block B, provided that the underlying groups of the character tables in question are available.

Extended block induction

If B is the unique block of G with the property $(\lambda_b \circ s_H)((f_B)^*) \neq 0$, where $f_B \in \mathrm{Z}(RG)$ is the block idempotent of B (see `CoefficientsOfOsimaIdempotent` (3.3.13)), then we say that extended block induction (or block induction in the sense of Wheeler) is defined for b, and B is the induced block. It is shown in [Whe94] that this condition holds if block induction is defined in the sense of Brauer or of Alperin-Burry, thus it provides a common generalization of these concepts. (In fact it generalizes p-regular induction.)

The function `WheelerCorrespondent` (3.8.4) can be used to compute the number of the block B. Using [Whe94, Lemma 1.3], we can compute the induced block via the equality $(\lambda_b \circ s_H)((f_B)^*) = (\chi^B(1)/\chi^G(1))^*$, where χ is any character in b and χ^B is the restriction of χ^G to B. (With this approach, we need not compute reductions of cyclotomic numbers to finite fields, cf. Section 3.2.

The following functions implement the above concepts of block induction. In all cases, four arguments occur. The first two are the ordinary character tables of the two groups H and G, with $H \leq G$, the third is the prime integer p, and and the fourth is either a positive integer b denoting the number of a p-block of the character table of H or an irreducible character of H that lies in the b-th p-block of H.

Alternatively, the b-th p-block of the character table of H can be entered via a block object B. In this case, only the character table of G must be supplied in addition to B.

3.8.1 BrauerCorrespondent

▷ `BrauerCorrespondent(`*`B, tblG`*`)` (operation)
▷ `BrauerCorrespondent(`*`tblH, tblG, p, b`*`)` (operation)

Called with the arguments as described above, `BrauerCorrespondent` returns the number of the induced `p`-block of *`tblG`* if block induction to `G` is defined for the given block in the sense of Brauer (see above), otherwise `fail` is returned.

3.8.2 PRegularCorrespondent

▷ `PRegularCorrespondent(`*`B, tblG`*`)` (operation)
▷ `PRegularCorrespondent(`*`tblH, tblG, p, b`*`)` (operation)

Called with the arguments as described above, `PRegularCorrespondent` returns the number of the induced `p`-block of G if p-regular block induction to `G` is defined for the given block, otherwise `fail` is returned.

3.8.3 AlperinBurryCorrespondent

▷ `AlperinBurryCorrespondent(`*`B, tblG`*`)` (operation)
▷ `AlperinBurryCorrespondent(`*`tblH, tblG, p, b`*`)` (operation)

Called with the arguments as described above, `AlperinBurryCorrespondent` returns the number of the induced p-block of G if block induction to `G` is defined for the given block in the sense of Alperin-Burry (see above), otherwise `fail` is returned. The currently available methods require that underlying groups are stored for *`tblH`* and *`tblG`*, and that these groups are compatible.

3.8.4 WheelerCorrespondent

▷ `WheelerCorrespondent(`*`B, tblG`*`)` (operation)
▷ `WheelerCorrespondent(`*`tblH, tblG, p, b`*`)` (operation)

Called with the arguments as described above, `WheelerCorrespondent` returns the number of the induced p-block of G if extended block induction to `G` is defined for the given block, otherwise `fail` is returned.

3.9 Reality questions about blocks

A conjugacy class C of the group G is called *real* if C is equal to the class of inverses of the elements in C (see `RealClasses` (**Reference: RealClasses**)).

The class C is called *strongly real* if its elements are either the identity in G or are inverted by conjugation with an involution (that is, an element of order two) in G. Equivalently, the elements in a strongly real class C are the identity or products of at most two involutions in G; note that if $t \in G$ is an involution such that $tg = g^{-1}t$ holds for $g \in C$ then $(tg)^2$ is the identity, hence $g = t(tg)$ is a product of at most two involutions.

A p-block B of G is called *real* if $\mathrm{Irr}(B)$ is invariant under complex conjugation (see `ComplexConjugate` (**Reference: ComplexConjugate**)). In [Nav98, p. 75], real p-blocks are called *selfdual*.

The p-block B is called *strongly real* (see [Mur06]) if it is real and has a strongly real defect class (see `ClassPositionsOfDefectClasses` (3.4.11)). Since a class is strongly real if and only if all of its algebraic conjugates are strongly real, the real p-block B is strongly real if and only if the set of defect classes of B up to Galois conjugacy (see `ClassPositionsOfDefectClassesUpToGaloisConjugacy` (3.4.12)) contains a strongly real class.

3.9.1 IsRealClass

▷ `IsRealClass(`*`tbl, c`*`)` (operation)

Let `c` be the position of a class in the ordinary character table *`tbl`*. `IsRealClass` returns `true` if the `c`-th class in *`tbl`* is real, and `false` otherwise.

Example

```
gap> tbl:= CharacterTable( "M22" );;
gap> List( [ 1 .. NrConjugacyClasses( tbl ) ],
>          c -> IsRealClass( tbl, c ) );
[ true, true, true, true, true, true, true, false, false, true,
  false, false ]
```

3.9.2 IsStronglyRealClass

▷ IsStronglyRealClass(*tbl*, *c*) (operation)

Let *c* be the position of a class in the ordinary character table *tbl*. IsStronglyRealClass returns true if the *c*-th class in *tbl* is strongly real, and false otherwise.

This property can be checked using the class multiplication coefficients of *tbl*, see ClassMultiplicationCoefficient (**Reference: ClassMultiplicationCoefficient for character tables**).

Example

```
gap> tbl:= CharacterTable( "M22" );;
gap> List( [ 1 .. NrConjugacyClasses( tbl ) ],
>          c -> IsStronglyRealClass( tbl, c ) );
[ true, true, true, true, true, true, true, false, false, false,
  false, false ]
```

3.9.3 IsRealBlock

▷ IsRealBlock(*B*) (property)
▷ IsRealBlock(*tbl*, *p*, *b*) (operation)

returns true if the *b*-th *p*-block of the ordinary character table *tbl* is real, and false otherwise.

The block can also be entered as a block object *B*.

Example

```
gap> IsRealBlock( CharacterTable( "A7" ), 5, 1 );
true
gap> IsRealBlock( Block( CharacterTable( "A7" ), 5, 2 ) );
false
```

3.9.4 IsStronglyRealBlock

▷ IsStronglyRealBlock(*B*) (property)
▷ IsStronglyRealBlock(*tbl*, *p*, *b*) (operation)

returns true if the *b*-th *p*-block of the ordinary character table *tbl* is strongly real, and false otherwise.

The block can also be entered as a block object *B*.

Example

```
gap> tbl:= CharacterTable( "2.A5" );;
gap> IsStronglyRealBlock( Block( tbl, 2, 1 ) );
true
gap> IsStronglyRealBlock( tbl, 2, 2 );
false
```

The principal block is always strongly real. The unique nonprincipal 2-block of $2.A_5 \cong SL(2,5)$ is not strongly regular; note that the corresponding defect zero block of the factor group A_5 is strongly regular.

Example

```
gap> tbl:= CharacterTable( "A5" );;
gap> IsStronglyRealBlock( tbl, 2, 2 );
true
```

(This example can be generalized: The group $G = SL(2,q)$, for odd q, contains exactly one involution, which is central in G. Thus G cannot have strongly real p-blocks of nonmaximal defect.)

3.10 Selecting blocks according to their invariants

The idea behind the functions described in this section is that one selects those p-blocks from a given set that satisfy certain conditions, where the candidates can be given either by a list of block objects or by a list of character tables whose p-blocks are considered, for all prime divisors p of the group order. (In the case of character tables from GAP's character table library, one can enter also a list of admissible names of these tables, for example their `Identifier` (**Reference: Identifier for character tables**) values.)

For example all p-blocks with positive defect or with trivial kernel can be considered, or more generally all p-blocks with prescribed values for some properties and attributes that have been introduced in the previous sections.

`AllPBlocks` (3.10.1) and `OnePBlock` (3.10.2) return all blocks and one block with the desired properties, respectively.

`BlockInvariants` (3.10.3) returns a record that contains a matrix of block invariants (one row for each block with the desired properties) and a description of the invariants corresponding to the columns. `DisplayBlockInvariants` (3.10.4) and `BrowseBlockInvariants` (3.10.5) (this function requires the Browse package [BL18]) show a tabular overview of these invariants. The columns of these matrices of block invariants can be customized, see `CTBlocks.BlockInvariantsColumns` (3.10.6).

3.10.1 AllPBlocks

▷ `AllPBlocks(`*`source[, fun1, val1, ...]`*`)` (function)

The first argument *`source`* must be either a list l_1 of block objects or (an admissible name of) an ordinary character table t or a list of (admissible names of) ordinary character tables l_2. If there is just this argument then `AllPBlocks` returns l_1 or the list of all p-blocks of t, for all primes dividing the order of the underlying group of t, or the concatenation of these lists for the tables in l_2.

Additional arguments *`fun1`*, *`val1`*, *`fun2`*, *`val2`* ... can be used to restrict the output. Each of *`fun1`*, *`fun2`*, ... must be a unary function that takes a block object as its argument, and each of *`val1`*, *`val2`*, ... describes the admissible values, as follows: The i-th value is admissible for the result of the i-th function applied to a given block object if the result is either equal to the value or (if the value is a list) is contained in the value or (if the value is a function) yields `true` when the value is applied to the result. `AllPBlocks` returns the list of those blocks for which the values of all functions are admissible.

Example

```
gap> t:= CharacterTable( "A5" );;
gap> AllPBlocks( t );
[ Block( CharacterTable( "A5" ), 2, 1 ),
```

```
    Block( CharacterTable( "A5" ), 2, 2 ),
    Block( CharacterTable( "A5" ), 3, 1 ),
    Block( CharacterTable( "A5" ), 3, 2 ),
    Block( CharacterTable( "A5" ), 3, 3 ),
    Block( CharacterTable( "A5" ), 5, 1 ),
    Block( CharacterTable( "A5" ), 5, 2 ) ]
  gap> AllPBlocks( t, Defect, 2 );
  [ Block( CharacterTable( "A5" ), 2, 1 ) ]
  gap> AllPBlocks( t, Defect, IsPosInt );
  [ Block( CharacterTable( "A5" ), 2, 1 ),
    Block( CharacterTable( "A5" ), 3, 1 ),
    Block( CharacterTable( "A5" ), 5, 1 ) ]
```

3.10.2 OnePBlock

▷ `OnePBlock(`*source[, fun1, val1, ...]*`)` (function)

The difference between `OnePBlock` and `AllPBlocks` (3.10.1) is that `OnePBlock`, when called with the same arguments, returns the first entry of the result of `AllPBlocks` (3.10.1) if this result is nonempty, and `fail` otherwise.

Example

```
gap> t:= CharacterTable( "A5" );;
gap> OnePBlock( t );
Block( CharacterTable( "A5" ), 2, 1 )
gap> OnePBlock( t, UnderlyingCharacteristic, 2 );
Block( CharacterTable( "A5" ), 2, 1 )
gap> OnePBlock( t, Defect, 2 );
Block( CharacterTable( "A5" ), 2, 1 )
gap> OnePBlock( t, Defect, IsPosInt );
Block( CharacterTable( "A5" ), 2, 1 )
```

3.10.3 BlockInvariants

▷ `BlockInvariants(`*source[, fun1, val1, ...][, arec]*`)` (function)

This function returns a record with the following components.

`list`
: a matrix of block invariants whose rows correspond to the block objects that are returned by `AllPBlocks` (3.10.1) when this function is called with the same arguments,

`columns`
: a list af records that correspond to the columns of the matrix stored in the `list` component; for the format of these records, see `CTBlocks.BlockInvariantsColumns` (3.10.6),

`header`
: a list of strings that describe the contents.

For the meaning of the arguments of this function except *arec*, see `AllPBlocks` (3.10.1). The optional argument *arec* can be used to customize the columns of the `list` component, see `CTBlocks.BlockInvariantsColumns` (3.10.6).

Example

```
gap> t:= CharacterTable( "A5" );;
gap> inv:= BlockInvariants( t );;
gap> List( inv.columns, r -> r.label );
[ "p", "b", "d", "k", "l", "c", "a", "n", "f", "r", "sr" ]
gap> Display( inv.list );
[ [ 2, 1, 2, 4, 3, false, true, false, true, true, true ],
  [ 2, 2, 0, 1, 1, true, true, true, true, true, true ],
  [ 3, 1, 1, 3, 2, true, true, false, true, true, true ],
  [ 3, 2, 0, 1, 1, true, true, true, true, true, true ],
  [ 3, 3, 0, 1, 1, true, true, true, true, true, true ],
  [ 5, 1, 1, 4, 2, true, true, false, true, true, true ],
  [ 5, 2, 0, 1, 1, true, true, true, true, true, true ] ]
```

The following values of *fun1*, *fun2*, ... have the additional effect that the corresponding column is omitted from the `list` component if exactly one admissible value is prescribed.

- `Defect` (3.4.1)
- `IsBlockWithAbelianDefectGroup` (3.4.10)
- `IsBlockWithCyclicDefectGroup` (3.4.9)
- `IsBlockWithNormalDefectGroup` (3.4.7)
- `IsFaithful` (3.3.12)
- `NumberOfBlock` (3.3.2)
- `UnderlyingCharacteristic` (3.3.2)

Example

```
gap> inv:= BlockInvariants( t, UnderlyingCharacteristic, 2 );;
gap> List( inv.columns, r -> r.label );
[ "b", "d", "k", "l", "c", "a", "n", "f", "r", "sr" ]
gap> Display( inv.list );
[ [ 1, 2, 4, 3, false, true, false, true, true, true ],
  [ 2, 0, 1, 1, true, true, true, true, true, true ] ]
```

3.10.4 DisplayBlockInvariants

▷ `DisplayBlockInvariants(`*source[, fun1, val1, ...][, arec]*`)` (function)

This function prints the return value of `BlockInvariants` (3.10.3), when this function is called with the same arguments, in tabular form.

If there is just one argument then `DisplayBlockInvariants` prints tabular information about the p-blocks of the given character table(s), for all primes p dividing the group order(s). Each row corresponds to a p-block, and each column corresponds to a block invariant.

(First we set the user preference `DisplayFunction` to the value `"Print"` in order to make sure that only ASCII characters are shown in the following example output. The value will be reset at the end of the examples in this manual chapter.)

Example

```
gap> origpref:= UserPreference( "AtlasRep", "DisplayFunction" );;
gap> SetUserPreference( "AtlasRep", "DisplayFunction", "Print" );
gap> t:= CharacterTable( "2.A5" );;
gap> DisplayBlockInvariants( t );
Block invariants for 2.A5

-----------------------------------------------
| p | b | d | k | l | c | a | n | f | r | sr |
-----------------------------------------------
| 2 | 1 | 3 | 7 | 3 | - | - | - | + | + |  + |
|   | 2 | 1 | 2 | 1 | + | + | + | + | + |  - |
| 3 | 1 | 1 | 3 | 2 | + | + | - | - | + |  + |
|   | 2 | 0 | 1 | 1 | + | + | + | - | + |  - |
|   | 3 | 0 | 1 | 1 | + | + | + | - | + |  - |
|   | 4 | 1 | 3 | 2 | + | + | - | + | + |  + |
|   | 5 | 0 | 1 | 1 | + | + | + | + | + |  - |
| 5 | 1 | 1 | 4 | 2 | + | + | - | - | + |  + |
|   | 2 | 0 | 1 | 1 | + | + | + | - | + |  - |
|   | 3 | 1 | 4 | 2 | + | + | - | + | + |  + |
-----------------------------------------------
```

See `CTBlocks.BlockInvariantsColumns` (3.10.6) for the meaning of the columns.

The meaning of the arguments *fun1*, *val1*, *fun2*, *val2* ... for restricting the rows and the meaning of *arec* for customizing the columns and the header is the same as described for `BlockInvariants` (3.10.3).

Example

```
gap> DisplayBlockInvariants( t, UnderlyingCharacteristic, 2 );;
Block invariants for 2.A5
p = 2

-----------------------------------------
| b | d | k | l | c | a | n | f | r | sr |
-----------------------------------------
| 1 | 3 | 7 | 3 | - | - | - | + | + |  + |
| 2 | 1 | 2 | 1 | + | + | + | + | + |  - |
-----------------------------------------
```

The user preference `DisplayFunction` from the package AtlasRep controls whether the output is just printed or shown using a pager. Depending on the value of this user preference and the terminal capabilities, nicer table borders may be used than are shown in the above examples.

3.10.5 BrowseBlockInvariants

▷ `BrowseBlockInvariants(`*source[, fun1, val1, ...][, arec]*`)` (function)

The arguments of this function are equal to the ones for `DisplayBlockInvariants` (3.10.4). The difference to that function is that `BrowseBlockInvariants` does not print the overview to the screen but shows it in a browse table, see (**Browse: Browsing Tables in GAP using ncurses –The User Interface**).

`BrowseBlockInvariants` can be used only if the Browse package [BL18] is loaded.

The full functionality of the function `NCurses.BrowseGeneric` (**Browse: NCurses.BrowseGeneric**) is available.

Example

```
gap> n:= [ 14, 14, 14, 14, 14, 14 ];;  # ‘‘do nothing’’
gap> enter:= [ NCurses.keys.ENTER ];;
gap> BrowseData.SetReplay( Concatenation(
>        "sc",                      # select the ’p’ column,
>        "sc",                      # categorize by this column,
>        "X",                       # expand all categories
>        n,                         # wait a little
>        "scrrrrrrr",               # select the ’f’ column
>        "f+", enter,               # restrict to the faithful blocks
>        n,                         # wait a little
>        "!",                       # reset the filtering etc.
>        n,                         # wait a little
>        "Q" ) );                   # and quit
gap> t:= CharacterTable( "2.A5" );;
gap> BrowseBlockInvariants( t );
gap> BrowseData.SetReplay( false );
```

3.10.6 Customizing the columns of block invariants overviews

▷ `CTBlocks.BlockInvariantsColumns` (global variable)

This variable defines the columns of the overviews that are computed by `BlockInvariants` (3.10.3), `DisplayBlockInvariants` (3.10.4), and `BrowseBlockInvariants` (3.10.5). Each column is described by a record with the following components.

`label`
: a string that serves as a column label in the table header,

`fun` a unary function that takes a block object as its argument, for example `Defect` (3.4.1),

`align`
: one of `"left"` or `"right"`, meaning the alignment of the values in the printed table,

`replace` **(optional)**
: if present, this must be a list of length three, the first entry being the list of admissible values, the second entry being the list of the corresponding strings that shall be printed in the overview, and the third entry being the list of the corresponding strings that shall be shown in the table header when the output is restricted to the given value; the same strings are also shown in the category rows of the browse table shown by `BrowseBlockInvariants` (3.10.5). If no component `replace` is present then `String` (**Reference: String**) is applied to the invariant in question, and the label together with the value will be used in the header and in category rows of the printed table.

Example

```
gap> CTBlocks.BlockInvariantsColumns[1];
rec( align := "left", fun := function( B ) ... end, label := "G" )
```

The columns of the overview tables can be customized via the optional record that can be entered as the last argument to `BlockInvariants` (3.10.3), `DisplayBlockInvariants` (3.10.4), and `BrowseBlockInvariants` (3.10.5). If this record contains the component `columns` with value a list of records as described above then this list of columns is considered instead of the default list `CTBlocks.BlockInvariantsColumns`. This way one can reduce, extend, or reorder the columns that are shown.

Example

```
gap> cols:= List( [ "c", "a", "n", "f" ],
>                 l -> First( CTBlocks.BlockInvariantsColumns,
>                             r -> r.label = l ) );;
gap> t:= CharacterTable( "2.A5" );;
gap> DisplayBlockInvariants( t, rec( columns:= cols ) );
Block invariants for 2.A5

-----------------
| c | a | n | f |
-----------------
| - | - | - | + |
| + | + | + | + |
| + | + | - | - |
| + | + | + | - |
| + | + | + | - |
| + | + | - | + |
| + | + | + | + |
| + | + | - | - |
| + | + | + | - |
| + | + | - | + |
-----------------
```

By default, the following columns appear.

`G` **(omitted if only one character table is given)**
: the identifier of the underlying character table,

`b`
: the block number,

`d`
: the defect of the block,

`k`
: the number of ordinary irreducibles in the block,

`l`
: the number of modular irreducibles in the block,

`c`
: + if the defect groups of the block are cyclic, - otherwise,

`a`
: + if the defect groups of the block are abelian, - if they are nonabelian, ? if the character table does not determine this property,

`n`
: + if the defect groups of the block are normal, - otherwise,

`f`
: + if the block is faithful, - otherwise,

`r`
: + if the block is real, - otherwise,

`sr` + if the block is strongly real, - otherwise,

Example

```
gap> List( CTBlocks.BlockInvariantsColumns, r -> r.label );
[ "G", "p", "b", "d", "k", "l", "c", "a", "n", "f", "r", "sr" ]
```

Another supported component of the optional record is `header`. If it is bound then its value must be a list of strings which apper as the `header` component of the record returned by `BlockInvariants` (3.10.3) and which are shown as a header above the tables printed by `DisplayBlockInvariants` (3.10.4) and `BrowseBlockInvariants` (3.10.5). The default for the header is `[ "Block invariants" ]` except that the line is extended by the identifier of the character table if exactly one character table is entered as the first argument.

Example

```
gap> DisplayBlockInvariants( t, Defect, IsPosInt,
>      rec( columns:= cols,
>           header:= [ "only Boolean invariants" ] ) );
only Boolean invariants

-----------------
| c | a | n | f |
-----------------
| - | - | - | + |
| + | + | + | + |
| + | + | - | - |
| + | + | - | + |
| + | + | - | - |
| + | + | - | + |
-----------------
```

(The user preference `DisplayFunction` from the AtlasRep had been changed before showing examples for `DisplayBlockInvariants` (3.10.4). Now we reset this preference to its original value.)

Example

```
gap> SetUserPreference( "AtlasRep", "DisplayFunction", origpref );
```

3.11 Centres of *p*-blocks as algebras

The functions described in this section create algebra objects (see (**Reference: Algebras**)) which are isomorphic with the centres of blocks or group algebras. The structure constants of the centre of a group algebra can be computed from the character table of the group in question, see `ClassMultiplicationCoefficient` (**Reference: ClassMultiplicationCoefficient for character tables**), and the centre of a *p*-block is a suitable subalgebra.

3.11.1 SCAlgebraCentreOfGroupAlgebra

▷ `SCAlgebraCentreOfGroupAlgebra(`*G, p*`)` (function)
▷ `SCAlgebraCentreOfGroupAlgebra(`*G, F*`)` (function)
▷ `SCAlgebraCentreOfGroupAlgebra(`*tbl, p*`)` (function)
▷ `SCAlgebraCentreOfGroupAlgebra(`*tbl, F*`)` (function)

Let either *G* be a group or *tbl* be the ordinary character table of a group.

If the second argument is a prime integer *p* then SCAlgebraCentreOfGroupAlgebra returns a s. c. algebra (see (**Reference: Constructing Algebras by Structure Constants**) that is isomorphic with the centre of the group algebra of the group, over the field with *p* elements.

If the second argument is a finite field *F* then the same holds, except that the coefficients domain of the algebra is *F*.

The canonical basis of the algebra corresponds to the class sums in the group; the basis vectors appear in the same ordering as in the ConjugacyClasses (**Reference: ConjugacyClasses attribute**) value for *G* or *tbl*, respectively.

Example

```
gap> tbl:= CharacterTable( "M11" );
CharacterTable( "M11" )
gap> a:= SCAlgebraCentreOfGroupAlgebra( tbl, 3 );
<algebra-with-one of dimension 10 over GF(3)>
gap> a.2 * a.3;
(Z(3))*C2+(Z(3))*C4+C7+C8+(Z(3))*C9+(Z(3))*C10
gap> Dimension( RadicalOfAlgebra( a ) );
8
gap> SCAlgebraCentreOfGroupAlgebra( tbl, GF(9) );
<algebra-with-one of dimension 10 over GF(3^2)>
```

3.11.2 BlockDecompositionSCAlgebraCentreOfGroupAlgebra

▷ BlockDecompositionSCAlgebraCentreOfGroupAlgebra(*tbl, p*) (function)

▷ BlockDecompositionSCAlgebraCentreOfGroupAlgebra(*tbl, F*) (function)

Let *tbl* be an ordinary character table.

If the second argument is a prime integer *p* then this function returns a list of subalgebras of the s. c. algebra returned by SCAlgebraCentreOfGroupAlgebra (3.11.1) when this is called with *tbl* and the smallest field that contains the *p*-modular reductions of all block idempotents of *tbl* in characteristic *p*. The *i*-th subalgebra corresponds to the *i*-th *p*-block of *tbl*.

If the second argument is a finite field *F* then the same holds, except that the coefficients domain of the algebras is *F*. If *F* is too small then fail is returned.

Example

```
gap> tbl:= CharacterTable( "M11" );
CharacterTable( "M11" )
gap> dec:= BlockDecompositionSCAlgebraCentreOfGroupAlgebra(
>              tbl, 5 );
[ <algebra of dimension 5 over GF(5^2)>,
  <algebra of dimension 1 over GF(5^2)>,
  <algebra of dimension 1 over GF(5^2)>,
  <algebra of dimension 1 over GF(5^2)>,
  <algebra of dimension 1 over GF(5^2)>,
  <algebra of dimension 1 over GF(5^2)> ]
gap> List( dec, Dimension );
[ 5, 1, 1, 1, 1, 1 ]
gap> dec:= BlockDecompositionSCAlgebraCentreOfGroupAlgebra(
>              tbl, GF(5) );
fail
```

3.11.3 SCAlgebraCentreOfBlock

▷ SCAlgebraCentreOfBlock(*tbl*, *p*, *b*) (function)
▷ SCAlgebraCentreOfBlock(*tbl*, *F*, *b*) (function)

Let *tbl* be an ordinary character table, and *b* be a positive integer.

If the second argument is a prime integer *p* then SCAlgebraCentreOfBlock returns a subalgebra of the s. c. algebra returned by SCAlgebraCentreOfGroupAlgebra (3.11.1) when this is called with *tbl* and the smallest field that contains the *p*-modular reductions of the *b*-th block idempotent of *tbl* in characteristic *p*.

If the second argument is a finite field *F* then the same holds, except that the coefficients domain of the algebra is *F*. If *F* is too small then fail is returned.

Example

```
gap> tbl:= CharacterTable( "M11" );
CharacterTable( "M11" )
gap> SCAlgebraCentreOfBlock( tbl, 5, 1 );
<algebra of dimension 5 over GF(5)>
gap> SCAlgebraCentreOfBlock( tbl, 5, 3 );
<algebra of dimension 1 over GF(5^2)>
gap> SCAlgebraCentreOfBlock( tbl, GF(5), 3 );
fail
```

Chapter 4

Utilities

This chapter lists the documentation about functions which have been developed for the current package but are of more general interest (see Section 4.1) and functions which are referenced by the package documentation, whose code is available in the main GAP library, but which are currently undocumented (see `ReductionToFiniteField` (4.2.4)).

4.1 Generalized Straight Line Programs

Generalized straight line programs (in the following abbreviated as *gslps*) are a generalization of the straight line programs that are introduced in Section (**Reference: Straight Line Programs**). Like the latter objects, gslps describe an efficient way for evaluating an abstract word at concrete generators. The difference is that gslps can be built from existing (generalized) straight line programs. So the advantages of using gslps are

- that available objects are reused,
- that the internal structure is retained, and
- that intermediate results of an evaluation are not kept longer than until the relevant straight line program inside is evaluated.

A gslp in GAP is represented by an object in the category `IsGeneralizedStraightLineProgram` (4.1.1). This object has exactly one of the following forms.

- It is a straight line program, that is, it lies in the category `IsStraightLineProgram` (**Reference: IsStraightLineProgram**), and evaluation at some group elements is defined by `ResultOfStraightLineProgram` (**Reference: ResultOfStraightLineProgram**).
- It is of "union" kind, that is, the defining data are a nonempty list of gslps, and evaluation at some group elements means to evaluate these defining programs at these group elements, and to return the concatenation of the results.
- It is of "compose" kind, that is, the defining data are a nonempty list of gslps, and evaluation at some group elements means to evaluate the first of them at these elements, then to evaluate the second of them at the result of the first evaluations, and so on, and to return the last result.

Gslps can be constructed using `GeneralizedStraightLineProgram` (4.1.2).

Defining attributes for gslps are `NrInputsOfGeneralizedStraightLineProgram` (4.1.4) and `DataOfGeneralizedStraightLineProgram` (4.1.3). The probably most interesting operation for gslps is `ResultOfGeneralizedStraightLineProgram` (4.1.6).

Special methods applicable to gslps are installed for the operations `IsInternallyConsistent` (**Reference: IsInternallyConsistent**), `ViewString` (**Reference: ViewString**), and `String` (**Reference: String**).

Here are typical situations where gslps arise:

1. Suppose that a list l of standard generators for a group G is given, and that we know a straight line program for computing generators l' of a maximal subgroup M of G from l. For example, these data may be taken from the ATLAS of Group Representations [WWT+]. If M is also a group for which the ATLAS of Group Representations contains generators and straight line programs, we may be interested in computing standard generators l'' for M from l'. For that, a second straight line program can be needed, and it makes sense to encode the computation of l'' from l via a gslp of "compose" kind.

2. Now suppose that we are in fact interested in a downward extension H of G, and that π is the natural epimorphism from H to G, which maps a list L, say, of standard generators of H to l. Then the above gslp can be applied to L, but the result L' may generate a proper subgroup of $\pi^{-1}(M)$ because some part of the kernel of π is missing. A list K of generators of the kernel of π can be described by a straight line program that takes L as its input, and it makes sense to encode the computation of the concatenation of L' and K from L via a gslp of "union" kind.

A remark on the name "generalized straight line program": We could have taken the viewpoint that these objects are the ones that one wants to deal with, and that they should therefore be called "straight line program", whereas GAP's straight line programs could be called "special straight line programs", However, several functions are applicable to GAP's straight line programs (such as `IntermediateResultOfSLP` (**Reference: IntermediateResultOfSLP**)) for which we do not intend to provide methods applicable to our generalized straight line programs.

4.1.1 IsGeneralizedStraightLineProgram

▷ `IsGeneralizedStraightLineProgram(`*obj*`)` (category)

Each generalized straight line program in GAP lies in the category `IsGeneralizedStraightLineProgram`. Examples are straight line programs, that is, objects in the category `IsStraightLineProgram` (**Reference: IsStraightLineProgram**).

Example

```
gap> gslp:= GeneralizedStraightLineProgram( "union",
>               [ [ [[1,2]]], 1 ], [ [[1,3]]], 1 ] ] );
<generalized straight line program>
gap> IsGeneralizedStraightLineProgram( gslp );
true
gap> slp:= StraightLineProgram( [[[1,2]]], 1 );
<straight line program>
gap> IsGeneralizedStraightLineProgram( slp );
true
gap> IsGeneralizedStraightLineProgram( [ slp, slp ] );
false
```

4.1.2 GeneralizedStraightLineProgram

▷ GeneralizedStraightLineProgram(*lines[, nrgens]*) (function)
▷ GeneralizedStraightLineProgram(*kind, list*) (function)

In the first form, *lines* must be a list of lists that defines a unique straight line program (see IsStraightLineProgram (**Reference: IsStraightLineProgram**)); in this case GeneralizedStraightLineProgram delegates to StraightLineProgram (**Reference: StraightLineProgram for a list of lines (and the number of generators)**).

In the second form, *kind* must be one of the strings "union" or "compose", and *list* must be a nonempty list such that each of its entries is either a gslp or a list *l*, say, such that CallFuncList (**Reference: CallFuncList**) applied to GeneralizedStraightLineProgram and l returns a gslp.

Example

```
gap> GeneralizedStraightLineProgram( [[[1,2]]], 1 );
<straight line program>
gap> GeneralizedStraightLineProgram( "union",
> [ [ [[[1,2]]], 1 ], [ [[[1,3]]], 1 ] ] );
<generalized straight line program>
gap> GeneralizedStraightLineProgram( "compose",
> [ [ [[[1,2]]], 1 ], [ [[[1,3]]], 1 ] ] );
<generalized straight line program>
```

4.1.3 DataOfGeneralizedStraightLineProgram

▷ DataOfGeneralizedStraightLineProgram(*gslp*) (attribute)

For a generalized straight line program *gslp* that is *not* a straight line program, DataOfGeneralizedStraightLineProgram returns a list of length two, the first entry being either "union" or "compose" and the second being the list of defining generalized straight line programs.

If *gslp* is a straight line program then this attribute is not set in *gslp*. There is no default method to compute the value if it is not stored.

Example

```
gap> gslp:= GeneralizedStraightLineProgram( "union",
>               [ [ [[[1,2]]], 1 ], [ [[[1,3]]], 1 ] ] );
<generalized straight line program>
gap> DataOfGeneralizedStraightLineProgram( gslp );
[ "union", [ <straight line program>, <straight line program> ] ]
```

4.1.4 NrInputsOfGeneralizedStraightLineProgram

▷ NrInputsOfGeneralizedStraightLineProgram(*gslp*) (attribute)

For a generalized straight line program *gslp*, this function returns the number of generators that are needed as input.

If *gslp* is a straight line program then it may be necessary that the value is set in the construction of *gslp*, see NrInputsOfStraightLineProgram (**Reference: NrInputsOfStraightLineProgram**). If *gslp* is not a straight line program then the value is determined by the (generalized) straight line programs from which *gslp* is constructed.

Example

```
gap> NrInputsOfGeneralizedStraightLineProgram(
>        GeneralizedStraightLineProgram( "union",
>            [ [ [[[1,2]]], 1 ], [ [[[1,3]]], 1 ] ] ) );
1
```

In order to avoid the introduction of unnecessary filters, we define `NrInputsOfGeneralizedStraightLineProgram` just as a synonym of `NrInputsOfStraightLineProgram` (**Reference: NrInputsOfStraightLineProgram**).

4.1.5 NrOutputsOfGeneralizedStraightLineProgram

▷ `NrOutputsOfGeneralizedStraightLineProgram(`*`gslp`*`)` (attribute)

For a generalized straight line program *`gslp`*, this function returns the number of elements returned by `ResultOfGeneralizedStraightLineProgram` (4.1.6) when *`gslp`* is evaluated.

Example

```
gap> NrOutputsOfGeneralizedStraightLineProgram(
>        GeneralizedStraightLineProgram( "union",
>            [ [ [[[1,2]]], 1 ], [ [[[1,3]]], 1 ] ] ) );
2
gap> NrOutputsOfGeneralizedStraightLineProgram(
>        GeneralizedStraightLineProgram( "compose",
>            [ [ [[[1,2]]], 1 ], [ [[[1,3]]], 1 ] ] ) );
1
```

4.1.6 ResultOfGeneralizedStraightLineProgram

▷ `ResultOfGeneralizedStraightLineProgram(`*`gslp, gens`*`)` (operation)

`ResultOfGeneralizedStraightLineProgram` evaluates the generalized straight line program (see `IsGeneralizedStraightLineProgram` (4.1.1)) *`gslp`* at the group elements in the list *`gens`*, as follows.

- If *`gslp`* is a straight line program then the value of `ResultOfStraightLineProgram` (**Reference: ResultOfStraightLineProgram**) is returned.
- If *`gslp`* is of "union" kind then `ResultOfGeneralizedStraightLineProgram` is applied to each of the involved generalized straight line programs, with second argument *`gens`*, and the concatenation of the results is returned.
- If *`gslp`* is of "compose" kind then `ResultOfGeneralizedStraightLineProgram` is first called with the first involved generalized straight line program and *`gens`*, then the operation is called with the second involved generalized straight line program and the result of this call, and so on; the last such result is returned.

Example

```
gap> gens:= [ (1,2,3,4,5,6) ];;
gap> gslp:= GeneralizedStraightLineProgram( "union",
>               [ [ [[[1,2]]], 1 ], [ [[[1,3]]], 1 ] ] );
```

```
<generalized straight line program>
gap> ResultOfGeneralizedStraightLineProgram( gslp, gens );
[ (1,3,5)(2,4,6), (1,4)(2,5)(3,6) ]
gap> gslp:= GeneralizedStraightLineProgram( "compose",
>               [ [ [[[1,2]]], 1 ], [ [[[1,3]]], 1 ] ] );
<generalized straight line program>
gap> ResultOfGeneralizedStraightLineProgram( gslp, gens );
[ () ]
```

In order to avoid the introduction of unnecessary operations, we define `ResultOfGeneralizedStraightLineProgram` just as a synonym of `ResultOfStraightLineProgram` (**Reference: ResultOfStraightLineProgram**).

4.1.7 EquivalentStraightLineProgram

▷ `EquivalentStraightLineProgram(`*gslp*`)` (attribute)

For a generalized straight line program *gslp*, `EquivalentStraightLineProgram` returns a straight line program such that evaluating *gslp* and this straight line program with `ResultOfGeneralizedStraightLineProgram` (4.1.6) yields the same output, for any list of input elements.

Example

```
gap> gslp:= GeneralizedStraightLineProgram( "union",
>               [ [ [[[1,2]]], 1 ], [ [[[1,3]]], 1 ] ] );
<generalized straight line program>
gap> slp:= EquivalentStraightLineProgram( gslp );
<straight line program>
gap> Display( slp );
# input:
r:= [ g1 ];
# program:
# return values:
[ r[1]^2, r[1]^3 ]
gap> gslp:= GeneralizedStraightLineProgram( "compose",
>               [ [ [[[1,2]]], 1 ], [ [[[1,3]]], 1 ] ] );
<generalized straight line program>
gap> slp:= EquivalentStraightLineProgram( gslp );
<straight line program>
gap> Display( slp );
# input:
r:= [ g1 ];
# program:
r[2]:= r[1]^2;
r[1]:= r[2];
# return values:
[ r[1]^3 ]
```

4.2 Miscellaneous

4.2.1 PRegularTable

▷ PRegularTable(*tbl*, *p*) (function)
▷ ComputedPRegularTables(*tbl*) (attribute)

For an ordinary character table *tbl* and a prime integer *p*, PRegularTable returns the same as CharacterTableRegular (**Reference: CharacterTableRegular**). The only difference is that the results of PRegularTable are cached in *tbl*, via the attribute ComputedPRegularTables.

The purpose of this attribute is to store *p*-regular character tables (see CharacterTableRegular (**Reference: CharacterTableRegular**) for those primes *p* for which the irreducible characters of the *p*-modular character table are not available. It may still be possible to provide the irreducibles of certain blocks, and for that, it is necessary to assign these characters to a suitable modular character table.

The cached tables are used for example by IBr (3.3.6).

4.2.2 PrintOverviewOfDefectOneNormalizers

▷ PrintOverviewOfDefectOneNormalizers(*tbl*) (function)

Let *tbl* be the ordinary character table of a finite group *G*, say. For each prime integer *p* that divides the order of *G* exactly once, PrintOverviewOfDefectOneNormalizers prints one or two lines of information about the normalizers of Sylow *p*-subgroups in *G*.

If such a Sylow *p*-normalizer is maximal in *G* and if the attribute Maxes (**CTblLib: Maxes**) is set in *tbl* then one line is printed for *p*, saying that the normalizer is maximal, and showing the name of the character table.

Otherwise, if the character table of the Sylow *p*-normalizer is known then the availability of the character table and its name are mentioned, and if not then an approximation of the structure is shown. If the attribute Maxes (**CTblLib: Maxes**) is set in *tbl* then also the list of maximal subgroups is shown that contain a Sylow *p*-normalizer.

Example

```
gap> PrintOverviewOfDefectOneNormalizers( CharacterTable( "A5" ) );
3: max. subgroup S3
5: max. subgroup D10
gap> PrintOverviewOfDefectOneNormalizers( CharacterTable( "A6" ) );
5: char. table available as D10,
   contained in max. subgroups A5, A6M2
gap> PrintOverviewOfDefectOneNormalizers( CharacterTable( "S10" ) );
7: structure (7x[6]).6,
   contained in max. subgroups S7xS3
```

4.2.3 FrobeniusCharacterValueExt

▷ FrobeniusCharacterValueExt(*value*, *p*) (function)

The GAP library function FrobeniusCharacterValue (**Reference: FrobeniusCharacterValue**) works only for those cyclotomics *value* whose conductor is coprime to *p*. In order to implement the ring homomorphism from all cyclotomics whose coefficients are coprime to *p*, we use the

fact that this ring homomorphism is defined via the Conway polynomials on the m-th roots of unity, for m coprime to p, and is defined by mapping all $p\hat{}k$-th roots of unity to the identity element of the finite field.

This means that for $q = p^k$ and $n = mq$, with m coprime to p, the image of ζ_n^i is equal to the image of ζ_m^j, where $i \equiv jq \bmod m$ holds.

Example

```
gap> FrobeniusCharacterValue( E(4), 2 );
fail
gap> FrobeniusCharacterValueExt( E(4), 2 );
Z(2)^0
```

4.2.4 ReductionToFiniteField

▷ `ReductionToFiniteField(`*value, p*`)` (function)

Let *value* be a cyclotomic whose coefficients over the rationals are in the ring $\mathbb{Z}_p$ of p-local numbers, where p is a prime integer.

`ReductionToFiniteField` returns either `fail` or a pair $[f,m]$, where f is a polynomial over the field with p elements and m is a positive integer.

In the latter case, the meaning is as follows. Let F be the finite field with p^m elements, given as a set of residue classes modulo the ideal I that is spanned by the Conway polynomial (see `ConwayPolynomial` (**Reference: ConwayPolynomial**)) of degree m in characteristic p. The coset $f+I$ represents the image of *value* under the ring homomorphism $*$ defined in Section 3.2, and F is the minimal field that contains this image.

`fail` is returned if the conductor of *value* is divisible by p, if the denominator of some coefficient of *value* is divisible by p. or if the Conway polynomial of the degree in question is not known and would be hard to compute, in the sense of `IsCheapConwayPolynomial` (**Reference: IsCheapConwayPolynomial**).

Example

```
gap> ReductionToFiniteField( E(5), 2 );
[ x_1^3, 4 ]
gap> ReductionToFiniteField( Sqrt(5), 2 );
[ Z(2)^0, 1 ]
gap> ReductionToFiniteField( E(7), 2 );
[ x_1, 3 ]
gap> ReductionToFiniteField( Sqrt(-7), 2 );
[ Z(2)^0, 1 ]
gap> ReductionToFiniteField( Sqrt(7), 2 );  # conductor is 28
fail
```

4.2.5 InfoBlocks

▷ `InfoBlocks` (info class)

Currently only the info levels 0, 1, and 2 are supported. If the level is 1 then info messages about `fail` results are printed. If the level is 2 then additionally info messages about the progress of computations are printed. The default info level of `InfoBlocks` is zero.

References

[AB80] J. L. Alperin and D. W. Burry. Block theory with modules. *J. Algebra*, 65(1):225–233, 1980. 49

[BH01] T. Breuer and E. Horváth. On block induction. *J. Algebra*, 242(1):213–224, 2001. 4, 11, 12

[BL18] T. Breuer and F. Lübeck. Browse, ncurses interface and browsing applications, Version 1.8.9. `http://www.math.rwth-aachen.de/~Browse`, Jun 2018. GAP package. 52, 56

[CCN+85] J. H. Conway, R. T. Curtis, S. P. Norton, R. A. Parker, and R. A. Wilson. *Atlas of finite groups*. Oxford University Press, Eynsham, 1985. Maximal subgroups and ordinary characters for simple groups, With computational assistance from J. G. Thackray. 23

[GAP19] GAP – Groups, Algorithms, and Programming, Version 4.10.2. `http://www.gap-system.org`, Jun 2019. 4

[GM00] R. Gow and J. Murray. Real 2-regular classes and 2-blocks. *J. Algebra*, 230(2):455–473, 2000. 16

[GW84] D. Gluck and T. R. Wolf. Brauer's height conjecture for p-solvable groups. *Trans. Amer. Math. Soc.*, 282(1):137–152, 1984. 40

[Isa76] I. M. Isaacs. *Character theory of finite groups*. Academic Press [Harcourt Brace Jovanovich Publishers], New York, 1976. Pure and Applied Mathematics, No. 69. 29, 35, 36

[JLPW95] C. Jansen, K. Lux, R. Parker, and R. Wilson. *An atlas of Brauer characters*, volume 11 of *London Mathematical Society Monographs. New Series*. The Clarendon Press Oxford University Press, New York, 1995. Appendix 2 by T. Breuer and S. Norton, Oxford Science Publications. 29

[KM13] R. Kessar and G. R. Malle. Quasi-isolated blocks and Brauer's height zero conjecture. *Ann. of Math. (2)*, 178(1):447, 2013. 40

[LP10] K. Lux and H. Pahlings. *Representations of groups*, volume 124 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, Cambridge, 2010. A computational approach. 13, 29, 30, 33, 34, 35, 36, 40, 42, 45, 46, 47, 48

[Mur06] J. Murray. Strongly real 2-blocks and the Frobenius-Schur indicator. *Osaka J. Math.*, 43(1):201–213, 2006. 50

[Nav98] G. Navarro. *Characters and blocks of finite groups*, volume 250 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, Cambridge, 1998. 28, 29, 33, 34, 35, 36, 40, 42, 46, 48, 50

[NST15] G. Navarro, R. Solomon, and P. H. Tiep. Abelian Sylow subgroups in a finite group, II. *J. Algebra*, 421:3–11, 2015. 40

[NT89] H. Nagao and Y. Tsushima. *Representations of finite groups*. Academic Press Inc., Boston, MA, 1989. Translated from the Japanese. 34

[Sch16] I. Schwabrow. *The center of a block*. Phd thesis, School of Mathematics, University of Manchester, 2016. 18, 23

[Sul08] I. Suleiman. Strongly real elements in sporadic groups and alternating groups. *Jordan J. Math. Stat.*, 1(2):97–103, 2008. 15

[Was97] L. C. Washington. *Introduction to cyclotomic fields*, volume 83 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, second edition, 1997. 30

[Whe94] W. W. Wheeler. Extended block induction. *J. London Math. Soc. (2)*, 49(1):73–82, 1994. 13, 49

[Wil98] R. A. Wilson. The McKay conjecture is true for the sporadic simple groups. *J. Algebra*, 207(1):294–305, 1998. 25

[WWT$^+$] R. A. Wilson, P. Walsh, J. Tripp, I. Suleiman, R. A. Parker, S. P. Norton, S. Nickerson, S. Linton, J. Bray, and R. Abbott. ATLAS of Finite Group Representations. `http://brauer.maths.qmul.ac.uk/Atlas/`. 62

Index